HW5: Theme: Addressing modes

1. Fill in the following memory diagram with the data provided below. Please assume that the data will begin being assigned at 00404000, which is the bottom row of the grid.

```
.data
Alpha  WORD   54, 76h <- is it supposed to be 54 decimal??
Beta   BYTE   1h
Gamma  DWORD 56789h
Delta  BYTE   2h
```

AddressVariable	*Data*
00404009***Delta***	***02h***
00404008***Gamma***	***00h***
00404007***Gamma***	***05h***
00404006***Gamma***	***67h***
00404005***Gamma***	***89h***
00404004***Beta***	***01h***
00404003***Alpha***	***00h***
00404002***Alpha***	***76h***
00404001***Alpha***	***00h***
00404000***Alpha***	***54***

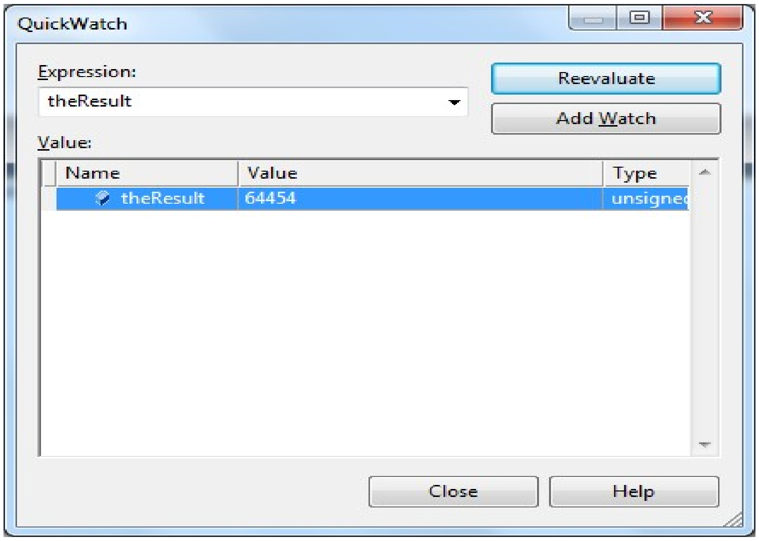
2. Copy the following code into your assembly development environment and single-step through it. For those instructions referencing memory, write the linear address.

```
TITLE Homework 5, Question 1                    (main.asm)

; Description: Memory reference exercise.
; Author: Matthew J Swann
; Version: 1.0, 2012-08-02

INCLUDE Irvine32.inc
.data
alpha  DWORD         1h, 2h
beta   DWORD         3h, 4h
gamma  DWORD         5

.code
main PROC
       mov eax, 0Ah;              Immediate
       mov ecx, eax;              register to register
       mov edi, OFFSET beta;      Immediate
       mov [gamma], eax;          Indirect      00404013
       mov esi, [gamma];          Direct        00404013
       mov esi, 4;                Immediate
       mov eax, beta[esi];        Indirect-offset      0040400C
       mov ebx, OFFSET alpha;     Immediate
       mov eax, [ebx];            Indirect      00404000
       mov eax, 4[ebx];           Indirect-displacement      00404004
       mov eax, 4[ebx][esi];      Base-Indirect-displacement  00404008
       mov eax, 8[ebx][esi];      Base-Indirect-displacement  0040400C
       mov eax, 12[ebx][esi];     Base-Indirect-displacement   00404010
```

```
exit
main ENDP
END main
```

3. Draft the `.code` section of a program that subtracts each element of an array from a single value. The `.data` section of the code is provided below. The program should: 1) iterate through "*theArray*", 2) subtracts the value at each index from "*theSource*', and 3) stores the resulting value in "*theResult*". Please embed your code into your homework submission along with a screenshot showing the final value.

```
TITLE Sum of elements of a DWORD array
; Author: Matthew J Swann
; Version 1.0, 2012-08-02

INCLUDE Irvine32.inc
.data
theArray WORD 1h, 2h, 4h, 8h, 16h, 32h, 64h, 128h, 256h
theSource WORD 0FFFFh
theResult WORD ?

.code

       main PROC

       MOV edi, OFFSET theArray
       MOV cx, LENGTHOF theArray
       MOV ax, theSource

       L1:
              SUB ax, [edi]
              ADD di, TYPE theArray
       LOOP L1

       exit
main ENDP
END main      64454d is FBC6h!!!
```

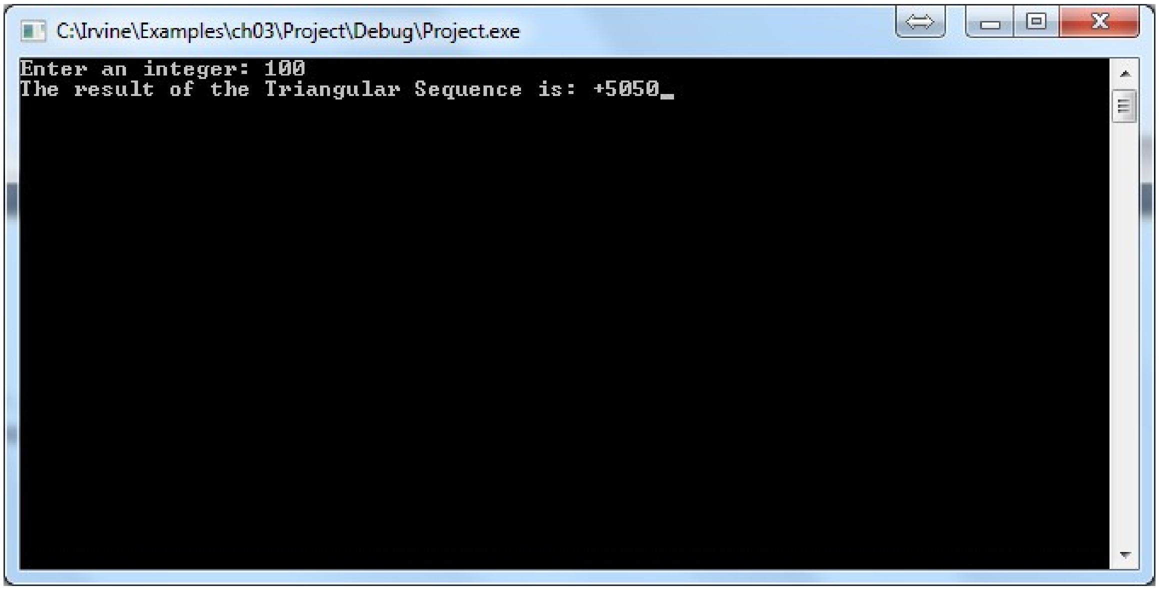


Expression:
theResult

Value:

Name	Value	Type
theResult	64454	unsigned

4. A Triangular Sequence is calculated as the summation of all positive integer values up to and including *n*. As such, $t_n = n + (n - 1) + (n - 2) + ... + 2 + 1$. Draft a program that:
1) Prompts the user for integer input,
2) Takes integer input from the user,
3) Stores that value in a variable called "n",
4) Calculates t_n, and;
5) Prints the final value to the screen.
Use the "call WriteInt" invocation, not "call DumpRegs". Other invocations that are likely necessary include: "call ReadInt", "call WriteString." The calculation can be done numerous ways, and all submissions that evidence proper programming practice are acceptable (including loops, recursion, etc.). In your homework submission, please embed both the code and one screen shot with user input supplied as 100.



```
Enter an integer: 100
The result of the Triangular Sequence is: +5050_
```

```
TITLE Triangular Sequence

INCLUDE Irvine32.inc
.data
       n DWORD ?
       prompt BYTE "Enter an integer: ", ?
       result BYTE "The result of the Triangular Sequence is: ", ?
.code
       main PROC
       MOV edx, OFFSET prompt
       call WriteString
       call ReadInt

       MOV n, eax
       MOV ecx, n
       MOV eax, 0
       L1:
              ADD eax, ecx
       LOOP L1
       MOV edx, OFFSET result
       call WriteString
       call WriteInt
       exit
main ENDP
END main
```