

LAB 04

Working with Data related Operator and Directives, Addressing



STUDENT NAME

ROLL NO

SEC

LAB ENGINEER'S SIGNATURE & DATE

MARKS AWARDED: /

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
(NUCES), KARACHI

Prepared by: Qurat ul ain & Rabia Ansari

Lab Session 04: Working with Data Related Operators and Directives, Addressing

OBJECTIVES:

- Observing effect of Arithmetic Instructions on Flag Register
- Direct-offset operands
- OFFSET operator
- PTR operator
- TYPE operator
- LENGTHOF operator
- SIZEOF operator
- Indirect operands
- Indexed operands

Effect of Arithmetic Instructions on Flag Registers

- Status flags are updated to indicate certain properties of the result
- Once a flag is set, it remains in that state until another instruction that affects the flags is executed

Z-Zero Flag:

The Zero flag is set when the result of an operation produces zero in the destination operand.

```
mov cx,1
sub cx,1          ; CX = 0, ZF = 1
mov ax,0FFFFh
inc ax            ; AX = 0, ZF = 1
inc ax            ; AX = 1, ZF = 0
```

Remember...

- A flag is **set** when it equals 1.
- A flag is **clear** when it equals 0.

C-Carry Flag:

This flag is set, when there is a carry out of MSB in case of addition and borrow in case of subtraction.

The Carry flag is set when the result of an operation generates an unsigned value that is out of range (too big or too small for the destination operand).



```

mov al,0FFh
add al,1                ; CF = 1, AL = 00

; Try to go below zero:

mov al,0
sub al,1                ; CF = 1, AL = FF

```

S-Sign Flag:

This flag indicates the sign of the result of an operation. A 0 for positive number and 1 for a negative number.

<pre> mov AL, 15 add AL, 97 clears the sign flag as the result is 112 (or 0111000 in binary) </pre>	<pre> mov AL, 15 sub AL, 97 sets the sign flag as the result is -82 (or 10101110 in binary) </pre>
---	--

AC-Auxiliary Carry Flag:

This flag is set, if there is a carry from the lowest nibble, i.e., bit three during addition, or borrow for the lowest nibble, i.e. bit three, during subtraction.

Suppose we add 1 to 0Fh. The sum (10h) contains a 1 in bit position 4 that was carried out of bit position 3:

```

mov al,0Fh
add al,1                ; AC = 1

```

P Parity Flag:

The Parity flag (PF) is set when the least significant byte of the destination has an even number of 1 bits. The following ADD and SUB instructions alter the parity of AL:

```

mov al,10001100b
add al,00000010b        ; AL = 10001110, PF = 1
sub al,10000000b        ; AL = 00001110, PF = 0

```

O-Over flow Flag:



The Overflow flag is set when the result of a signed arithmetic operation over-flows or

underflows the destination operand. For example, the largest possible integer signed byte value is +127; adding 1 to it causes overflow:

```
mov al,+127
add al,1           ; OF = 1
```

Similarly, the smallest possible negative integer byte value is 128. Subtracting 1 from it causes underflow. The destination operand value does not hold a valid arithmetic result, and the Overflow flag is set:

```
mov al,-128
sub al,1           ; OF = 1
```

Direct-offset Operands:

You can add a displacement to the name of a variable, creating a direct-offset operand.

Example:

```
.data
arrayB          BYTE
10h,20h,30h,40h,50h  arrayW
WORD 100h,200h,300h

.code
mov al,arrayB      ; AL = 10h
mov al,[arrayB+1]   ; AL = 20h
mov ax,arrayW       ; AX = 100h
mov ax,[arrayW+2]    ; AX = 200h
```

Similarly, the second element in a doubleword array is 4 bytes beyond the first one.

DATA-RELATED OPERATORS AND DIRECTIVES

OFFSET Operator:

The OFFSET operator returns the offset of a data label.

Syntax:

MOV reg32, OFFSET mem ; reg32 points to count



Example:

```
.data
bVal BYTE ?
wVal WORD ?
dVal DWORD ?
dVal2 DWORD ?
```

If bVal is located at offset 00404000h, we would get:

```
mov esi, OFFSET bVal      ; ESI = 00404000
mov esi, OFFSET wVal      ; ESI = 00404001
mov esi, OFFSET dVal      ; ESI = 00404003
mov esi, OFFSET dVal2     ; ESI = 00404007
```

PTR Operator:

We can use the PTR operator to override the declared size of an operand. Note PTR must be used in combination with one of the standard assembler data types.

For example, that we would like to move the lower 16 bits of a doubleword variable named myDouble into AX. The assembler will not permit the following move because the operand sizes do not match:

```
.data
myDouble DWORD 12345678h
.code
mov ax, myDouble    ; error
```

But the WORD PTR operator makes it possible to move the low-order word (5678h) to AX:

```
mov ax, word ptr myDouble      ; AX = 5678H
```

and higher word (1234h) to AX:

```
mov dx, word ptr myDouble+2    ; DX = 1234H
```

Moving Smaller Values into Larger Destinations

We might want to move two smaller values from memory to a larger destination operand. In the next example, the first word is copied to the lower half of EAX and the second word is copied to the upper half.

The DWORD PTR operator makes this possible:



```
.data
wordList WORD 5678h, 1234h
.code
mov eax, DWORD PTR wordList           ; EAX = 12345678h
```

TYPE Operator:

The TYPE operator returns the size, in bytes, of a single element of a variable.

Syntax:

MOV reg16, TYPE mem

Example 1:

```
.data
var1 BYTE ?      ; TYPE var1 = 1
var2 WORD ?      ; TYPE var2 =
2 var3 DWORD ?   ; TYPE var3 =
4 var4 QWORD ?   ; TYPE var4 = 8
```

Example 2:

```
.data
var1 BYTE 20h
var2 WORD
1000h var3
DWORD ?
var4 BYTE 10, 20, 30, 40, 50
msg BYTE 'File not found', 0
.code
mov ax, type var1      ; AX = 0001
mov ax, type var2      ; AX = 0002
mov ax, type var3      ; AX = 0004
mov ax, type var4      ; AX = 0001
mov ax, type msg       ; AX = 0001
```

LENGTHOF Operator:

The LENGTHOF operator counts the number of individual elements in a variable that has been defined using DUP.

Syntax:



MOV reg16 , LENGTHOF mem

Example:

```
.data
val1 WORD 1000h
val2 SWORD 10, 20, 30
array WORD 10 DUP(?),0
array2 WORD 5 DUP(3 DUP(0))
message BYTE 'File not found', 0

.code
mov ax, LENGTHOF val1 ; AX = 1
mov ax, LENGTHOF val2 ; AX = 3
mov ax, LENGTHOF array ; AX = 11
mov ax, LENGTHOF array2 ; AX = 15
mov ax, LENGTHOF message ; AX = 15
```

SIZEOF Operator:

The SIZEOF operator returns the number of bytes an array takes up. It is similar in effect to multiplying LENGTHOF with TYPE.

Syntax:

MOV reg16/32 , SIZEOF mem

Example:

```
.data
intArray WORD 32 DUP(0)
.code
mov eax,SIZEOF intArray ; EAX = 64
```

Indirect Operands

In protected mode, an indirect operand can be any 32-bit general-purpose register (EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP) surrounded by brackets. The register is assumed to contain the address of some data.

Example:

```
.data
byteVal BYTE 10h
.code
mov esi,OFFSET byteVal
mov al,[esi] ; AL = 10h
```



If the destination operand uses indirect addressing, a new value is placed in memory at the location pointed to by the register.

```
mov [esi],bl
```

Using PTR with Indirect Operands

```
inc [esi] ; error: operand must have size
```

The assembler does not know whether ESI points to a byte, word, doubleword, or some other size. The PTR operator confirms the operand size:

```
inc BYTE PTR [esi]
```

Arrays

Indirect operands are ideal tools for stepping through arrays.

Example:

```
.data
```

```
arrayB BYTE 10h,20h,30h
```

```
.code
```

```
mov esi,OFFSET arrayB
```

```
mov al,[esi] ; AL = 10h
```

```
inc esi
```

```
mov al,[esi] ; AL = 20h
```

If we use an array of 16-bit integers, we add 2 to ESI to address each subsequent array element.

```
.data
```

```
arrayW WORD 1000h,2000h,3000h
```

```
.code
```

```
mov esi,OFFSET arrayW
```

```
mov ax,[esi] ; AX = 1000h
```

```
add esi,2
```

```
mov ax,[esi] ; AX = 2000h
```



If we use an array of 32-bit integers, we add 4 to ESI to address each subsequent array element.

Indexed Operands

An indexed operand adds a constant to a register to generate an effective address. Any of the 32-bit general-purpose registers may be used as index registers.

SYNTAX:

constant [reg32] ; reg32 can be any of the 32-bit general registers

[constant + reg32]

EXAMPLE:

```
.data
```

```
arrayB BYTE 20, 40, 60, 80
```

```
.code
```

```
mov esi, 1
```

```
mov al, arrayB[esi]
```

```
inc esi
```

```
mov al, arrayB[esi]
```

```
mov esi, 3
```

```
mov al, [arrayB + esi]
```

Adding Displacements: The second type of indexed addressing combines a register with a constant offset. The index register holds the base address of an array.

```
INCLUDE Irvine32.inc
```

```
.data
```


```
arrayW WORD 1000h,2000h,3000h
```

```
.code
```

```
main PROC
```

```
mov eax,0
```

```
mov ebx,0
```

```
 mov ecx,0
```

```
mov esi,OFFSET arrayW  
mov ax,[esi] ; AX = 1000h  
mov bx,[esi+2] ; AX = 2000h  
mov cx,[esi+4] ; AX = 3000h
```

Scale Factors in Indexed Operands

Indexed operands must take into account the size of each array element when calculating offsets.

SYNTAX:

constant [reg32 * TYPE constant]

EXAMPLE:

```
INCLUDE Irvine32.inc
```

```
.data
```

```
arrayW WORD 1000h, 2000h, 3000h, 4000h
```

```
.code
```

```
main PROC
```

```
mov eax,0
```

```
mov ebx,0
```

```
mov ecx,0
```

```
mov esi, 1
```

```
mov ax, arrayW[esi * TYPE arrayW]
```

```
mov esi, 2
```

```
mov bx, arrayW[esi * TYPE arrayW]
```

```
mov esi, 3
```

```
mov cx, arrayW[esi * TYPE arrayW]
```



call DumpRegs

Exercises:

1. Declare a 32-bit signed integer `val1` and initialize it with the eight thousand. If `val1` is incremented by 1 using the `ADD` instruction, what will be the values of the Carry and Sign flags?
2. Write down the values of the Carry, Sign, Zero, and Overflow flags after each instruction has executed:

```
mov ax,7FF0h
add al,10h      ; a. CF = SF = ZF = OF =
add ah,1        ; b. CF = SF = ZF = OF =
add ax,2        ; c. CF = SF = ZF = OF =
```

3. Initialize a double word array consisting of elements 8, 5, 1, 2, 6. Sort the given array in ascending order directly with the help of registers. Use direct-offset addressing to access the elements.
4. Use following array declarations:
`arrayB BYTE 10, 20, 30`
`arrayW WORD 150, 250, 350`
`arrayD DWORD 600, 1200, 1800`

Now initialize three double word variables `SUM1`, `SUM2`, `SUM3` and perform following operations (expressed in pseudo-code here):

`SUM1 = arrayB[0] + arrayW[0] + arrayD[0]`

`SUM2 = arrayB[1] + arrayW[1] + arrayD[1]`

`SUM3 = arrayB[2] + arrayW[2] + arrayD[2]`

5. Initialize two arrays:
`array1 BYTE 10, 20, 30, 40`
`array2 BYTE 4 DUP (?)`

Copy elements of `array1` into `array2` in reverse order using either indirect addressing or direct-offset addressing.

6. Subtract an array of 5 doublewords using indirect operands.
7. Use following array declarations:



arrayB BYTE 60, 70, 80

arrayW WORD 150, 250, 350

arrayD DWORD 600, 1200, 1800

For each array, add its 1st and last element using scale factors and display the result in a separate register.



LAB 4

TASK 1:

Code + Output

```
Title Task 1
INCLUDE Irvine32.inc

.data
val1 sdword 8000h

.code
main PROC
mov eax, val1
add eax, 1
call DumpRegs

exit
main ENDP
end main
```

Microsoft Visual Studio Debug Console

EAX=00008001 EBX=00C90000 ECX=007E10AA EDX=007E10AA
ESI=007E10AA EDI=007E10AA EBP=00F3F868 ESP=00F3F85C
EIP=007E366D EFL=00000202 CF=0 SF=0 ZF=0 OF=0 AF=0 PF=0

D:\Uni\3rd Semester\Coal\Practice\Project1\Debug\Project1.exe (process 19908) exited with code 0.
Press any key to close this window . . .

TASK 2:

Code + Output

```
1 Title Task 2
2 Include Irvine32.inc
3 .code
4 main PROC
5
6 mov ax, 7FF0h
7 add al, 10h
8 call dumpregs
9 add ah, 1
10 call dumpregs
11 add ax, 2
12 call dumpregs
13
14
15 exit
16 main ENDP
17 end main
```

Select Microsoft Visual Studio Debug Console

EAX=007D7F00 EBX=00804000 ECX=000610AA EDX=000610AA
ESI=000610AA EDI=000610AA EBP=007DF78C ESP=007DF780
EIP=0006366B EFL=00000247 CF=1 SF=0 ZF=1 OF=0 AF=0 PF=1

EAX=007D8000 EBX=00804000 ECX=000610AA EDX=000610AA
ESI=000610AA EDI=000610AA EBP=007DF78C ESP=007DF780
EIP=00063673 EFL=00000A92 CF=0 SF=1 ZF=0 OF=1 AF=1 PF=0

EAX=007D8002 EBX=00804000 ECX=000610AA EDX=000610AA
ESI=000610AA EDI=000610AA EBP=007DF78C ESP=007DF780
EIP=0006367C EFL=00000282 CF=0 SF=1 ZF=0 OF=0 AF=0 PF=0

D:\Uni\3rd Semester\Coal\Practice\Project1\Debug\Project1.exe (process 7700) exited with code 0.
Press any key to close this window . . .

LAB 4

TASK 3:

Code + Output

```
1  TITLE Task3
2  INCLUDE Irvine32.inc
3  .data
4  arr DWORD 8,5,1,2,6
5
6  .code
7  main PROC
8  mov eax,arr+8
9  xchg arr,eax
10 mov arr+8,eax; arr=1,5,8,2,6
11 mov eax,arr+12
12 xchg arr+4,eax
13 mov arr+12,eax; arr=1,2,8,5,6
14 mov eax,arr+12
15 xchg arr+8,eax
16 mov arr+12,eax; arr=1,2,5,8,6
17 mov eax,arr+16
18 xchg arr+12,eax
19 mov arr+16,eax; arr=1,2,5,6,8
20 call DumpRegs
21 exit
22 main ENDP
23 END main
24
```

Microsoft Visual Studio Debug Console

EAX=00000008 EBX=00C39000 ECX=005A10AA EDX=005A10AA
ESI=005A10AA EDI=005A10AA EBP=00EFF960 ESP=00EFF954
EIP=005A36A5 EFL=00000246 CF=0 SF=0 ZF=1 OF=0 AF=0 PF=1

D:\Uni\3rd Semester\Coal\Lab03\Project1\Debug\Project1.exe (process 27184) exited w
Press any key to close this window . . .

Before Sort

```
08 00 00 00 05 00 00 00 01 00 00 00 02 00 00 00 06
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

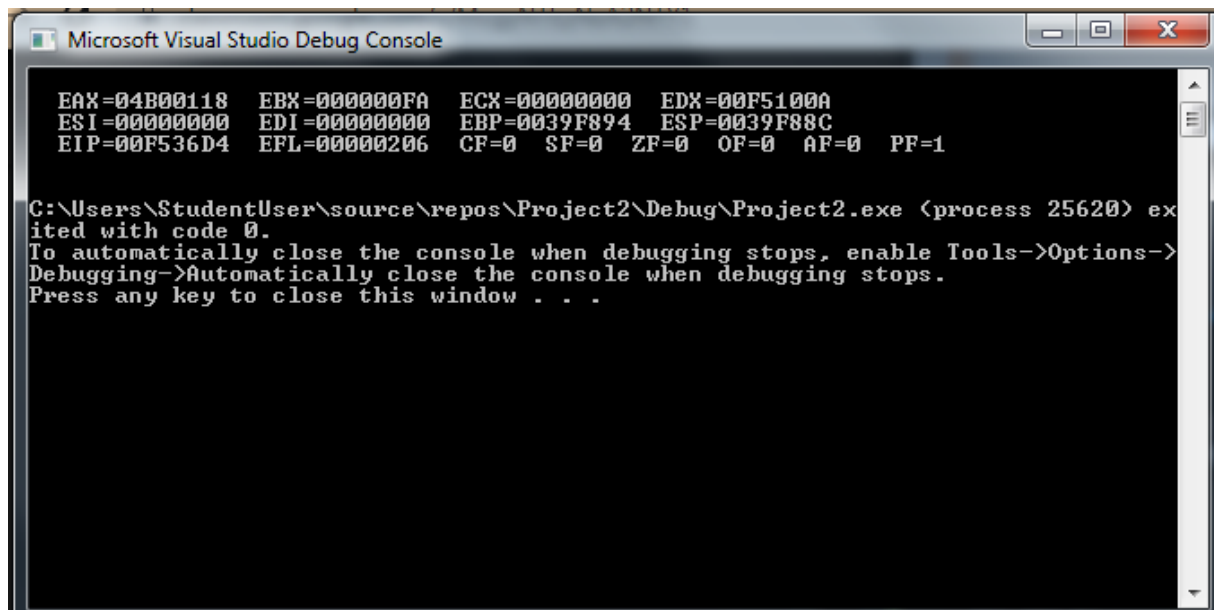
After Sort

```
01 00 00 00 02 00 00 00 05 00 00 00 06 00 00 00 03
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

TASK 4:

Code + Output

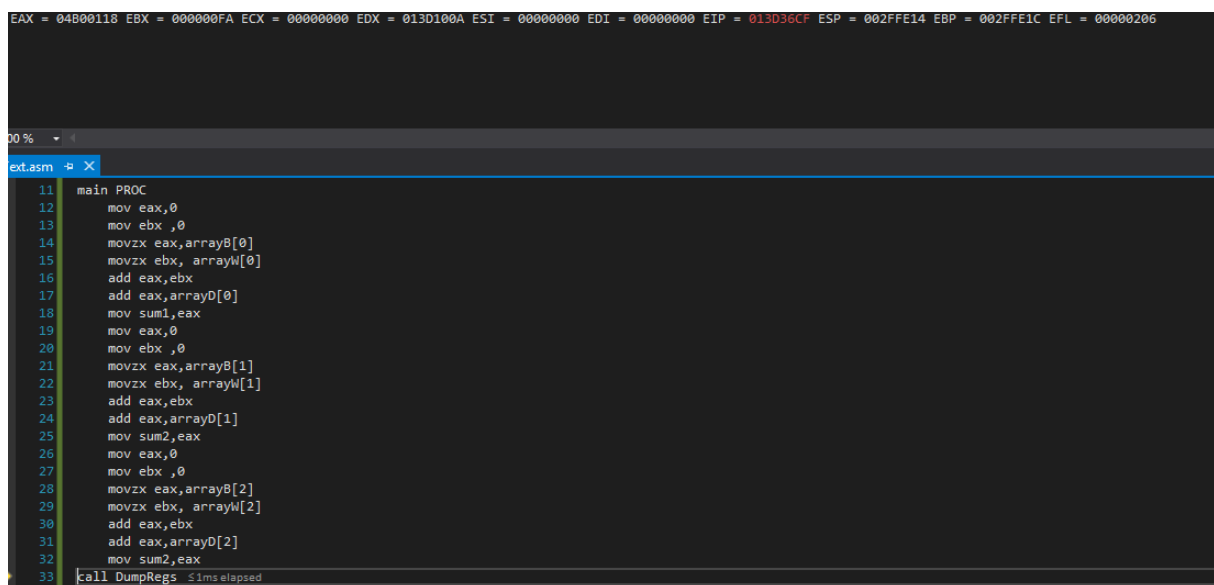
LAB 4



```
Microsoft Visual Studio Debug Console

EAX=04B00118  EBX=000000FA  ECX=00000000  EDX=00F5100A
ESI=00000000  EDI=00000000  EBP=0039F894  ESP=0039F88C
EIP=00F536D4  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1

C:\Users\StudentUser\source\repos\Project2\Debug\Project2.exe (process 25620) ex
ited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```



```
EAX = 04B00118 EBX = 000000FA ECX = 00000000 EDX = 013D100A ESI = 00000000 EDI = 00000000 EIP = 013D36CF ESP = 002FFE14 EBP = 002FFE1C EFL = 00000206

00 %
ext.asm
11  main PROC
12      mov eax,0
13      mov ebx,0
14      movzx eax,arrayB[0]
15      movzx ebx,arrayW[0]
16      add eax,ebx
17      add eax,arrayD[0]
18      mov sum1,eax
19      mov eax,0
20      mov ebx,0
21      movzx eax,arrayB[1]
22      movzx ebx,arrayW[1]
23      add eax,ebx
24      add eax,arrayD[1]
25      mov sum2,eax
26      mov eax,0
27      mov ebx,0
28      movzx eax,arrayB[2]
29      movzx ebx,arrayW[2]
30      add eax,ebx
31      add eax,arrayD[2]
32      mov sum2,eax
33      call DumpRegs 51ms elapsed
```

TASK 5:

Code + Output

LAB 4

```
Microsoft Visual Studio Debug Console

EAX=75A9330A  EBX=7EFDE000  ECX=00000000  EDX=011D100A
ESI=011D6000  EDI=00000000  EBP=0044FB40  ESP=0044FB38
EIP=011D3695  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1

C:\Users\StudentUser\source\repos\Project2\Debug\Project2.exe (process 48700) ex
ited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

```
Registers
EAX = 75A9330A EBX = 7EFDE000 ECX = 00000000 EDX = 011D100A ESI = 011D6000 EDI = 00000000 EIP = 011D3690 ESP = 0022F7B0 EBP = 0022F7B8 EFL = 00000206

100 %
Text.asm
5      array2 byte 5 dup(?)
6      .code
7      main PROC
8          mov esi,OFFSET array1+0
9          inc esi
10         inc esi
11         inc esi
12         inc esi
13         mov al,[esi]
14         mov array2[0],al
15         dec esi
16         mov al,[esi]
17         mov array2[1],al
18         dec esi
19         mov al,[esi]
20         mov array2[2],al
21         dec esi
22         mov al,[esi]
23         mov array2[3],al
24         dec esi
25         mov al,[esi]
26         mov array2[4],al
27         call DumpRegs 51ms elapsed
28         exit
29     main ENDP
30     END main
```

Before:

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 30 31 32 33 34 3
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 2
```

After:

LAB 4

28	1e	14	0a	00	00	00	00	00	00	00	00	00	01	30	31	32	33	34	3
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	2
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	2
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	2
20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	2

TASK 6:

Code + Output

```
EAX=FDE61BDC  EBX=0030E000  ECX=007C10AA  EDX=007C10AA
ESI=007C6010  EDI=007C10AA  EBP=0053FBBC  ESP=0053FBB0
EIP=007C3680  EFL=00000282  CF=0  SF=1  ZF=0  OF=0  AF=0  PF=0

C:\Users\hp\source\repos\Project4\Debug\Project4.exe (process 13096) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

```
Registers
EAX = FEC21D64  EBX = 011DE000  ECX = 007C10AA  EDX = 007C10AA  ESI = 007C6010  EDI = 007C10AA  EIP = 007C3678  ESP = 012FFD38  EBP = 012FFD44  EFL = 00000282

92 %
file1.asm  X
1  TITLE My First Program (test.asm)
2  INCLUDE Irvine32.inc
3  .data
4      array1 dword 10, 20, 30, 40, 50
5  .code
6  main PROC
7      mov esi, OFFSET array1+0
8      sub eax, esi
9      add esi, 4
10     sub eax, esi
11     add esi, 4
12     sub eax, esi
13     add esi, 4
14     sub eax, esi
15     add esi, 4
16     sub eax, esi
17     call DumpRegs 51ms elapsed
18     exit
19 main ENDP
```

TASK 7:

LAB 4

Code + Output

```
Registers
EAX = 42413DE8 EBX = 005040DC ECX = 002310AA EDX = 002310AA ESI = 00000008 EDI = 002310AA EIP = 002336A7 ESP = 0033FE4C EBP = 0033FE58 EFL = 00000206
```



```
92 %
file1.asm
12 inc esi
13 add bl,arrayB[esi*TYPE arrayB]
14 mov esi,1
15 mov ax,arrayW[esi*TYPE arrayW]
16 mov esi,4
17 add ax,arrayW[esi*TYPE arrayW]
18 mov esi,1
19 mov eax,arrayD[esi*TYPE arrayD]
20 mov esi,8
21 add eax,arrayD[esi*TYPE arrayD]
22 call DumpRegs 51ms elapsed
23 exit
24 main ENDP
25 END main
26
```



```
Microsoft Visual Studio Debug Console
EAX=42413DE8 EBX=006080DC ECX=002310AA EDX=002310AA
ESI=00000008 EDI=002310AA EBP=008FF974 ESP=008FF968
EIP=002336AC EFL=00000206 CF=0 SF=0 ZF=0 OF=0 AF=0 PF=1

C:\Users\hp\source\repos\Project4\Debug\Project4.exe (process 20024) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```