Comp 3350: Computer Organization & Assembly Language
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 0x0038E000.

```
.data
Alpha       DWORD       1A2B3C4Dh, 12AB34CCh
Beta        BYTE        7Ch
Gamma       DWORD       9ABCDEF0h
Delta       BYTE        12h
```

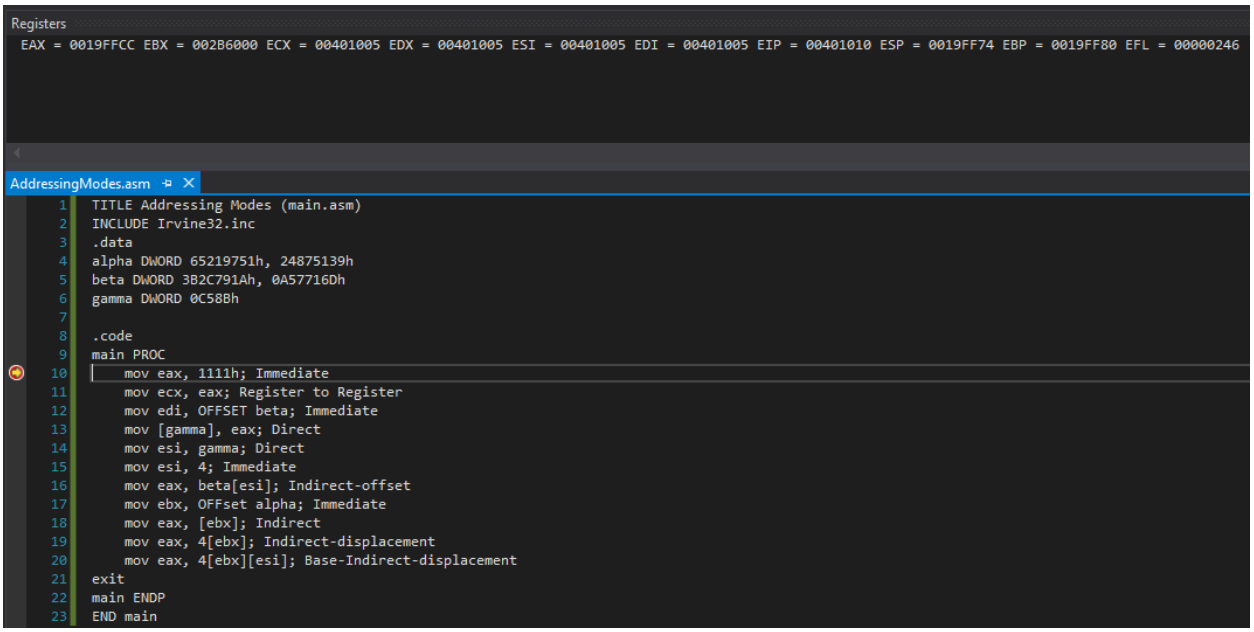
Address	*Variable*	*Data*
0038E000	***Alpha***	***4Dh***
0038E001	***Alpha***	***3Ch***
0038E002	***Alpha***	***2Bh***
0038E003	***Alpha***	***1Ah***
0038E004	***Alpha***	***CCh***
0038E005	***Alpha***	***34h***
0038E006	***Alpha***	***ABh***
0038E007	***Alpha***	***12h***
0038E008	***Beta***	***7Ch***
0038E009	***Gamma***	***F0h***
0038E00A	***Gamma***	***DEh***
0038E00B	***Gamma***	***BCh***
0038E00C	***Gamma***	***9Ah***
0038E00D	***Delta***	***12h***

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 (typewritten).

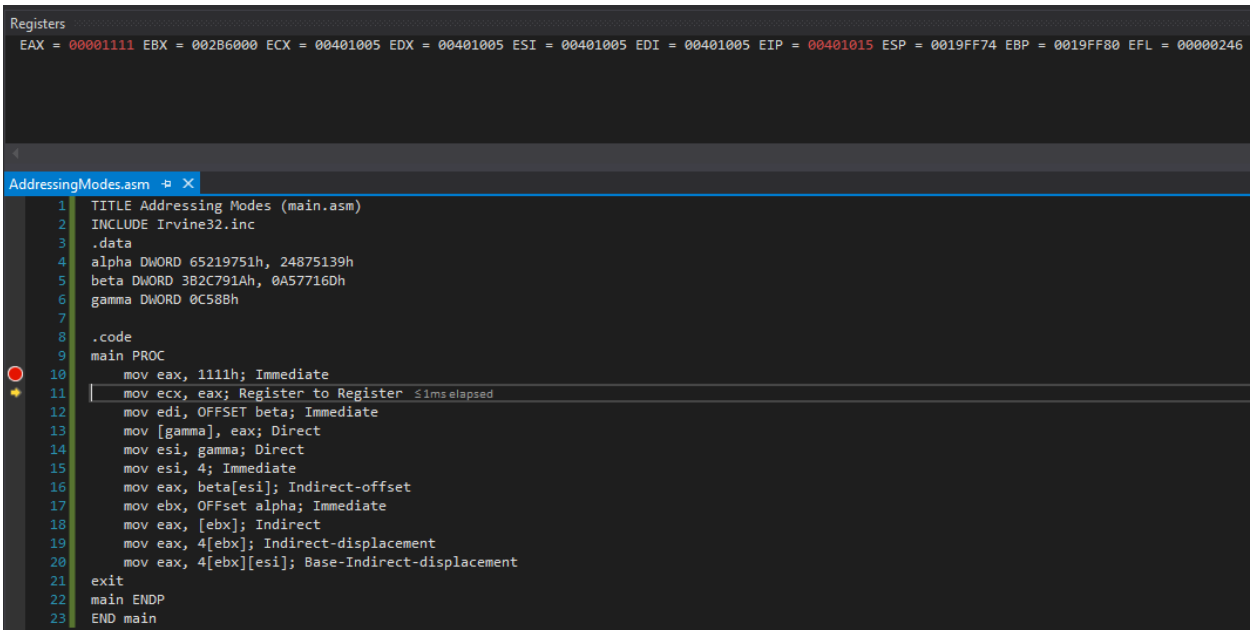
```
TITLE Addressing Modes                  (main.asm)
INCLUDE Irvine32.inc
.data
alpha       DWORD       65219751h, 24875139h
beta        DWORD       3B2C791Ah, 0A57716Dh
gamma       DWORD       0C58Bh

.code
main PROC
      mov eax, 1111h;               Immediate
      mov ecx, eax;                 Register to Register
      mov edi, OFFSET beta;         Immediate
```

```
        mov [gamma], eax;           Direct
        mov esi, gamma;             Direct
        mov esi, 4;                 Immediate
        mov eax, beta[esi];         Indirect-offset
        mov ebx, OFFSET alpha;      Immediate
        mov eax, [ebx];             Indirect
        mov eax, 4[ebx];            Indirect-displacement
        mov eax, 4[ebx][esi];       Base-Indirect-displacement
    exit
    main ENDP
    END main
```

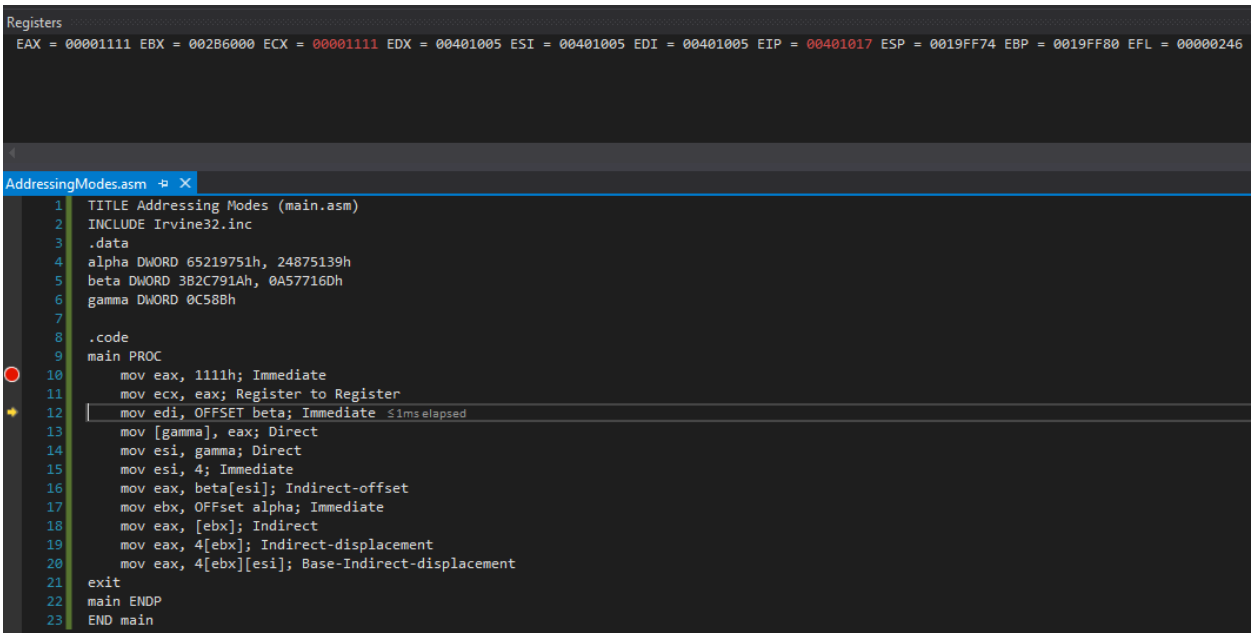
Before line 10:



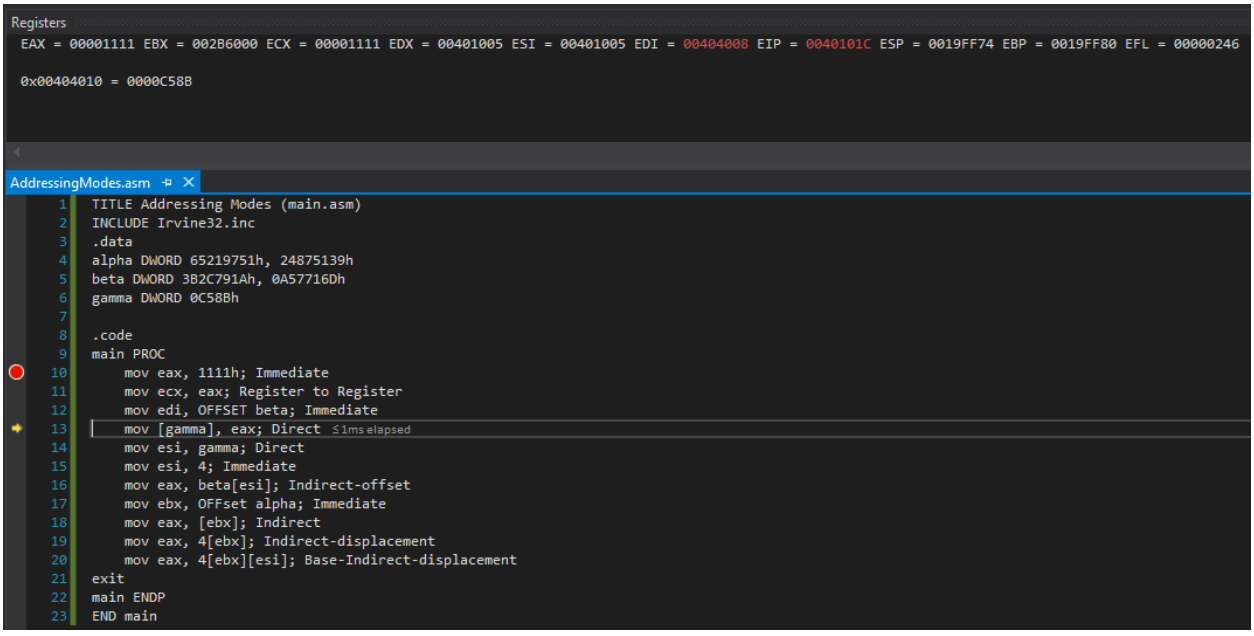
After line 10:



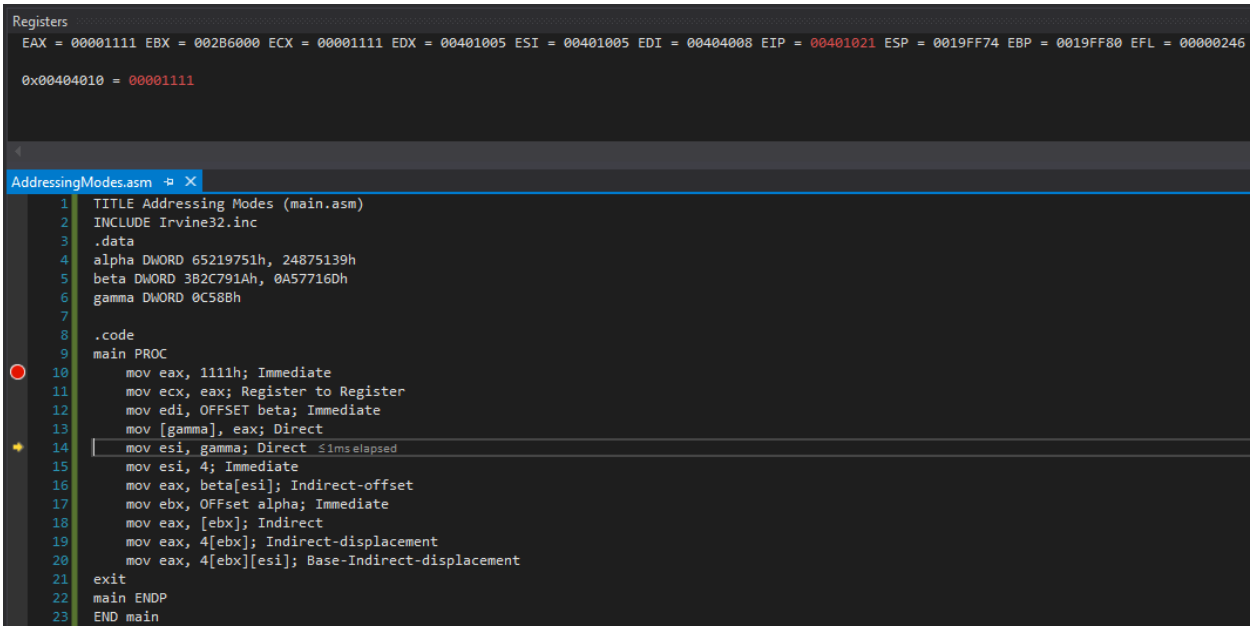
After line 11:



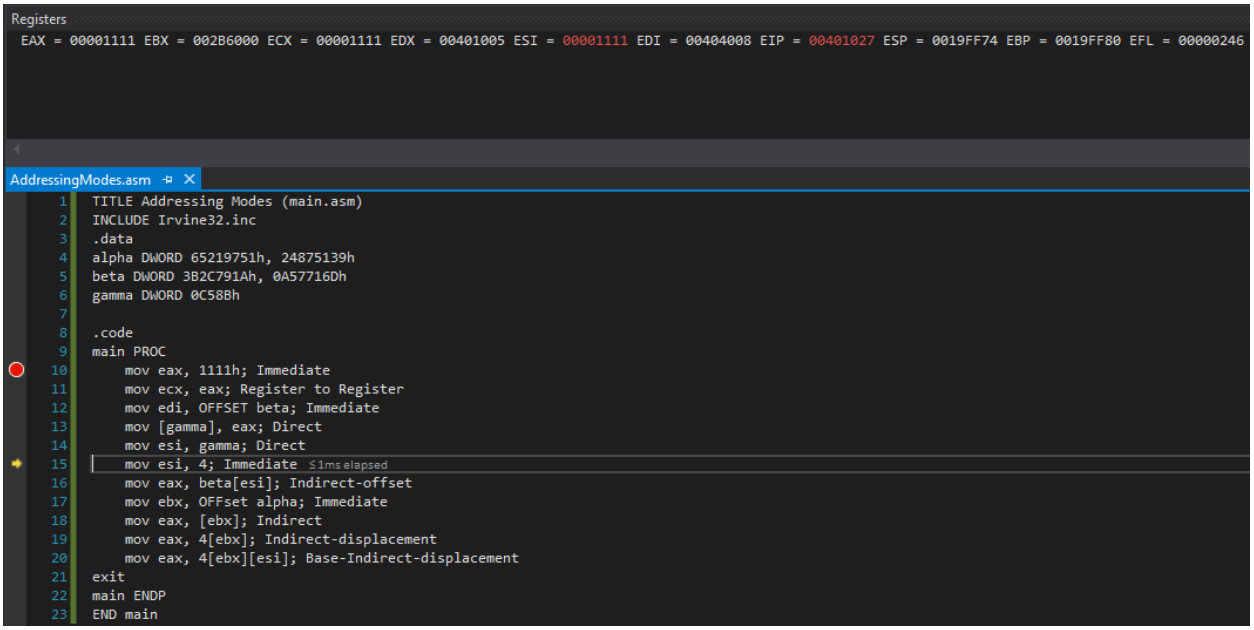
After line 12:



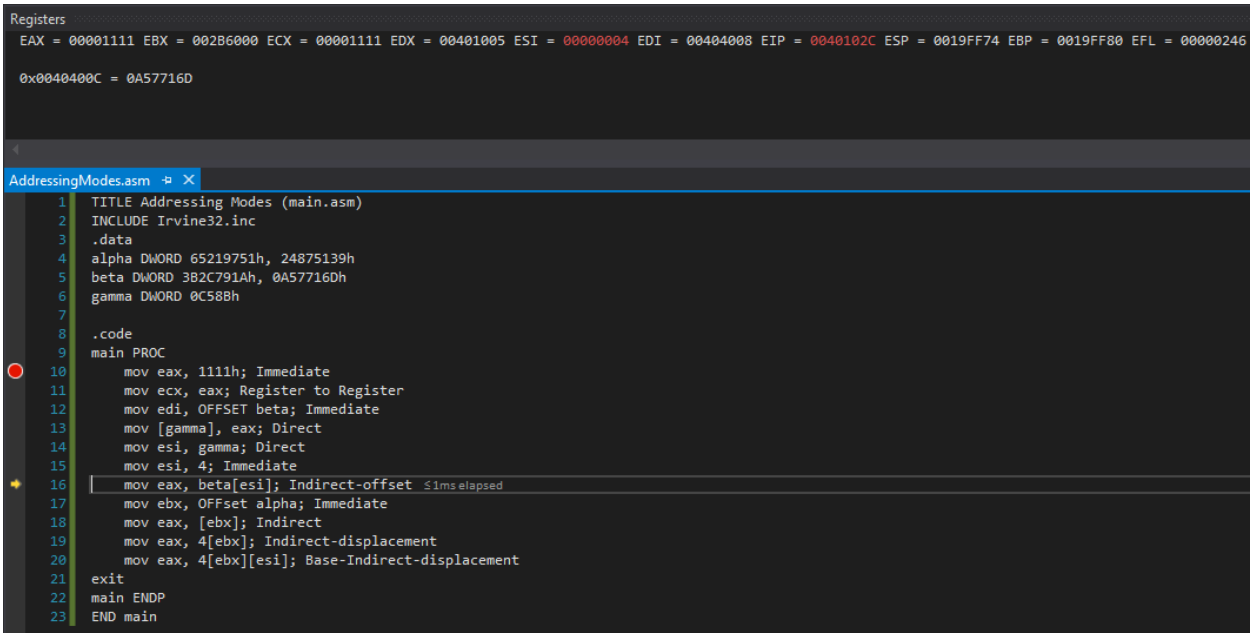
After line 13:



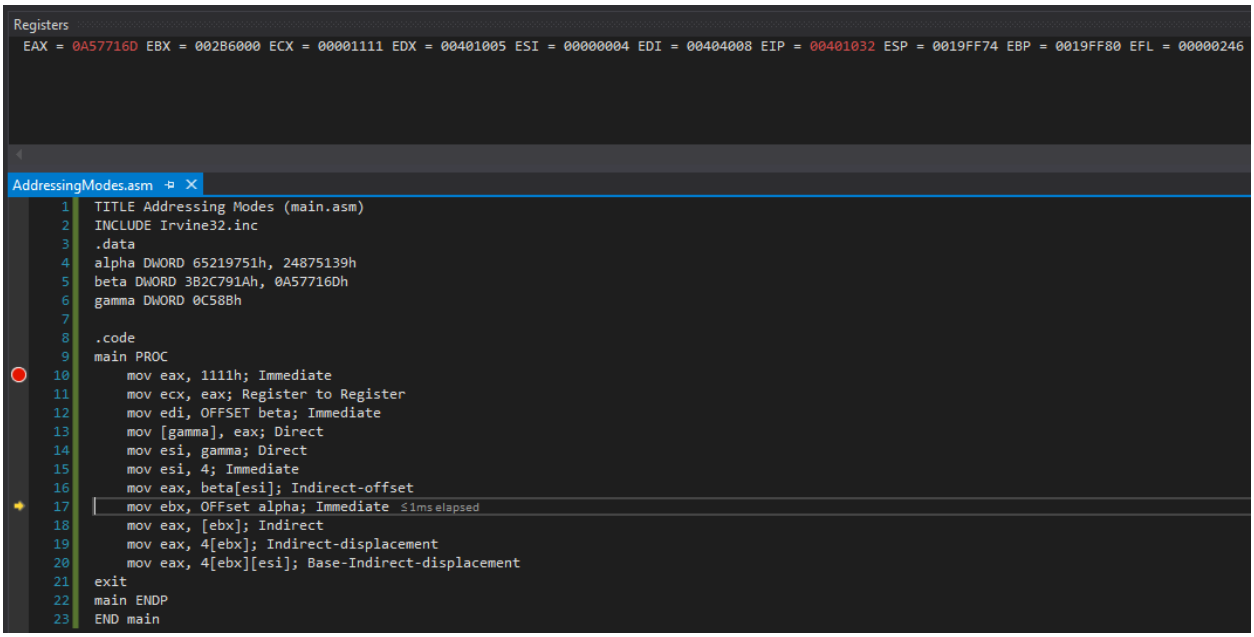
After line 14:



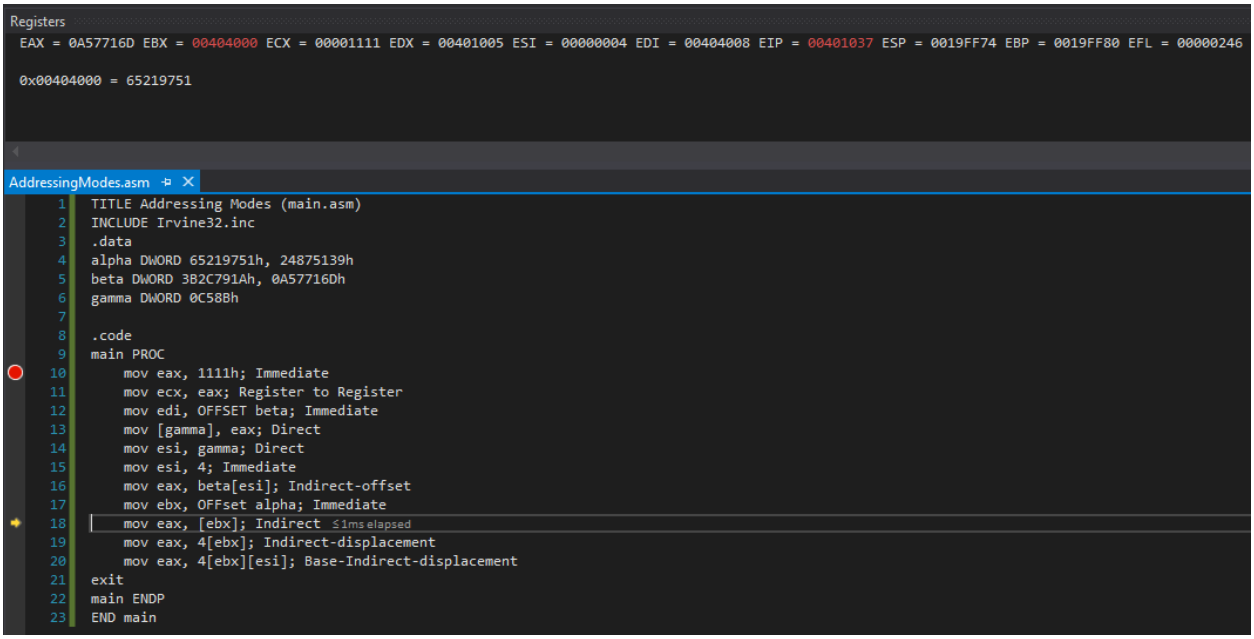
After line 15:



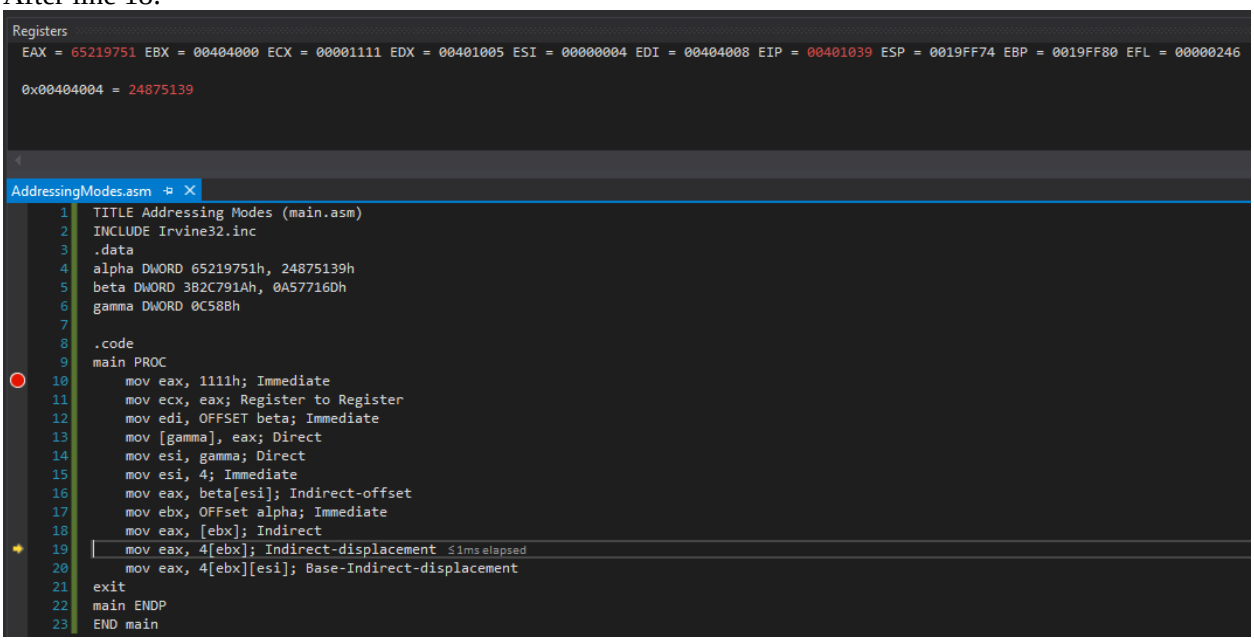
After line 16:



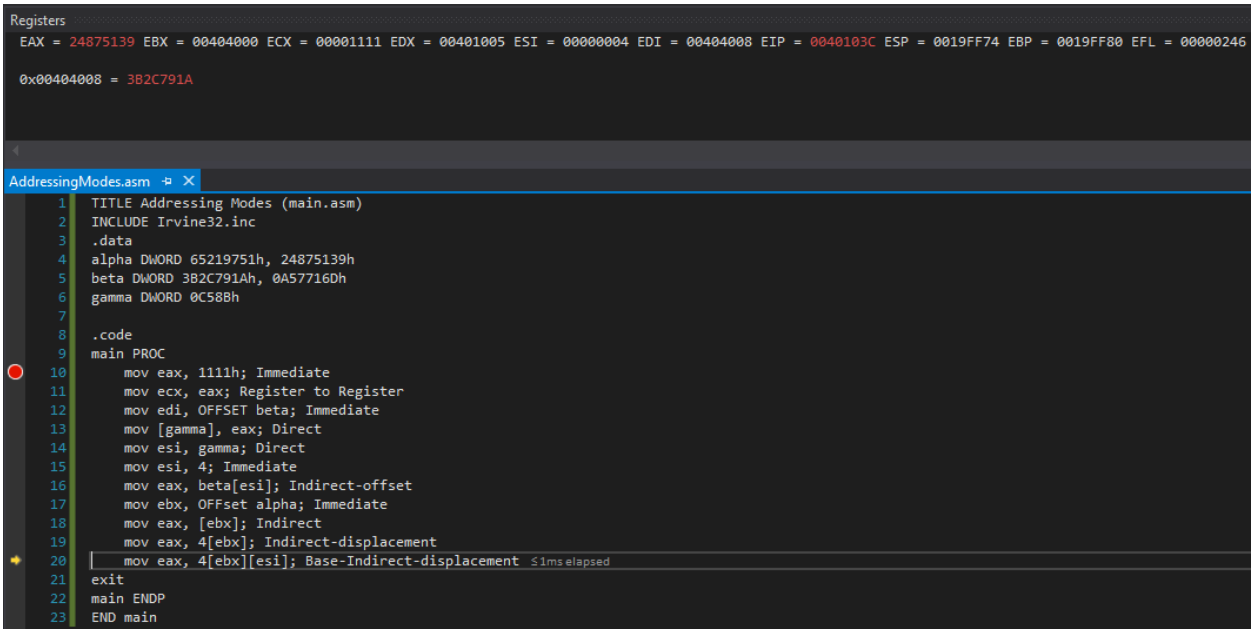
After line 17:



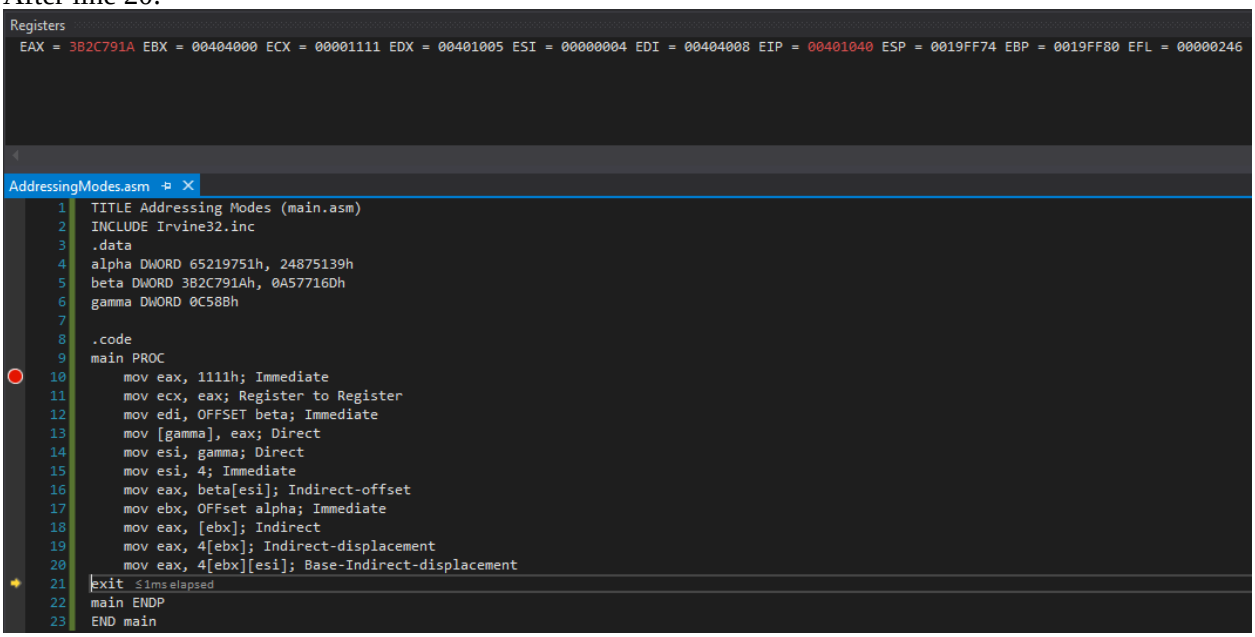
After line 18:



After line 19:



After line 20:



Linear Addresses for Each:

```
mov eax, A1C3h;
mov ecx, eax;
mov edi, OFFSET beta;
mov [gamma], eax;            0x00404010 = 1111h
mov esi, gamma;              esi = 01111h (0x00404010)
mov esi, 4;
mov eax, beta[esi];          beta + 4, 00404012 (0x00404012)
mov ebx, OFFSET alpha;
mov eax, [ebx];              eax = 65219751 (0x00404000)
```

```
mov eax, 4[ebx];              4 + ebx, eax = 24875139 (0x00404004)
mov eax, 4[ebx][esi];         4 + ebx + esi, eax = 3B2C791A (0x00404008)
```

3. [Indirect addressing] Write a program that subtracts the corresponding even indexed elements of Array2 from Array1 and stores the results in Array3; e.g. for the 8th element, Array3 [7] ← Array1 [7] - Array2 [7]. Note that Array3 will have about half the number of elements of the other two arrays. Include commands to display the elements of all the arrays. Submit screenshot 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.

```
TITLE SubArrays - Cameron Mathis (AddArrays.asm)
INCLUDE Irvine32.inc
.data
Array1 WORD 7h, 8h, 7h, 6h, 5h, 4h, 3h, 2h, 1h, 0h, 1h
Array2 WORD 4h, 2h, 3h, 4h, 5h, 6h, 7h, 8h, 9h, 8h, 7h
Array3 WORD 10 DUP (?)
a1message BYTE "Array1 Values: ",0
a2message BYTE "Array2 Values: ",0
a3message BYTE "Array3 Values: ",0

.code
main PROC
    mov esi, OFFSET Array1
    mov ecx, 5
    mov edx, OFFSET Array2
    mov ebx, OFFSET Array3
L1:
    call ArraySub
    add edx, TYPE DWORD
    add esi, TYPE DWORD
    mov [ebx], ax
    add ebx, TYPE DWORD
    loop L1

    call crlf
    mov edx, OFFSET a1message
    call WriteString
    call crlf
    movzx eax, [Array1 + 0]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 2]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 4]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 6]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 8]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 10]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 12]
    call WriteHex
    call crlf
    movzx eax, [Array1 + 14]
    call WriteHex
```

```
52      call crlf
53      movzx eax, [Array1 + 16]
54      call WriteHex
55      call crlf
56      movzx eax, [Array1 + 18]
57      call WriteHex
58      call crlf
59      movzx eax, [Array1 + 20]
60      call WriteHex
61      call crlf
62
63      call crlf
64      mov edx, OFFSET a2message
65      call WriteString
66      call crlf
67      movzx eax, [Array2 + 0]
68      call WriteHex
69      call crlf
70      movzx eax, [Array2 + 2]
71      call WriteHex
72      call crlf
73      movzx eax, [Array2 + 4]
74      call WriteHex
75      call crlf
76      movzx eax, [Array2 + 6]
77      call WriteHex
78      call crlf
79      movzx eax, [Array2 + 8]
80      call WriteHex
81      call crlf
82      movzx eax, [Array2 + 10]
83      call WriteHex
84      call crlf
85      movzx eax, [Array2 + 12]
86      call WriteHex
87      call crlf
88      movzx eax, [Array2 + 14]
89      call WriteHex
90      call crlf
91      movzx eax, [Array2 + 16]
92      call WriteHex
93      call crlf
94      movzx eax, [Array2 + 18]
95      call WriteHex
96      call crlf
97      movzx eax, [Array2 + 20]
98      call WriteHex
99      call crlf
100
101     call crlf
```

```
102     mov edx, OFFSET a3message
103     call WriteString
104     call crlf
105     movzx eax, [Array3 + 0]
106     call WriteHex
107     call crlf
108     movzx eax, [Array3 + 4]
109     call WriteHex
110     call crlf
111     movzx eax, [Array3 + 8]
112     call WriteHex
113     call crlf
114     movzx eax, [Array3 + 12]
115     call WriteHex
116     call crlf
117     movzx eax, [Array3 + 16]
118     call WriteHex
119     call crlf
120     movzx eax, [Array3 + 20]
121     call WriteHex
122     call crlf
123
124 exit
125 main ENDP
126
127 ArraySub PROC USES esi
128 mov eax, 0
129 add eax, [esi]
130 sub eax, [edx]
131
132 ret
133 ArraySub ENDP
134
135 END main
```

4. [Loops] Write a program to compute the sum of first *n* integers of the series: *Sum = 1 + 2 - 3 + 4 -5 + 6 -7 +8 - 9...* Your program must:

    a. Prompt user for integer *n*,
    b. Read the value of *n* from user input
    c. Calculate *Sum*, and;
    d. Print *Sum* on screen.

    Please use the "WriteInt" procedure, not "DumpRegs". Other relevant procedures: "ReadInt" and "WriteString." The calculation can be done in many ways, and all submissions that evidence proper programming practice are acceptable. In your homework submission, please embed both the code and one screen shot for *n = 10*.

```
1  TITLE Program - Cameron Mathis (Program.asm)
2  INCLUDE Irvine32.inc
3
4  .data
5  message1 BYTE "Please enter your desired number: ",0
6  message2 BYTE "Your output is: ",0
7  intVal SDWORD ?
8
9  .code
10 main PROC
11     mov edx, OFFSET message1
12     call WriteString
13     call ReadInt
14     mov intVal, eax
15     call crlf
16     mov ecx, intVal
17     mov eax, 0
18
19 L1:
20     add eax, ecx
21     dec ecx
22     .if ecx == 2
23         dec ecx
24     .endif
25     loop L1
26     mov ecx, intVal
27
28 L2:
29     dec ecx
30     sub eax, ecx
31     .if ecx == 3
32         dec ecx
33         dec ecx
34     .endif
35     loop L2
36
37     inc eax
38
39     mov edx, OFFSET message2
40     call WriteString
41     call WriteInt
42     call crlf
43
44 exit
45 main ENDP
46 END main
```

Microsoft Visual Studio Debug Console

```
Please enter your desired number: 10

Your output is: +7
```