## 8086 Board Design Project

**Due: On Last Day of classes (Mainly an in class exercise)**

**Project Description:**
You will be programming the UMBC 8086 Trainer Kit that includes all the features that you designed during the hardware project along with extra functionality. The non-volatile memory has been pre-programmed with a monitoring programing that will load your code and execute it from the RAM. Most of the programming will be done during class/lab meeting times and you will submit the code that you have written using 'submit' before the due date. We will discuss the board, the programming details and various jumper setting during the discussion session.

**Programming Details**
Your coding will be focussed towards the following chips, these devices are decoded at the address mentioned in your hardware project specification document:

- 8279 (Keyboard/Display Controller)
- 8259 (Interrupt Controller)
- 8254 (Programmable Counters)
- 8255 (PPI)
- 16650 (UART)
- 2 7-Segment LEDs and 8 discrete LEDs
- 20 character x 4 line LCD display (can be accessed using software interrupt 0x10)

**Connecting the kit to a PC and running your code**
The monitor program is designed to communicate with the PC using the serial port and a null-modem cable. The serial port marked Serial 1 on the board should be connected to the serial port on the PC.

***Setting***

- Connect using to the serial port (usually COM1) on the PC.
- Set bits per second to 38400, data bits to 8, parity bits to none, stop bits to 1 and flow control to hardware (the monitor program uses CTS/RTS handshaking for data transfer)

***Board Setup***

- Connect a 5V power supply to the system, using either of the two power connectors, current requirement for the board are quiet high so use a power supply in the lab
- On power up or if the Reset key on the board is pressed the monitor program performs a RAM test and configures various devices on the board. The following will display on the LCD screen:

- Once the last line turns blank the board is ready for use
- Pressing the Reset key on the board at any time will reset the system and reload the monitor program

***Prompts***

- On successful boot, the terminal window should display the following message:

**UMBC 8086 Minimum Mode System v1.1**
**Copyright (c) Chintan Patel and Ekarat Laohavaleeson**
**Press 'H' for Help**

**UMBC-86>**

- Press ***'L'*** and follow the instructions to load an Intel Hex File containing your code.
- Once the hex file is received, it is processed and loaded at the right location in the RAM. A message saying 'Intel Hex File Received' is displayed on the terminal window and the control is transferred to your code.
- Reset the system to load another file.

**Programming Considerations**

We will be discussing the programming model extensively during the class discussion, however, here are a few things to keep in mind

- This is a 8086 based system, so only 16-bit and 8-bit registers are available (32-bit register e.g. eax, edx etc. are not present). The system runs in minimum mode without a floating point co-processor so floating points instructions are not allowed
- You will be programming in real mode. This will require that you keep track of your segment registers. If you use one code and one data segment, the monitor program will initialize them properly. Your code segment should be relocated to 0x10000 and your data segment at 0x18000. The stack segment is setup for you at 0x00000 and your stack pointer is initialized so that you have1KB in your stack segment. Interrupt vectors can be loaded in the lower RAM address by initializing the extra segment register to 0x00000. You have access to a total of 128KB of RAM, however, you will not require that much space in your programs.
- Only instructions that run on the 8086 can be used in your code. We will be using a directive in nasm to enforce this. However, you should be careful when you are using indirect addressing. Only a subset of instructions that you have used before are available on the 8086. All memory addressing modes are not available and offsets can be placed in defined registers. All the valid instructions are given at the end of the 8086 datasheet posted on the webpage. These are also given in appendix B of the textbook, if you see a '-' next to the 8086 column for a particular instruction, then it cannot be used in your code.
- We will be using a different output format for nasm for this programming assignments. Also we will be using different linking and locating tools to relocate you code. The lab discussion will cover the details about programming.
- Use interrupt vector numbers 0x08-0x0F for the interrupt controller.
- Binaries and sample code is provided on the webpage.

**Using the LCD display via software interrupt 0x10**

You can display messages on the LCD screen using a software interrupt 0x10. The following parameters need to be passed in the registers mentioned below to the interrupt service routine:

- ***ax: A value between 0 and 4***
  0 for writing to the entire screen (20 characters x4 lines, lines 2 and 3 are swapped) for a total of 80 characters
  1 through 4 to print upto 20 characters on that particular line
- ***bx: Pointer to the string to print in memory***
  pointer to a memory location with 80 characters or less for entire display, 20 or less for a particular line
- ***cx: Number of characters to print***
  max 80 for entire screen, 20 for a particular line
- ***dx: Offset from the beginning of line to start displaying***
  This is used only in the line mode, the sum of number of characters plus the offset should be less than 20
- ***si: Segment base for the string***
  The base of the segment where your string is located. This will usually be the data segment register, however, if you have multiple data segments defined, you can print from any one.

***Return Value***

The return value is 0 for no errors, -1 if there was an error in the line number, -2 if there was an error in count + offset. This is returned in ax, however, you don't have to check it.

***Examples***

Call:

```
mov ax, 0
mov bx, welcome
mov cx, len1
mov dx, 0
mov si, ds
int 10H
```

Data (data segment):

```
welcome: db " Welcome to CMPE 310"        ; 20 characters per line
         db "    Section 0101, 0102   "
         db "          Fall 07          "
         db "     8086 Project          "
len1:     equ $ - welcome
```

Output:


Example Cont. :

Next Call:

```
mov ax, 4
mov bx, change_project
```

```
    mov cx, 6
    mov dx, 11
    mov si, ds
    int 10H
```

Data (data segement):

```
change_project: db " is fun"      ;6 characters, displayed starting at character 12 (offset+1)
```

Output:

**Welcome to CMPE 310**
**Section 0101, 0102**
**Fall 07**
**8086 is fun**