**HW # 5:  Theme: Data Definitions, Addressing Modes, Arrays**

*All main questions carry equal weight.*

*(Credit awarded to only those answers for which work has been shown.)*

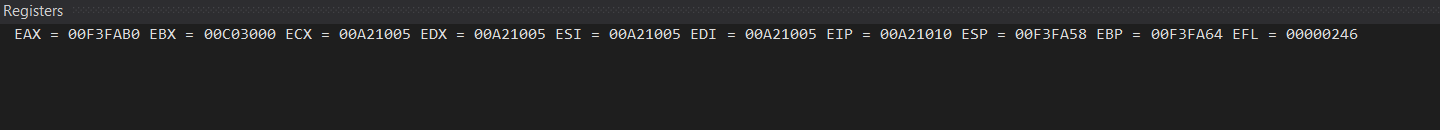
1. [Memory Map] Fill in the following memory diagram with the data provided below. Please assume that the data segment begins at 0x0045B200.

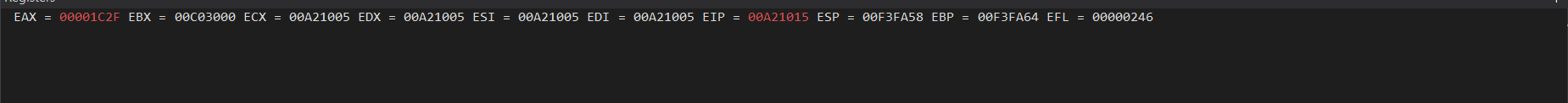
.data  
Rose       BYTE       0C7h  
Magnolia   WORD       0AB43h  
Cannas   DWORD     8612F137h, 0A2B2h

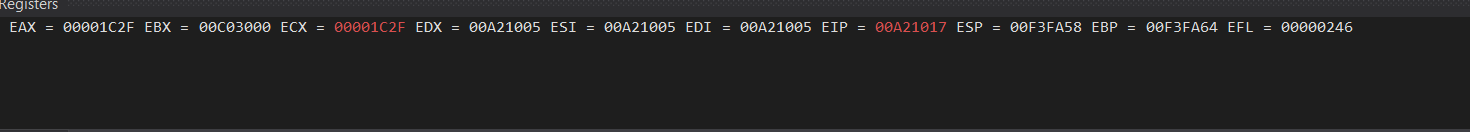
|  |  |  |
| --- | --- | --- |
| *Variable* | *Address* | *Data* |
| Rose | 0x0065A300 | ***C7*** |
| Magnolia | 0x0065A301 | ***43*** |
| Cannas | 0x0065A302 | ***AB*** |
|  | 0x0065A303  0x0065A304  0x0065A305  0x0065A306  0x0065A307  0x0065A308  0x0065A309  0x0065A310 | ***37***  F1  12  86  B2  A2  00  00 |

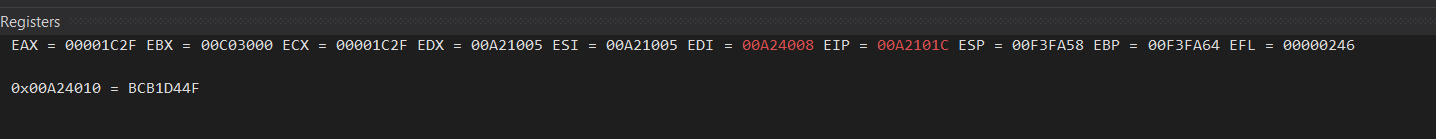
2. [Addressing Modes] Copy the following code into your assembly development environment and single-step through it.  For each single step execution, submit the screenshot.  For those instructions referencing memory, do the linear address computation by hand and typewrite it.

TITLE Addressing Modes              (main.asm)  
  
INCLUDE Irvine32.inc  
.data  
   alpha       DWORD       0A1B1C1D1h, 87654321h    
   beta        DWORD      67EED9FCh, 21A220C2h  
  gamma       DWORD       0BCB1D44Fh  
.code         
  
main PROC     
      mov eax, 1C2Fh;         Immediate  
      mov ecx, eax;           Register to Register      
      mov edi, OFFSET beta;   Immediate     
      mov [gamma], eax;       Direct      0x00A24010 = 00001C2F  
      mov esi, gamma;         Direct      0x00A24010  
      mov esi, 4;             Immediate   0x00A2400C  
      mov eax, beta[esi];     Indirect-offset     
      mov ebx, OFFSET alpha;  Immediate     
      mov eax, 4[ebx];        Indirect-displacement  0x00A24004 = 87654321   
 mov eax, [ebx];         Indirect    0x00A24000 = A1B1C1D1  
      mov eax,4[ebx][esi]; Base-Indirect-displacement 0x00A24008 = 67EED9FC  
exit  
main ENDP  
END main

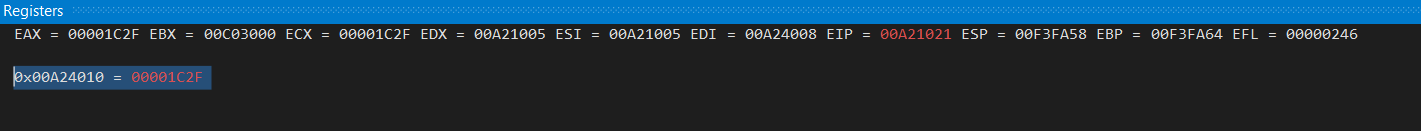
1.

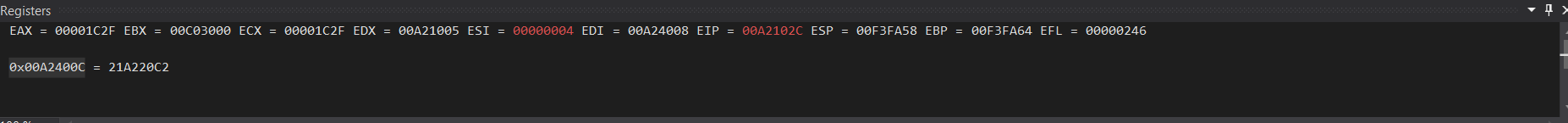
2.

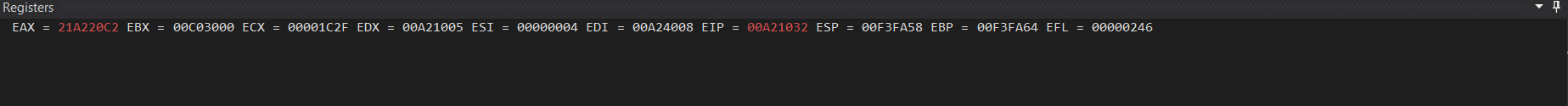
3.

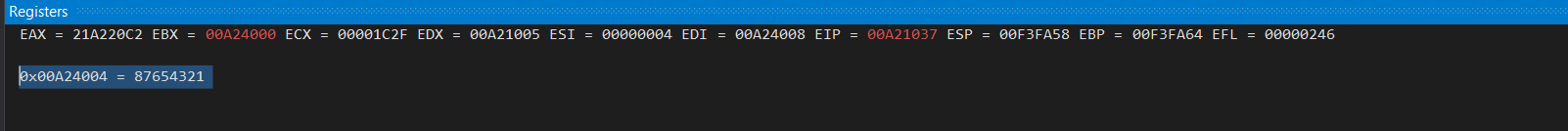
4.

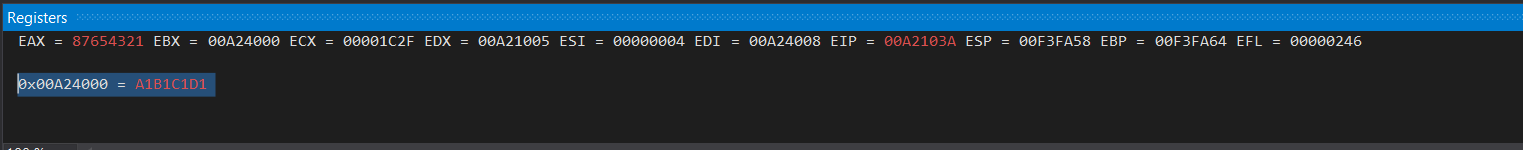
5.

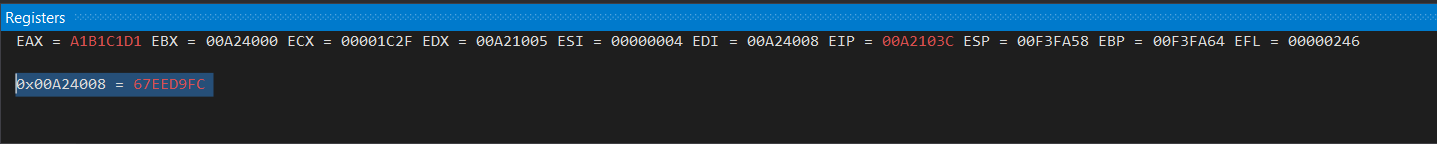


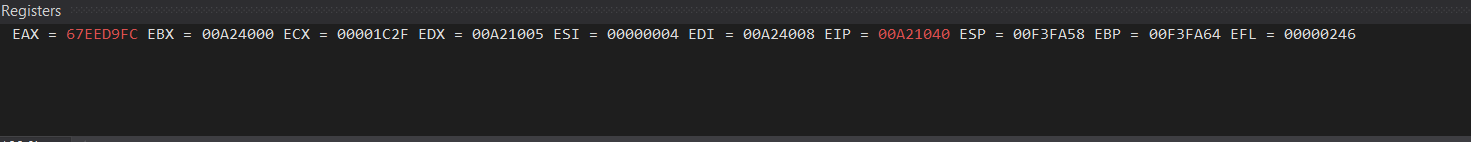
6.

7.

8.

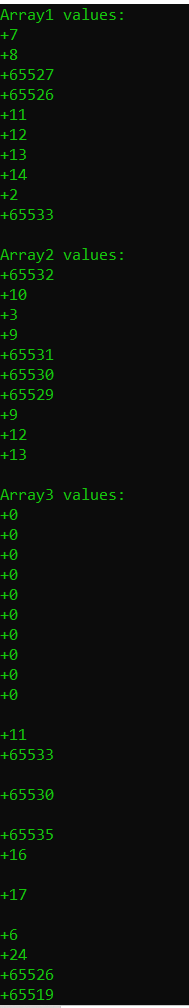
9.

10.

11.

3. [Indirect addressing] Write a program that first displays  all the elements of Array1, Array2 and Array3.  Then, the program should subtract all the odd indexed elements of Array2 from the odd indexed elements of Array1 and store the result in Array3; e.g. Array3 [7] = Array1 [7] - Array2 [7].  Next, it must add the even indexed elements of Array1 and Array 2 and store them in the corresponding even indexed elements of Array3, e.g. Array3 [4] = Array1 [4] + Array2 [4]. Next, display the elements of all the arrays after these operations.  Submit screenshots of the displays of the elements of all the arrays.  You can use WriteInt or WriteHex to display the elements of the arrays.  Fill in Array1 and Array2 each by your own ten numbers each using both positive and negative integers.

.data  
Array1      SWORD 7, …  
Array2      SWORD -4, …  
Array3      SWORD 10 DUP (?)



4. [Loops] Declare a signed word array. Write a program to print on screen the firs*t n* positive elements of the array, using the Loop instruction.  One sample array is given below.  You should test with other sample created arrays and with multiple sized arrays.  Make sure you have a good mix of positive and negative integers.

.data

MySigned Array  SWORD  -1, 78,  0AC, 4567, -7, -273, 92

* 1. Prompt user for integer *n*,
  2. Read the value of *n* from user input

Please use the “WriteInt” procedure, not “DumpRegs”. Other relevant procedures: “ReadInt” and “WriteString.” In your homework submission, please embed both the code and one screen shot for *n = 6*.

TITLE Addressing Modes (main.asm)

INCLUDE Irvine32.inc

.data

MySignedArray SWORD -1, 78, 0ACh, 4567, -7, -273, 92

message1 BYTE "n: ",0

n DWORD ?

.code

main PROC

mov edx, OFFSET message1

call WriteString

call ReadInt

mov n, eax

mov ecx, n

mov eax, 0

mov esi, OFFSET MySignedArray

L1:

mov eax, [esi]

call WriteInt

add esi, 4

loop L1

exit

main ENDP

END main

