CSUS

SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Department of Computer Science

CSC 35

Spring 2021

Dr. Ghansah

**Lab #3: Your First Program - Arithmetic Instructions**

**Purpose:** This Lab is intended to help you get started with writing your own Assembly Language Program. You will do so by using appropriate instructions, registers, and memory variables in your program. You will also learn how to use the Debugger if you have not already done so.

A debugger is useful when you want to inspect the values in register or variable at run time or see how the flow of the program takes place. The debugger allows you can **step** through the whole program and notice the change in the register values as you do so. You can verify that the registers are getting loaded with the correct values as specified in your program. You can also verify variable values by setting a ‘**watch’** over them in the debugger.

**HOW TO SET a WATCH in Microsoft Visual Studio Debugger**

While you're debugging, you can use Watch windows and QuickWatch to watch variables and expressions. The windows are only available during a debugging session.

You can use Watch windows to set a watch to show variables and register values in decimal inside the debugger as follows.

Open a **Watch** window by selecting **Debug** > **Windows** > **Watch** > **Watch 1**, or pressing **Ctrl**+**Alt**+**W** > **1**.

You can open additional **Watch** windows by selecting windows **2**, **3**, or **4**. I recommend you leave the additional watch window option for now.

In the Watch window, select an empty row, type name of the variable you want to watch, and press enter. It will display the initial value of the variable in decimal. Use the down arrow key to enter the rest of the variables. You can also watch the value of registers especially if you want to see them displayed in decimal. For instance, you can type EAX and press enter in the watch window to see the values in the EAX register displayed in decimal form.

**Introduction:**

In this assignment you will learn how to write your first assembly language program involving Intel x86 instructions including addition, subtraction, and multiplication. You will also learn how to declare variables in Assembly language.

The following table shows high level language statements (in C-like notation) and the corresponding Intel assembly language statement.

|  |  |  |
| --- | --- | --- |
| High Level | Assembly | Meaning |
| Int X | X DWORD ? | Variable, X, declared as 32bit integer and Uninitialized |
| Int X = 9 | X DWORD 9 | Variable, X, declared as 32bit integer and Initialized to 9 |
| X=2 | Mov X, 2 | Assigning value to a variable |
| X=X+2 | Add X, 2 | Addition |
| X=X-2 | Sub X, 2 | Subtraction |
| X=X\*2 | Multiplication (Product=Multiplicand \* Multiplier) uses the *MUL* instruction which has a single operand. The format of the MUL instruction is:  *MUL reg/mem*  where the single operand *reg/mem (*meaning *register* or *memory*) is the multiplier. The  Multiplicand must be in an Accumulator register (AL, AX, or EAX). The size of accumulator used depends on the size of the MUL operand chosen by the programmer. If the programmer uses 8bit operand for the MUL instruction the multiplicand must be placed in AL register. After the MUL instruction is executed the CPU will place the product in AX register (at most, a 16 bit value). For 16 bit operands the multiplicand must be placed in AX and the product will be in DX:AX registers (at most a 32 bit value where the upper 16 bits are in DX register and the lower 16 bits are in the AX register). Similarly, for 32 bit operands the multiplicand must be placed in EAX before the MUL instruction is executed. Afterwards, the CPU will place the product in EDX:EAX registers (at most, 64 bit value where the upper 32 bits are in EDX register and the lower 32 bits are in the EAX register). Here is an example for code for X\*2.  *.data*  *X DWORD ?*  *.code*  *Mov eax, 2*  *Mul X*  NOTE: The CPU will place the result of this MUL instruction in e*dx:eax* but if the result is small enough to fit just e*ax* register, you can ignore whatever is in the *edx* and only access *eax* after the MUL instruction is executed. | Multiplication. The information provided and the code example should help you with this lab assignment but if you need more information you should see section 7.3.1 of the Irvine 7th Ed text book.  Summary for how the CPU uses the MUL instruction.  For 8 bit operands (multipliers)  *[AX] 🡨 [AL] \* [8bit reg/mem operand]*  For 16 bit operand (multiplier)  *[DX:AX] 🡨 [AX] \* [16bit reg/mem operand]*  For 32 bit operand (multiplier)  *[EDX:EAX] 🡨 [EAX] \* [16bit reg/mem operand]* |
|  |  |  |

Details of the complete format of MOV, ADD, SUB, and MUL instructions can be found in Appendix B.2 of Irvine 7Ed text.

**Example Program**

Here is an example program you can use as a guide

;THIS PROGRAM, IS 32bit application for MASM on MSVC++

;It is used to introduce ARITHmetic operations and VARIABLE ;declarations. Specific Arithmetic Operations implemented are: ;ADD, SUB, MUL, INC

;The program computes Result = X + Y + X\*Y

;The COMMENT instruction is used for group comments. The group to ;be commented is delimited with & character.

COMMENT &

Objectives: Students will learn to use Registers to do Step by Step ASM Instructions

to handle Intel Architecture Restrictions such as: 1) No memory-memory ADD; 2) Instructions have no more than 2 operands

3) Thus We Cannot Add more than 2 items at a time in same instruction

The following High Level C-like code describes what we want to do

void main()

{

int X;

int Y=9;

int Result;

X=2;

X=X+1;

Y=Y+2;

Result=X+Y+X\*Y;

}

&

;

;Here is the corresponding x86 Assembly code

.386

.model flat, stdcall

.stack 4096

ExitProcess PROTO ,dwExitCode:DWORD

;The following include statements may not be necessary in a lab ;environment. In that case they can be deleted or modified.

INCLUDE c:\irvine\Irvine32.inc

include c:\irvine\macros.inc

includelib c:\irvine\irvine32.lib

includelib c:\irvine\kernel32.lib

includelib c:\irvine\user32.lib

.data

X DWORD 0

Y DWORD 9

Result DWORD ?

.code

main proc

mov X,2 ;X=2

inc X ;X++ i.e. X=X+1

add Y,2 ;Y=Y+2

mov eax,X ;eax=eax+X Note:eax will be used to accumulate result

add eax,Y ; eax = eax+Y

mov ebx,eax ;ebx=X+Y. save eax, will need it to do X\*Y (multiplication)

mov eax,X

mul Y ; eax=X\*Y

add eax,ebx ;eax=X\*Y + X+Y

mov Result,eax ;Result= X\*Y + X+Y

invoke exitprocess,0 ;EXIT TO OS

main endp

end main

**Procedure:**

a) Write an assembly program to implement the following instructions. HINT: YOU CAN USE REGISTERS FOR STORING INTERMEDIATE INFORMATION IF NEEDED. *EAX, EBX, ECX, EDX, ESI, EDI* REGISTERS ARE AVAILABLE.

void main()

{

int X, Y, loc1, loc2, loc3;

int A=90;

X = 40

Y = 24

Loc1 = Y \* 160 + X \* 2

Loc2 = A\* 950

Loc3 = loc2 – loc1

}

b) Recall that to allow your code to run properly in our lab environment the you should use the following format. Complete the it.

*; Student name:*

*;Course:*

*;Semester*

*;Instructor: Dr Ghansah*

*;Lab Day, Date and Time*

*;Lab section #*

*;Program #*

*;*

*; This program … WHAT IT DOES HERE …..*

*.386*

*.model flat, stdcall*

*.stack 4096*

*ExitProcess PROTO ,dwExitCode:DWORD*

*.data*

*X WORD ?*

*…*

*DECLARE REST OF VARIABLES HERE*

*…*

*.code*

*Main proc*

*Mov X,40*

*…*

*REST OF YOUR CODE SHOULD BE HERE*

*invoke exitprocess,0 ;EXIT TO Operating System*

*Main endp*

*End main*

c) Assemble your program using MASM assembler. Recall that filename must use *.asm* extension.

d) Link the object code (This is automatically done if you used “Build Solution” from the Build Menu in part c above).

e) If there are no assembler errors Use Debug Menu to run your program.

f) **Step** through the program (F10 key) using Debug and see how register values change. Set watches for variables and appropriate registers as you step through for verification.

g) **Demonstration**: Please provide a screen shot showing all the variables at the last step (ie. Just before exiting to OS). In addition, upload your source code to Canvas. This will be run your program to verify that it works according to specification.

h) **Submission**: Submit electronic copy of your **documented** program to CANVAS including the program (source code whose filename has .asm extension) and output (screen shots in a Word Document). **Filenames must be according to the format specified in the syllabus.**