**P1: Addressing Data in Memory and Segments (16 bits)**

**Steps to configure MASM Project in Visual Studio 2017**

1. Create C++ empty project
2. Configure Linker for ‘General’, ‘Input’ to add libraries.

**General :** Additional directory libraries - c:\Irvine

**Input:** Additional dependencies: Irvine32.lib;

**System:** Subsystem - Console

1. Add Microsoft Macro Assembler into the project
   1. Right-click project file name - Build dependencies - build customization
   2. Check masm
2. Add item, C++ file, name it as main.asm/ main.cpp to embed assemble language in C/C++ files.
3. To Include Path c:\Irvine for ASM file

* Project Properties - Microsoft Macro Assembler -> General -> Include Paths - c:\Irvine

1. Add breakpoints in your code file (registers value can only be shown during debugging).
2. To enable the Memory windows, Enable address-level debugging must be selected in Tools > Options (or Debug > Options) > Debugging > General.
3. While debugging:

To show **register values:** Debug > Windows > Registers

To show **memory values**:Debug > Windows > Memory

Assembly language template:

.386

.model flat, stdcall

.stack 4096

ExitProcess PROTO, dwExitCode: DWORD

.data

;define variables here

.code

main PROC

;write assembly code here

INVOKE ExitProcess,0

main ENDP

END main

**Debug**

Objective : To understand the machine operation

Tools : Microsoft Visual Studio

# Part I

1. Write the program below and examine the register contents of **eax, ebx, ecx** and **edx** for each step (Press F10: Step Over).

a) mov cl, 42

mov dl, 29

add cl, dl

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command** | **EAX** | **EBX** | **ECX** | **EDX** |
| mov cl, 42 | CF58C5D0 | 00287000 | 0018102A | 00181005 |
| mov dl, 29 | CF58C5D0 | 00287000 | 0018102A | 0018101D |
| add cl, dl | CF58C5D0 | 00287000 | 00181047 | 0018101D |

b) mov ax,0123

add ax,0025

mov bx,ax

add bx,ax

mov cx,bx

sub cx,ax

sub ax,ax

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Command** | **EAX** | **EBX** | **ECX** | **EDX** |
| mov ax, 0123 | 52F3007B | 00592000 | 008E1005 | 008E1005 |
| add ax, 0025 | 52F30094 | 00592000 | 008E1005 | 008E1005 |
| mov bx, ax | 52F30094 | 00590094 | 008E1005 | 008E1005 |
| add bx, ax | 52F30094 | 00590128 | 008E1005 | 008E1005 |
| Mov cx, bx | 52F30094 | 00590128 | 008E0128 | 008E1005 |
| Sub cx, ax | 52F30094 | 00590128 | 008E0094 | 008E1005 |
| Sub ax, ax | 52F30000 | 00590128 | 008E0094 | 008E1005 |

1. To obtain the memory address that stores the executing command, examine the register contents of **EIP.**

Write the following program and list the memory addresses that stores each line of codes. (HINT: disassembly)

mov ax, 56h (00071010)

EIP = 00071014

mov bx, 02h (00071014)

mul bx (00071018)

1. Disassembly the following machine code to assembly code/symbolic code:

a) B8 54 01 05 25 00

**01301038 mov eax,25050154h**

**0130103D add ah,cl**

b) B8 05 1B 00 2C EB F8

**00C31038 mov eax,2C001B05h**

**00C3103D jmp \_ExitProcess@4+1Fh (0C31037h)**

4. Consider the machine language instructions

B0 1C D0 E0 B3 0C F6 E3 6A

Which instruction performs the following operations?

* 1. Move hex value 1C to the AL register.

00A71038 mov al,1Ch

* 1. Shift the contents of AL one bit to the left.

00A7103A shl al,1

* 1. Move the hex value 12 to BL.

00A7103C mov bl,0Ch

* 1. Multiply AL by BL.

00A7103E mul al,bl

Trace the program and find out the final product in AX? Confirm the result by manual calculation.

5. What is the output in AX?

MOV AL, 5 ; AL = multiplicand

MOV BL, 10 ; BL = multiplier (operand) MUL BL

AX = 0370

6. What is the output in AX and DX?

MOV AX, 0083 ; dividend

MOV BL, 2 ; divisor (8 bits) DIV BL

AX = 0129, DX = 1005

# Part II

1. Enter the following instructions:

MOV AX, 0010

MOV BX, 0020

MOV CX, 0030

ADD AX, BX

INC BX

SUB CX, AX

DEC CX

What is the content of register AX, BX, CX and IP for each instruction?

Value of registers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **AX** | **BX** | **CX** |
| MOV | AX,010 | 000A | E000 | 0000 |
| MOV | BX,020 | 000A | 0014 | 0000 |
| MOV | CX,030 | 000A | 0014 | 001E |
| ADD | AX,BX | 001E | 0014 | 001E |
| INC | BX | 001E | 0015 | 001E |
| SUB | CX,AX | 001E | 0015 | 0000 |
| DEC | CX | 001E | 0015 | FFFF |
|  |  |  |  |  |

What is the value in decimal for CX register?

CX = 65535

1. What is the final value of AX and BX?

MOV CX,3 ;Initialize for 3 loops

L1: MOV AX,00

MOV BX,00

ADD BX,AX

LOOP L1 ;Decrement CX ;Repeat if nonzero

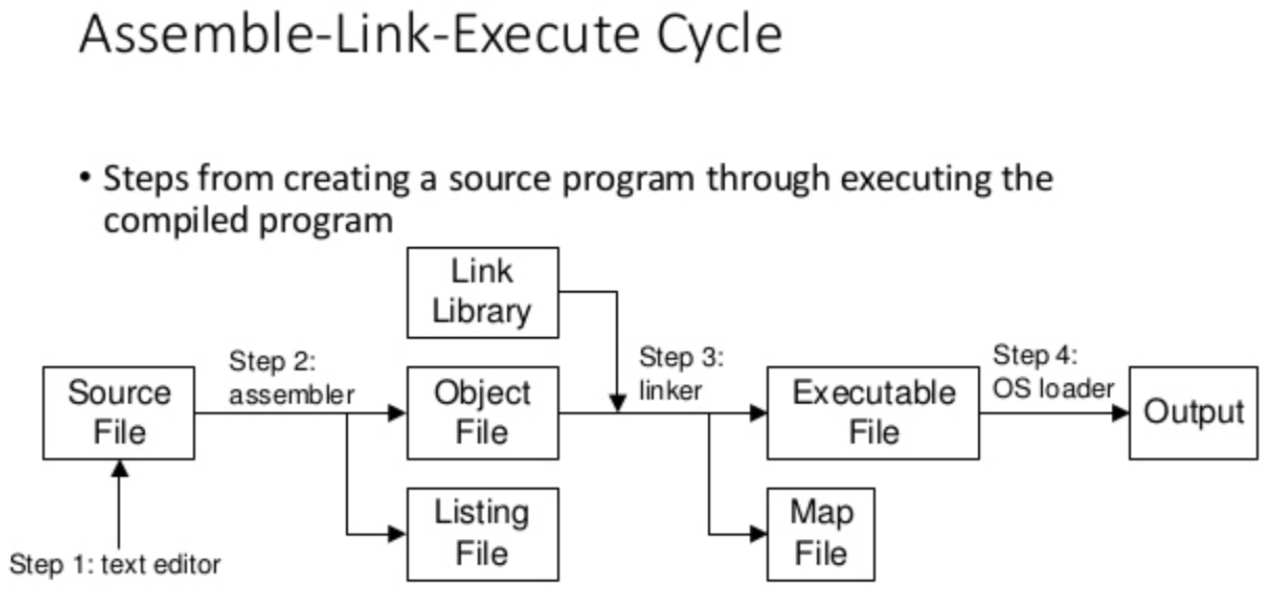
|  |  |  |
| --- | --- | --- |
| **AX** | **BX** | **CX** |
| 3358 | E000 | 0003 |
| 0000 | E000 | 0003 |
| 0000 | 0000 | 0003 |
| 0000 | 0002 | 1005 |
| 0000 | 0000 | 0002 |
| 0000 | 0000 | 0002 |
| 0000 | 0000 | 0001 |
| 0000 | 0000 | 0001 |
| 0000 | 0000 | 0001 |
|  |  |  |

**16 Bits Assembly Programming (Real Address Mode)**

Objective : Experience on 16 Bits Assembly Programming (Real Address Mode) by using DOSBox before covering 32 Bits (Flat Memory Model).

DOSBox is used for real address mode because lots of configurations are required for 64 bits windows to write 16 Bits program.

Tools : Notepad++ , MASM.exe, LINK.exe



1) Using Notepad++ 2) Assemble XXX.asm 3) Linker converts obj to exe 4) Run exe file

**XXX.asm MASM XXX.asm LINK XXX.obj XXX.exe**

**General x86 Instruction Set**

Example of complete instruction set:

<http://www.gabrielececchetti.it/Teaching/CalcolatoriElettronici/Docs/i8086_instruction_set.pdf>

**General x86 Interrupt Command**

Example of complete interrupt list: <http://www.gabrielececchetti.it/Teaching/CalcolatoriElettronici/Docs/i8086_and_DOS_interrupts.pdf>

For INT 21H

|  |  |  |
| --- | --- | --- |
| AX | DX | Function |
| 01--H | ----H | Read single character, result stored in AL |
| 02--H | --61H | Write single character, e.g., ‘a’= 61H |
| 0A--H | BUF | Read an array of characters, requires to define a pointer to array, e.g., BUF DB 10 DUP(‘ ’) |
| 09--H | MSG | Write an array of characters, requires to define a pointer to array, e.g., MSG DB “TESTING” |

- Refer to the American Standard Code for Information Interchange (ASCII) codes for character display.

Reference: <https://commons.wikimedia.org/wiki/File:ASCII-Table.svg>

- $ dollar symbol to indicate the end of characters array

- Each new line (action of “ENTER” keyboard button) = carriage return (CR) + line feed (LF)

**Assembly Language Programming I**

# 1. Arithmetic Expression

Write a program that implements the following arithmetic expression:

*result = v al2 + 5– val1 + val3*

Using the following data definitions:

|  |  |  |
| --- | --- | --- |
| val1 | DB | 6 |
| val2 | DB | 3 |
| val3 | DB | 4 |
| result | DB | ? |

In comments next to each instruction, write the hexadecimal value of AL. Print the final result on screen.

# 2. Uppercase to lowercase conversion

Defines a symbolic constant for uppercase letter ‘A’ and create a variable that uses the symbol as initialize.

Write a program that converts this uppercase letter to lowercase. Print the output as the format below:

|  |
| --- |
| A , a |

Modify the constant value and check the result.

# 3. Lowercase to uppercase conversion

Modify the program in question 2 to allow a conversion from a lowercase letter to an uppercase letter.

# 4. Exchanging two character

Write a program that defines two initialized character and exchanges their contents. Print the output as the format below:

|  |
| --- |
| (a , k) ▶ (k , a) |

# 5. Multiplication (product in single digit)

# Write a program that prompts the user for a decimal digit and display the digit and its self multiplication result with an appropriate message. Sample output:

|  |
| --- |
| Please enter a digit: 2  2 times 2 returns: 4 |

# 

# 6. Multiplication (product in double digit)

# Modify the program from Question 5, prompts the user for a decimal digit (4-9) and display the digit and its self multiplication result (2 digits) with an appropriate message.. Sample output:

|  |
| --- |
| Please enter a digit: 6  6 times 6 returns: 36 |

# 7. Division (product in single digit)

# Write a program that calculates and displays the quotient and remainder of a division operation. For this exercise, use single digit dividend and divisor. Sample output:

|  |
| --- |
| Divided : 8  Divisor : 5  Quotient : 1  Remainder : 3 |