ECS502U Lab: Week 8

The purpose of this lab is to practise solving small problems using 8051 assembler code on actual 8051 hardware. On the QMPlus module page you can find 8051 assembler instruction documentation, including a brief "cheat sheet." We suggest you create a directory for all your code for ECS502U, and then a sub-directory for this week.

The Development Board

The development board provides a typical 8051 configuration. In this case, the ANDing of the RD and PSEN signals from the 8051 microcontroller effectively mean that the code and data memories share a common 64Kbyte address space. This is a Von Neumann architecture. Address decoding ensures that access to the lower 32Kbytes of memory is directed at the external EPROM device, whereas access to the upper 32Kbytes is directed to the RAM.

The development board is shown in Figure 1. Upon reset, the microcontroller begins to execute instructions, starting from location 0000 hex whenever it is first powered up or when the reset button is pressed. This corresponds to an address in the EPROM. The EPROM holds a small program, called a monitor, which initializes the development board and allows it to communicate with a PC. The monitor program supports a number of commands that it receives through the serial interface. These include commands to download an Intel Hex formatted application program and place it at a chosen location within the memory. The monitor can then be instructed to execute the program starting from this given address. This provides a straightforward way to download and execute programs developed within the MCU8051 environment.

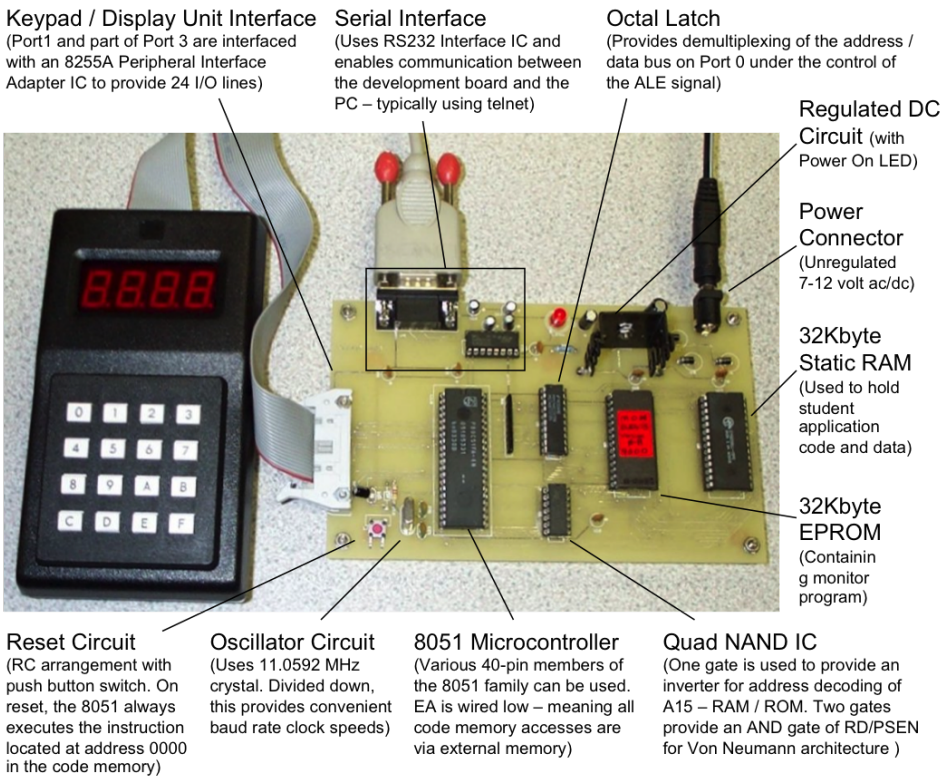


Figure 1: The 8051 Development Board

Running on the Development Board

To run a program on the development board it is necessary to change the origin of the code as follows:

```
ORG 8000h ; origin at 8000 hex
```

See below for the rationale of this change.

1. On the Development Board, connect the LED/Switch Module (as shown below in Figure 2) to the connector labelled Keypad/Display Unit Interface in Figure 1 using the ribbon cable provided.

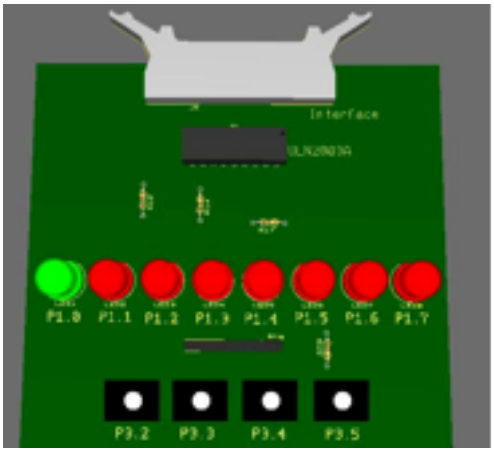


Figure 2: The LED/Switch Module

2. Connect the board to the PC host via a modem cable and to a power supply (AC/DC 7–12V) and check that the power LED is "ON".

3. When we build a program, the Intel Hex file includes information about where the code will be placed in the memory map. On our development board, we must ensure this is at location 8000 hex or above as the monitor program occupies the lower 32K of the memory map. We can do this by modifying the program so that the **ORG** directive is adjusted to "8000h"[1]. Click the menu item "Tool/Compile". As well as loading the code into the simulator it also causes MCU8051 to generate an Intel HEX file. This will have the same name as the source code file but with a ".hex" extension.

4. Due to the compatibility of the original code in the chip and the MCU8051 software, the automatic generated hex file needs to be modified. Open the hex file in the directory when saving the asm file, delete the first line of the data, then save the hex file again.

5. On the PC start up HyperTerminal (from `\\teaching\ELE475`) and create a new connection called "8051". On the Connect To screen select COM1 and on the following screen, Port Settings, select 9600 bits per second and turn off Flow Control. This configures the serial interface appropriately to talk with the Development Board. Press the RETURN

[1] Indeed it is possible to include multiple **ORG** directives within an assembly language program. For example you may wish to have a data block (such as a lookup table) placed at addresses starting from 8000 hex upwards, followed by the code for the program starting at address 9000 hex upwards. In this case we would precede the table (typically listed as a series of **DB** to define data bytes) with **ORG** 8000h and a further **ORG** 9000h before the code listing.

key several times and the reset button on the board until the "CMD>" prompt appears in the terminal window. This confirms that the PC is able to communicate with the Development Board. The board setting can be saved and retrieved for later use by using File→Save, File→Open.

6. The Development Board responds to a number of single letter commands. To obtain a summary, just type "h".

7. To download a file from the PC to the Development Board we will use the "d" command. Assume we wish to download the Intel Hex file "test.hex". The location the program will be loaded to is determined by information within the Intel Hex file when it was created. For example, assume you wish an application program to be loaded starting from the address 8000 hex[2]. To do this it is essential that the assembly language program include the assembler directive **ORG** 8000h at the start of your source code. When the Intel HEX file is created, this "load address" will be encoded into the file.

8. Once you type "d" the monitor program will await the transfer of the Intel HEX file. In the HyperTerminal menu bar you need to select Transfer→Send Text File and select the file you wish to download. This will initiate the transfer. When complete you should be able to get the monitor program prompt by hitting the RETURN key.

9. The application program is now ready to run. To do so we need to issue a "g" for GO command. However we have to instruct the monitor of the start (entry) address of the application program. Assuming it starts at location 8000 hex, we would do this as follows: "g" "80" "00". At this point, the microcontroller will set the program counter to 8000 and commence running the program from that address location.

10. To halt your application and return control to the monitor program, simply press the reset button on the Development Board.

QUESTION 1 Download the program `q1.asm` from QMPlus and do the following:

1. Compile the code using MCU8051 and download it onto the development board as explained above. Do not forget to change the origin to 8000h beforehand. Describe the observed behaviour. *(1–2 sentences)*

2. Modify the program such that P3.2 controls the state of the P1.0 pin. Each time P3.2 goes from High to Low the P1.0 pin should toggle its state. Thus, if P3.2 goes High to Low causing P1.0 to go High, the next time P3.2 goes High to Low, P1.0 should go Low and so on. To toggle P1.0 you will need to remember P3.2's previous state. This can be stored in a variable. A typical way to do this is to use part of the internal RAM to hold the previous status. To use one of the general-purpose bit addressable memory locations we might use:

   {**name** of your choice} **EQU** 0x00 *; the first bit−addressable location in internal RAM*

   Recompile and run the program to confirm the desired behaviour. Provide your modified program (it suffices to copy the "main program" part).

[2]Memory Locations from 0000-7FFF hex on the Development Board are associated with the external ROM. The RAM address range is from 8000-FFFF hex. As it is not possible to write to the ROM, the application code must be stored in locations from 8000 hex or above.

3. Optional: As the compile and debug loop via hardware is time consuming, how could you use MCU8051 to simulate the behaviour? *(1–2 sentences)*

Peripheral Interfacing

The purpose of the next experiment is to enable you to use the development board to control and monitor simple peripherals, in this case a Keypad/Display Unit, accessible via an 8255 Peripheral Interface Adapter.

Expanding the I/O with the 8255 Device

Inside the Keypad/Display Unit is an 8255 device. The 8255 contain three 8 bits ports A, B and C that we can connect or not to the system data bus D. The three ports can be programmed in input or output. In the case of output, all ports are their output latch/buffer. There are three basic modes of operation that can be selected by the system software:

- Mode 0 – basic input/output
- Mode 1 – strobed input/output
- Mode 2 – bi-directional bus

When the reset input goes high all ports will be set to the input mode with all 24 port lines held at a logic one level by the internal bus hold devices. After the reset is removed the 8255 can remain in the input mode with no additional initialization required. This eliminates the need for pull-up or pull-down devices in all-CMOS designs. During the execution of the system program, any of the other modes may be selected by using a single output instruction.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions as shown in Figure 3. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be tailored to almost any I/O structure. For instance; Group B can be programmed in Mode 0 and Group A could be programmed in Mode 1.

In our application, we want to program the ports as follows: port A, B and C7-C4 in output and C3-C0 as input. So, according to the data sheet, after a reset, we must send the control word #1. This is shown in Figure 4. This table also shows the port selection details.

Keypad and Display Application

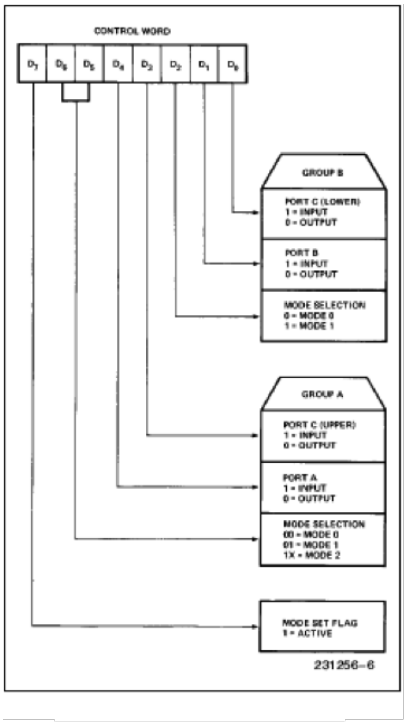
By now, you will be expected to have a good understanding of the 8255 operation as well as the operation of the 8051 device.

This final part of the experiment is concerned with reading input from a small keypad and then displaying the data on a simple multiplexed seven-segment display. Both of these modules are accessed via an 8255 Peripheral Interface Adaptor (PIA) IC as described in above. A block diagram of the system is shown in Figure 5.

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	⟷	
PA_1	IN	OUT	IN	OUT	⟷	
PA_2	IN	OUT	IN	OUT	⟷	
PA_3	IN	OUT	IN	OUT	⟷	
PA_4	IN	OUT	IN	OUT	⟷	
PA_5	IN	OUT	IN	OUT	⟷	
PA_6	IN	OUT	IN	OUT	⟷	
PA_7	IN	OUT	IN	OUT	⟷	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Figure 3: 8255 Mode Selection



A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Input Operation (Read)
0	0	0	1	0	Port A - Data Bus
0	1	0	1	0	Port B - Data Bus
1	0	0	1	0	Port C - Data Bus
1	1	0	1	0	Control Word - Data Bus
					Output Operation (Write)
0	0	1	0	0	Data Bus - Port A
0	1	1	0	0	Data Bus - Port B
1	0	1	0	0	Data Bus - Port C
1	1	1	0	0	Data Bus - Control
					Disable Function
X	X	X	X	1	Data Bus - 3 - State
X	X	1	1	0	Data Bus - 3 - State

Figure 4: 8255 Control Register Values and Port Addressing

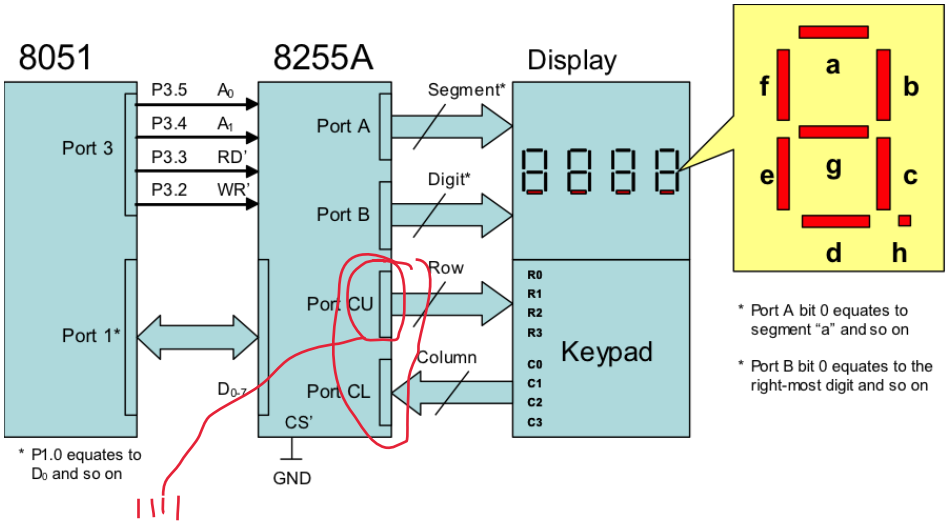


Figure 5: Block Diagram for Keypad and Display Interfacing

QUESTION 2 Download the program `q2.asm` from QMPlus and do the following:

(a) Add a procedure "write" that sets up the 8255, connected as shown in Figure 5, for writing. That is, RDpin and WRpin need to be set/cleared. Add an **NOP** instruction before returning to permit the 8255 to actually send the data across.

(b) Add a wrapper, named "write_C" that configures the port being written to to Port C, and then invokes the above "write" procedure.

(c) Add a procedure "read" that sets up the 8255 for reading from Port C. The result of reading should be stored in the accumulator.

Provide the code for the above in your submission. Ensure that your code includes comments that explain the intention of the most important instructions. For example,

NOP *; wait for the 8255 to complete the read*

would be considered a comment clarifying the intentions, whereas

NOP *; do nothing*

is not helpful.

QUESTION 3

(a) Explain what is meant by an interrupt in a microcontroller and state the possible sources of an interrupt supported by an 8051.

(b) Write some lines of 8051 assembly code to toggle pin 3 of port 1 when the timer 1 overflow occurs.

(c) What bits need to be set in the TMOD register if timer 0 is to be used as a counter counting pulses from the INT0 pin up to a maximum of at least 50,000 counts?

Practice Projects The following mini-projects are for training purposes only and do *not* need to be submitted as part of any coursework. Instead, bring your (partial) solutions with you to the lecture on Friday, where they will be discussed.

(a) *Two-byte left shift:* Shift the contents of registers R0 and R1 (with R1 being the more significant byte) one bit to the left, as if they were a single 16-bit value. The result should be stored in R0 and R1.

(b) *Rounding unsigned bytes to the next multiple of* 8*:* Given an input in register R0, round its value to the nearest multiple of 8. Store the result in the accumulator.

(c) *Button indicator:* Assume a button connected to P3.3, and an LED at P1.0. Toggle the LED after every seventh key press.