Homework
CSE 331 Lab
MPU 8086
Section 7

Summer 2020
North South University

Name	:	Md. Fahad Mojumder
Student ID	:	1712145642
Email Address	:	fahad.mojumder@northsouth.edu
Submission Date	:	17-07-2020

CSE 331 Lab

Lab - 01

Introduction

In this session, you will be introduced to assembly language programming and to the emu8086 emulator software. emu8086 will be used as both an editor and as an assembler for all your assembly language programming.

Steps required to run an assembly program:

1. Write the necessary assembly source code.
2. Save the assembly source code.
3. Compile / Assemble source code to create machine code.
4. Emulate / Run the machine code.

Microcontrollers vs. Microprocessors.

- A microprocessor is a cpu on a single chip.
- If a microprocessor, its associated support circuitry, memory, and peripheral I/O components are implemented on a single chip, it is a microcontroller.

Features of 8086

- 8086 is a 16 bit processor. Its ALL internal registers work with 16 bit binary word.

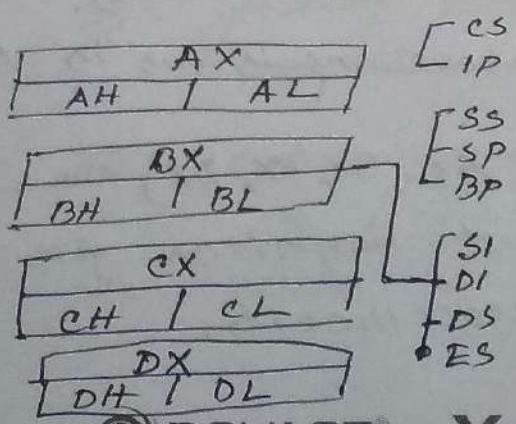
Date: ...../...../.....

- 8086 has a 16 bit data bus. It can read or write data to a memory/port either 16 bits or 8 bits at a time.
- 8086 has a 20 bit address bus, which means it can address up to $2^{20} = 1$ MB memory location.

Register - Register - Register

- Both ALU & FPU have a very small amount of super-fast private memory placed right next to them for their exclusive use. These are called registers.
  - The ALU & FPU store intermediate and final results from their calculations in these registers.
  - Processed data goes back to the data cache and then to the main memory from these registers.

Inside the CPU.



Registers are basically the CPU's own internal memory. They are used, among other purpose, to store temporary data while performing calculations.

GPR

The 8086 CPU has 8 general-purpose registers; each register has its own name:

- AX - The accumulator register.
- BX - The Base Address register
- CX - The count "
- DX - The Data "
- SI - Source Index "
- DI - Destination Index "
- BP - Base Pointer
- SP - Stack ".

Despite the name of a register, it is the programmer who determines the usage for each general-purpose register. The main purpose of a register is to keep a number (variable). The size of the above registers is 16 bits.

4 general-purpose registers (AX, BX, CX, DX) are made of two separate 8-bit registers. for example if AX = 0011000000111001b,

Date: ...../...../.....

then AH = 00110000b and AL = 00111001b. Therefore, when you modify any of the 8-bit registers 16-bit registers are also updated, and vice-versa. The same is for other 3 registers, "H" is for high and "L" is for low part.

Since registers are located inside the CPU, they are much faster than a memory. Accessing a memory location requires the use of a system bus, so it takes much longer. Accessing data in a register usually takes no time. Therefore, you should try to keep variables in the registers. Register sets are very small and most registers have special purposes which limit their use as variables, but they are still an excellent place to store temporary data of calculations.

Segment registers

CS - points at the segment containing the current program.

DS - generally points at the segment where variables are defined.

ES - extra segment register, it's up to a coder to define its usage.

SS - points at the segment containing the stack.

Special Purpose registers

IP - The Instruction Pointer. Points to the next location of instruction in the memory.

Flag Register - Determine the current state of the microprocessor. Modified automatically by the cpu after some mathematical operations, determines certain types of results and determines how to transfer control a program.

Writing your First Assembly Code

In order to write programs in assembly language, you will need to familiarize yourself with most, if not all, of the instructions in the 8086 - instructions set.

The following table shows the instruction name, the syntax of its use, and its description. The operands heading refers to the type of operands that can be used with the instruction along with their proper order.

- REG: Any valid register.
- Memory: Referring to a memory location in RAM.
- Immediate: Using direct value.

Date: ...../...../.....

Instruction	Operands	Description
MOV	REG, memory memory, REG REG, REG memory, immediate REG, immediate	copy Operand2 to Operand1. • The MOV instruction can't set the value of the CS and IP registers. • Copy value of one segment register to another segment register (should copy to GIS first) • Copy an immediate value to segment register (should copy to general register first). Algorithm: operand1 = operand2.
ADD	REG, memory memory, REG REG, REG memory, immediate REG, immidiate	Adds two numbers. Algorithm: operand1 = operand1 + operand2

X

Lab-2

Variables, I/O, Array

Creating Variables:

Syntax for a variable declaration:

name DB value
name DW value

DB - stands for define Byte.
DW - stand for define word.

- name - can be any letter or digit combination, though it should start with a letter. It is possible to declare unnamed variables by not specifying the name.

- Value - can be any numeric value in any supported numbering system (hexa, binary, or decimal), or "?" symbol for variables that are not initialized.

Creating Constants.

Constant are just like variables, but they exists only until your program is compiled. After definition of a constant its value can't be changed.

Date: ...../...../.....

To define a constant EQU directive is used :-

name EQU < any expression >

For example :

E EQU 5
mov AX, K

Creating Arrays

Arrays can be seen as chain of variables. A text string is an example of a byte array, each character is presented as an ASCII code value (0-255).

Here are some array definition examples :

a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h

b DB 'Hello', 0

- You can access the value of any element in array using square brackets, for example :

mov AL, a[3]

- You can also use any of the memory index registers BX, SI, DI, BP, for example :

mov SI, 3
mov AL, a[SI]

- If you need to declare a large array you can use DUP operator.

The Syntax for DUP:

number DUP (value(s))

number — number of duplicates to make (any constant value).

value — expression that DUP will duplicate.

for example:

c DB 5 DUP(9)

is an alternative way of declaring:

c DB 9,9,9,9,9.

Memory Access

To access memory, we can use these four registers: BX, SI, DI, BP. Combining these registers inside [] symbols, we can get different memory locations.

Date: ...../...../.....

[BX+SI]	[SI]	[BX+SI+d8]
[BX+DI]	[DI]	[BX+DI+d8]
[BP+SI]	d16 (offset only)	[BP+SI+d8]
[BP+DI]	[BX]	[BP+DI+d8]
[SI+d8]	[BX+SI+d16]	[SI+d16]
[DI+d8]	[BX+DI+d16]	[DI+d16]
[BP+d8]	[BP+SI+d16]	[BP+d16]
[BX+d8]	[BP+DI+d16]	[BX+d16]

- Displacement can be an immediate value or offset of a variable, or even both. if there are several values, assembler evaluates all values and calculates a single immediate value.
- Displacement can be inside or outside of the [] symbol, assembler generates the same machine code for both ways.
- Displacement is a signed value, so it can be both positive or negative.

Instructions

Instruction	Operands	Description
INC	REG MEM	Increment. Algo: operand = operand + 1 Ex: MOV AL, 4 INC AL; AL=5 RET
DEC	REG MEM	Decrement. Algo: ope = ope - 1 Ex: MOV AL, 86 DEC AL; AL=85 RET

LEA	REG
	MEM
		Load effective address.
		Algo:
		REG = address of memory (offset)
		Ex:
		MOV BX, 35h
		MOV DI, 12h
		LEA SI, [BX+DI]

Declaring Array:

Array Name db size DUP(?)

Value initialize:

arr1 db 50 dup (5, 10, 12)

Index values:

```
mov bx, offset array 0
mov [bx], 6 ; inc bx
mov [bx+1], 10
mov [bx+9], 9
```

Date: ...../...../.....

OFFSET:

"offset is an assembler directive in x86 assembly language. It actually means "address" and is a way of handling the overloading of the "mov" instruction. Allow me to illustrate the usage.

1. mov si, offset variable.
2. mov si, variable.

The first line loads SI with the address of variable. The second line loads SI with the value stored at the address of variable.

1. mov si, offset variable
2. mov si, [variable]

The square brackets aren't necessary, but they made it much clearer while loading the contents rather than the address.

LEA is an instruction that load "offset variable" while adjusting the address between 16 and 32 bits are necessary. "LEA load the lower 16 bits of the address into the register, and "LEA loads the 32 bits register with the address extended to 32 bits.

Lab-3

Print and I/O

In the Assembly Language Programming, A single program is divided into four segments which are –

1. Data segment
2. Code "
3. Stack "
4. Extra "

Print : Hello World in Assembly Language.

```
DATA SEGMENT
        MESSAGE DB "HELLO WORLD !$"
ENDS
CODE SEGMENT
        ASSUME DS:DATA CS:CODE
START:
    MOV AX, DATA
    MOV DS. AX
     LEA DX, MESSAGE
    MOV AH, 9
    INT 21H
    MOV AH, 4CH
    INT 21H
ENDS
 END START
```

Date: ...../...../.....

Now, from these one is compulsory i.e Code segment if all of you don't need variable(s) for your program. If you need variable(s) for your program you will need two segment i.e. Code segment and Data segment.

First Line - Data segment.

Data segment is the starting point of the Data segment in a program and Data is the name given to this segment and SEGMENT is the Keyword for defining segments, where we can declare our variables.

Next Line - MESSAGE DB "HELLO WORLD!$"

MESSAGE is the variable name given to a Data Type(Size) that is DB. DB stands for Define Byte and is of one byte (8 bits). In Assembly Language programs, variables are defined by Data size not it's type. Characters need one byte to to store Character or string we need DB only that don't mean DB can't hold numbers or numerical value. The string is given in

double quotes. $ is used as NULL character in C programming. so that compiler can understand where to stop.

Next Line - CODE SEGMENT

Code segment is the starting point of the code segment in a program and CODE is the name given to this segment and SEGMENT is the keyword for defining segments, where we can write the coding of the program.

Next Line - ASSUME DS:DATA CS:CODE

There are different registers present for different purpose. So we have to assume Data in the name given to Data segment register and CODE is the name given to Code segment register (SS, ES are used in the same way as CS, DS)

Next Line - START

START is the label used to show the starting point of the code which is written in the Code segment. : is used to define a label as in C programming.

Date: ...../...../.....

Next Line - MOV AX, DATA

MOV DX, AX.

After assuming data and code segment, still it is compusory to initialize DATA segment to DS register. MOV is a keyword to move to move the second element into the first element. But we can't move Data directly to DS due to MOV commands restriction, hence we move data to AX and then from AX to DS. Ax is the first and most important register in the ALU unit. the part is also called INITIALIZATION OF DATA SEGMENT and it is important so that the Data elements or variables in the Data segment are made accessible. Other segment are not needed to be initialized, only assuming is inhale.

Next line - LEA DX, MESSAGE

MOV AH, 9

INT 21H

The above three line code is used to print the string inside the MESSAGE variable. LEA stands for Load Effective Address which is used to assign Address of variable to DX register.

17

To do input and output in Assembly Language we use interrupts. Standard input and standard output related interrupts are found in INT 21 H which is also called as DOS interrupt. It works with the value of AH register. If the value is 9 or 9h or 9H, that means PRINT the string whose Address is Loaded in DX register.

Next Line - MOV AH, 4CH
INT 21 H

The above two line code is used to exit to dos or exit to operating system. Standard input and standard output related interrupts are found in INT 21 H which is also called as DOS interrupt.

Next Line - CODE ENDS

CODE ENDS is the End point of the code segment in a program. We can write just ENDS but to differentiate the end of which segment it is of which we have to write the same name given to the Code segment.

Date: ...../...../.....

Last Line - END START

END START is the end of the label used to show the ending point of the code which is written in the code segment.

Now try this -

```
DATA SEGMENT
    MESSAGE DB "HELLO WORLD $"

START:
    MOV AX, DATA
    MOV DS, AX
    LEA DX, MESSAGE
    MOV AH, 9
    INT 21H
    MOV AH, 4CH
    INT 21H

END START.
```

Ex. 1 - Print 2 strings

```
.MODEL SMALL
.STACK 100H
.DATA
    STRING_1  DB 'I hate CSE331 $'
    STRING_2  DB 'But I Love Kacchi !!! $'
```

19

~~- Data~~

~~STRING-1 DB~~

- CODE

MAIN PROC

MOV AX, @DATA ; Initialize DS

MOV DS, AX

LEA DX, STRING-1 ; Load and display the STRING-1

MOV AH, 9

INT 21H

MOV AH, 2 ; carriage return

MOV DL, 0DH

INT 21H

MOV DL, 0AH ; Line feed

INT 21H

LEA DX, STRING-2 ; load and display the STRING-2

MOV AH, 9

INT 21H

MOV AH, 4CH ; return control to DS.

INT 21H

MAIN ENDP

END MAIN.

Ex. 2 - Read a string and Print it

Date: ...../...../.....

```
.MODEL SMALL
.STACK 100H
.Data
  MSG-1 EQU "Enter the character : $"
  MSG-2 EQU 0DH,0AH, 'The given character is : $'

  PROMPT_1 DB .MSG-1
  PROMPT-2 DB . MSG-2
.CODE
  MAIN PROC
    Mov AX, @DATA ; initial DS
  MOV DS, AX
  LEA DX, PROMPT-1 ; Load and display PROMPT-1
  MOV AH, 9
  INT 21H
  MOV AH, 1 ; read a character
  INT 21H
  MOV BL, AL ; save the given character into BL
  LEA DX , PROMPT_2 ; LOAD and display PROMPT-2
  MOV AH, 9
  INT 21 H
  MOV AH, 2 ; disply the character
  MOV DL, BL
  INT 21 H
  MOV AH, 4CH ; return control to DOS
  INT 21H
MAIN ENDP
END MAIN.
```

Ex-3 - Read a string from user and display this strings in a new line.

```
.MODEL SMALL
.STACK 100H

.CODE
MAIN PROC
  MOV AH,1 ; read character
  INT 21H
  MOV BL,AL ; save input character into BL
  MOV AH,2 ; carriage return
  MOV DL,0DH
  INT 21H
  MOV DL,0AH ; line feed
  INT 21H
  MOV AH,2 ; display the character stored in BL
  MOV DL,BL
  INT 21H
  MOV AH,4CH ; return control to DOS
  INT 21H
MAIN ENDP
END MAIN.
```

Ex-5 - Printing string using MOV instruction.

```
.MODEL Small
;.STACK
.DATA
  MSG1 DB 'KI!!! Kemon lage:D$'
.CODE
```

Date: ...../...../.....

```
MOV AX, @DATA
MOV DS, AX
MOV dx, OFFSET MSG1 ; LEA dx, msg1
mov ah, 09h
int 21 H
mov ah, 4ch
int 21 h
END.
```

EX.6 - Print Digit from 0-9

```
.MODEL SMALL
.STACK 100H
.DATA
    PROMPT DB \'The counting from 0 to 9 is: $'
.CODE
MAIN PROC
    MOV AX, @DATA ; initialize DS
MOV DS, AX
LEA DX, PROMPT ; load the print PROMPT
MOV AH, 9
INT 21 H
MOV CX, 10 ; initialize CX.
MOV AH, 2 ; set output function
MOV DL, 48 ; set DL with 0
@LOOP: ; Loop label
    INT 21H ; print character.
```

```
    INC DL      ; increment DL to next ASCII character
    DEC CX      ; decrement CX
JNZ @LOOP       ; Jump to label @LOOP if CX is 0

MOV AH, 4CH     ; return control to DOS
INT 21H
MAIN ENDP
END MAIN
```