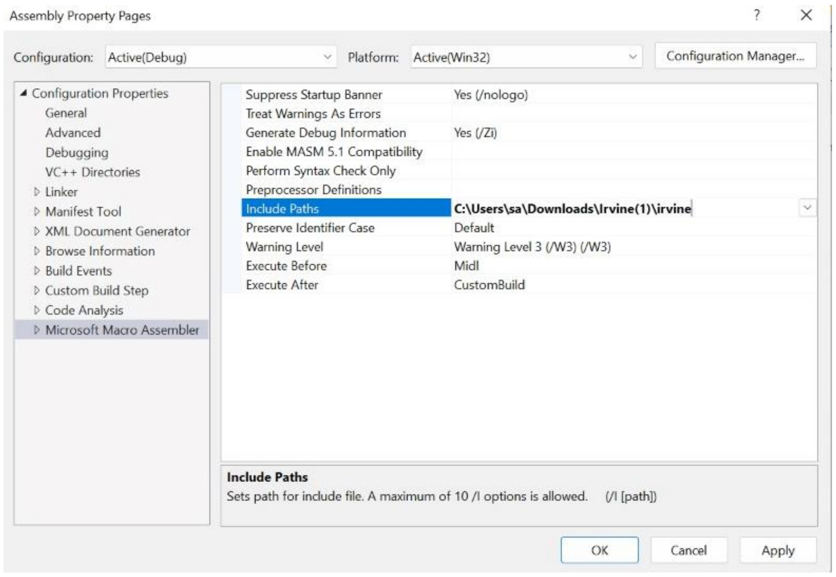
LAB 4 - 6

Name: Kulsoom Khurshid

Reg #: Sp20-BCS-044

LAB # 4:

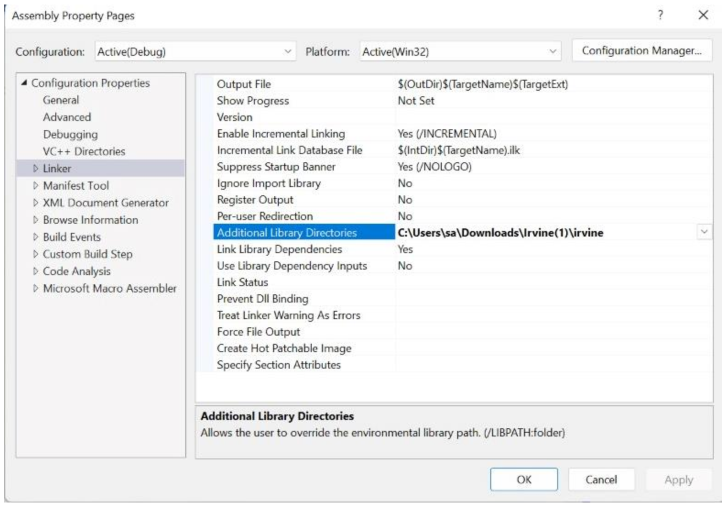
1) Assuming that our sample project is still open, select Project Properties from the Project menu. Click the entry named General under Microsoft Macro Assembler. Notice that the Include Paths option has been set to the c:\Irvine directory.



Suppress Startup Banner	Yes (/nologo)
Treat Warnings As Errors	
Generate Debug Information	Yes (/Zi)
Enable MASM 5.1 Compatibility	
Perform Syntax Check Only	
Preprocessor Definitions	
Include Paths	C:\Users\sa\Downloads\Irvine(1)\irvine
Preserve Identifier Case	Default
Warning Level	Warning Level 3 (/W3) (/W3)
Execute Before	Midl
Execute After	CustomBuild

Include Paths
Sets path for include file. A maximum of 10 /I options is allowed. (/I [path])

2) Next, select the Listing File entry, also in the Microsoft Macro Assembler group. Find the Linker entry under Configuration Properties. Select the Input entry, and notice that two filenames have been added to the Additional Dependencies entry.

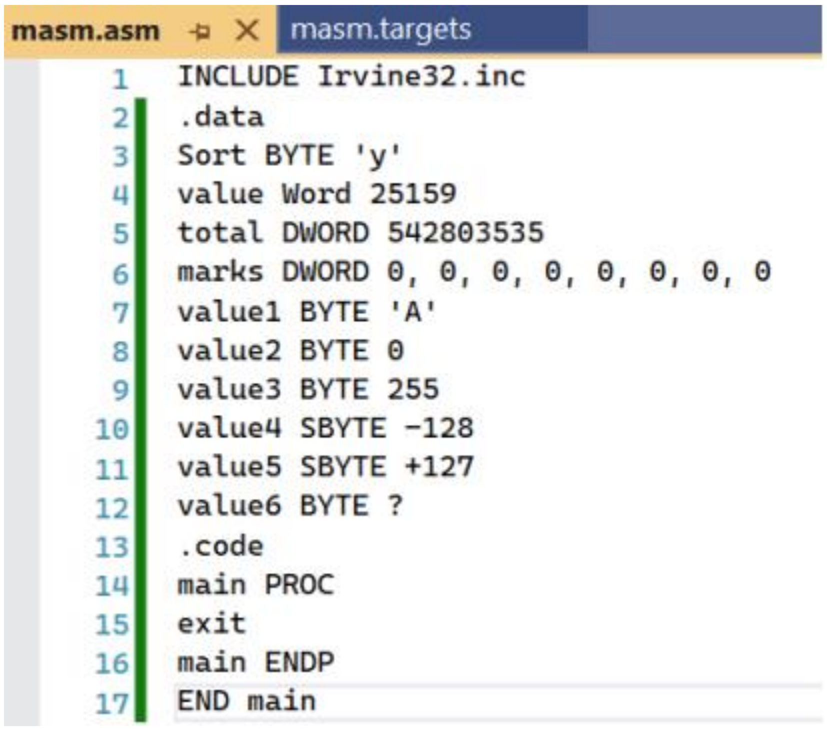


Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	Yes (/INCREMENTAL)
Incremental Link Database File	$(IntDir)$(TargetName).ilk
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	C:\Users\sa\Downloads\Irvine(1)\irvine
Link Library Dependencies	Yes
Use Library Dependency Inputs	No
Link Status	
Prevent Dll Binding	
Treat Linker Warning As Errors	
Force File Output	
Create Hot Patchable Image	
Specify Section Attributes	

Additional Library Directories
Allows the user to override the environmental library path. (/LIBPATH:folder)

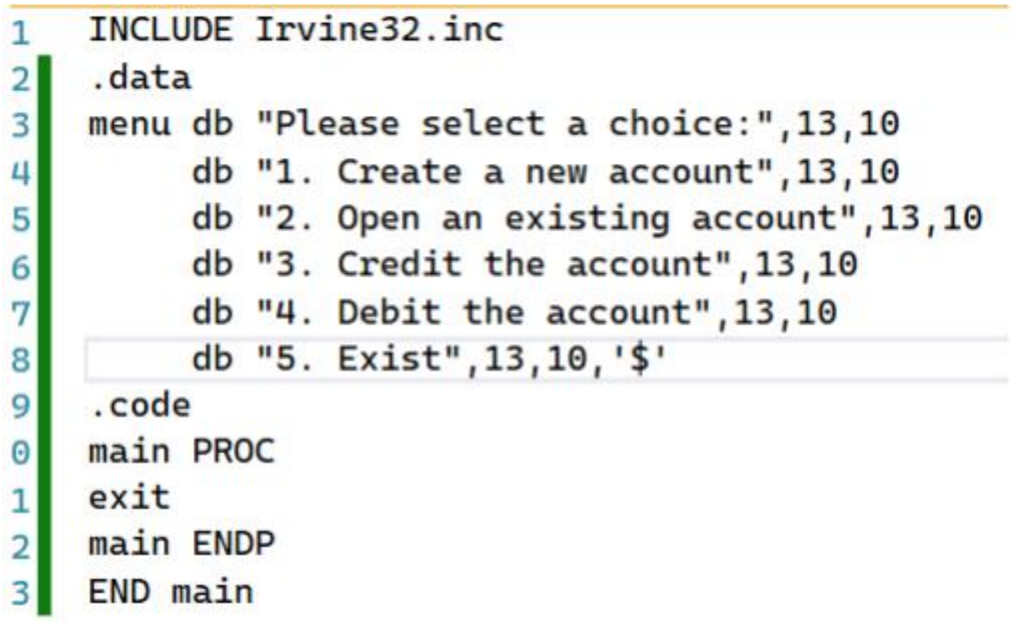
LAB # 5:

Home Task 1: Define the following data in Assembly



```
INCLUDE Irvine32.inc
.data
Sort BYTE 'y'
value Word 25159
total DWORD 542803535
marks DWORD 0, 0, 0, 0, 0, 0, 0, 0
value1 BYTE 'A'
value2 BYTE 0
value3 BYTE 255
value4 SBYTE -128
value5 SBYTE +127
value6 BYTE ?
.code
main PROC
exit
main ENDP
END main
```

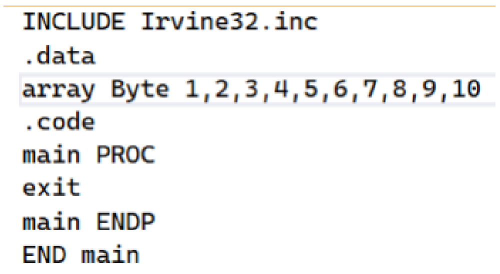
Home Task 2: Create the following menu by declaring a string in Assembly Language



```
INCLUDE Irvine32.inc
.data
menu db "Please select a choice:",13,10
     db "1. Create a new account",13,10
     db "2. Open an existing account",13,10
     db "3. Credit the account",13,10
     db "4. Debit the account",13,10
     db "5. Exist",13,10,'$'
.code
main PROC
exit
main ENDP
END main
```

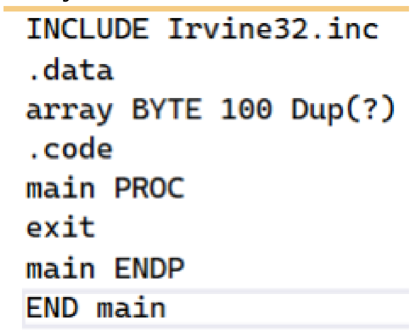
Home Task 3: Create the following arrays in Assembly Language.

1) Array of ten integers.



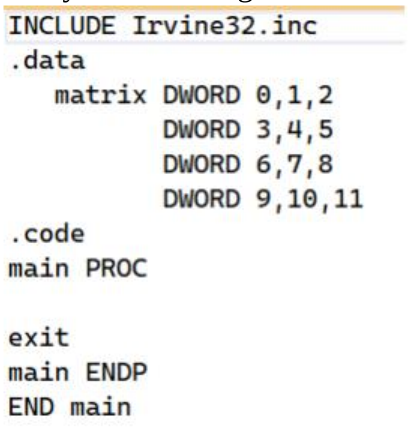
```
INCLUDE Irvine32.inc
.data
array Byte 1,2,3,4,5,6,7,8,9,10
.code
main PROC
exit
main ENDP
END main
```

2) Array of 100 characters.



```
INCLUDE Irvine32.inc
.data
array BYTE 100 Dup(?)
.code
main PROC
exit
main ENDP
END main
```

3) Arrays of 4x3 integers

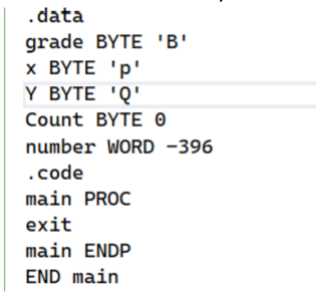


```
INCLUDE Irvine32.inc
.data
   matrix DWORD 0,1,2
          DWORD 3,4,5
          DWORD 6,7,8
          DWORD 9,10,11
.code
main PROC

exit
main ENDP
END main
```

Home Task 4: Implement each of the following declarations in assembly language:

1) char initial;
2) char grade = 'B';
3) char x = 'P', y = 'Q';
4) int amount;
5) int count = 0;
6) int number = -396;



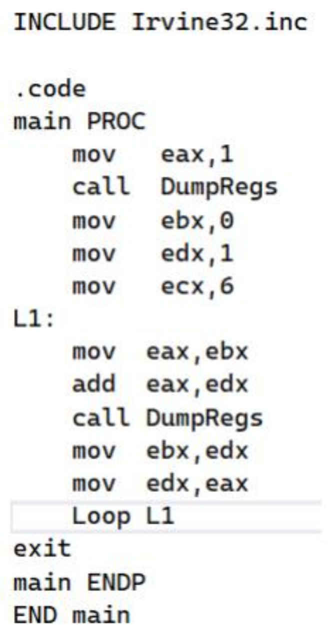
```
.data
grade BYTE 'B'
x BYTE 'p'
Y BYTE 'Q'
Count BYTE 0
number WORD -396
.code
main PROC
exit
main ENDP
END main
```

LAB # 6:

Home Task 1:

Write a program that uses a loop to calculate the first seven values in the Fibonacci number sequence { 1,1,2,3,5,8,13 }. Place each value in the EAX register and display it with a call DumpRegs statement inside the loop.

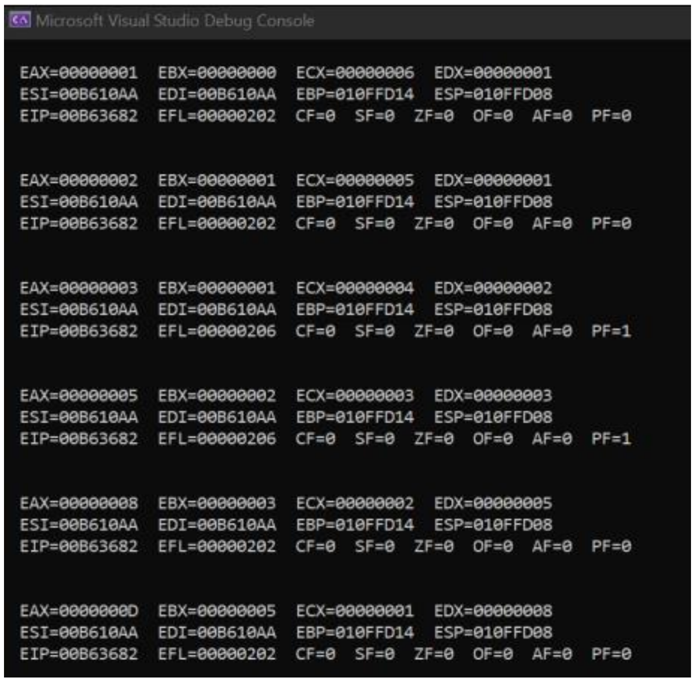
CODE:



```
INCLUDE Irvine32.inc

.code
main PROC
    mov   eax,1
    call  DumpRegs
    mov   ebx,0
    mov   edx,1
    mov   ecx,6
L1:
    mov  eax,ebx
    add  eax,edx
    call DumpRegs
    mov  ebx,edx
    mov  edx,eax
    Loop L1
exit
main ENDP
END main
```

OUTPUT:



```
EAX=00000001  EBX=00000000  ECX=00000006  EDX=00000001
ESI=00B610AA  EDI=00B610AA  EBP=010FFD14  ESP=010FFD08
EIP=00B63682  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


EAX=00000002  EBX=00000001  ECX=00000005  EDX=00000001
ESI=00B610AA  EDI=00B610AA  EBP=010FFD14  ESP=010FFD08
EIP=00B63682  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


EAX=00000003  EBX=00000001  ECX=00000004  EDX=00000002
ESI=00B610AA  EDI=00B610AA  EBP=010FFD14  ESP=010FFD08
EIP=00B63682  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1


EAX=00000005  EBX=00000002  ECX=00000003  EDX=00000003
ESI=00B610AA  EDI=00B610AA  EBP=010FFD14  ESP=010FFD08
EIP=00B63682  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1


EAX=00000008  EBX=00000003  ECX=00000002  EDX=00000005
ESI=00B610AA  EDI=00B610AA  EBP=010FFD14  ESP=010FFD08
EIP=00B63682  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0


EAX=0000000D  EBX=00000005  ECX=00000001  EDX=00000008
ESI=00B610AA  EDI=00B610AA  EBP=010FFD14  ESP=010FFD08
EIP=00B63682  EFL=00000202  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=0
```

Home Task 2:

Write a program that implements the following arithmetic expression:

EAX = -val2 + 7 - val3 + val

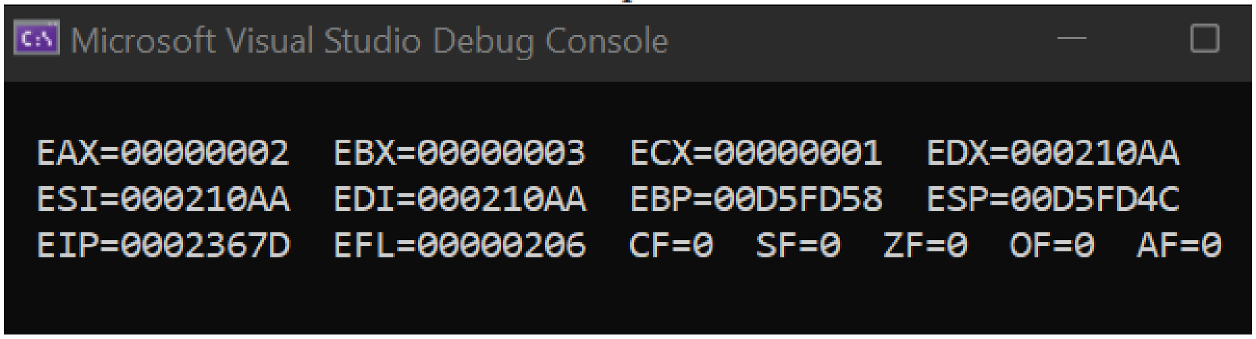
In comments next to each instruction, write the hexadecimal value of EAX. Insert a call DumpRegs statement at the end of the program.

CODE:

```
INCLUDE Irvine32.inc
.data
val1 DWORD 2
val2 DWORD 9
val3 DWORD 1

.code
main PROC
    mov eax, val1
    mov ebx, val2
    mov ecx, val3
    sub ebx,7
    sub ebx, ecx
    add ebx, eax
    call DumpRegs
    exit
main ENDP
END main
```

OUTPUT:



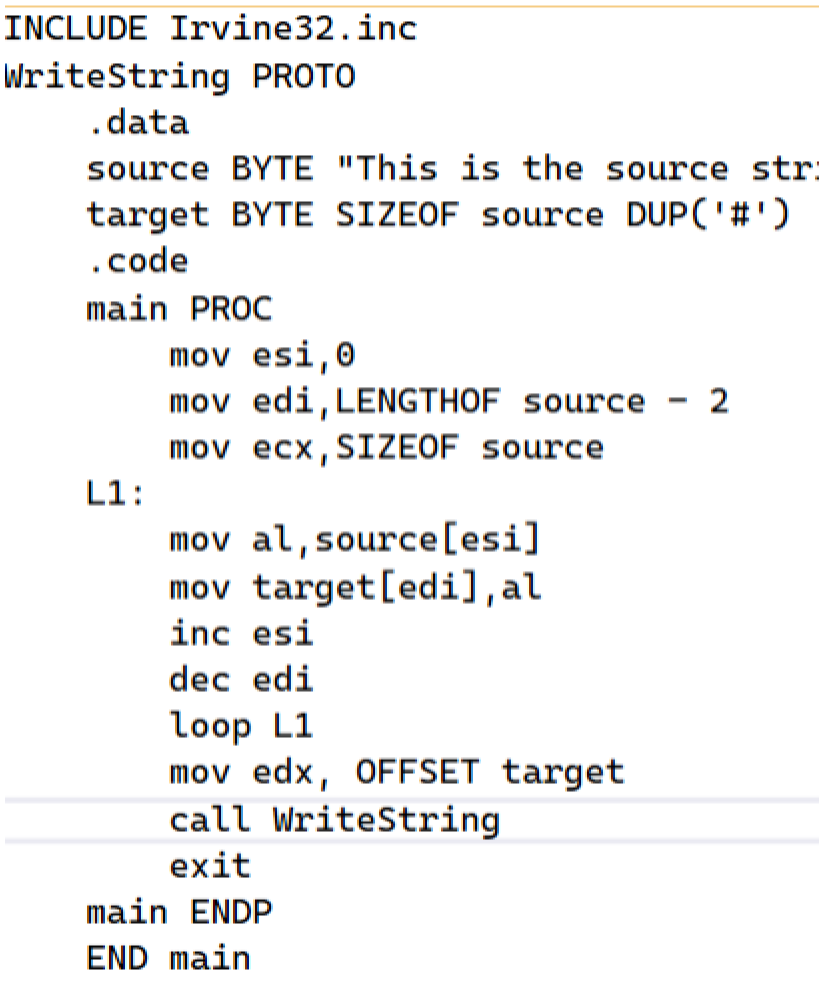
```
Microsoft Visual Studio Debug Console

  EAX=00000002  EBX=00000003  ECX=00000001  EDX=000210AA
  ESI=000210AA  EDI=000210AA  EBP=00D5FD58  ESP=00D5FD4C
  EIP=0002367D  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0
```

Home Task 3:

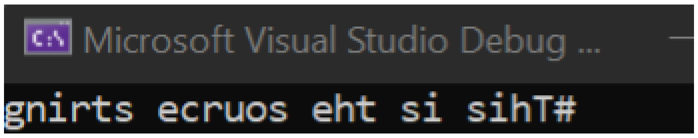
Write a program using the LOOP instruction with indirect addressing that copies a string from source to target, reversing the character order in the process.

CODE:



```
INCLUDE Irvine32.inc
WriteString PROTO
    .data
    source BYTE "This is the source str
    target BYTE SIZEOF source DUP('#')
    .code
    main PROC
        mov esi,0
        mov edi,LENGTHOF source - 2
        mov ecx,SIZEOF source
    L1:
        mov al,source[esi]
        mov target[edi],al
        inc esi
        dec edi
        loop L1
        mov edx, OFFSET target
        call WriteString
        exit
    main ENDP
    END main
```

OUTPUT:



```
gnirts ecruos eht si sihT#
```