BRAC UNIVERSITY
Inspiring Excellence

Department of Computer Science and Engineering

Course Code: CSE341	Credits: 1.5
Course Name: Microprocessors	Semester: Summer'24

Lab 01

Introduction

I. Topic Overview:

The lab is designed to introduce to the basics of a microprocessor and intel8086 chipset, along with the emu8086, the simulator used to code in assembly language. Furthermore, representation of numbers and characters in assembly language will also be introduced to the students.

II. Lesson Fit:

The lab is partially related to the introductory theory class of Microprocessors.

III. Learning Outcome:

After this lecture, the students will be able to:

a. Understand the basics of an Intel 8086 microprocessor structure.
b. Interact with the EMU8086 Simulator for Intel 8086 Microprocessor.
c. Represent various number systems, string, characters and declare data types in assembly language.

IV. Anticipated Challenges and Possible Solutions

a. Difference between a Micro-processor and Micro-controller.

   Possible Solutions:

   i. Use a figure to distinguish between the two.

b. How the EMU8086 stores the numbers in hexadecimal formats

V. Acceptance and Evaluation

If a task is a continuing task and one couldn't finish within time limit, then he/she will continue from there in the next Lab, or be given as a home work and in the next Lab you have to submit the code and have to face a short viva. A deduction of 30% marks is applicable for late submission. The marks distribution is as follows:

Code: 0%

Viva: 100%

Registers

Registers of the 8086/80286 by Category

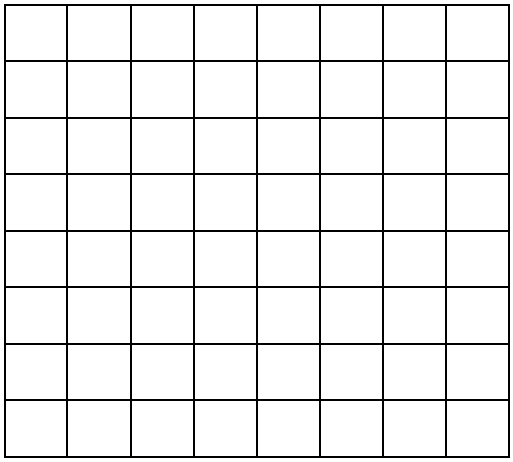
Category	Bits	Register Names
General	16	AX,BX,CX,DX
	8	AH,AL,BH,BL,CH,CL,DH,DL
Pointer	16	SP (Stack Pointer), Base Pointer (BP)
Index	16	SI (Source Index), DI (Destination Index)
Segment	16	CS(Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)
Instruction	16	IP (Instruction Pointer)
Flag	16	FR (Flag Register)

Memory:

- It stores the binary codes for the sequence of instructions and binary coded data. Example: ROM, RAM and Magnetic disks
- RAM can be read and written to anytime the CPU commands it, but ROM is pre-loaded with data and software that never changes, so the CPU can only read from it.
- ROM is typically used to store the computer's initial start-up instructions.

- In general, the contents of RAM are erased when the power to the computer is turned off, but ROM retains its data indefinitely.
- In a PC, the ROM contains a specialized program called the BIOS that orchestrates loading the computer's operating system from the hard disk drive into RAM whenever the

Let’s us look at the memory and addressing, how data are stored inside each slot.



The memory of 8086 looks something like the above diagram. This is just like a 2d array we have studied in our earlier courses. The tine square boxes are of 1 bit and in one row there are 8 tiny boxes, that it each row can hold 8 bits of data. Each row is said to be a location and address of each location is unique. In 8086 there are 2^20 locations, where each location is of 1 byte. Therefore the size of the memory is 1 Mb. On the other hand, the size of the address bus is 20bit.

Discussion: Introduction to Registers and Data Types in Assembly Language.

Registers

Assembly language:

Assembly language is used for most programming because it is difficult to program a microprocessor in its native language, that is hexadecimal machine language.

Assembler:

An assembler is a program that converts software written in symbolic machine language (the source program) into hexadecimal machine language (object program). The primary reason to use assembler is because development and modification are always difficult in machine language.

Assembly Language vs. Machine Language:

- Machine language or object code is the only code a computer can execute but it is nearly impossible for a human to work with.
  E4 27 88 C3 E4 27 00 D8 E6 30 F4 the object code for adding two numbers input from keyboard.

- When programming a microprocessor, programmers often use assembly language. This involves 3-5 letter abbreviations for the instruction codes (mnemonics) rather than the binary or hex object codes. A sample is shown below comparing the two languages.

Address	Hex Object Code				Mnemonics		Comment
					Op-Code	Operand	
0100	E4	27			IN	AL, 27H	Input first number from port 27H and store in AL
0102	88	C3			MOV	BL, AL	Save a copy of register AL in register BL
0104	E4	27			IN	AL, 27H	Input second number to AL
0106	00	D8			ADD	AL, BL	Add AL and BL and store the sum in AL
0107	E6	30			OUT	30H, AL	Output AL to port 30H
0109	F4				HLT		Halt the computer

Temporary memory with small capacity but are very fast. They are used to hold data temporarily for carrying out instructions. 8086 has registers which are of 16 bits long.

Data Types

1. **Numbers**: Must specify the base of the number at the end. If not specified the numbers will be considered as decimal.

For **binary**, add a ‘b’ at the end of the bit-string. e.g. 1110b

For **decimal**, just type the number without any suffix or prefix, e.g. 1101

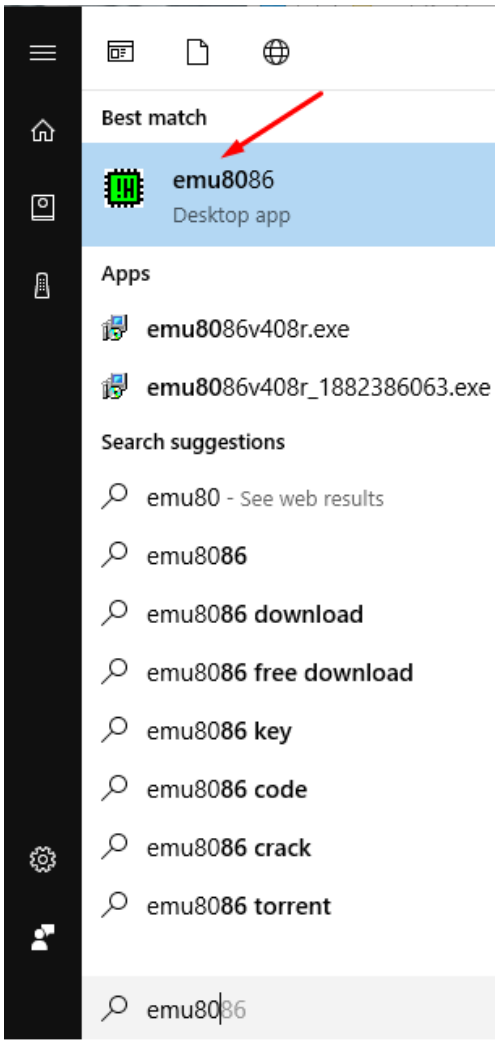
For **hexadecimal**, start with a 0 and end with ‘h’. A06 = invalid, 0A06h = valid

2. **Characters**: Enclosed by single quotes. For example, 'a'. The assembler converts every character into its ASCII value while storing
3. **Strings**: Array of character ends with a $ and enclosed by double quotes. e.g, **"hjshj$"**
4. **Data types: DB** = define byte, **DW** = define word
5. **Variables**: In java the syntax for defining a variable is:
   data_type var_name; /= value;
   *In assembly it is **var_name data_type value.** e.g.: m db 4*
   **Variables are saved in the data segment of the memory*

Introduction to EMU8086

1. Double click on the icon on desktop or search ‘emu8086’ on ‘Start menu’.

Desktop Icon

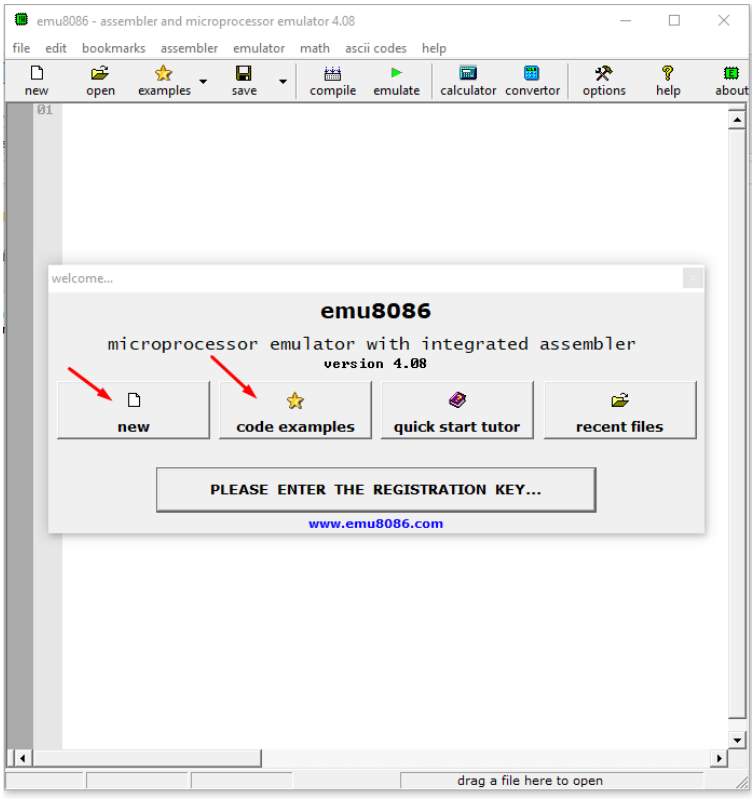


Search on 'Start Menu'

2. The following window will pop-up after you open the program.

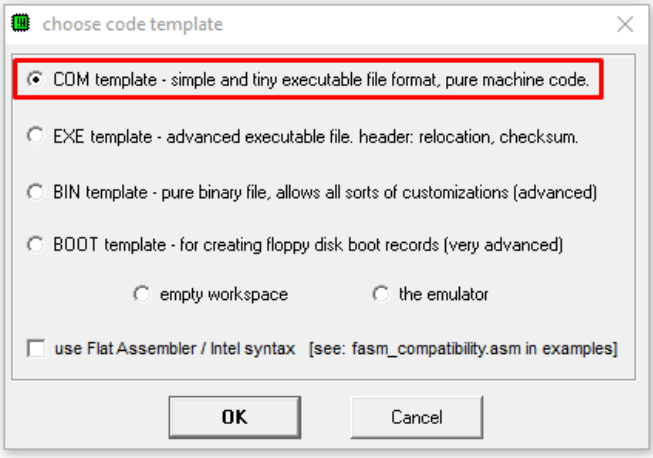
   ‘new’ will lead you to Step 3.

   ‘code examples’ will give you a list of example codes already done in the emu8086

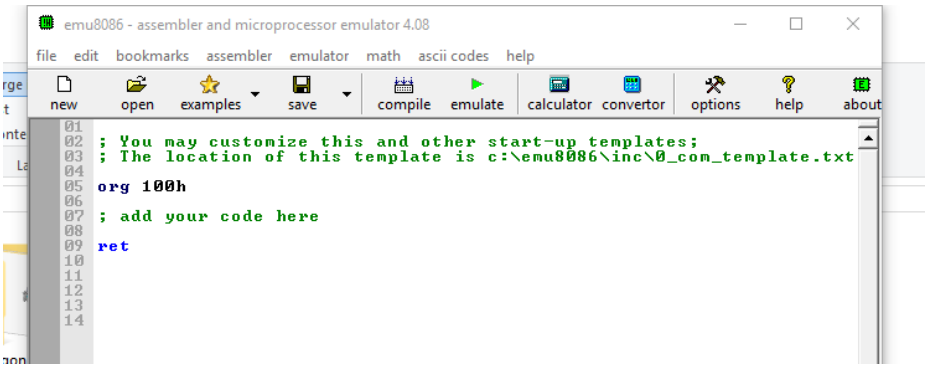


3. After clicking on **‘new’** the following window pop-up.

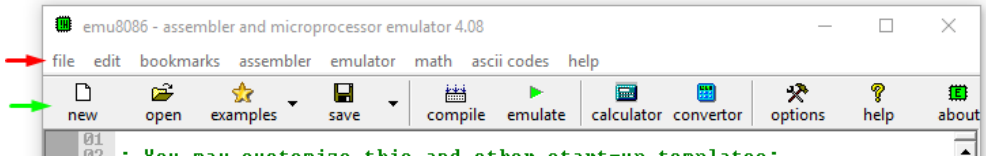
   Click on **‘COM template’**, this will open up window of Step 4.



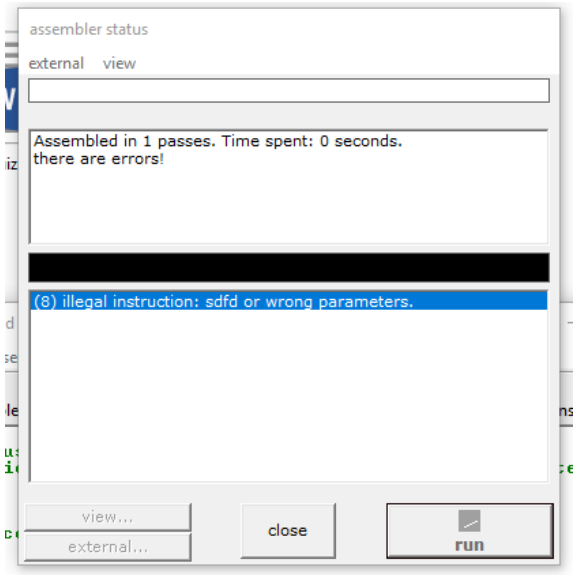
4. Step 3 will open the following window with already written codes.
   This is similar to the 'public class {public static void main…..' at the start of a java class file. **You may also choose to use a blank page to code*



5. On the top of the window, the first row has the basic menu options. On the second row some of the options are,
   '**examples**' opens up some pre-coded examples.
   '**compile**' compiles the code
   '**emulate**' compiles and then runs the code
   '**calculator**' opens up a basic calculator
   '**converter**' is for number system conversions.



6. When you click '**emulate**', the following window will pop-up for a second if there are no errors, otherwise it will stay and show errors.



7. If there were no errors on Step 6, the following window (next page) will appear.

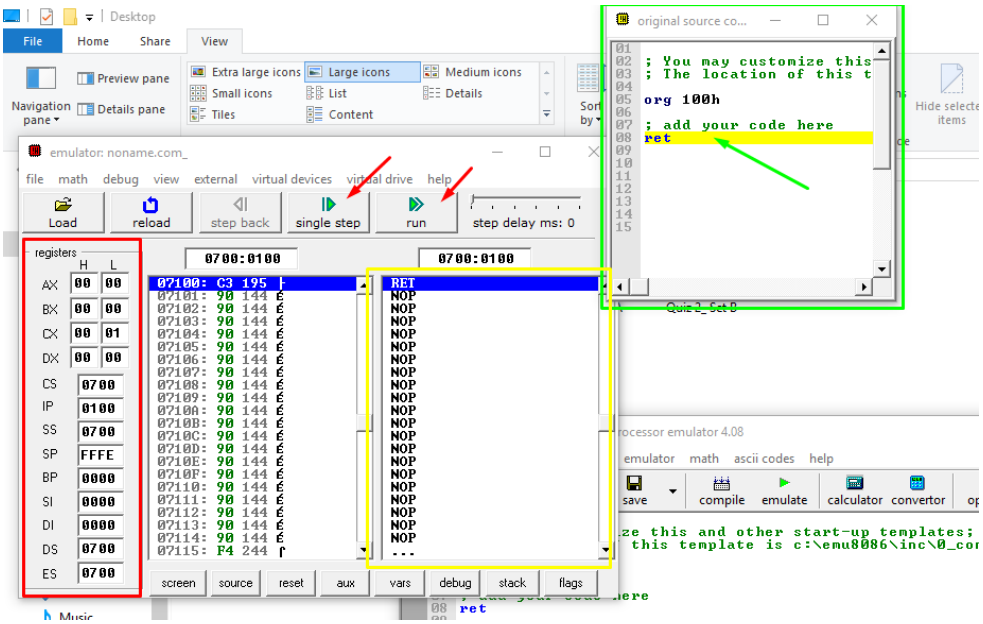
   The *red box* shows the values of all the registers. If you notice, some of the registers are divided into two columns, each of 8 bit (2 hex numbers each, i.e. 4 bit). Each value is in the hexadecimal format.

   The *yellow box* shows the converted code that was compiled.

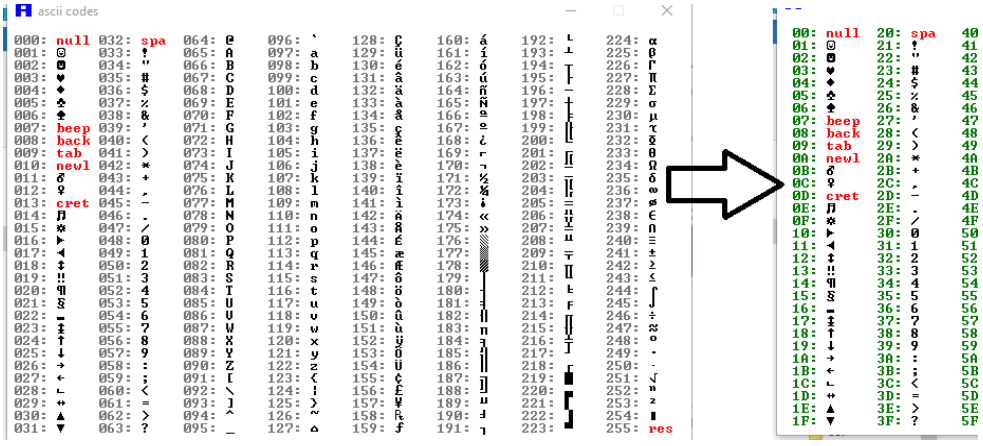
   The *green box* shows your code and highlights the line that is currently being executed.

   '**Run'** will run the code and give your output without any interruption.

   '**Single Step'** will interrupt on each line execution. Letting you to essentially debug the code and trace the register values side by side with seeing which line is being executed.



8. On the main window, clicking on **'ascii codes'** on the top bar will show up the following window. Double clicking on it will toggle the code between decimal and hexadecimal number system.



Using Registers in Assembly

For our course, we will stick to using registers in emu 8086.

Registers

16-bit storage for holding data temporarily during operations. Do not confuse with RAM. Registers can be divided into 4 types.

i. Data Registers: AX, BX, CX, DX

These 4 registers are used to hold data for carrying out mathematical and logical operations. These 4 registers are byte accessible. This means each 16-bit register can be used as two separate8-bit registers. AX = AH and AL, same goes for others. These four registers also perform other special functions. AX = During multiplication and division, one of the numbers must be inside AX or AL. BX = Used as address registers. CX = used in loop. CX increases automatically by 1DX = Used in multiplication/division and i/o operations.

ii. Segment Registers: CS, DS, SS, ES

To understand the these registers we must understand what memory segments are. The 8086 is a 16-bit Microprocessor and contains a memory of 20 bits. Now to store a value of10 bits the registers had to be of 20 bits each but they are not. To overcome this problem the whole memory is divided into a 3 segments data segment, stack segment and code segment. The segment registers will contain the address of the segments. Each segment register will be combined with index registers (16 bit) to form a 20-bit storage to store memory locations. [The combination of 2 16-bit registers to form a 20-bit location will be covered in theory class].

iii. Index and Pointer Registers: SP, BP, SI, DI

They contain the offset addresses mentioned in the previous section.
There are restrictions of using a segment register and index registers.

SP used with SS

BP used with DS, SS

SI used with DS

DI used with DS

Discussion: Instructions (mov, add, sub, inc, dec, neg)

i. **MOV**: used to transfer data between registers; to a register from a memory location, from a memory location to register. Moving constants to register or memory locations.

Syntax: mov destination, source

mov al,5 - the decimal value 5 will be stored in al

	Destination			
Source	**General registers**	**Segment Register**	**Memory Location**	**Constants**
General registers	**valid**	**valid**	**valid**	**invalid**
Segment Registers	**valid**	**invalid**	**valid**	**invalid**
Memory Location	**valid**	**valid**	**invalid**	**invalid**
Constant	**valid**	**invalid**	**valid**	**invalid**

ii. **ADD/SUB:** addition/subtraction between numbers. These operations occur between 2 registers, register/memory location and a number, register and memory location.

Syntax: add/sub destination, source

i.e. destination = destination +/ - source

	Destination	
Source	**General registers**	**Memory Location**
General registers	**valid**	**valid**
Memory Location	**valid**	**invalid**
Constant	**valid**	**valid**

iii. **INC**: This is used to increment a value of a register by 1.

inc ax; // this would increase the value of ax by 1.

iv. **DEC**: This is used to decrement a value of a register by 1.

dec ax; // this would decrease the value of ax by 1.

v. **NEG**: This is used to negate the contents of the destination. NEG does this by replacing the contents by its two's. complement.

The syntax is: NEG destination

Discussion: Instructions (MUL and DIV)

i. **MUL:** multiplication between 2 numbers. You must be very careful while carrying out multiplication. This is because 2 n bit numbers will result a 2n bit number. Therefore, multiplication is divided into2branches. byte multiplication where the operands are 8-bit numbers and word multiplication where the numbers are 16-bit numbers.

For byte multiplication one number is contained in the source and the other number has to be in AL. The result is saved in AX. The source can be a register or a memory location but not a constant.
mul BL; the 8-bit number is AL is multiplied by the 8-bit number in BL. The result is in AX.

For word multiplication one number is contained in the source and the other number has to be in AX. The result will be a 32-bit number. The higher 16 bits will be in DX and the lower 16-bit will be in AX. The source can be a register or a memory location but not a constant.

ii. **DIV:** When division is performed there are 2 results- quotient and remainder. Division is also divided into byte and word form.

In the byte form the divisor is an 8 bit register or memory location. The dividend is 16 bit and it must be in AX. After division the 8-bitquotient is in AL and the 8-bit remainder is in AH. The divisor can't be a constant. Div BL; the 16-bit number is AX is divided by the 8-bit number I n BL. The quotient is in AL and the 8-bit remainder is in AH.

In the word form, the divisor is a 16-bit register or memory location. The dividend is 32 bit and it must be in DX:AX. After division the 16-bit quotient is in AX and the 16-bit remainder is in DX. The divisor can't be a constant.

Problems

Task 01

Take input in the register AX, and then move it to BX using the MOV instruction.

Task 02

Swap two numbers, using a maximum of 3 registers.

Hint: Use the MOV instruction.

Task 03

Add two numbers using two registers.

Task 04

Subtract two numbers using two registers. Do you always get the correct answer? What happens when you subtract larger number from the smaller one?

Task 05

Swap two numbers using ADD/SUB instructions only.

Task 06

Perform the following arithmetic instructions. A, B, C are three variables to be declared beforehand

1. A = B - A
2. A = -(A + 1)
3. C = A + (B + 1); use inc
4. A = B – (A – 1); use dec

Task 07

Perform the following arithmetic operations

1. X * Y
2. X / Y
3. X * Y / Z

Task 08

Perform the following arithmetic operations and analyze the answers

1. 36DF * AF
2. F4D5 / C9A5
3. CA92 * BAF9
4. C2A2 * ABCD / BED

Task 09

Write two examples for each combination of registers possible for the 'mov' instruction.
Hint: See the table above to see all the possible combinations.

Task 10

Write two examples for each combination of registers possible for the 'add' and 'sub' instructions.
Hint: See the table above to see all the possible combinations.

Task 11

Perform the following arithmetic operation: (1 + 2) * (3 – 1) / 5 + 3 + 2 – (1 * 2)