`

LAB 03

DATA TYPES & ASSEMBLY INSTRUCTIONS

STUDENT NAME ROLL NO SEC

SIGNATURE & DATE

MARKS AWARDED: ______________

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

Prepared by: Amin Sadiq

Version: 1.0

Date: 17th Sep 2021

EL 2003 COMP ORG & ASSEMBLY LANGUAGE LAB

LABORATORY MANUAL Fall 2021Instructor: Aamir Ali, Aashir Mehboob, Amin Sadiq, Qurat-ul-Ain, M Kariz, Rabia

Lab Session 03: DATA TYPE & ASSEMBLY INSTRUCTIONS

Objectives:

- Defining Data
- Data Definition Statement
- Data Initializations
- Multiple Initializations
- String Initialization
- Assembly language Instructions: MOV , ADD , SUB
- Sample Program
- Exercise

Data Types:

MASM defines **intrinsic data types**, each of which describes a set of values that can be assigned to variables and expressions of the given type.

BYTE	8-bit unsigned integer
SBYTE	8-bit signed integer. S stands for signed
WORD	16-bit unsigned integer
SWORD	16-bit signed integer
DWORD	32-bit unsigned. D stands for double
SDWORD	32-bit signed integer
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit integer. T stands for ten

Data definition statement:

A data definition statement sets aside storage in memory for a variable, with an optional name. Data definition statements create variables based on intrinsic data types.
A data definition has the following syntax:

[name] directive initializer [,initializer]...

Initializer: At least one initializer is required in a data definition, even if it is zero. Additional initializers, if any, are separated by commas. For integer data types, initializer is an integer constant or expression matching the size of the variable's type, such as BYTE or WORD. If you prefer to leave the variable uninitialized (assigned a random value), the ? symbol can be used as the initializer.

Examples:

```
value1 BYTE 'A'                        ; character constant
value2 BYTE 0                          ; smallest unsigned byte
value3 BYTE 255                        ; largest unsigned byte
value4 SBYTE −128                      ; smallest signed byte
value5 SBYTE +127                      ; largest signed byte
greeting1 BYTE "Good afternoon”, 0     ; String constant  with null terminated string
greeting2 BYTE 'Good night'            ; String constant
greeting1 BYTE 'G','o','o','d'         ; String constant
```

The hexadecimal codes 0Dh and 0Ah are alternately called CR/LF (carriage-return line-feed) or end-of-line characters.

```
list BYTE 10,20,30,40                  ; Multiple initializers
```

Note: A question mark (?) initializer leaves the variable uninitialized, implying it will be assigned a value at runtime:

```
value6 BYTE ?
```

DUP Operator

The DUP operator allocates storage for multiple data items, using a constant expression as a counter. It is particularly useful when allocating space for a string or array, and can be used with initialized or uninitialized data.

Examples:

```
v1 BYTE 20 DUP(0)              ; 20 bytes, all equal to zero
v2 BYTE 20 DUP(?)              ; 20 bytes, uninitialized
v3 BYTE 4 DUP("STACK")         ;20 bytes, "STACKSTACKSTACKSTACK"
```

Operand Types:

As x86 instruction formats:

```
[label:] mnemonic [operands][ ; comment ]
```

Because the number of operands may vary, we can further subdivide the formats to have zero, one, two, or three operands.

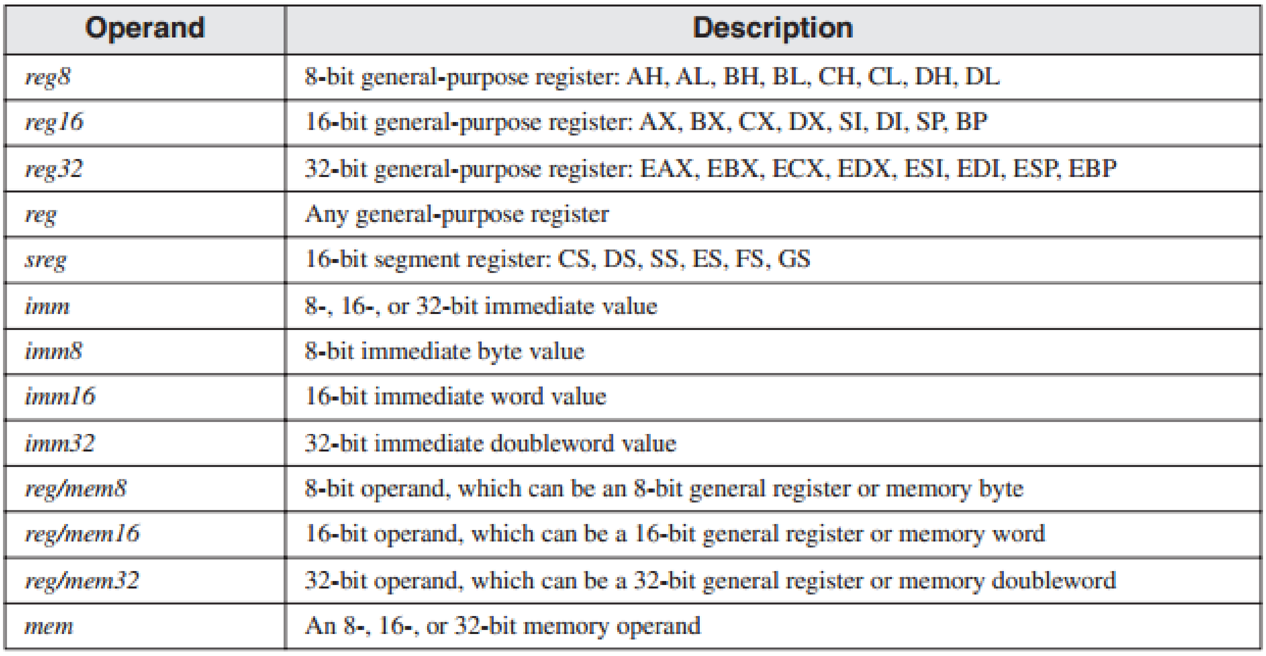
Here, we omit the label and comment fields for clarity:

mnemonic
mnemonic [destination]
mnemonic [destination],[source]
mnemonic [destination],[source-1],[source-2]

x86 assembly language uses different types of instruction operands. The following are the easiest to use:

- Immediate—uses a numeric literal expression
- Register—uses a named register in the CPU
- Memory—references a memory location

Following table lists a simple notation for operands. We will use it from this point on to describe the syntax of individual instructions.



Operand	Description
reg8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
reg16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
reg32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
reg/mem8	8-bit operand, which can be an 8-bit general register or memory byte
reg/mem16	16-bit operand, which can be a 16-bit general register or memory word
reg/mem32	32-bit operand, which can be a 32-bit general register or memory doubleword
mem	An 8-, 16-, or 32-bit memory operand

MOV Instruction:

It is used to move data from source operand to destination operand

- Both operands must be the same size.
- Both operands cannot be memory operands.

- CS, EIP, and IP cannot be destination operands.
- An immediate value cannot be moved to a segment register.

Syntax:

MOV destination, source

Here is a list of the general variants of MOV, excluding segment registers:

```
MOV reg,reg
MOV mem,reg
MOV reg,mem
MOV mem,imm
MOV reg,imm
```

Example:

MOV bx, 2
MOV ax, cx

Example:

'A' has ASCII code 65D (01000001B, 41H)

The following MOV instructions stores it in register BX:

MOV bx, 65d

MOV bx, 41h

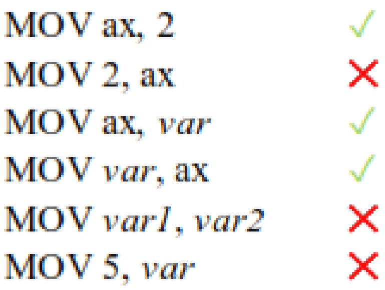
MOV bx, 01000001b

MOV bx, 'A'

All of the above are equivalent.

Examples:

The following examples demonstrate compatibility between operands used with MOV instruction:



MOV ax, 2	✓
MOV 2, ax	✗
MOV ax, *var*	✓
MOV *var*, ax	✓
MOV *var1*, *var2*	✗
MOV 5, *var*	✗

ADD Instruction

The ADD instruction adds a source operand to a destination operand of the same size. Source is unchanged by the operation, and the sum is stored in the destination operand

Syntax:

```
ADD dest,source
```

SUB Instruction

The SUB instruction subtracts a source operand from a destination operand.

Syntax:

```
SUB dest,source
```

Sample Program:

```
TITLE Add and Subtract (AddSub.asm)
; This program adds and subtracts 32-bit integers.
INCLUDE Irvine32.inc
.code
main PROC
mov eax,10000h      ; EAX = 10000h
add eax,40000h      ; EAX = 50000h
sub eax,20000h      ; EAX = 30000h

call DumpRegs       ; display registers
exit
main ENDP
END main
```

Lab Exercise:

1. Write an uninitialized data declaration for a16-bit signed integer val1. Initialize 8-bit signed integer val2 with -10.
2. Declare a 32-bit signed integer val3 and initialize it with the smallest possible negative decimal value. (Hint: Use SDWORD)
3. Declare an unsigned 16-bit integer variable named wArray that uses three Initializers.

4. Declare a string variable containing the name of your favorite color. Initialize it as a null terminated string. Initialize five 16-bit unsigned integers varA, varB, varC, varD & varE with the following values: 12, 2, 13, 8, 14.
5. Convert the following high-level instruction into Assembly Language:

   ebx = { (a+b) – (a-b) + c } +d

   a= 10h , b=15h, c=20h, d=30h
6. Convert the given values of a,b,c,d into binary and then use in 8-bit data definition and implement in the equation.
7. Write a program in assembly language that implements following expression:

   Eax = imm8 + data1 – data3 + imm8 + data2

   Use these data definitions:

   Imm8 = 20

   Data1 word 8

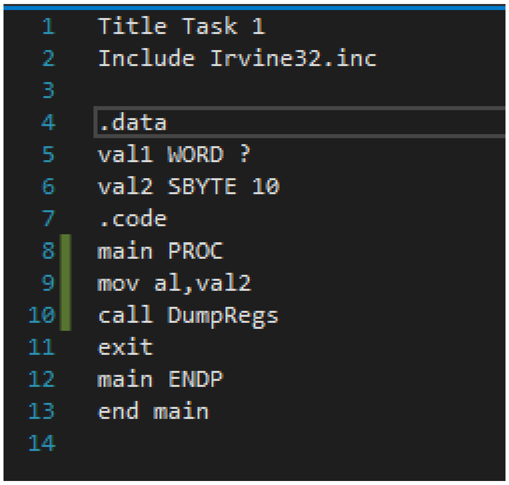
   Data2 word 15

   Data3 word 20

LAB 3

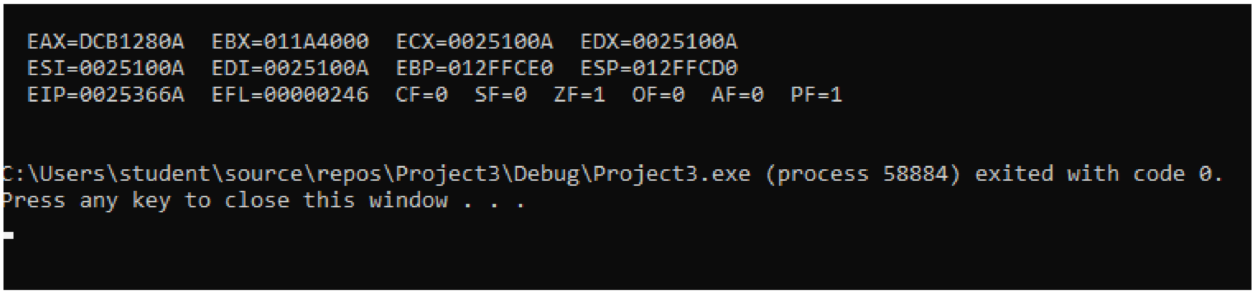
TASK 1

Input



```
Title Task 1
Include Irvine32.inc

.data
val1 WORD ?
val2 SBYTE 10
.code
main PROC
mov al,val2
call DumpRegs
exit
main ENDP
end main
```

Output

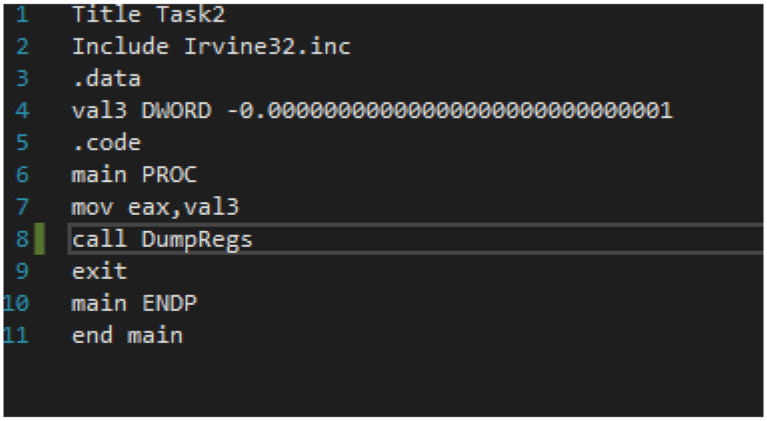


```
  EAX=DCB1280A  EBX=011A4000  ECX=0025100A  EDX=0025100A
  ESI=0025100A  EDI=0025100A  EBP=012FFCE0  ESP=012FFCD0
  EIP=0025366A  EFL=00000246  CF=0  SF=0  ZF=1  OF=0  AF=0  PF=1


C:\Users\student\source\repos\Project3\Debug\Project3.exe (process 58884) exited with code 0.
Press any key to close this window . . .
```

Comment: Learned How To Declare Variables

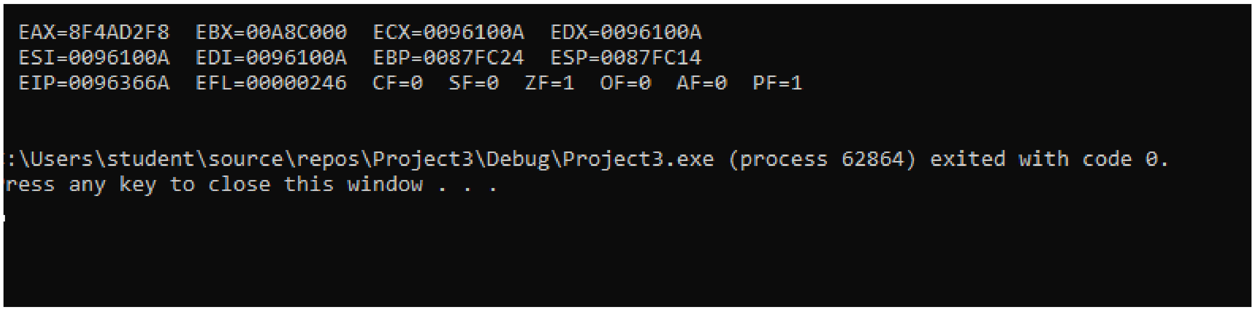
TASK 2

Input



```
Title Task2
Include Irvine32.inc
.data
val3 DWORD -0.00000000000000000000000000001
.code
main PROC
mov eax,val3
call DumpRegs
exit
main ENDP
end main
```

Output

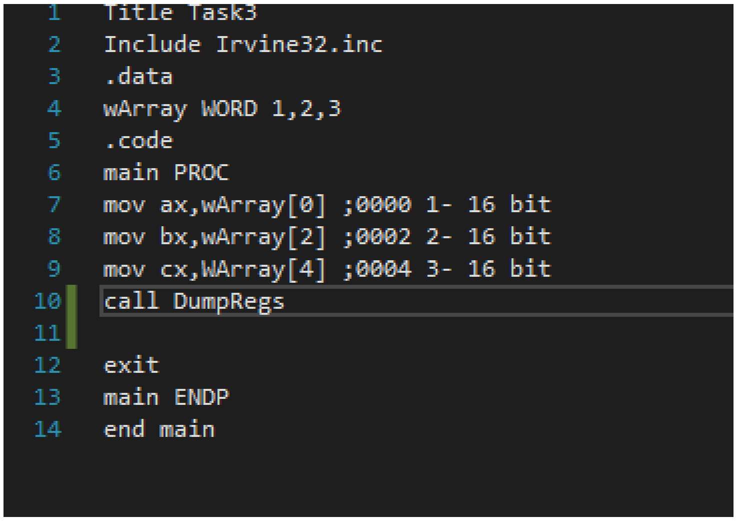


```
EAX=8F4AD2F8  EBX=00A8C000  ECX=0096100A  EDX=0096100A
ESI=0096100A  EDI=0096100A  EBP=0087FC24  ESP=0087FC14
EIP=0096366A  EFL=00000246  CF=0  SF=0  ZF=1  OF=0  AF=0  PF=1


:\Users\student\source\repos\Project3\Debug\Project3.exe (process 62864) exited with code 0.
ress any key to close this window . . .
```

Comment: Learned How to Initialize Variables

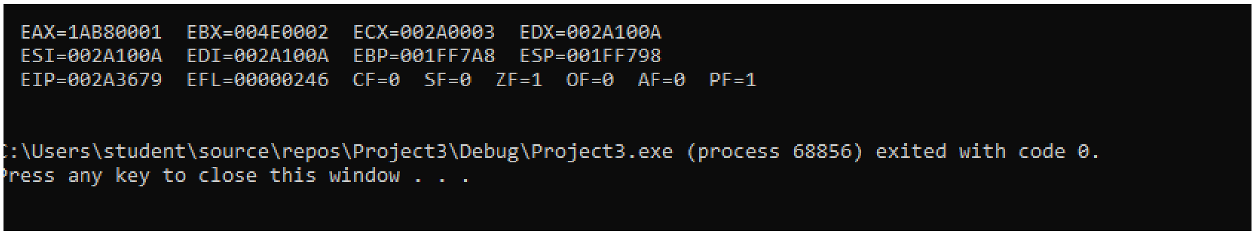
TASK 3

Input



```
Title Task3
Include Irvine32.inc
.data
wArray WORD 1,2,3
.code
main PROC
mov ax,wArray[0] ;0000 1- 16 bit
mov bx,wArray[2] ;0002 2- 16 bit
mov cx,WArray[4] ;0004 3- 16 bit
call DumpRegs

exit
main ENDP
end main
```

Output



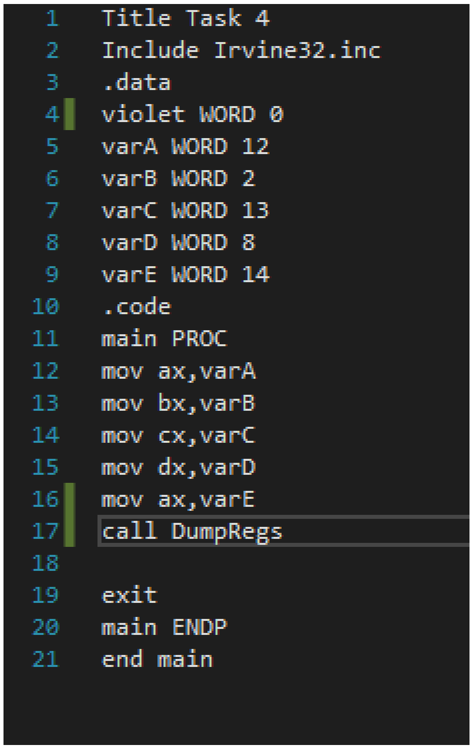
```
EAX=1AB80001  EBX=004E0002  ECX=002A0003  EDX=002A100A
ESI=002A100A  EDI=002A100A  EBP=001FF7A8  ESP=001FF798
EIP=002A3679  EFL=00000246  CF=0  SF=0  ZF=1  OF=0  AF=0  PF=1


:\Users\student\source\repos\Project3\Debug\Project3.exe (process 68856) exited with code 0.
ress any key to close this window . . .
```

Comment: Learned How To Use Array Of Variables

LAB 3

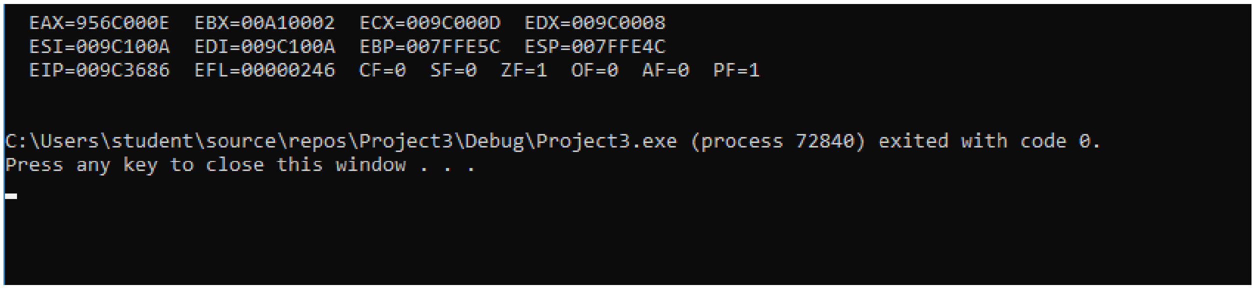
TASK 4

Input



```
Title Task 4
Include Irvine32.inc
.data
violet WORD 0
varA WORD 12
varB WORD 2
varC WORD 13
varD WORD 8
varE WORD 14
.code
main PROC
mov ax,varA
mov bx,varB
mov cx,varC
mov dx,varD
mov ax,varE
call DumpRegs

exit
main ENDP
end main
```

Output

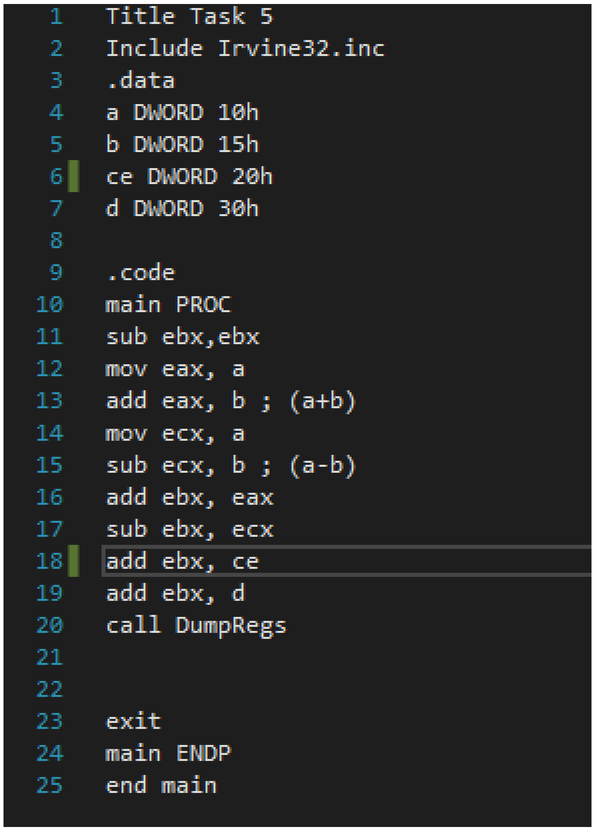


```
  EAX=956C000E  EBX=00A10002  ECX=009C000D  EDX=009C0008
  ESI=009C100A  EDI=009C100A  EBP=007FFE5C  ESP=007FFE4C
  EIP=009C3686  EFL=00000246  CF=0  SF=0  ZF=1  OF=0  AF=0  PF=1


C:\Users\student\source\repos\Project3\Debug\Project3.exe (process 72840) exited with code 0.
Press any key to close this window . . .
```

Comment: The Code Was Running as expected

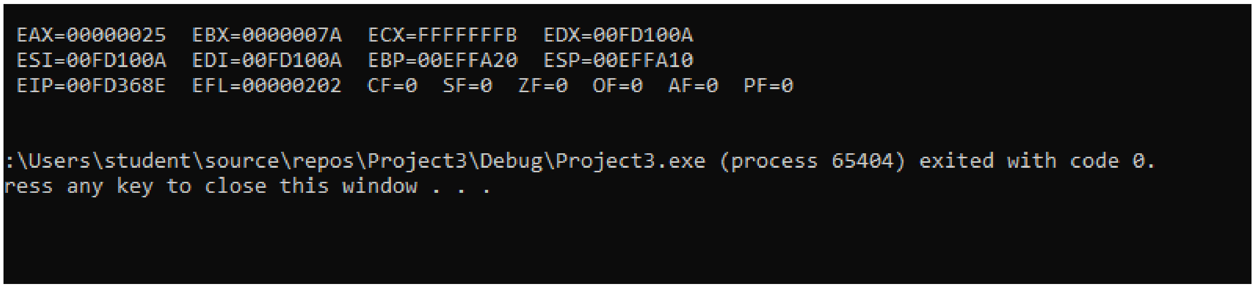
TASK 5

Input



```
Title Task 5
Include Irvine32.inc
.data
a DWORD 10h
b DWORD 15h
ce DWORD 20h
d DWORD 30h

.code
main PROC
sub ebx,ebx
mov eax, a
add eax, b ; (a+b)
mov ecx, a
sub ecx, b ; (a-b)
add ebx, eax
sub ebx, ecx
add ebx, ce
add ebx, d
call DumpRegs


exit
main ENDP
end main
```

Output



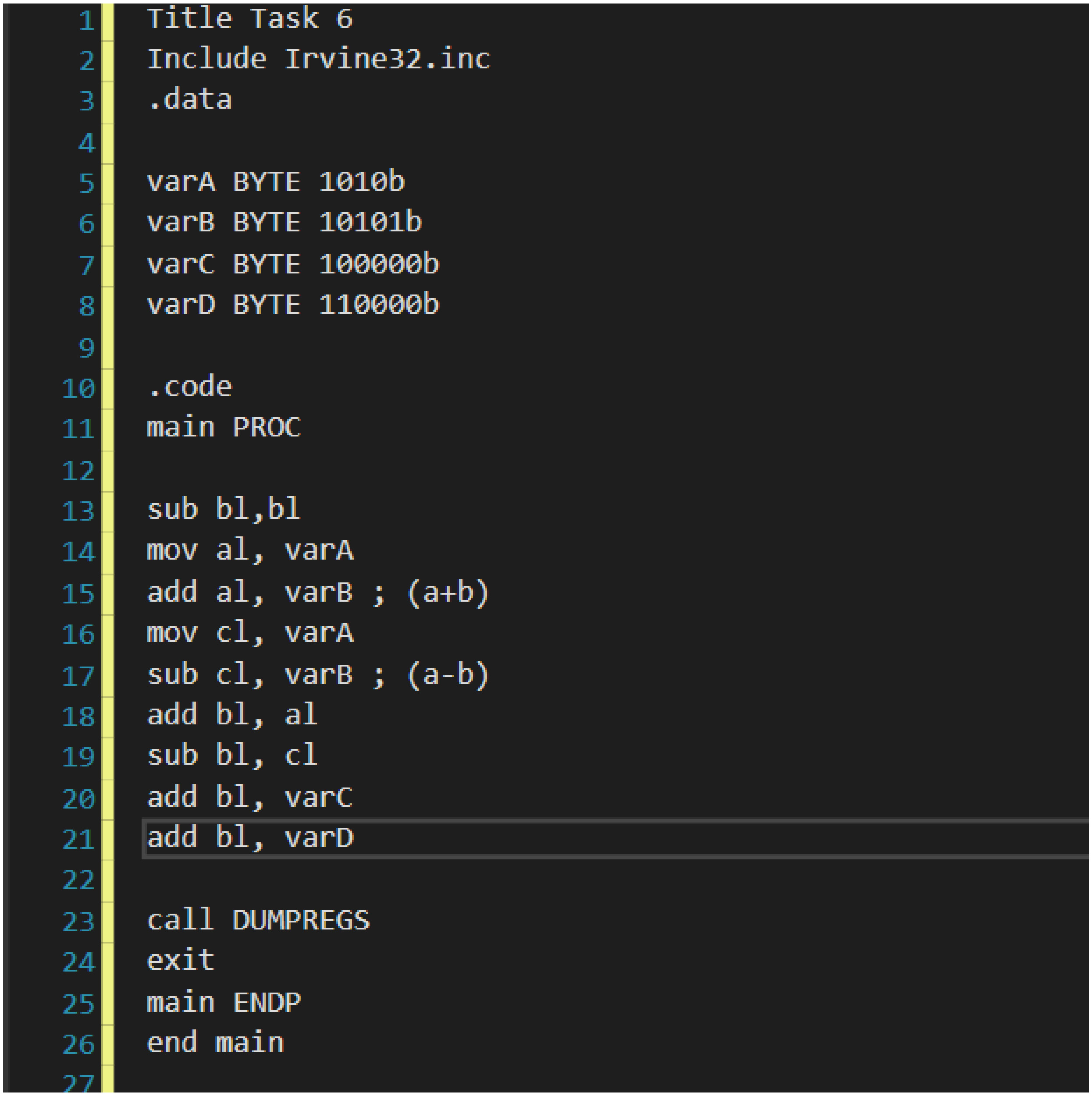
```
 EAX=00000025  EBX=0000007A  ECX=FFFFFFFB  EDX=00FD100A
 ESI=00FD100A  EDI=00FD100A  EBP=00EFFA20  ESP=00EFFA10
 EIP=00FD368E  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


:\Users\student\source\repos\Project3\Debug\Project3.exe (process 65404) exited with code 0.
ress any key to close this window . . .
```

Comment: The Code was running successfully

TASK 6

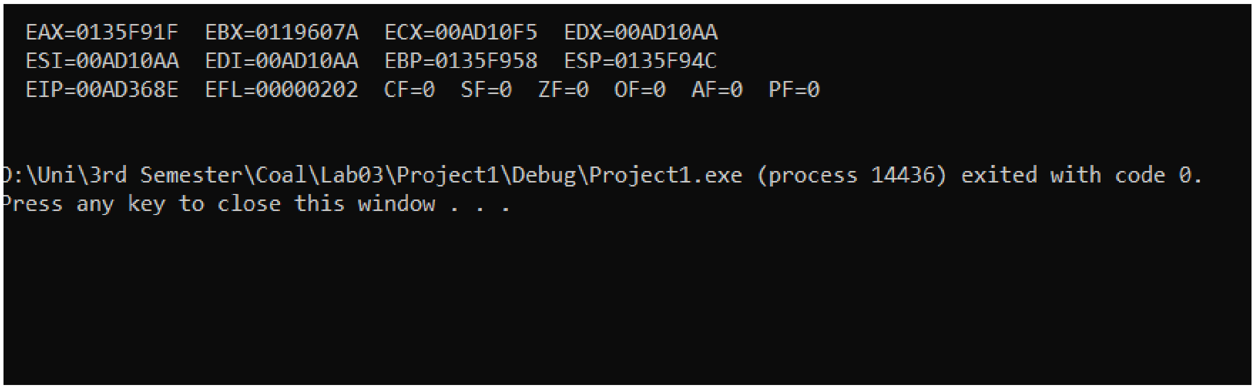
Input

LAB 3



```
Title Task 6
Include Irvine32.inc
.data

varA BYTE 1010b
varB BYTE 10101b
varC BYTE 100000b
varD BYTE 110000b

.code
main PROC

sub bl,bl
mov al, varA
add al, varB ; (a+b)
mov cl, varA
sub cl, varB ; (a-b)
add bl, al
sub bl, cl
add bl, varC
add bl, varD

call DUMPREGS
exit
main ENDP
end main
```

Output

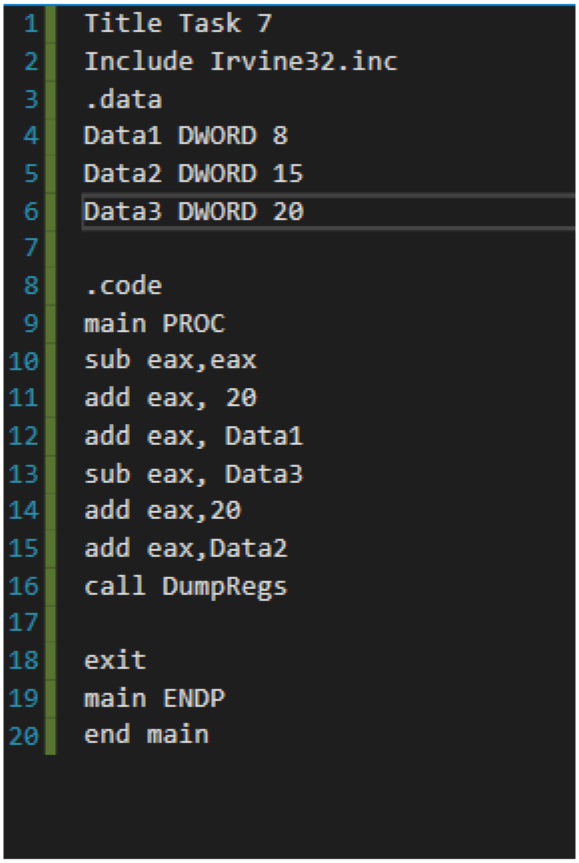


```
  EAX=0135F91F  EBX=0119607A  ECX=00AD10F5  EDX=00AD10AA
  ESI=00AD10AA  EDI=00AD10AA  EBP=0135F958  ESP=0135F94C
  EIP=00AD368E  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


D:\Uni\3rd Semester\Coal\Lab03\Project1\Debug\Project1.exe (process 14436) exited with code 0.
Press any key to close this window . . .
```

Comment: The Code was running as predicted

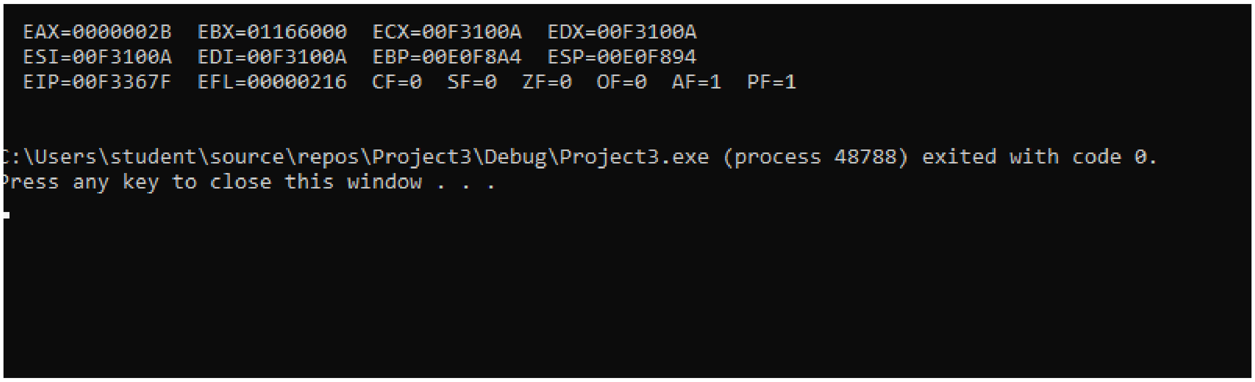
TASK 7

Input



```
Title Task 7
Include Irvine32.inc
.data
Data1 DWORD 8
Data2 DWORD 15
Data3 DWORD 20

.code
main PROC
sub eax,eax
add eax, 20
add eax, Data1
sub eax, Data3
add eax,20
add eax,Data2
call DumpRegs

exit
main ENDP
end main
```

Output



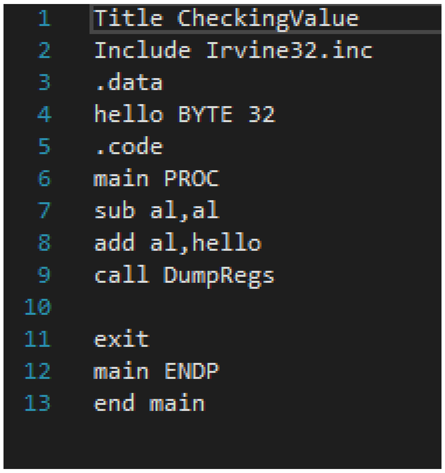
```
  EAX=0000002B  EBX=01166000  ECX=00F3100A  EDX=00F3100A
  ESI=00F3100A  EDI=00F3100A  EBP=00E0F8A4  ESP=00E0F894
  EIP=00F3367F  EFL=00000216  CF=0  SF=0  ZF=0  OF=0  AF=1  PF=1


C:\Users\student\source\repos\Project3\Debug\Project3.exe (process 48788) exited with code 0.
Press any key to close this window . . .
```

Comment: The Code was running successfully

LAB 3

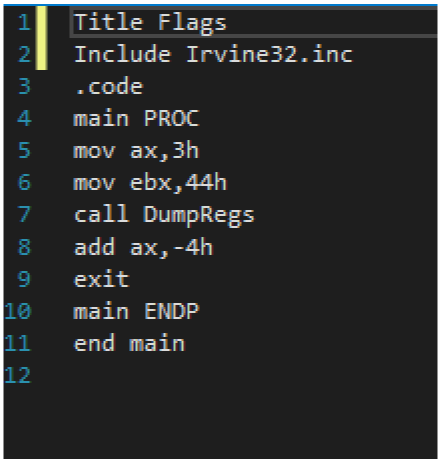
OTHER TASKS

Checking Value



```
Title CheckingValue
Include Irvine32.inc
.data
hello BYTE 32
.code
main PROC
sub al,al
add al,hello
call DumpRegs

exit
main ENDP
end main
```

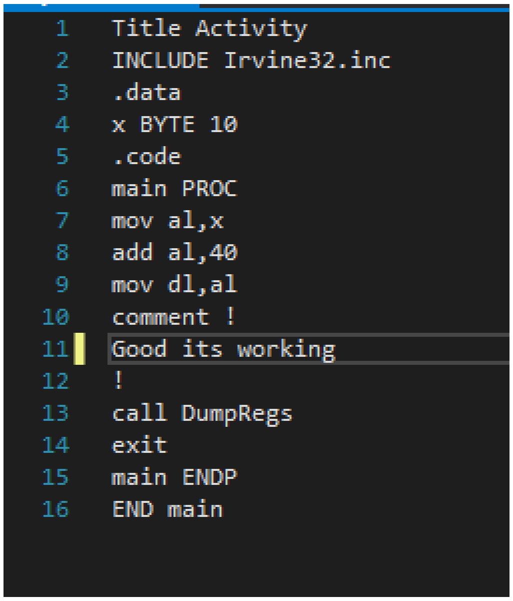
Flags



```
Title Flags
Include Irvine32.inc
.code
main PROC
mov ax,3h
mov ebx,44h
call DumpRegs
add ax,-4h
exit
main ENDP
end main

```

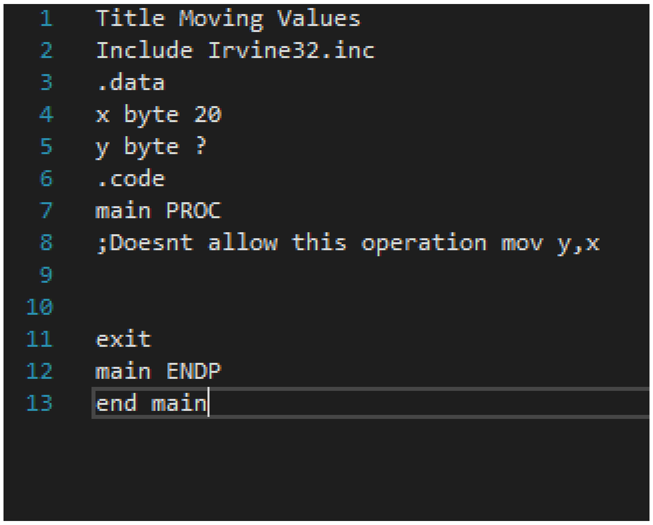
Activity

LAB 3



```
Title Activity
INCLUDE Irvine32.inc
.data
x BYTE 10
.code
main PROC
mov al,x
add al,40
mov dl,al
comment !
Good its working
!
call DumpRegs
exit
main ENDP
END main
```

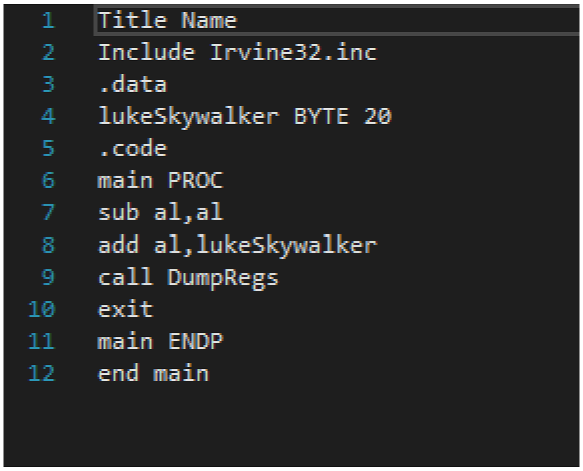
Moving Values



```
Title Moving Values
Include Irvine32.inc
.data
x byte 20
y byte ?
.code
main PROC
;Doesnt allow this operation mov y,x


exit
main ENDP
end main
```

Variables

LAB 3



```
Title Name
Include Irvine32.inc
.data
lukeSkywalker BYTE 20
.code
main PROC
sub al,al
add al,lukeSkywalker
call DumpRegs
exit
main ENDP
end main
```