

LAB 02

COMPUTER ORGANIZATION AND ASSEMBLY LANG(COAL)



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:

Lab Session 02

Objectives:

- Debugging of programs
- Basic elements of Assembly language
- Defining Data
- Intrinsic Data Types

Steps Involved in Creating and Running a Program:

ASSEMBLER:

It converts the assembly language to machine language (Object Code) May contain unresolved references (i.e. file contains some or all of complete program)

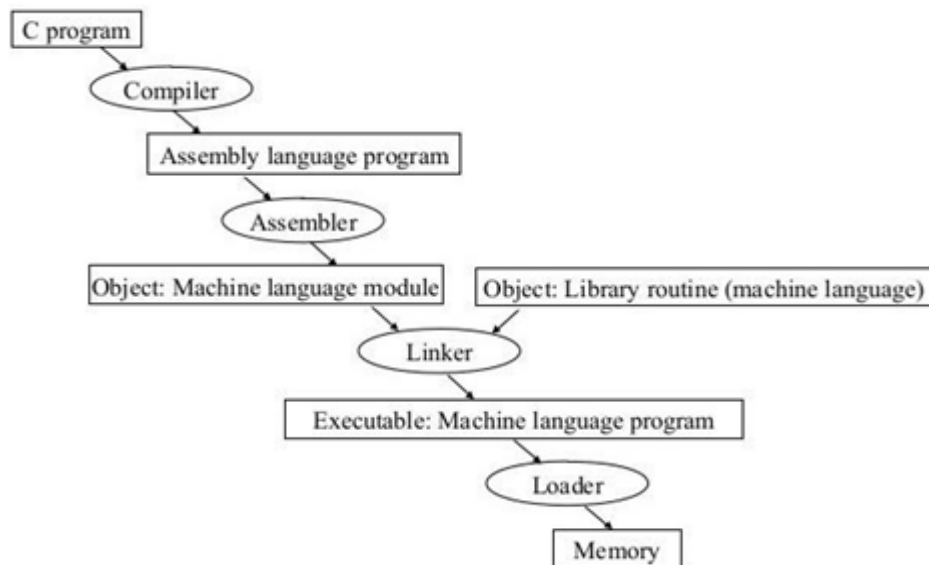
LINKER:

A program that combines object files to create a single “executable” file. Major functional difference is that all references are resolved. (i.e. Program contains all parts needed to run) A program that loads executable files into memory, and may initialize some registers (e.g. IP) and starts it going.

DEBUGGER:

A program that loads but controls the execution of the program. To start/stop execution, to view and modify state variables.

STEPS IN CREATING & RUNNING CODE:



SECTION 1: DEBUGGING OUR PROGRAM

We have seen how to configure Visual Studio 2019 for Assembly Language and tested it with a sample program. The output of our sample program was displayed using a console window but it is usually more desirable to watch the step by step execution of our program with each line of code using breakpoints.

Let us briefly define the keywords relevant to debugging in Visual Studio and then we will cover an example for understanding.

DEBUGGER

The (Visual Studio) debugger helps us observe the run-time behavior of our program and find problems. With the debugger, we can break the execution of our program to examine our code,



examine and edit variables, view registers, see the instructions created from our source code, and view the memory space used by our application.

BREAKPOINT

A breakpoint is a signal that tells the debugger to temporarily suspend execution of your program at a certain point. When execution is suspended at a breakpoint, your program is said to be in break mode.

CODE STEPPING

One of the most common debugging procedures is stepping: executing code one line at a time. The Debug menu provides three commands for stepping through code:

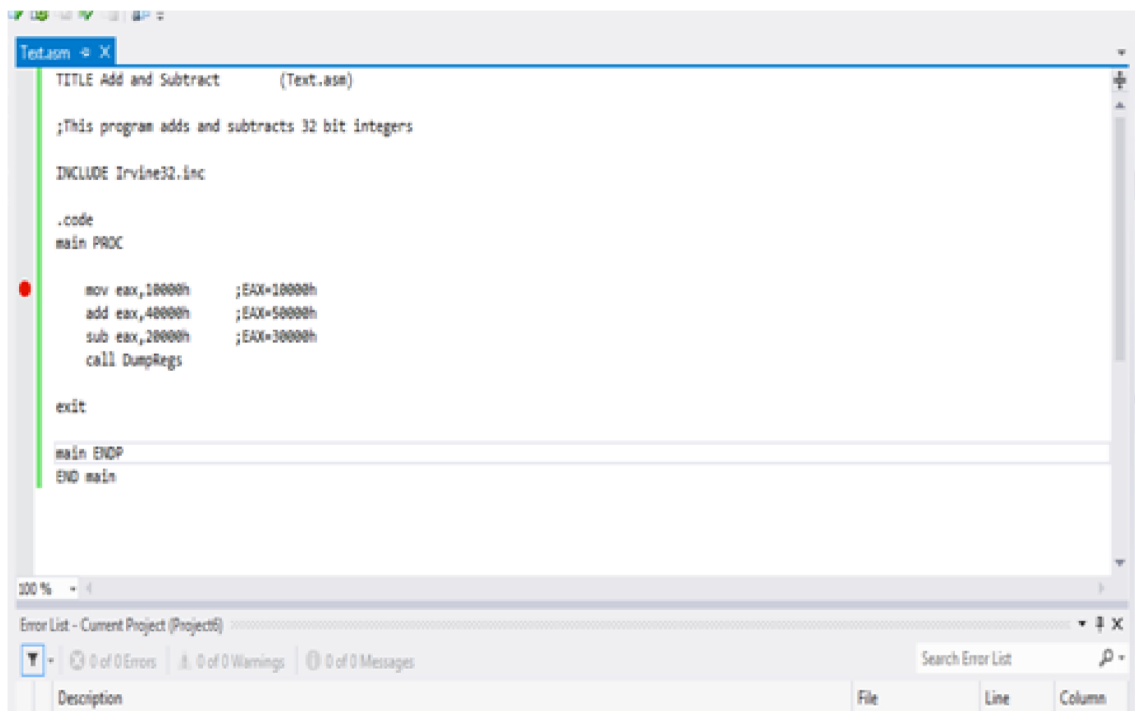
- Step Into (By pressing F11)
- Step Over (By pressing F10)
- Step Out (Shift+F11)

SINGLE STEPPING

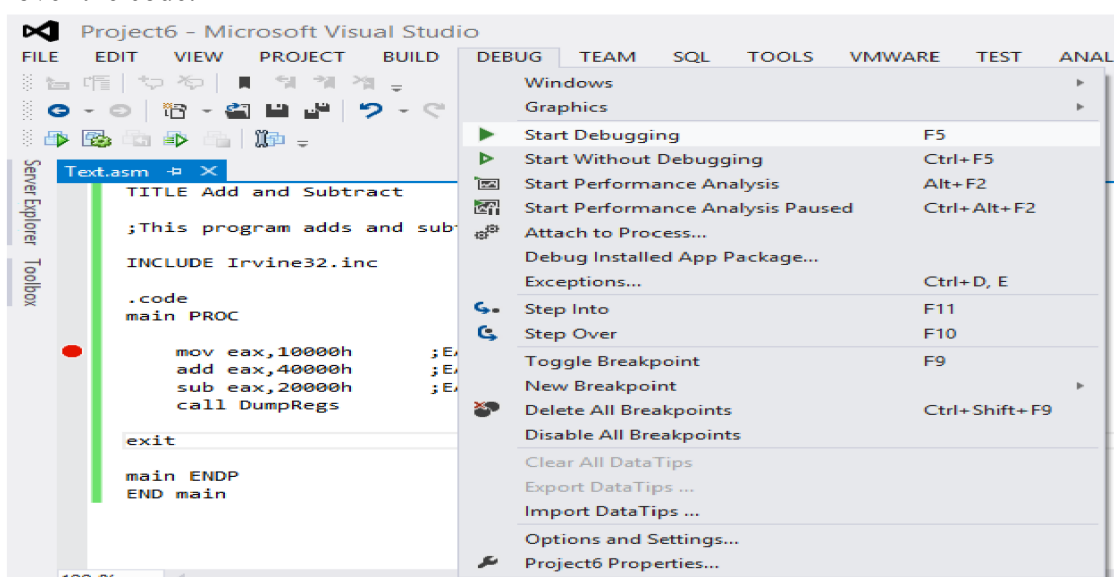
To see the values of internal registers and memory variables during execution, let us use an example. Copy the following code onto your Test.asm file.

```
TITLE Add and Subtract
INCLUDE Irvine32.inc
.code
main PROC
    mov eax, 10000h    ; EAX =10000h
    add eax, 40000h    ; EAX =50000h
    sub eax, 20000h    ; EAX =30000h
    call DumpRegs
    exit
main ENDP
END main
```

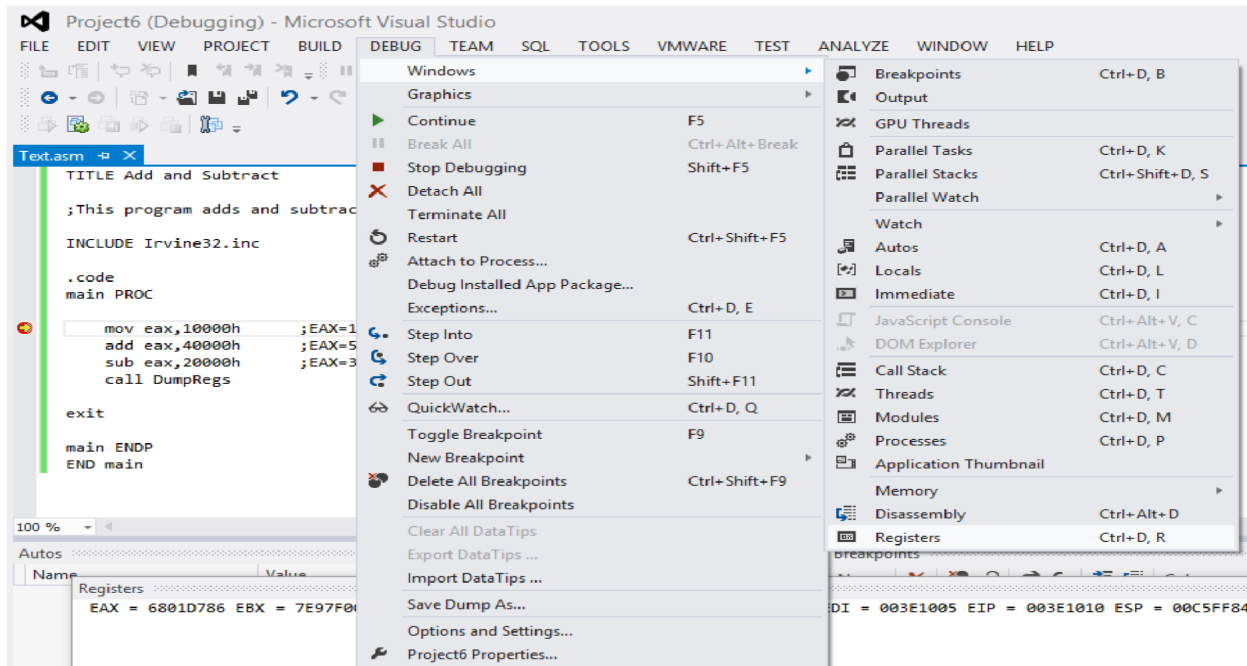
1. Right-click on line 6 to insert a breakpoint.



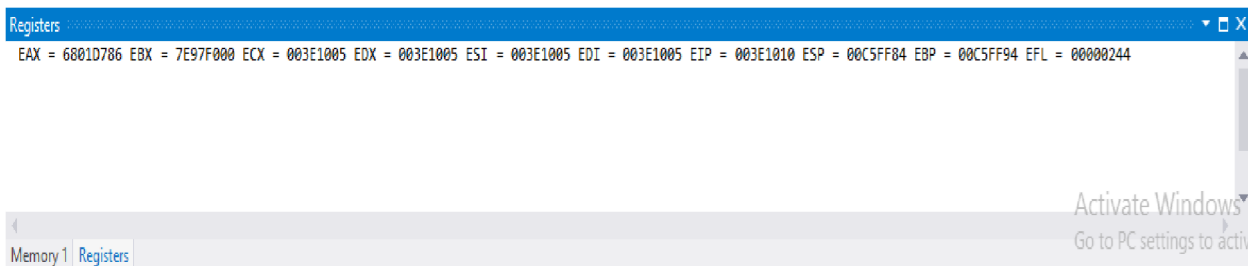
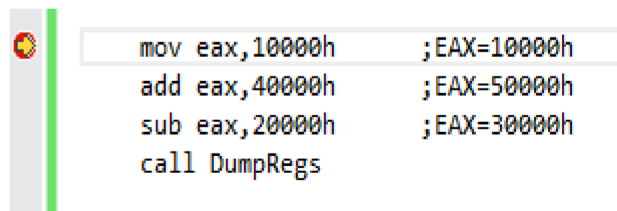
2. Click on **Debug** tab from the toolbar, select **Start Debugging** OR press **F10** to start stepping over the code.




3. Click on **Debug** tab than select Windows after that open menu and select **Registers** option.



4. Breakpoint set on 1st instruction



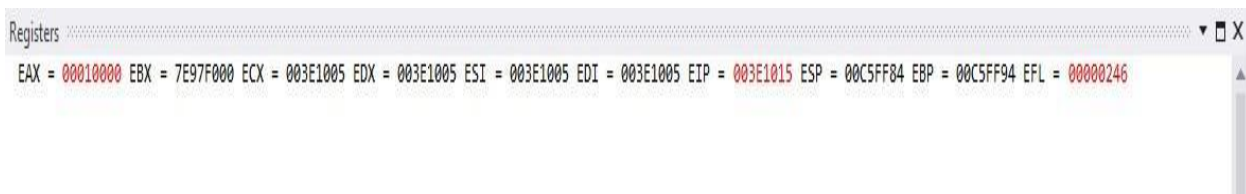
Press **F10** again to execute next line.



```

mov eax,10000h    ;EAX=10000h
add eax,40000h    ;EAX=50000h
sub eax,20000h    ;EAX=30000h
call DumpRegs


```



Registers

EAX = 00010000 EBX = 7E97F000 ECX = 003E1005 EDX = 003E1005 ESI = 003E1005 EDI = 003E1005 EIP = 003E1015 ESP = 00C5FF84 EBP = 00C5FF94 EFL = 00000246

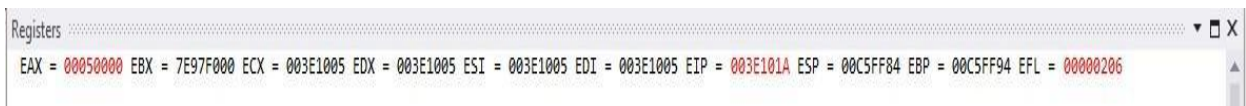
Again press **F10** key for next instruction execution.



```

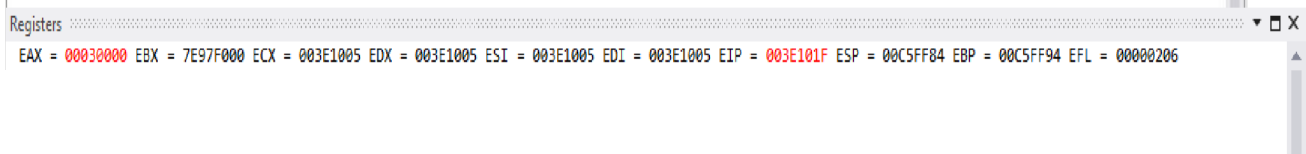
mov eax,10000h    ;EAX=10000h
add eax,40000h    ;EAX=50000h
sub eax,20000h    ;EAX=30000h
call DumpRegs

```



Registers

EAX = 00050000 EBX = 7E97F000 ECX = 003E1005 EDX = 003E1005 ESI = 003E1005 EDI = 003E1005 EIP = 003E101A ESP = 00C5FF84 EBP = 00C5FF94 EFL = 00000206



Registers

EAX = 00030000 EBX = 7E97F000 ECX = 003E1005 EDX = 003E1005 ESI = 003E1005 EDI = 003E1005 EIP = 003E101F ESP = 00C5FF84 EBP = 00C5FF94 EFL = 00000206

Press **F10** again, the program will not terminate after executing the current instruction and as soon as it reaches the line with a call to **DumpRegs**

SECTION 3: INTRODUCTION TO REGISTERS

To speed up the processor operations, the processor includes some internal memory storage locations, called **Registers**. The registers store data elements for processing without having to access the memory.

PROCESSOR REGISTERS

There are ten 32-bit and six 16-bit processor registers in IA-32 architecture. The registers are grouped into three categories:

- General registers.
- Control registers.
- Segment registers.

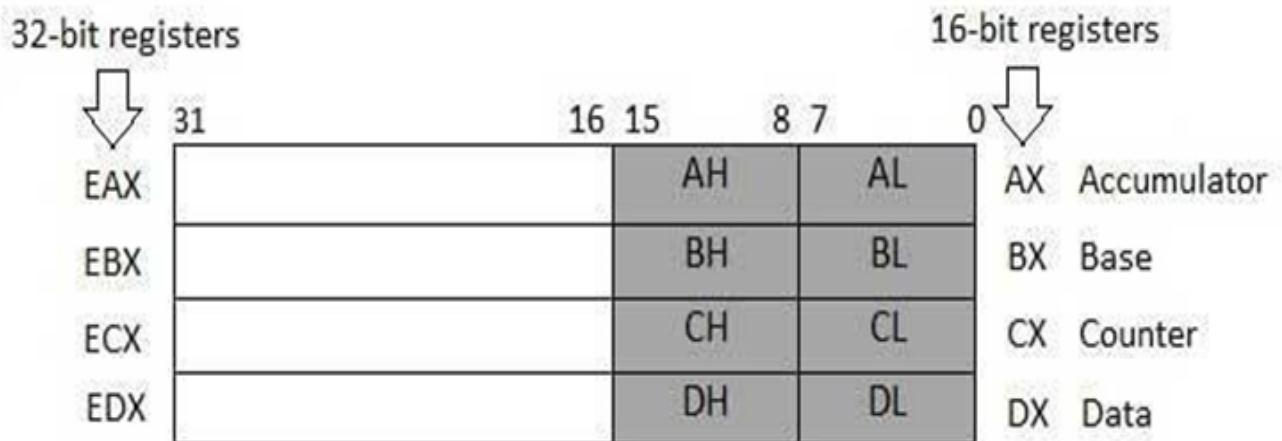
Furthermore, the general registers are further divided into the following groups:

- Data registers.
- Pointer registers.
- Index registers.

DATA REGISTERS

Four 32-bit data registers are used for arithmetic, logical, and other operations. These 32-bit registers can be used in three ways:

- As complete 32-bit data registers: EAX, EBX, ECX, EDX.
- Lower halves of the 32-bit registers can be used as four 16-bit data registers: AX, BX, CX and DX.
- Lower and higher halves of the above-mentioned four 16-bit registers can be used as eight 8-bit data registers: AH, AL, BH, BL, CH, CL, DH, and DL



AX (Accumulator): It is used in input/output and most arithmetic instructions. For example, in multiplication operation, one operand is stored in EAX or AX or AL register according to the size of the operand.

BX (Base register): It could be used in indexed addressing.

CX (Counter register): The ECX, CX registers store the loop count in iterative operations.

DX (Data register): It is also used in input/output operations. It is also used with AX register along with DX for multiply and division operations involving large values.

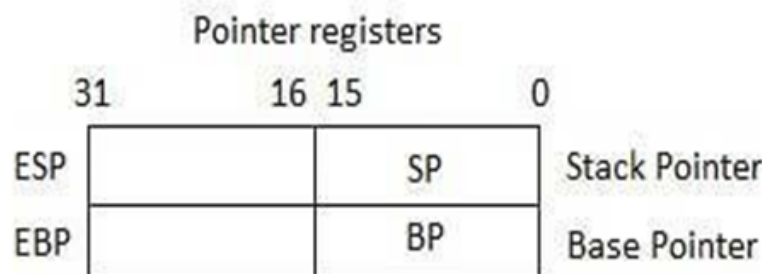
POINTER REGISTERS

The pointer registers are 32-bit EIP, ESP, and EBP registers and corresponding 16-bit right portions IP, SP, and BP. There are three categories of pointer registers:

Instruction Pointer (IP): The 16-bit IP register stores the offset address of the next instruction to be executed. IP in association with the CS register (as CS:IP) gives the complete address of the current instruction in the code segment.

Stack Pointer (SP): The 16-bit SP register provides the offset value within the program stack. SP in association with the SS register (SS:SP) refers to the current position of data or address within the program stack.

Base Pointer (BP): The 16-bit BP register mainly helps in referencing the parameter variables passed to a subroutine. The address in SS register is combined with the offset in BP to get the location of the parameter. BP can also be combined with DI and SI as base register for special addressing.

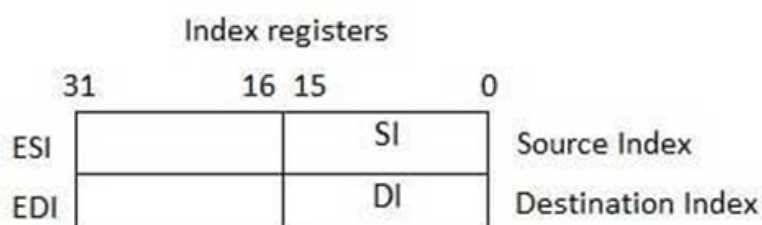


Index Registers

The 32-bit index registers, ESI and EDI, and their 16-bit rightmost portions, SI and DI, are used for indexed addressing and sometimes used in addition and subtraction. There are two sets of index pointers.

Source Index (SI): It is used as source index for string operations.

Destination Index (DI): It is used as destination index for string operations.



BASIC ELEMENT OF ASSEMBLY LANGUAGE

INTEGER CONSTANTS

Integer constants are made up of an optional leading sign, one or more digits and an optional suffix character.

Format:

[{+ | -}] digits radix

Examples:

26	for decimal
26d	for decimal
10111110b	for binary
42o	for octal
1Ah	for Hexadecimal
0A3h	for Hexadecimal

CHARACTER CONSTANTS

Character constants are made up of a single character enclosed in either single or double quotes.

Example:

'A' "d"

STRING CONSTANTS

A string of characters enclosed in either single or double quotes.

Example:

"Hello World"

IDENTIFIERS

An identifier is a programmer-defined name of a variable, procedure or code label.

Format:

They may contain between 1 and 247 characters. They are not case sensitive. The first character must be a letter (A..Z, a..z), underscore (_), @, ?, or \$. Subsequent characters may also be digits.

An identifier cannot be the same as an assembler reserved word. For example: reserved words are instruction mnemonics, directives, attributes, operators, predefined symbols.

Examples:

myVar
_abc
hello2

MEMORY SEGMENTS

A segmented memory model divides the system memory into groups of independent segments referenced by pointers located in the segment registers. Each segment defines the area of our program that contains data variables, code and stack, respectively.

Data segment: It is the memory region, where data elements are stored for the program. This section cannot be expanded after the data elements are declared, and it remains static throughout the program.

Code segment: This section defines an area in memory that stores the instruction codes. This is also a fixed area.

Stack segment: This segment contains data values passed to procedures within the program.

DIRECTIVES

A directive is a command embedded in the source code that is recognized and acted upon by the assembler. Directives do not execute at runtime. They can assign names to memory segments. In

MASM, directives are case insensitive. For example, it recognizes `.data`, `.DATA` and `.Data` as equivalent.

Let us see what different directives we can use to define segments of our program:

The **.DATA** directive identifies the area of a program containing variables:

Syntax:

`.data`

The **.CODE** directive identifies the area of a program containing executable instructions:

Syntax:

`.code`

The **.STACK** directive identifies the area of a program holding the runtime stack, setting its size:

Syntax:

`.stack 100h`

INSTRUCTIONS

An *instruction* is a statement that becomes executable when a program is assembled. Instructions are translated by the assembler into machine language bytes, which are loaded and executed by the CPU at runtime. An instruction contains four basic parts:

- Label (optional)
- Instruction mnemonic (required)
- Operand(s) (usually required)
- Comment (optional)

The basic syntax of an Assembly Language instruction is as:

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

where elements in square brackets are optional. We will now see what each of these elements.

Label: A *label* is an identifier that acts as a place marker for instructions and data. A label placed just before an instruction implies the instruction's address. Similarly, a label placed just before a variable implies the variable's address.

Mnemonics: An instruction mnemonic is a short word that identifies an instruction to perform an operation. Following are examples of instruction mnemonics:

mov: Moves (assigns) one value to another.

add: Adds two values

sub: Subtracts one value from another

mul: Multiplies two values

jmp: Jumps to a new location

call: Calls a procedure

Operands: Assembly language instructions can have between zero and three operands, each of which can be a register, memory operand, constant expression, or input-output port.

Example	Operand Type
96	Constant (<i>immediate value</i>)
2 + 4	Constant expression
eax	Register
count	Memory

Example:

The MOV instruction has two operands:

mov count, ebx ; move EBX to count

In a two-operand instruction, the first operand is called the destination. The second operand is the source. In general, the contents of the destination operand are modified by the instruction.

Comments: Comments are an important way for the writer of a program to communicate information about the program's design to a person reading the source code.

Comments can be specified in two ways:

- **Single-line comments:** beginning with a semicolon character (;). All characters following the semicolon on the same line are ignored by the assembler.
- **Block comments:** beginning with the COMMENT directive and a user-specified symbol. For example,

```
COMMENT !
```

```
This line is a comment.
```

```
This line is also a comment.
```

```
!
```

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
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit integer. T stands for ten

SECTION 4: EXERCISE

Implement all of these equations in assembly language.

- $47 + 39 + 60 + 85 + 64 + 54 - 0Ah$
- $30 - 9 + 186 - 150$
- $101110 + 50Ah + 6710d + 1010001 + F$
- $10001101 - D83h + 385 + 10 + 1111101 - E + F$

Write a program in assembly language that implements following expression:

- $edx = eax + 1 + ebx + edx - ecx + 0Ah - 65o + 73d$
- $eax = 5ADh - eax + 65o + 65d - 11110111 + 150$

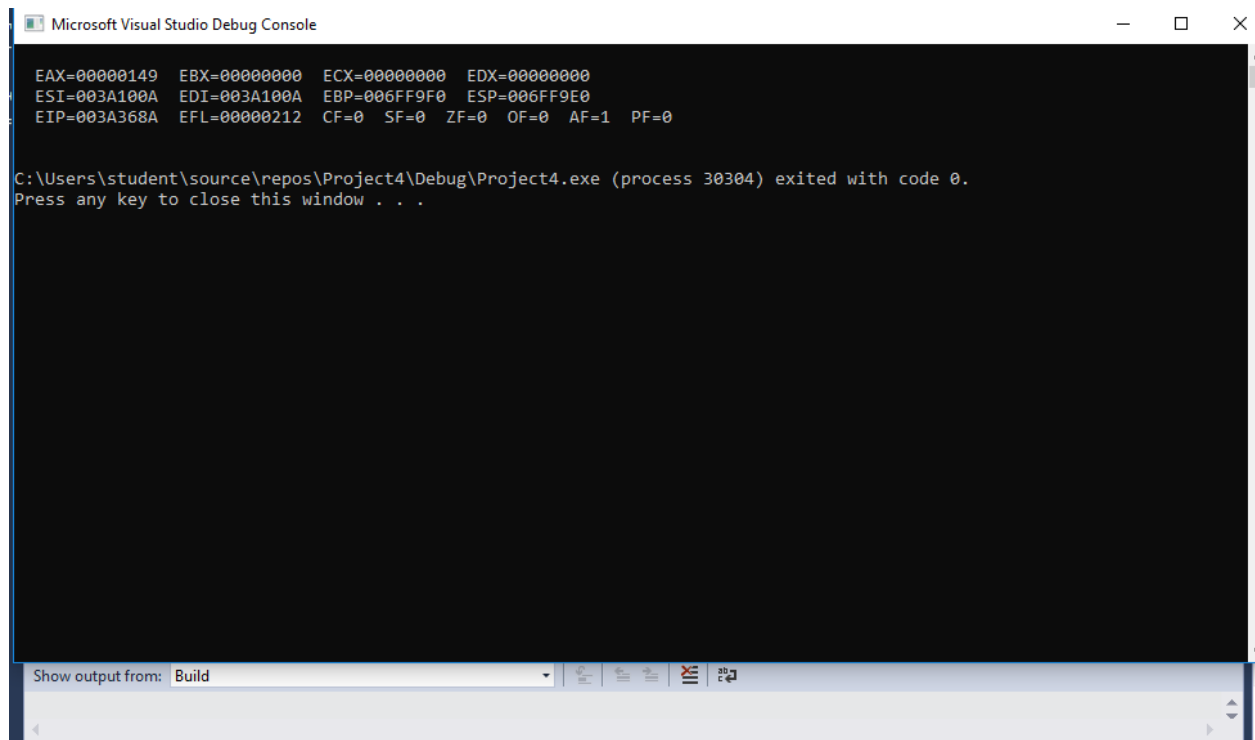
COAL LAB 2:

Task 1:



```
1  TITLE Task 1
2
3  INCLUDE Irvine32.inc
4
5  .code
6
7  main PROC
8
9  xor eax,eax ;initializing all the registers to zero
10 xor ebx,ebx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60
19 add eax,85
20 add eax,64
21 add eax,540
22 sub eax,0Ah
23
24 call DumpRegs
25 exit
26 main ENDP
27 END main
```

OUTPUT:



```
Microsoft Visual Studio Debug Console

EAX=00000149  EBX=00000000  ECX=00000000  EDX=00000000
ESI=003A100A  EDI=003A100A  EBP=006FF9F0  ESP=006FF9E0
EIP=003A368A  EFL=00000212  CF=0  SF=0  ZF=0  OF=0  AF=1  PF=0

C:\Users\student\source\repos\Project4\Debug\Project4.exe (process 30304) exited with code 0.
Press any key to close this window . . .
```

DEBUG OF TASK 1:

The image displays two screenshots of the Visual Studio debugger interface, showing the execution of an assembly program.

Top Screenshot:

- Registers:** EAX = 00000000, EBX = 00000000, ECX = 00000000, EDX = 00000000, ESI = 003A100A, EDI = 003A100A, EIP = 003A366E, ESP = 00CFFF04, EBP = 00CFFF14, EFL = 00000246.
- Assembly Code (Text.asm):**

```
10 xor ebx,ebx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60
19 add eax,85
20 add eax,64
21 add eax,540
22 csh eax,0Ah
```
- Autos:** Empty table.
- Call Stack:** Project4.exe!main() Line 16 [External Code].
- Diagnostic Tools:** Diagnostics session: 0 seconds. Events: 1 of 1.

Bottom Screenshot:

- Registers:** EAX = 0000002F, EBX = 00000000, ECX = 00000000, EDX = 00000000, ESI = 003A100A, EDI = 003A100A, EIP = 003A3673, ESP = 00CFFF04, EBP = 00CFFF14, EFL = 00000246.
- Assembly Code (Text.asm):**

```
10 xor ebx,ebx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39 51ms elapsed
18 add eax,60
19 add eax,85
20 add eax,64
21 add eax,540
22 csh eax,0Ah
```
- Autos:** Empty table.
- Call Stack:** Project4.exe!main() Line 17 [External Code].
- Diagnostic Tools:** Diagnostics session: 0 seconds (14 ms selected). Events: 2 of 2.

Process: [6488] Project4.exe Lifecycle Events Thread: [1020] Main Thread Stack Frame: main

Registers
EAX = 00000056 EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 003A100A EDI = 003A100A EIP = 003A3676 ESP = 00CFFF04 EBP = 00CFFF14 EFL = 00000216

Text.asm
10 xor edx,edx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60 51ms elapsed
19 add eax,85
20 add eax,64
21 add eax,540
22 sub eax,0Ah

100 %

Autos

Name	Value	Type
------	-------	------

Call Stack

Name	Lang
Project4.exe!main() Line 18	Un...
[External Code]	Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll]	Un...

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (3 of 3)

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Registers
EAX = 00000092 EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 003A100A EDI = 003A100A EIP = 003A3679 ESP = 00CFFF04 EBP = 00CFFF14 EFL = 00000212

Text.asm
10 xor edx,edx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60
19 add eax,85 51ms elapsed
20 add eax,64
21 add eax,540
22 sub eax,0Ah

100 %

Autos

Name	Value	Type
------	-------	------

Call Stack

Name	Lang
Project4.exe!main() Line 19	Un...
[External Code]	Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll]	Un...

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (4 of 4)

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Registers
EAX = 000000E7 EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 003A100A EDI = 003A100A EIP = 003A367C ESP = 00CFFF04 EBP = 00CFFF14 EFL = 00000206

Text.asm
10 xor edx,edx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60
19 add eax,85
20 add eax,64 51ms elapsed
21 add eax,540
22 sub eax,0Ah

100 %

Autos

Name	Value	Type
------	-------	------

Call Stack

Name	Lang
Project4.exe!main() Line 20	Un...
[External Code]	Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll]	Un...

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (5 of 5)

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Process: [6488] Project4.exe Lifecycle Events Thread: [1020] Main Thread Stack Frame: main

Registers
EAX = 00000127 EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 003A100A EDI = 003A100A EIP = 003A367F ESP = 00CFFF04 EBP = 00CFFF14 EFL = 00000206

Text.asm
10 xor edx,edx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60
19 add eax,85
20 add eax,64
21 add eax,540
22 sub eax,0Ah

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
13.2ms
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (6 of 6)

Autos
Name Value Type

Call Stack
Name Language
Project4.exe!main() Line 21 Un...
[External Code] Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll] Un...

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Loading symbols for sechost.dll Ln 21 Col 1 Ch 1 INS Add to Source Control Sign in

Process: [6488] Project4.exe Lifecycle Events Thread: [1020] Main Thread Stack Frame: main

Registers
EAX = 00000153 EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 003A100A EDI = 003A100A EIP = 003A3682 ESP = 00CFFF04 EBP = 00CFFF14 EFL = 00000216

Text.asm
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,47
17 add eax,39
18 add eax,60
19 add eax,85
20 add eax,64
21 add eax,540
22 sub eax,0Ah

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
13.2ms
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (7 of 7)

Autos
Name Value Type

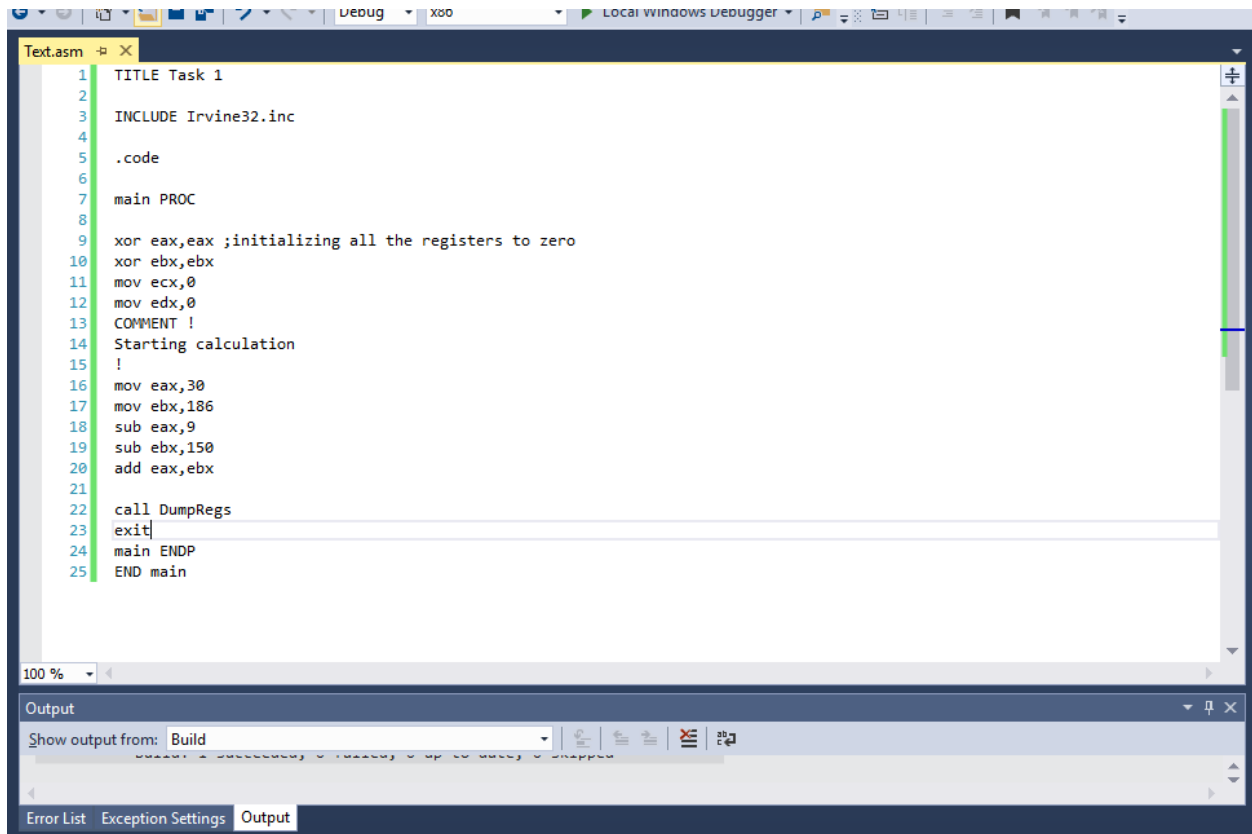
Call Stack
Name Language
Project4.exe!main() Line 22 Un...
[External Code] Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll] Un...

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Loading symbols for sechost.dll Ln 22 Col 1 Ch 1 INS Add to Source Control

TASK 2:

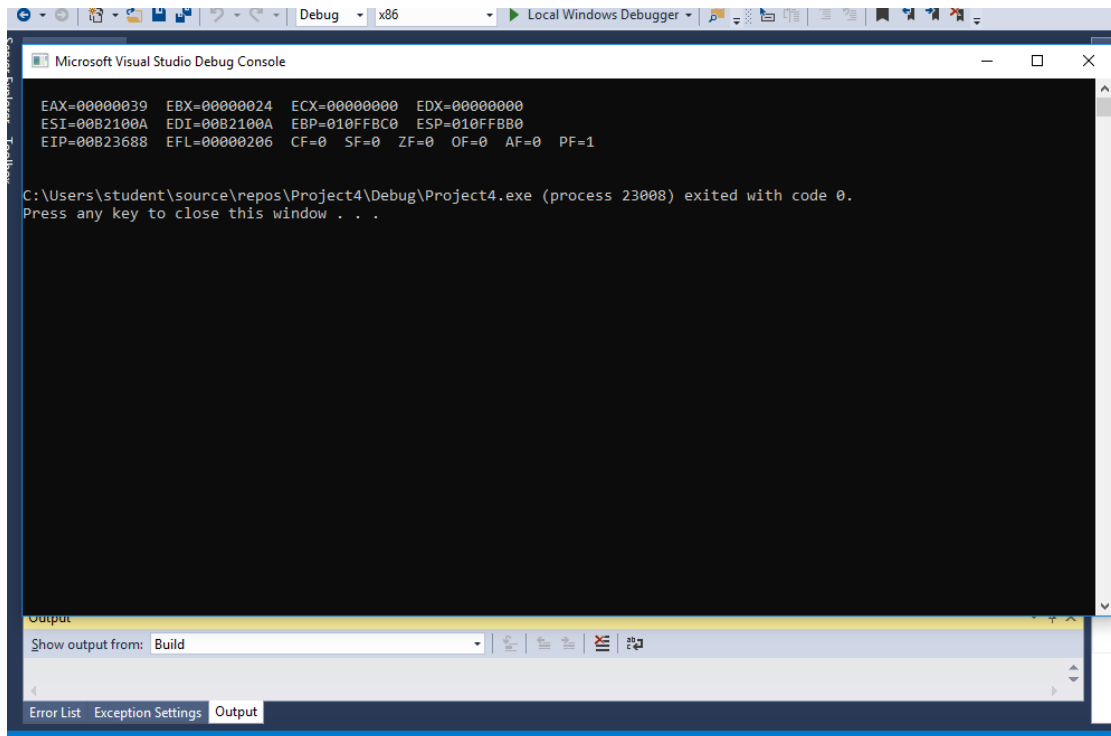


The screenshot shows a text editor window titled 'Text.asm' with the following assembly code:

```
1  TITLE Task 1
2
3  INCLUDE Irvine32.inc
4
5  .code
6
7  main PROC
8
9  xor eax,eax ;initializing all the registers to zero
10 xor ebx,ebx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,30
17 mov ebx,186
18 sub eax,9
19 sub ebx,150
20 add eax,ebx
21
22 call DumpRegs
23 exit
24 main ENDP
25 END main
```

Below the code editor is an 'Output' window showing the build output. The 'Error List' and 'Exception Settings' tabs are also visible at the bottom.

OUTPUT:



The screenshot shows the 'Microsoft Visual Studio Debug Console' window. It displays the following register values and status flags:

```
EAX=00000039  EBX=00000224  ECX=00000000  EDX=00000000
ESI=0002100A  EDI=0002100A  EBP=010FFB00  ESP=010FFB80
EIP=00B23688  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1
```

Below the register values, it shows the exit message:

```
C:\Users\student\source\repos\Project4\Debug\Project4.exe (process 23008) exited with code 0.
Press any key to close this window . . .
```

The 'Output' window at the bottom shows the build output, and the 'Error List' and 'Exception Settings' tabs are also visible.

DEBUG OF TASK 2:

The image displays two screenshots of the Visual Studio debugger interface, showing the execution of an assembly program.

Top Screenshot:

- Process:** [23424] Project4.exe
- Thread:** [47704] Main Thread
- Stack Frame:** main
- Registers:** EAX = 00000000 EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 00B2100A EDI = 00B2100A EIP = 00B2366E ESP = 00F3FC50 EBP = 00F3FC60 EFL = 00000246
- Assembly Code (ext.asm):**
 - 10 xor ebx,ebx
 - 11 mov ecx,0
 - 12 mov edx,0
 - 13 COMMENT !
 - 14 Starting calculation
 - 15 !
 - 16 mov eax,30
 - 17 mov ebx,186
 - 18 sub eax,9
 - 19 sub ebx,150
 - 20 add eax,ebx
 - 21
 - 22 call DumpRegs
- Diagnostic Tools:** Diagnostics session: 0 seconds. Events: 1 of 1.
- Call Stack:** Project4.exe!main() Line 16 [External Code].

Bottom Screenshot:

- Process:** [23424] Project4.exe
- Thread:** [47704] Main Thread
- Stack Frame:** main
- Registers:** EAX = 0000001E EBX = 00000000 ECX = 00000000 EDX = 00000000 ESI = 00B2100A EDI = 00B2100A EIP = 00B23673 ESP = 00F3FC50 EBP = 00F3FC60 EFL = 00000246
- Assembly Code (Text.asm):**
 - 10 xor ebx,ebx
 - 11 mov ecx,0
 - 12 mov edx,0
 - 13 COMMENT !
 - 14 Starting calculation
 - 15 !
 - 16 mov eax,30
 - 17 mov ebx,186
 - 18 sub eax,9
 - 19 sub ebx,150
 - 20 add eax,ebx
 - 21
 - 22 call DumpRegs
- Diagnostic Tools:** Diagnostics session: 0 seconds (14 ms selected). Events: 2 of 2.
- Call Stack:** Project4.exe!main() Line 17 [External Code].

Registers
EAX = 0000001E EBX = 0000000A ECX = 00000000 EDX = 00000000 ESI = 00B2100A EDI = 00B2100A EIP = 00B23678 ESP = 00F3FC50 EBP = 00F3FC60 EFL = 00000246

Text.asm
10 xor edx,edx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,30
17 mov ebx,186
18 sub eax,9 51ms elapsed
19 sub ebx,150
20 add eax,ebx
21
22 call DumpRegs

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
12.4ms 12.8ms
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (3 of 3)

Autos
Name Value Type

Call Stack
Name Language
Project4.exe\main() Line 18 Un...
[External Code]
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll] Un...

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Registers
EAX = 00000015 EBX = 0000000A ECX = 00000000 EDX = 00000000 ESI = 00B2100A EDI = 00B2100A EIP = 00B23678 ESP = 00F3FC50 EBP = 00F3FC60 EFL = 00000202

Text.asm
10 xor edx,edx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,30
17 mov ebx,186
18 sub eax,9
19 sub ebx,150 51ms elapsed
20 add eax,ebx
21
22 call DumpRegs

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
12.4ms
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (4 of 4)

Autos
Name Value Type

Call Stack
Name Language
Project4.exe\main() Line 19 Un...
[External Code]
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll] Un...

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Ready Ln 19 Col 1 Ch 1 INS Add to Source Control

Process: [23424] Project4.exe Lifecycle Events Thread: [47704] Main Thread Stack Frame: main

Registers
EAX = 00000015 EBX = 00000024 ECX = 00000000 EDX = 00000000 ESI = 00B2100A EDI = 00B2100A EIP = 00B23681 ESP = 00F3FC50 EBP = 00F3FC60 EFL = 00000206

Textasm
10 xor ecx,ecx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,30
17 mov ebx,186
18 sub eax,9
19 sub ebx,150
20 add eax,ebx ; 51ms elapsed
21
22 call DumpRegs

Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
12.8ms
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (5 of 5)

Autos
Name Value Type

Call Stack
Name Language
Project4.exe!main() Line 20 Un...
[External Code] Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll] Un...

Autos Locals Watch 1

Call Stack Breakpoints Exception Settings Command Window Immediate Window Output

Process: [23424] Project4.exe Lifecycle Events Thread: [47704] Main Thread Stack Frame: main

Registers
EAX = 00000039 EBX = 00000024 ECX = 00000000 EDX = 00000000 ESI = 00B2100A EDI = 00B2100A EIP = 00B23683 ESP = 00F3FC50 EBP = 00F3FC60 EFL = 00000206

Textasm
10 xor ecx,ecx
11 mov ecx,0
12 mov edx,0
13 COMMENT !
14 Starting calculation
15 !
16 mov eax,30
17 mov ebx,186
18 sub eax,9
19 sub ebx,150
20 add eax,ebx
21
22 call DumpRegs ; 51ms elapsed

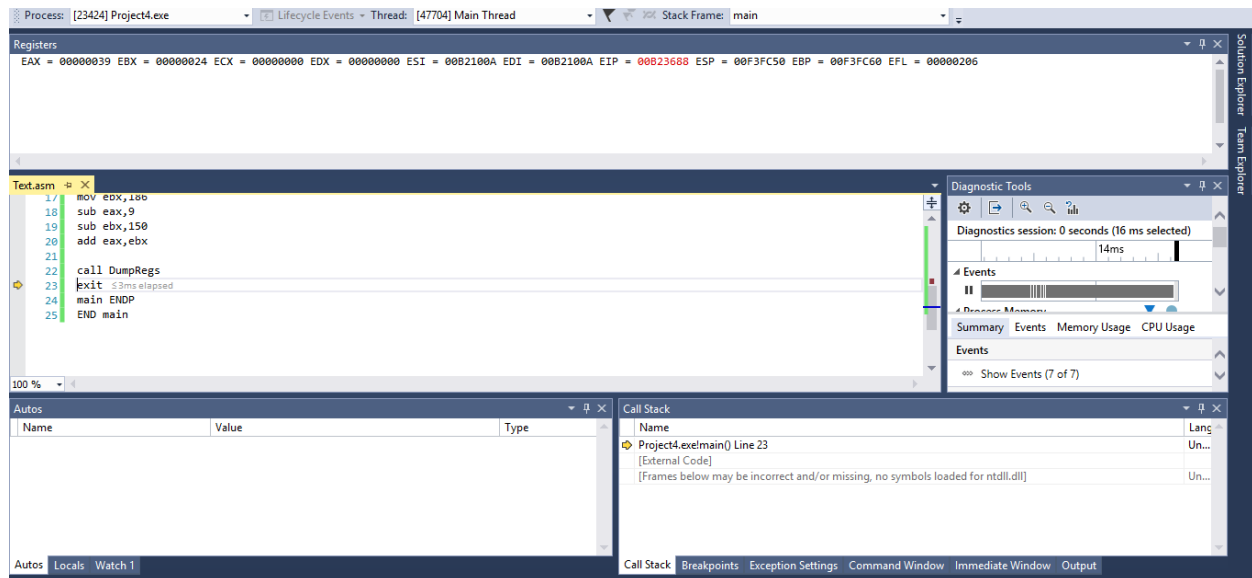
Diagnostic Tools
Diagnostics session: 0 seconds (14 ms selected)
12.4ms 12.4ms
Events
Summary Events Memory Usage CPU Usage
Events
Show Events (6 of 6)

Autos
Name Value Type

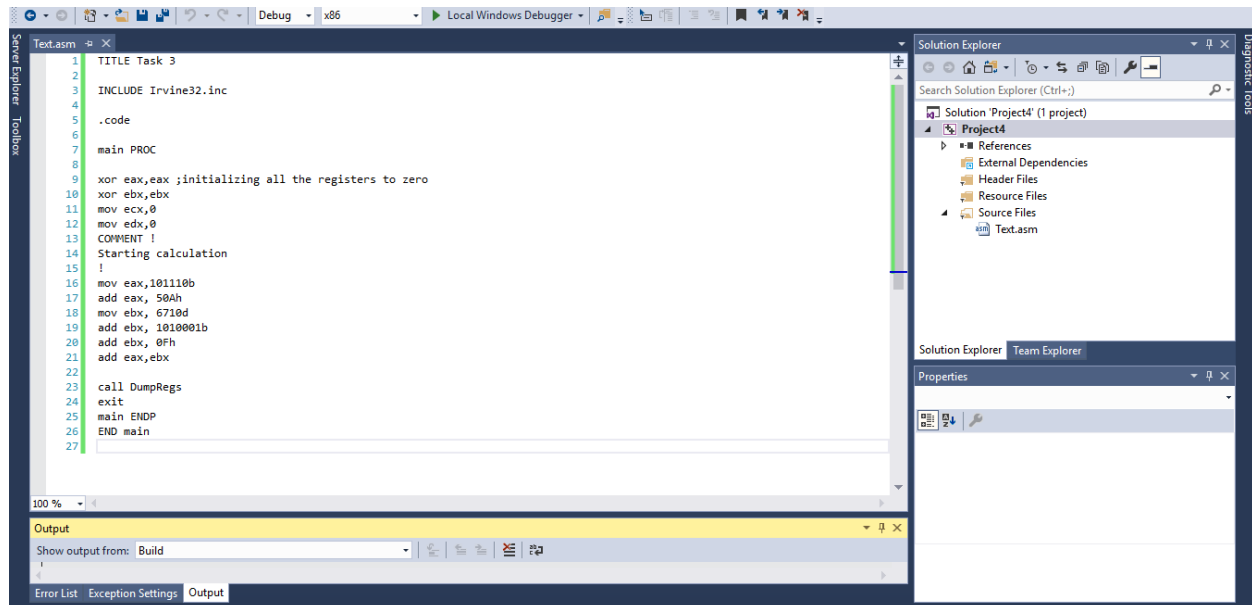
Call Stack
Name Language
Project4.exe!main() Line 22 Un...
[External Code] Un...
[Frames below may be incorrect and/or missing, no symbols loaded for ntdll.dll] Un...

Autos Locals Watch 1

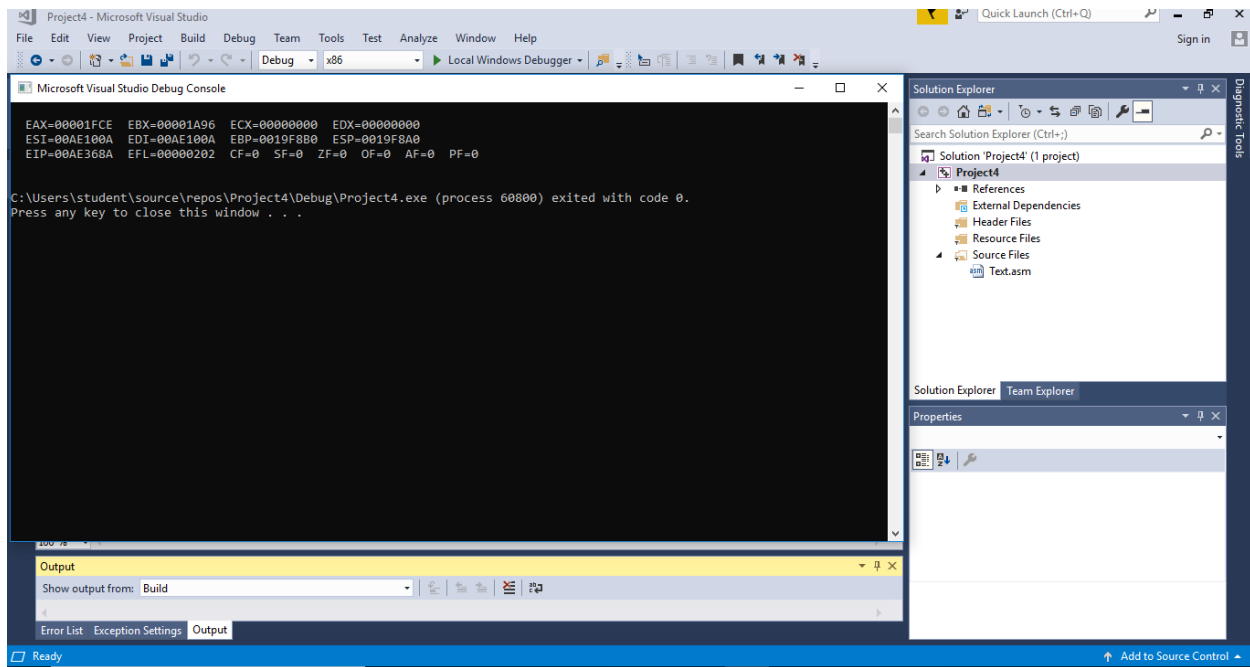
Call Stack Breakpoints Exception Settings Command Window Immediate Window Output



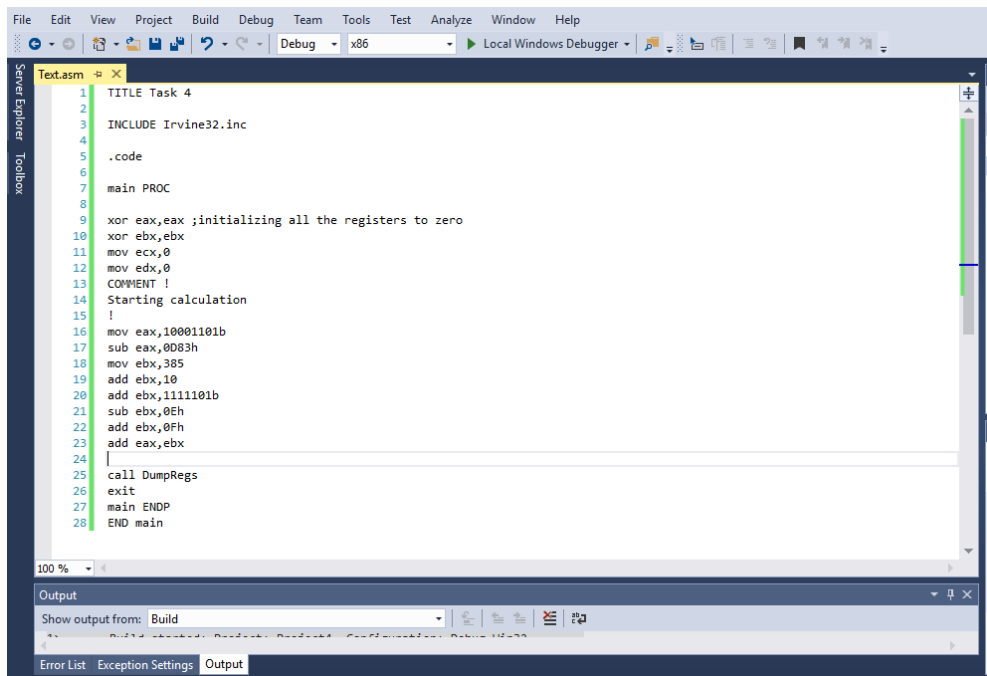
TASK 3:



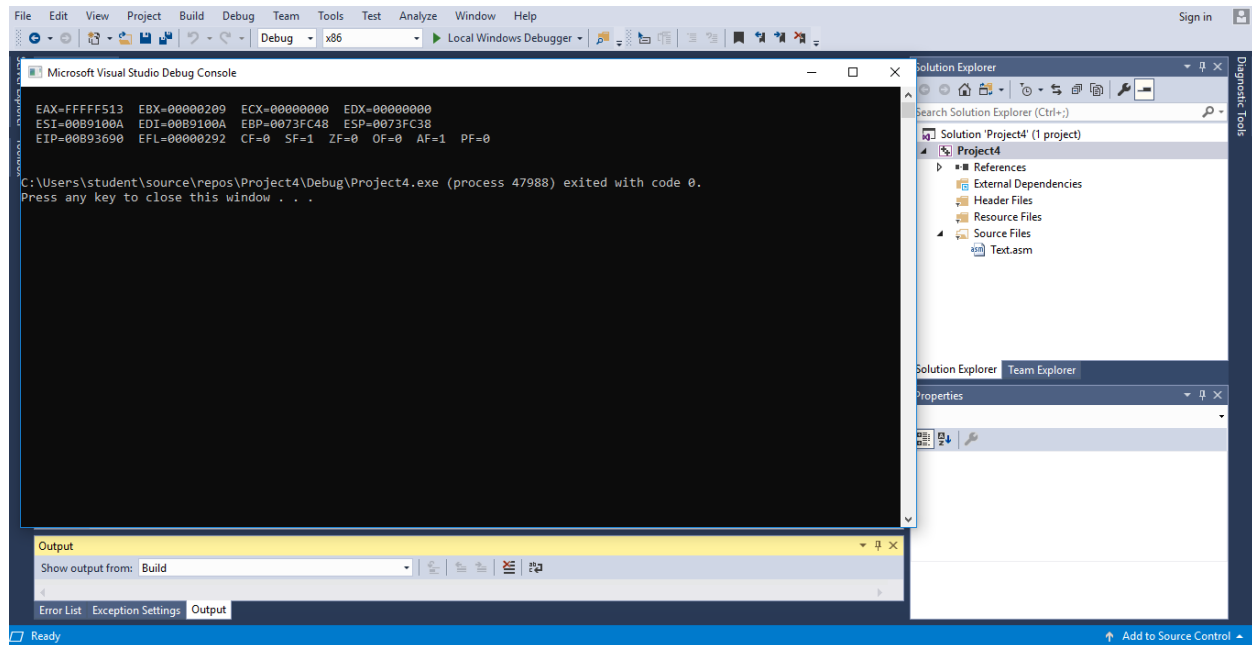
OUTPUT:



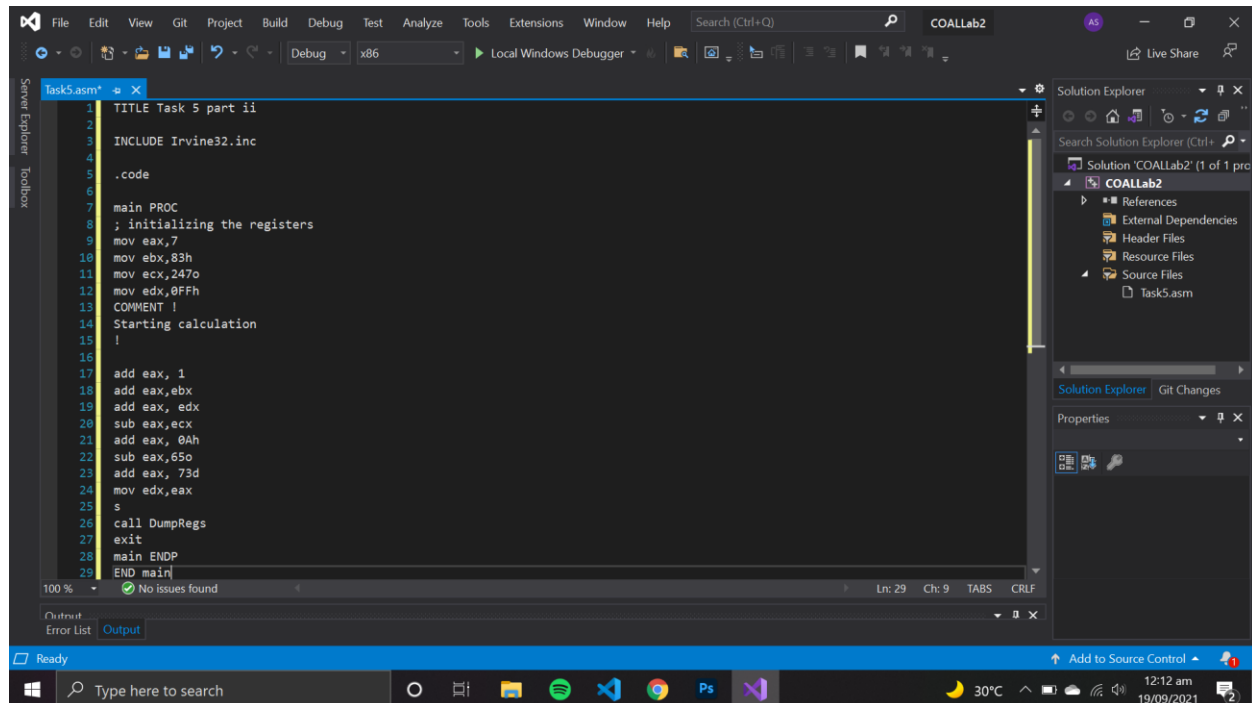
TASK 4:



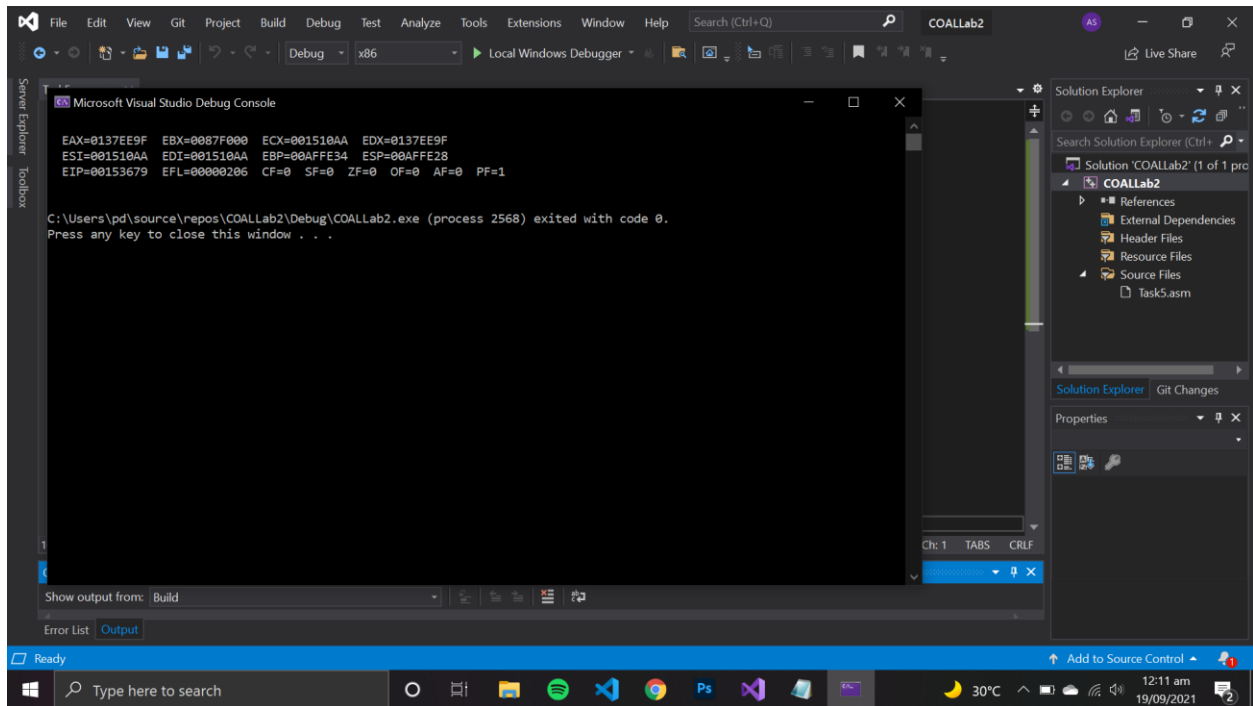
OUTPUT:



TASK 5 PART (i)



OUTPUT:



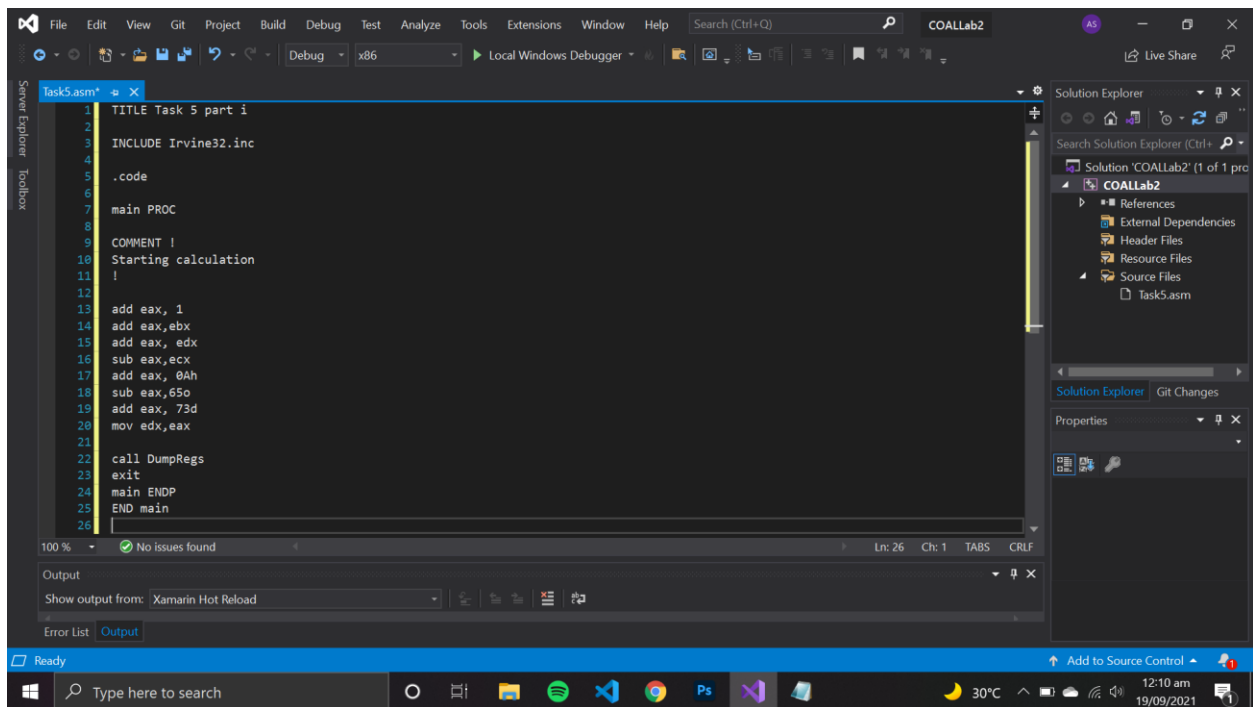
The screenshot shows the Visual Studio IDE with the 'Microsoft Visual Studio Debug Console' open. The console displays the following output:

```
EAX=0137EE9F  EBX=0087F000  ECX=001510AA  EDX=0137EE9F
ESI=001510AA  EDI=001510AA  EBP=00AFFE34  ESP=00AFFE28
EIP=00153679  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1

C:\Users\pd\source\repos\COALLab2\Debug\COALLab2.exe (process 2568) exited with code 0.
Press any key to close this window . . .
```

The Solution Explorer on the right shows the project structure for 'COALLab2', including 'References', 'External Dependencies', 'Header Files', 'Resource Files', and 'Source Files' (containing 'Task5.asm').

TASK 5 PART (ii)

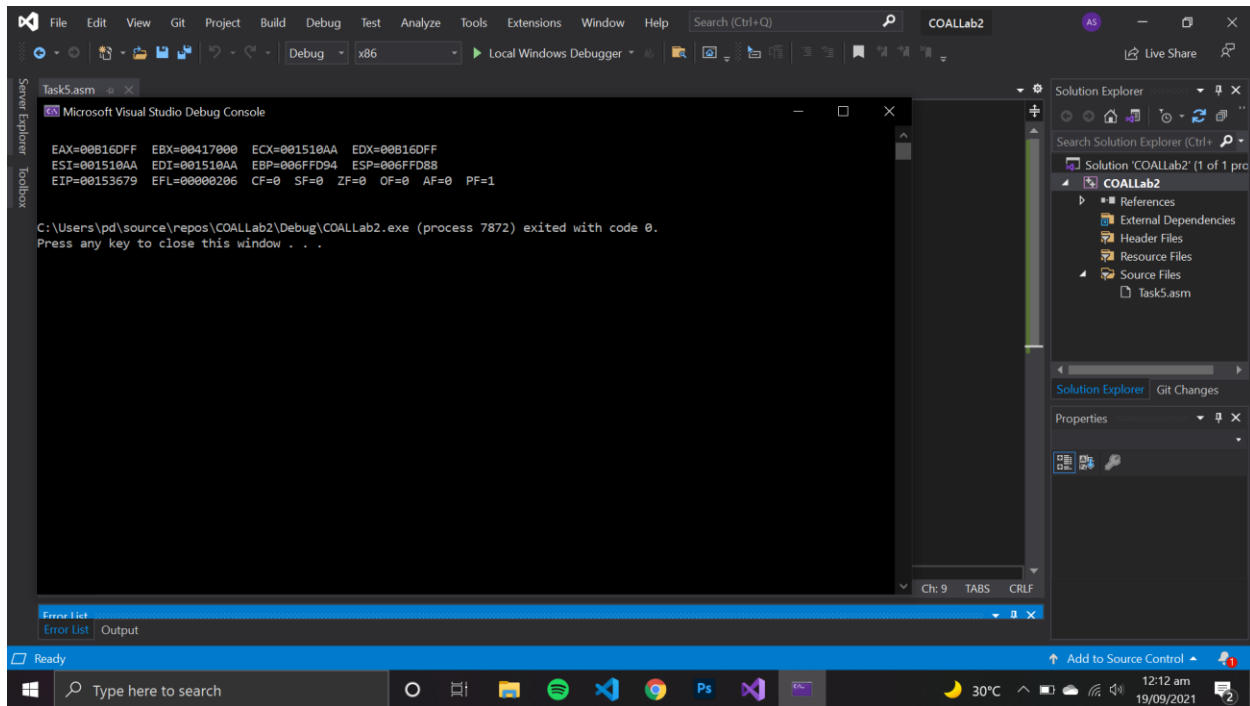


The screenshot shows the Visual Studio IDE with the 'Task5.asm' file open. The assembly code is as follows:

```
1  TITLE Task 5 part i
2
3  INCLUDE Irvine32.inc
4
5  .code
6
7  main PROC
8
9  COMMENT !
10 Starting calculation
11 !
12
13 add eax, 1
14 add eax, ebx
15 add eax, edx
16 sub eax, ecx
17 add eax, 0Ah
18 sub eax, 650
19 add eax, 73d
20 mov edx, eax
21
22 call DumpRegs
23 exit
24 main ENDP
25 END main
26
```

The Solution Explorer on the right shows the project structure for 'COALLab2', including 'References', 'External Dependencies', 'Header Files', 'Resource Files', and 'Source Files' (containing 'Task5.asm').

OUTPUT:

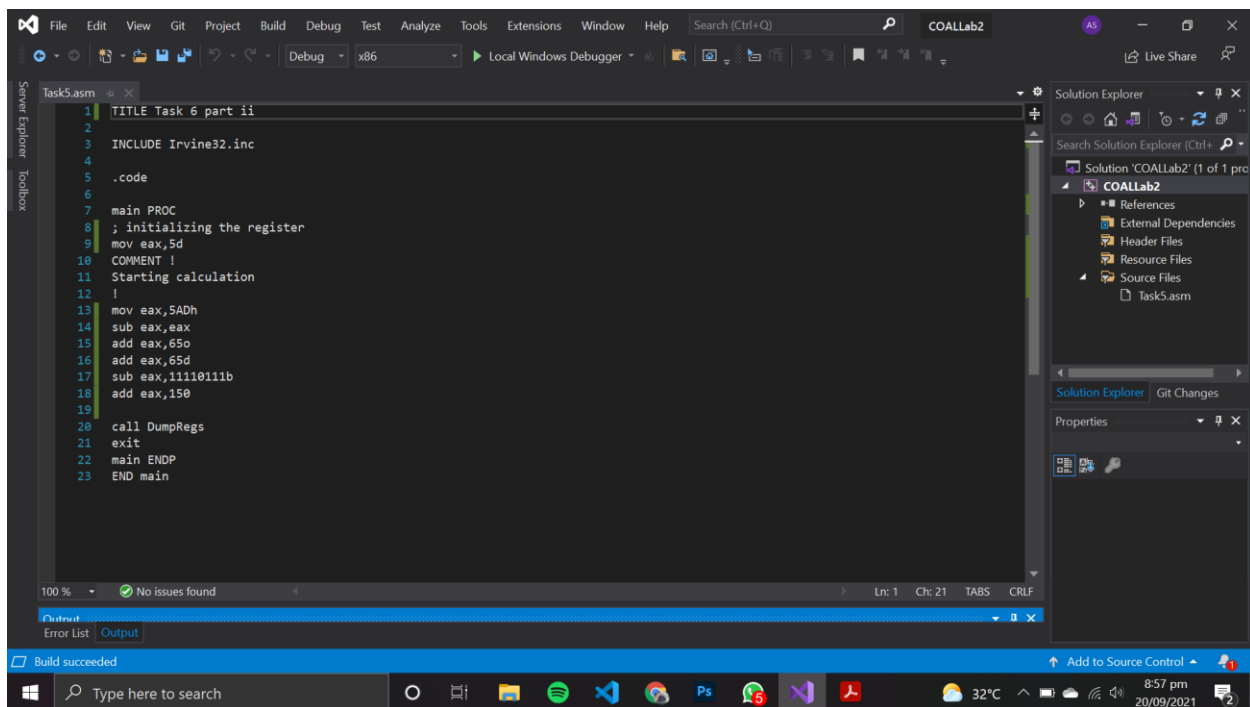


The screenshot shows the Visual Studio IDE with the 'Task5.asm' file open. The 'Debug Console' window displays the following output:

```
EAX=00B16DFF EBX=00417000 ECX=001510AA EDX=00B16DFF  
ESI=001510AA EDI=001510AA EBP=006FFD94 ESP=006FFD88  
EIP=00153679 EFL=00000206 CF=0 SF=0 ZF=0 OF=0 AF=0 PF=1  
  
C:\Users\pd\source\repos\COALLab2\Debug\COALLab2.exe (process 7872) exited with code 0.  
Press any key to close this window . . .
```

The 'Solution Explorer' on the right shows the project structure for 'COALLab2', including 'References', 'External Dependencies', 'Header Files', 'Resource Files', and 'Source Files' (containing 'Task5.asm'). The status bar at the bottom indicates 'Ready'.

TASK 6 PART (i)

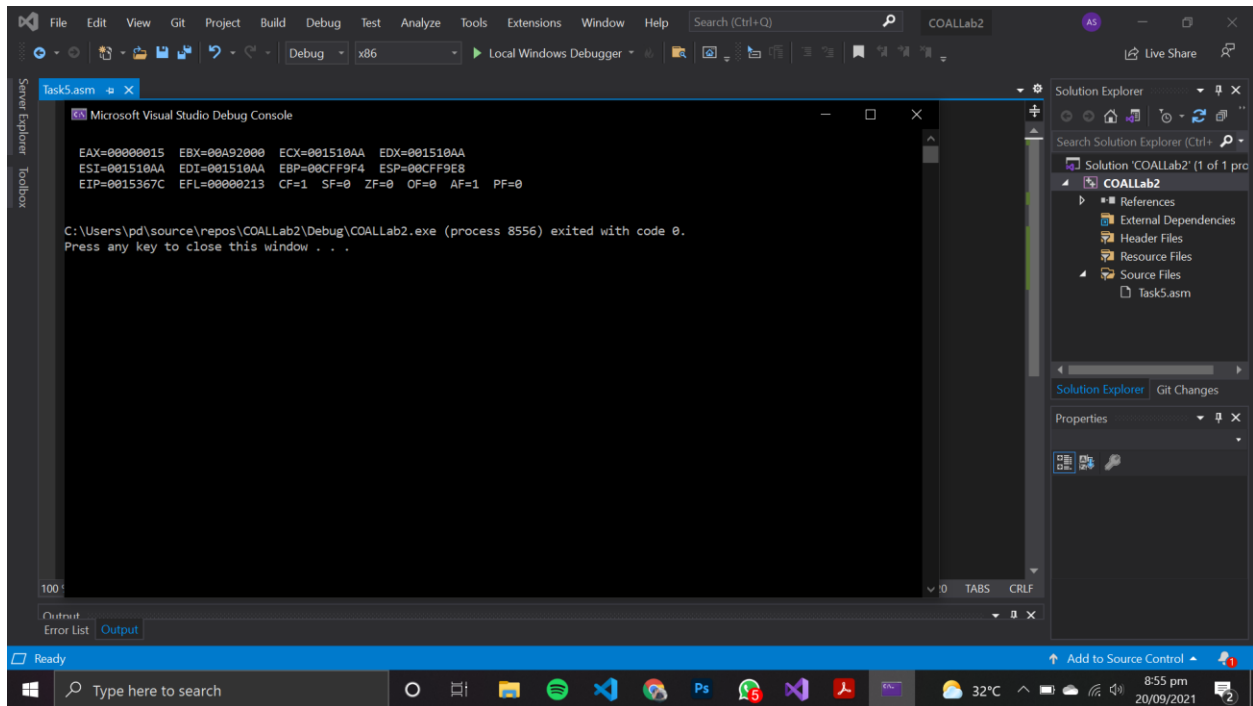


The screenshot shows the Visual Studio IDE with the 'Task5.asm' file open. The assembly code is as follows:

```
1  TITLE Task 6 part ii  
2  
3  INCLUDE Irvine32.inc  
4  
5  .code  
6  
7  main PROC  
8  ; initializing the register  
9  mov eax,5d  
10 COMMENT !  
11 Starting calculation  
12 !  
13 mov eax,5ADh  
14 sub eax,eax  
15 add eax,650  
16 add eax,65d  
17 sub eax,1111011b  
18 add eax,150  
19  
20 call DumpRegs  
21 exit  
22 main ENDP  
23 END main
```

The 'Solution Explorer' on the right shows the project structure for 'COALLab2', including 'References', 'External Dependencies', 'Header Files', 'Resource Files', and 'Source Files' (containing 'Task5.asm'). The status bar at the bottom indicates 'Build succeeded'.

OUTPUT:

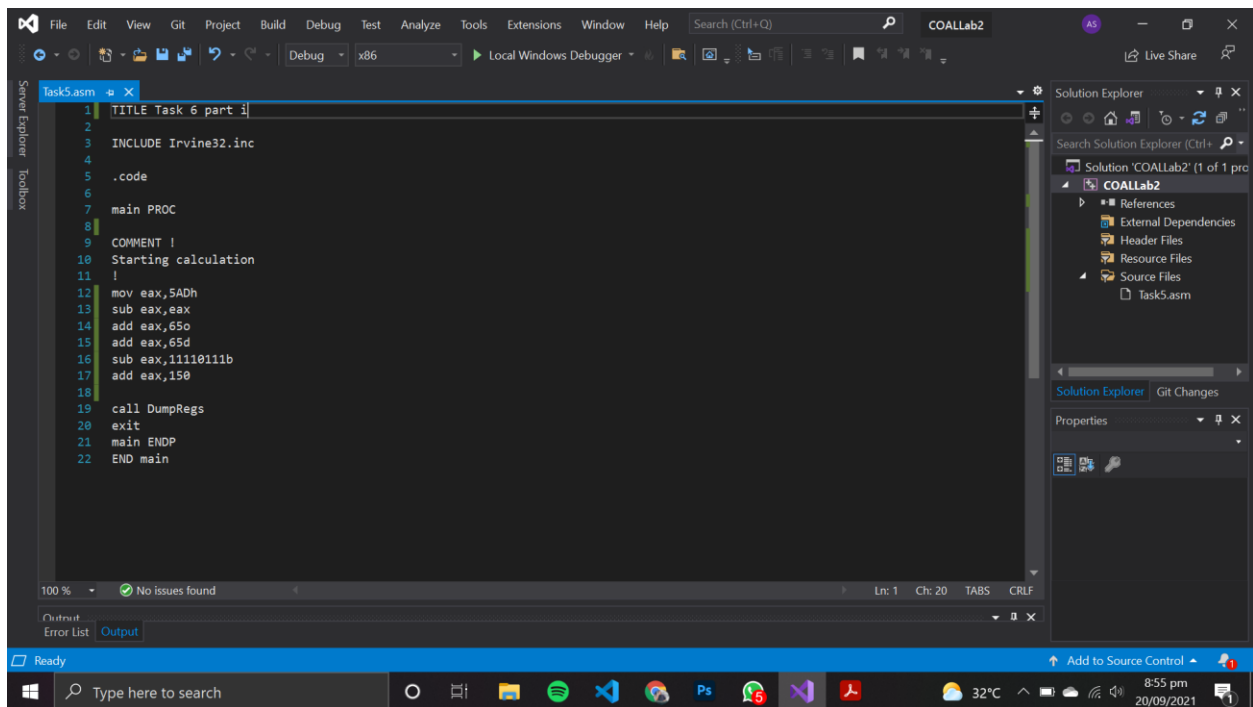


The screenshot shows the Visual Studio IDE with the 'Task5.asm' file open. The 'Debug Console' window displays the following output:

```
EAX=00000015 EBX=00A92000 ECX=001510AA EDX=001510AA  
ESI=001510AA EDI=001510AA EBP=00CFF9F4 ESP=00CFF9E8  
EIP=0015367C EFL=00000213 CF=1 SF=0 ZF=0 OF=0 AF=1 PF=0  
  
C:\Users\pd\source\repos\COALLab2\Debug\COALLab2.exe (process 8556) exited with code 0.  
Press any key to close this window . . .
```

The 'Solution Explorer' on the right shows the project structure for 'COALLab2', including 'References', 'External Dependencies', 'Header Files', 'Resource Files', and 'Source Files' (containing 'Task5.asm').

TASK 6 PART (ii)



The screenshot shows the Visual Studio IDE with the 'Task5.asm' file open. The assembly code is as follows:

```
1  TITLE Task 6 part i  
2  
3  INCLUDE Irvine32.inc  
4  
5  .code  
6  
7  main PROC  
8  
9  COMMENT !  
10 Starting calculation  
11 !  
12 mov eax,5ADh  
13 sub eax,eax  
14 add eax,650  
15 add eax,65d  
16 sub eax,1111011b  
17 add eax,150  
18  
19 call DumpRegs  
20 exit  
21 main ENDP  
22 END main
```

The 'Solution Explorer' on the right shows the project structure for 'COALLab2', including 'References', 'External Dependencies', 'Header Files', 'Resource Files', and 'Source Files' (containing 'Task5.asm').

OUTPUT:

