BRAC UNIVERSITY
Inspiring Excellence

Department of Computer Science and Engineering

Course Code: CSE341	Credits: 1.5
Course Name: Microprocessors	Semester: Summer 21

Lab 01

Introduction

I. Topic Overview:

The lab is designed to introduce 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%

VI. Activity Detail

a. **Hour: 1.5**

Discussion: Microprocessor vs. Microcontroller

All of us have come across the two terms **micro controller** and **micro-processor**. They are **programmable** digital circuits meant to carry out operations as we desire. Programmable means we hard code the operations and a micro-processor or a controller works according to the code. Both can be attached to external devices.

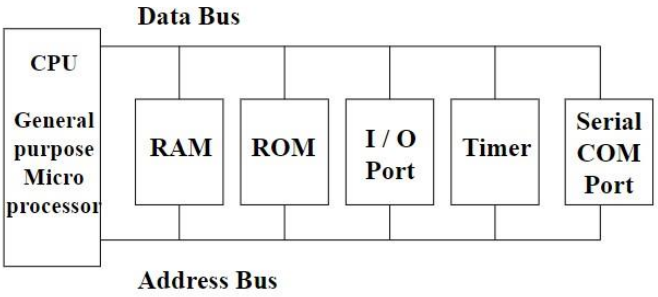
Both of the devices require memory to store the instructions and peripherals to input and output data. A controller has its memory and peripherals embedded inside it whereas a processor uses an external one. *Which one do you think is more capable of carrying out bigger and more complex instructions?* As the memory and the peripherals are built inside the controller it is also referred to as a minicomputer and it is a standalone device.

Micro-processors have larger **clock speeds** than microcontrollers as they are used to perform bigger and more complex tasks in small amount of time.

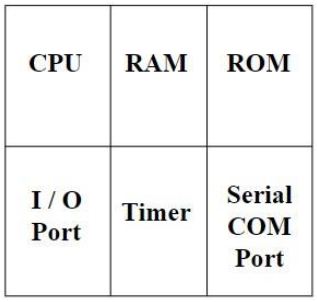
Clock speed: Also known as the clock rate is the measure how fast processor processes instructions. Measured in hertz (Hz).

Microprocessor vs. Microcontroller:

	µP	µC
Hardware Architecture	• Single-chip CPU • RAM-ROM ratio high • Interrupt, I/O, Timer – external. • Not much on Real-time	• Single-chip IC • ROM-RAM ration high • Interrupt, I/O, timer-Internal • Need to respond to Real time
Application	• Microcomputer system • Processing information	• Control oriented activities • Control of I/O devices
Instruction set	• Process intensive to handle Large volume of data • Operates on byte, words, pointers, and arrays. • Longer development time • ANDing, ORing, XORing in bit level is less easy	• Control intensive to handle I/O using single Bit • Operates mostly on Bit & byte • Shorter development time • ANDing, ORing, XORing in bit level is easy



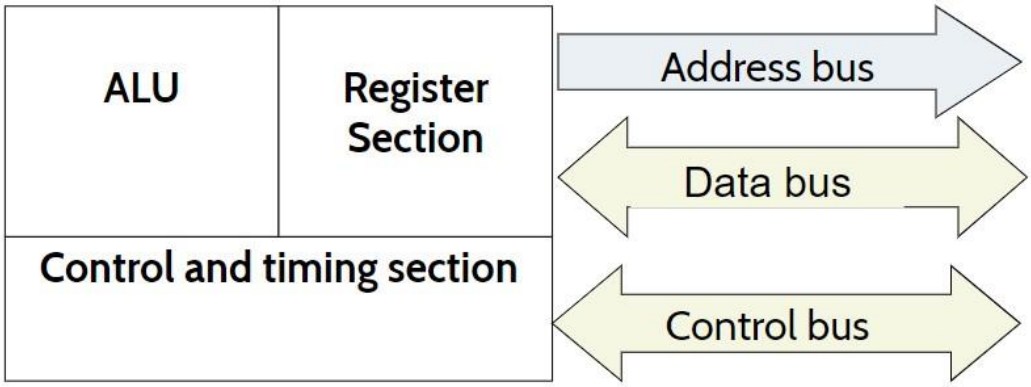
General-Purpose Microprocessor System



Microcontroller

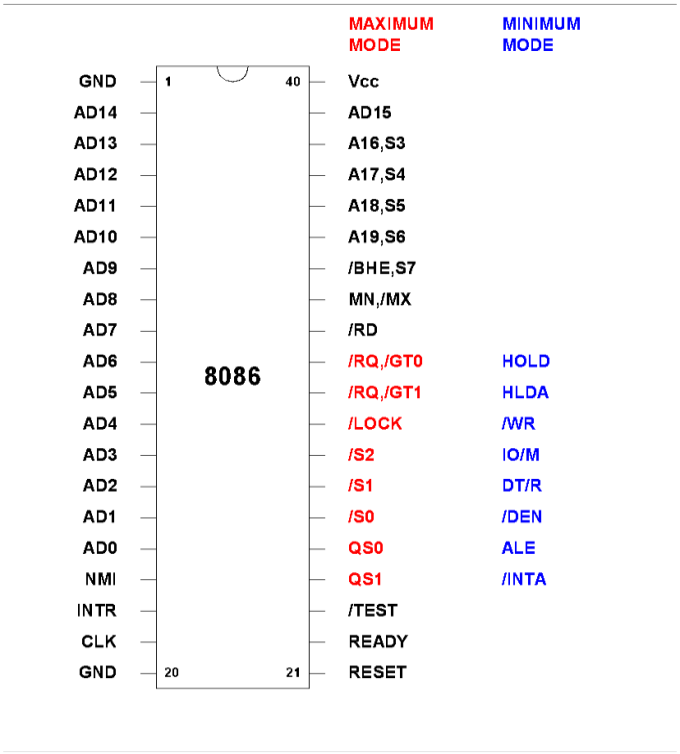
***This course is restricted to microprocessors.**

Let us break down a micro-processor. A microprocessor is a combination of CPU memory and bus interface unit. The heart of the microprocessor is called the Central Processing Unit (CPU). The CPU is made up of registers, control unit and the arithmetic and logical unit.



Intel 8086

A 16bit processor. It means it has 16 data lines (16 bits) inside the data bus. It means that the data, which can flow through the data bus can of 16 bits long. The address bus of 8086 is of 20 bits (20 address lines inside the address bus). Address but is used to access locations, as the address bus can carry 20 bits long address it means that the entire memory of the 8086 has 220 unique locations. A pin diagram of the processors is given below.



Bus Interface Unit: facilitates communication between the EU & the memory or I/O circuits.

Responsible for transmitting addresses, data, and control signals on the buses.

Registers (CS, DS, ES, SS, and IP) hold addresses of memory locations.

IP (instruction pointer) contain the address of the next instruction to be executed by the EU.

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

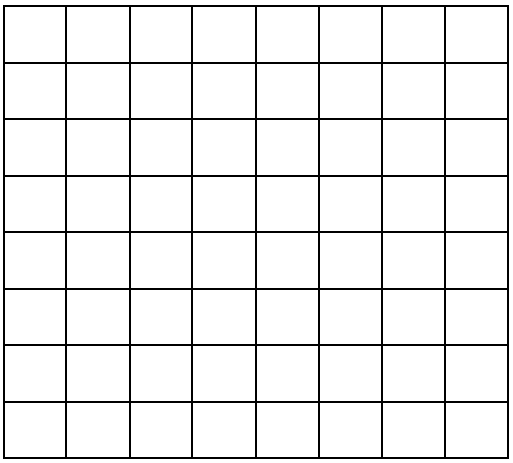
RAM can be read and written to anytime the CPU commands it, but ROM is pre-loaded with data and software that never changes, so the CPU can only read from it.

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 220 locations, that is, 1 Mb. The size of the address bus is 20bit.

b. **Hour: 2**

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, that is hexadecimal machine language.

Assembler: An assembler is a program that converts software written in symbolic machine language (the source programme) into hexadecimal machine language (object programme). 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_typevar_name; /= value;

In assembly it is **var_name data_type value.** e.g.: m db 4

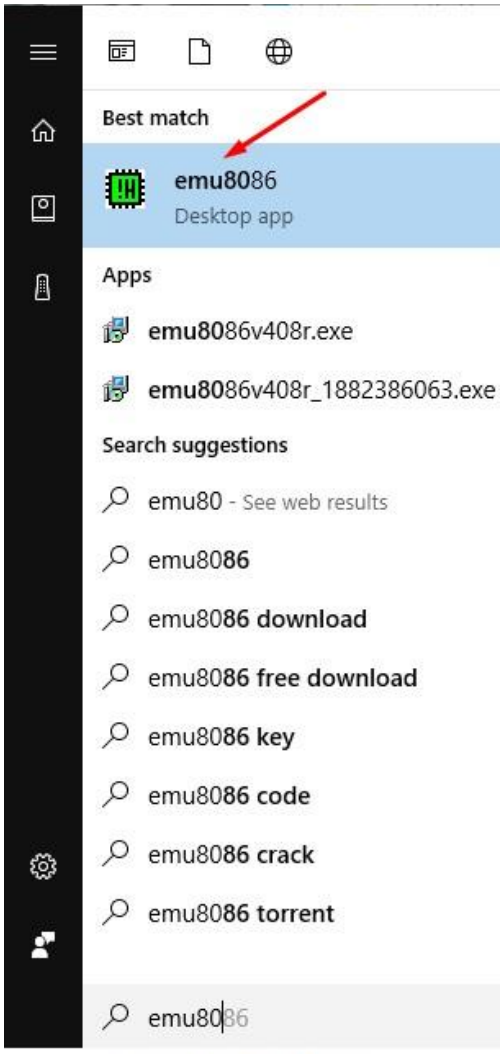
*Variables are saved in the data segment of the memory

c. **Hour: 2.5**

Discussion: Introduction to EMU8086

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

Desktop Icon

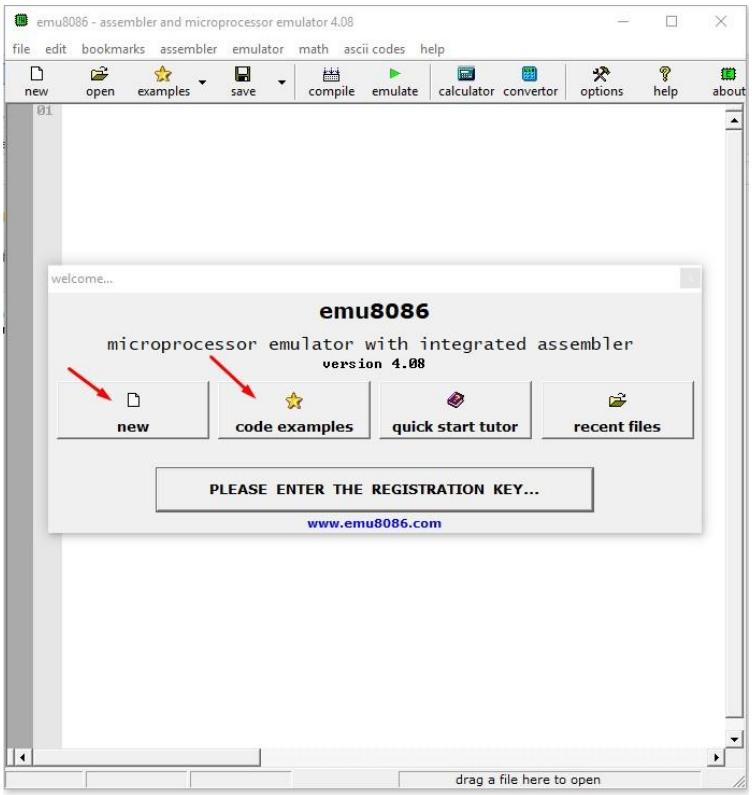


Search on 'Start Menu'

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

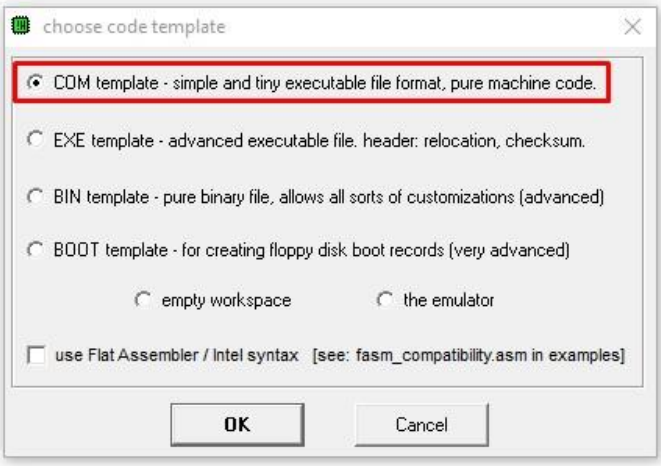
   'new' will lead you to Step 3.

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

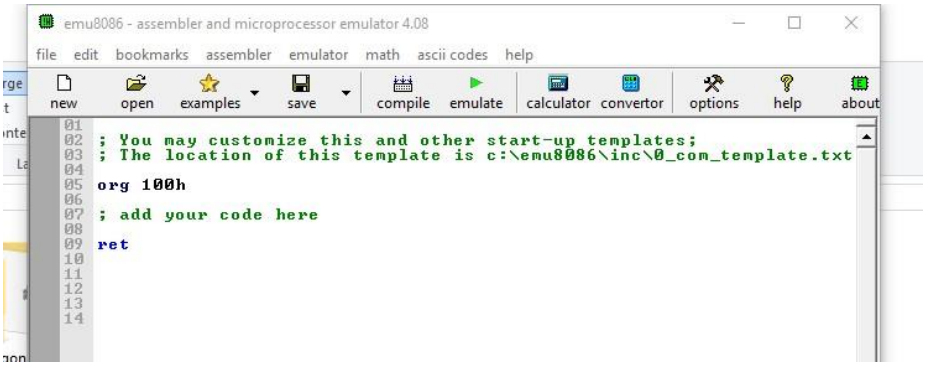


3. After clicking on **'new'** the following window pop-up.

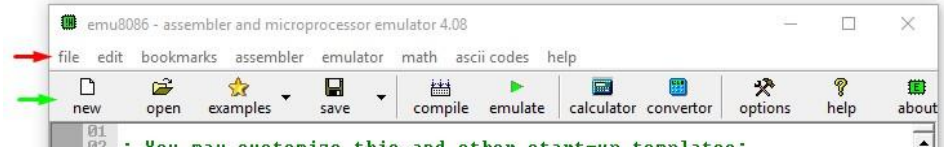
   Click on **'COM template'**, this will open up window of Step 4.



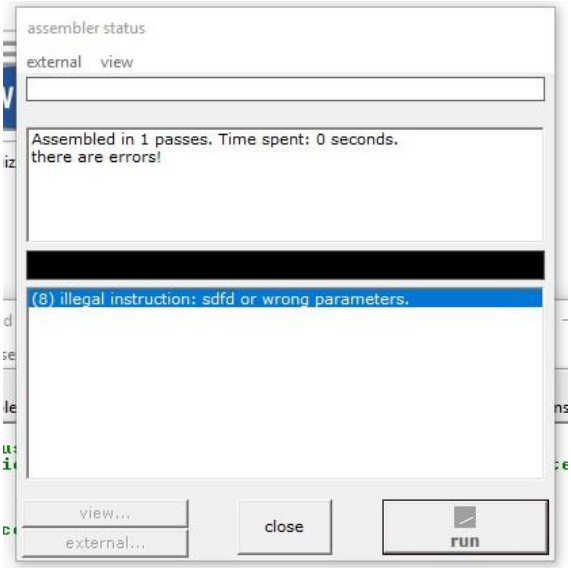
4. Step 3 will open the following window with already written codes.
   This is similar to the 'public class {public static void main…..' at the start of a java class file. **You may also choose to use a blank page to code*



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.

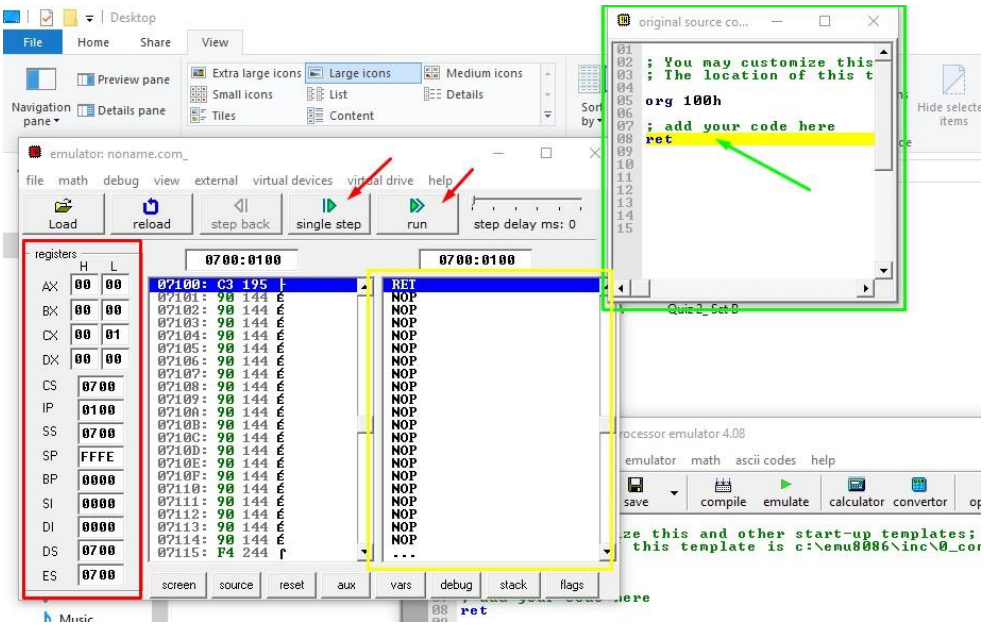
   The *red box* shows the values of all the registers. If you notice, some of the registers are divided into two columns, each of 8 bit (2 hex numbers each, i.e. 4 bit). Each value is in the hexadecimal format.

   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.