ECS502U Microprocessor Systems Design

Lab 2 Report - Peripheral Interfacing

By Harikrishna Soni

Introduction

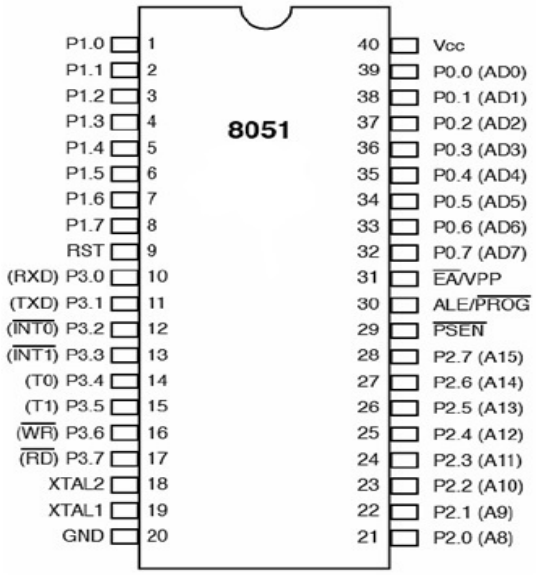
The purpose of this lab is to control and monitor the keypad/display unit using the provided development board. The 8255 peripheral interface adapter will be utilized to access the keypad/display unit. Source code is required so that when a number on the keypad is pressed, the 7 segment display unit displays it. When a new number is pressed, the previous number should shift to the left accordingly and whenever "F" is pressed the display should be reset. The 8051 microcontroller will be used to interface with the 8255, and the code will be written using the MCU 8051 IDE software to achieve this task.

Part A

The four main hardware devices used for this experiment: 8051 microcontroller, 8255A peripheral interface adapter, keypad unit and the four-digit 7 segment display.

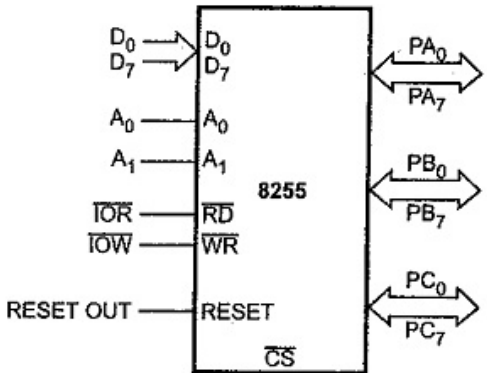
8051 Microcontroller:

The 8051 microcontroller offers four 8-bit input/output ports: ports 0-3. It also contains four different types of memory: Internal memory, external memory, program memory and special function register. Whenever the 8051 requires external memory, only two ports will remain(port 1 and port 3) thus only two input/output devices can be connected. This occurs as port 0 and port 2 are used to address the external memory. Our experiment requires three ports to connect the keypad/display unit. Therefore, we can use the 8255 peripheral interface adapter to get three additional 8-bit input/output lines to use.



8255 Peripheral Interface Adapter:

The 8255 consists of three 8-bit ports A, B and C alongside three operation modes: mode 0(basic input/output), mode 1(strobed input/output) and mode 2(bi-directional bus). In this experiment, we used mode 0, thus, ports A, B and C can be programmed as input or output ports. Port C is divided into two 4-bit ports; CU for the upper nibble of port C and CL for the lower nibble of port C. These two 4-bit ports are independent of each other and can be used for input or output. To interface the 8255 with the 8051 we have to connect the data lines(D0-D7),



address lines(A0-A1) and control signals(RD and WR). The chip select pin(CS') is tied low to activate the entire chip.

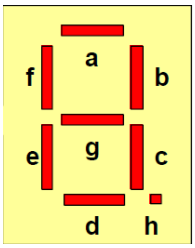
Data lines (D0-D7):
The data lines D0-D7 are directly connected with port 1 pins of the 8051. These pins are used for the transfer of data under the status of the control signals. All the data that is written to and read from the 8255 occurs via these data lines.

Address lines (A0-A1):
The A0 and A1 pins allow us to access different ports. The table below displays the port selection details utilising A0 and A1. Once the port is selected, it is ready for data to be written or read.

A1	A0	Port Selected
0	0	Port A
0	1	Port B
1	0	Port C
1	1	Control Register

Control signals (RD and WR):
The read and write pins are used to read or write data onto the ports of the 8255. These pins are enabled when they are active low. Therefore, in order to read or write data from a port the read or write pin must be 0.

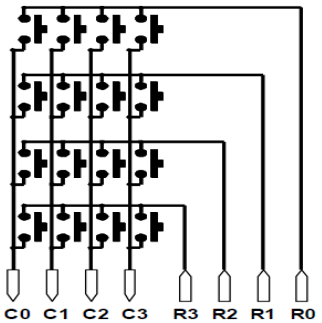
Display:
The display unit is made up of four 7-segment units. A unique segment code can be used to control an individual segment.



Keypad:
The keypad is intended for hexadecimal systems and consisted of a 4x4 matrix with 16 keys ranging from numbers 0-9 and letters A-F.

06h	5Bh	4Fh	71h
66h	6Dh	7Dh	79h
07h	7Fh	67h	5Eh
77h	3Fh	7Ch	58h

1	2	3	F
4	5	6	E
7	8	9	D
A	0	B	C



8255 ports A, B and C:

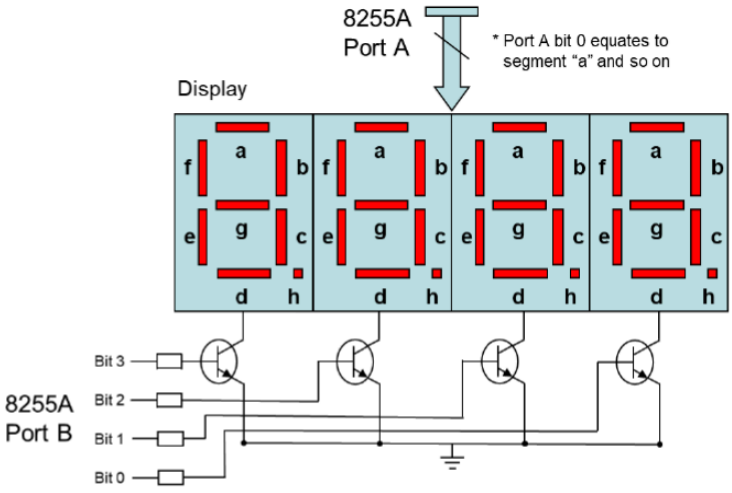
Port A is connected to the display unit in this experiment, and is used to display digits on the seven-segment display. Port A bits are equated to the different segments on the display, port A bit 0 is equal to segment “a”, port A bit 1 is equal to segment “b”, etc.

Port A bit	A0	A1	A2	A3	A4	A5	A6	A7
Segment	a	b	c	d	e	f	g	h

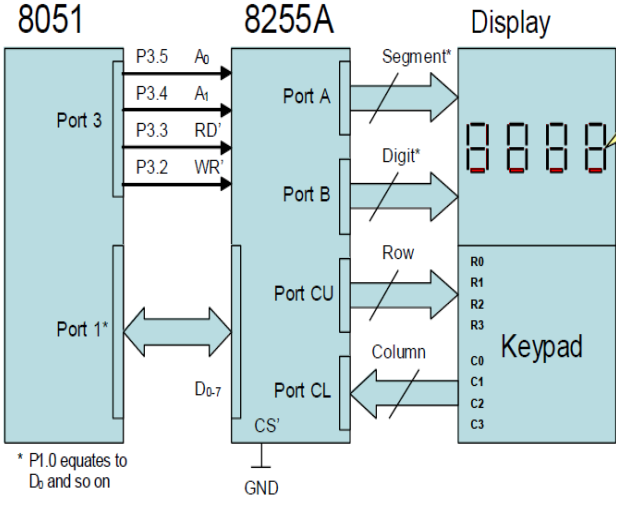
To display the value 2 we require segments a,b,d,e and g on and the rest turned off. Therefore, we transmit 01011011 to port A.

Port B is also connected to the display unit, however, it is used to choose where the digit will be displayed on the four-digit 7 segment display.

Unit position	**Port B code**
Unit on the left	00000001
Unit next to left unit	00000010
Unit next to right unit	00000100
Unit on the right	00001000

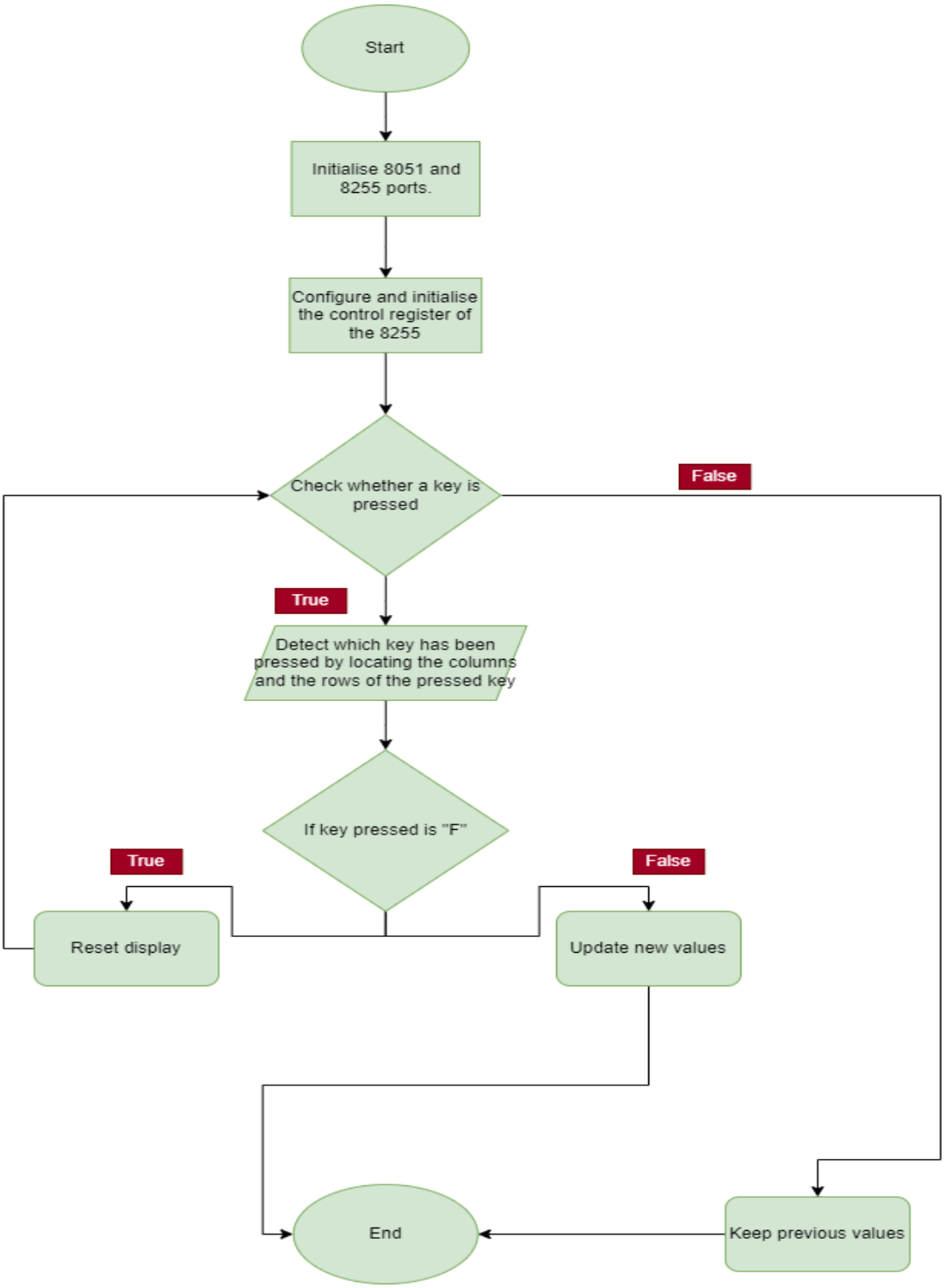


Port C is connected to the keypad consisting of CU for the upper nibble of port C and CL for the lower nibble of port C. CU is connected to the rows and CL is connected to the columns. To detect if a key has been pressed the value of CU is set to “1111” and the column values are read. If the value of CL is read to be” 0000” then no key has been pressed. However, when a key has been pressed it means that a column value has read “1”, thus, one of the four keys across this column has been pressed. The microcontroller then drives one row “1” at a time whilst reading column values to find out which one of the four keys has been pressed. The column outputs “1” when the row containing the pressed key is “1”.



Flowchart design of code

First we initialise the ports of the 8051 and the 8255 alongside the control register. We then check whether a key has been pressed, if it has been pressed we need to figure out which key has been pressed by locating the rows and columns. If a key has not been pressed the previous values will be kept. If the key pressed is “F” then the display is reset and the loop will continue, if the key pressed is not “F” then this value will be displayed on the display unit.



Code explanation:

Code	Description
RDpin equ 0B3h **WRpin equ 0B2h** **A0pin equ 0B5h** **A1pin equ 0B4h** **org 8000h** **KCODE0: db 06h,5Bh,4Fh,71h** **KCODE1: db 66h,6Dh,7Dh,79h** **KCODE2: db 07h,7Fh,6FH,5Eh** **KCODE3: db 77h,3Fh,7Ch,39h**	EQU instruction is used to name the pins of the 8051, which are linked to the 8255 control pins. A lookup table is then created for the keypad values with each line corresponding to a row on the keypad.
Org 8100h **Start:** **;setting the WR and RD pin low** **Setb RDpin** **Setb WRpin** **; setting the address to control register** **setb A0pin** **setb A1pin** **; setting the control word** **mov a,#81h ;81h control word** **mov P1,a** **; write pulse is sent to 8255** **lcall write** **; setting the word to 0** **mov a,#00h** **mov P2,a**	The RD and WR pin are turned off, A0 and A1 pins are set to high with chip select enabled. The control word 81h is sent to the data pin of the 8255, thus the 8255 now has I/O pins and CU is chosen as output whilst CL is chosen as input.
; Port writing to A **clr A0pin** **clr A1pin** **; write pulse is sent to 8255** **lcall write** **; Port to B** **setb A0pin** **clr A1pin** **; write pulse is sent to 8255** **lcall write** **; Port to C** **clr A0pin** **setb A1pin** **; write pulse is sent to 8255** **lcall write** **; clear registers (just to be on the safe side)** **mov R0,#00h** **mov R1,#00h** **mov R2,#00h** **mov R3,#00h** **mov R4,#00h** **mov R5,#00h**	Ports and registered are cleared.
;------- Start Keypad Loop ---------; **KLoop:**	Port c is chosen as we turn A1 high and A0 low. This is done to check the columns, we

clr A0pin **setb A1pin** **mov a, #00011111b** **mov P1, a** **lcall write** **lcall read**	move 0001111 (binary) into port c, then read the columns.
mov dptr, #KCODE0; set dptr to start of row 0 **anl a, #00001111b ; check which row is pressed** **cjne a,#00h,colcheck ; check whats been pressed** **mov a, #00101111b ;ground row 0** **mov P1, a ;read all columns** **lcall write** **clr a** **lcall read** **mov dptr, #KCODE1 ; set dptr to start of row 1** **anl a, #00001111b ; check which row is pressed** **cjne a,#00h,colcheck ; check whats been pressed** **mov a, #01001111b ;ground row 1** **mov P1, a ;read all columns** **lcall write** **clr a** **lcall read** **mov dptr, #KCODE2 ; set dptr to start of row 2** **anl a, #00001111b ; check which row is pressed** **cjne a,#00h,colcheck ; check whats been pressed** **mov a, #10001111b ;ground row 2** **mov P1, a ;read all columns** **lcall write** **clr a** **lcall read** **mov dptr, #KCODE3 ; set dptr to start of row 3** **anl a, #00001111b ; check which row is pressed** **cjne a,#00h,colcheck ; check whats been pressed** **acall display** **ljmp KLoop ;if none of the column row checks are successful we jump back to the start of the loop.**	We associate KCODE values with the DPTR. Therefore if a row was jumped to the dptr would store the address of the corresponding KCODE value. We also read the columns to see if any of them are active high. If any of the columns are not high in row 0 then we move onto row 1 etc.
colcheck: ; column value is checked **rrc a ; checks if any bit is low** **jc match ; if zero, get hexcode of button** **inc dptr ; point to next column address** **sjmp colcheck ; keep checking** **movc a, @a+dptr ; put segment code into A** **cjne a, #01110001b,match ; if key pressed code is not "F" it jumps to match subroutine** **sjmp refresh ; if key pressed code is "F" it jumps to reset subroutine.**	Column value check is done, we first check if the carry flag bit is 0, if it is 0 then we point to the next column address, increasing DPTR value. When we have a carry bit flag of 1, the DPTR will contain the value of the button pressed and jumps to subroutine "match". Then the code checks if the key pressed is "F" if it is not "F" the code functions as normal. If it is "F" everything is reset.
refresh: **CLR a** **Mov R0, a** **mov R0,#00h** **mov R1,#00h** **mov R2,#00h**	The refresh subroutine clears the registers and display is cleared as we send 00H to A. The match subroutine moves the value of R0

mov R3,#00h **mov R4,#00h** **mov R5,#00h** **match: ;** **mov a, R2** **mov R3, a ;value of R2 goes to R3** **mov a, R1** **mov R2, a ;value of R1 goes to R2** **mov a, R0** **mov R1, a ; value of R0 goes to R1** **clr a ;a set to zero** **movc a, @a+dptr ; put segment code into A** **mov R0, a**	to R1, R1 to R2 and R2 to R3.
bouncekey: ; debouncing **clr A0pin** **setb A1pin** **mov a,#0FFH** **mov p1,a** **lcall write** **lcall read** **anl a,#0FH** **cjne a,#00H,bouncekey** **lcall delay** **ljmp KLoop ; returns to the start of the loop** **;------- End Keypad Loop ---------;**	The bouncekey subroutine is used to help with debouncing and to check again a key on the keypad has been pressed.
;----- Start Display Subroutine -------; **display:** **mov a,R0 ; A contains character** **mov P1,a ; character from accumulator is sent to p1** **clr A0pin** **clr A1pin ; Port A selected** **acall write ; character from A is written to p1** **mov a,#00000001b** **mov P1,a** **clr A1pin** **setb A0pin ; Port B selected** **acall write ; first display unit is enabled** **lcall delay** **Ret** **;----- End Display Subroutine -------;**	The segment code from R0 is copied into the accumulator, this code is then copied into port 1. Port A is enabled and the segment code is written to port A. To choose the rightmost display unit we transmit #00000001b to port B. Delay is also added for debouncing. The same method is repeated to enable the other 3 display units and display a character.
write: **clr WRpin** **nop** **setb WRpin** **nop** **ret**	The write subroutine is utilized to send a write pulse, this is done by turning the WRpin on and off.
read: **clr RDpin;** **nop;** **mov a, p1;** **nop;** **setb RDpin;** **nop;** **RET;**	The read pin is activated by ensuring its value is 0. Data in port 1 is transferred to the accumulator. The read pin is turned off by ensuring its value is 1. We used this pin to read data from port 1.

delay: **mov a, #0ffh ; delay** **dly:** **djnz acc, dly ; debouncing** **ret** **end**	Delay subroutine produces a small delay and helps with debouncing.

Debouncing:

When a key is pressed on the keypad, we expect a single contact to be recorded. However, there is initial contact, a slight bounce, another contact as the bounce ends and then another bounce. To deal with debouncing a delay is added so that once a key is pressed, there is a slight delay before the microcontroller records another keypress.

Flicker-free display:

Flickers can occur on the display when a key is pressed. To reduce the effect of flickering we can add a delay to the code dealing with the display.

Concurrent key presses:

When two keys are pressed at the same time, the program displays the character that was pressed on the left. For example, if 5 and 6 were pressed at the same time on the matrix keypad the number 5 would be chosen and displayed. This occurs as a result of the order in which the subroutines are called.

Potential use of interrupt(s):

Interrupts halt the task that was being worked on and works on a different task, thus interrupting the initial task. If interrupts were utilised the 8051 microcontroller could work on another task whilst it is not in use. Thus, when an interrupt does occur, the 8051 is notified to halt the task it was working on and work on a different task.

How the code could be improved:

The use of interrupts could be used to reduce the amount of code required. Polling was used in this instance as the 8051 continuously monitored the status of the device and executed the task when the devices needed it. Polling increased the amount of code and reduced the efficiency of the 8051 microcontroller's resources.

Part B

Addition: To implement addition, the user presses a number and this number is displayed, the user then presses another number on the keypad and it is also displayed. The user can then choose to add these two numbers by pressing “A”.

```
MOV A, Number1 ; first number chosen
MOV B, Number2 ; second number chosen
ADD A, B ; numbers added together
DA, A ; adjusts for BCD
```

Subtraction:

To implement subtraction, the user presses a number and this number is displayed, the user then presses another number on the keypad and it is also displayed. The user can then choose to subtract these two numbers by pressing “B”.

```
MOV A, Number1 ; first number chosen
MOV B, Number2 ; second number chosen
SUBB A, B ; numbers subtracted
DA, A ; adjusts for BCD
```

Multiplication:

To implement multiplication, the user presses a number and this number is displayed, the user then presses another number on the keypad and it is also displayed. The user can then choose to multiply these two numbers by pressing “C”.

```
MOV A, Number1 ; first number chosen
MOV B, Number2 ; second number chosen
MUL AB ; numbers multiplied
```

Division:

To implement division, the user presses a number and this number is displayed, the user then presses another number on the keypad and it is also displayed. The user can then choose to divide these two numbers by pressing “C”.

```
MOV A, Number1 ; first number chosen
MOV B, Number2 ; second number chosen
DIV AB ; numbers divided
```

Equal:

Once the user has chosen the two numbers and decided on the arithmetic operation they can press equal “E” to display the result. The results of the operations can be stored in registers so calling back to them will help display the answer.

Clear:

“F” can be used to clear the display and registers. The display will be cleared as we send 00H to port A.