EE 337: Introduction to the Pt-51 Kit
Lab 3

Rajbabu Velmurugan and Shankar Balachandran

6^{th} August, 2015

This set of experiments has the following objectives:

- Familiarization with Pt-51 kit and simple peripherals.

You should browse through the manual for Keil software, the data sheet for 89C5131A, and the Pt-51 user manual.

89C5131A has 256 bytes of RAM, of which the top 128 bytes can only be accessed using indirect addressing. Since the stack is always accessed through indirect addressing using the stack pointer, we shall locate the stack in this part of the memory. In our assembly programs, we shall initialize the stack pointer to 0CFH, which provides 32 bytes for the stack. It is common to place arrays also in this part of this memory, because arrays are also easily accessed using a pointer. The address range from 80H to CFH is available for this (since memory beyond this is reserved for the stack).

Refer to the textbook by Mazidi and Mazidi for syntax of instructions.

1 Homework

1. Repeat this part from Lab 2 homework, but execute your program on the Pt-51 kit. Write an assembly program to blink an LED at port P1.4 at specific intervals. At location $4FH$ a user specified integer D is stored. You should write a subroutine called `delay`. When it is called it should read the value of D and insert a delay of $D/2$ seconds. Then write a main program which will call `delay` in a loop and blink an LED by turning it ON for $D/2$ seconds and OFF for $D/2$ seconds. D will satisfy the following constraint: $1 \leq D \leq 10$.

   You can use the code sequence below to introduce a delay of $\sim 50ms$.

```
MOV R2,#200
BACK1:
    MOV R1,#0FFH
    BACK :
         DJNZ R1, BACK
    DJNZ R2, BACK1
```

   The counters R1 and R2 are setup in such a way that the code above would take roughly $50ms$. Please see Appendix for how the delay is calculated.

   You have to nest this inside another loop with appropriate counters to get the required delay.

2. Repeat this part from Lab 2 homework, but execute your program on the Pt-51 kit. Write a subroutine `zeroOut` which will read a number N from the slider switches (P1.3 - P1.0) and a pointer P from $51H$. The subroutine should zero out the contents of memory in N consecutive locations starting at P. N will satisfy the following constraint: $1 \leq N \leq 16$. It should also display the slider input nibble value on the four LEDs.

   Note: Refer to the `main.asm` file in `lab3.zip` to initialize and read from the ports of the Pt-51 kit. Setup the slider switches before running your program.

3. Repeat this part from Lab 2 homework, but execute your program on the Pt-51 kit. Write a subroutine `display` which will read a number N from location $50H$ and a pointer P from $51H$. The subroutine must read the last four bits of the values at locations P to $P + N - 1$ and display on the LEDs, one at a time. There should be a delay of 1s between each such display. Use the ports P1.4 to P1.7 for the 4 LEDs. To demonstrate this routine, fill the memory locations P to $P + N - 1$ with values 1, 2, ..., N.

First you should assemble and debug these programs using Keil software on a PC or laptop. Next you should download these programs on the Pt-51 kit and check that the program is operating correctly.

2 Lab work

The lab work involves three exercises. For problems 2 and 3, make use of the template code provided.

2.1 Problem 1: Binary to Gray code

Write an assembly program to convert a 4-bit Binary code, read from onboard switches (P1.3-P1.0) of the Pt-51 kit, to Gray code and display it on onboard LEDs (P1.7-P1.4). This conversion should happen in a continuous loop.

2.2 Problem 2: Port I-O and Arrays

1. Write a subroutine `readNibble` (as per the algorithm template given next) which will read the binary value which is set on the port using slide switches (P1.3-P1.0). The subroutine should display this value on the LEDs for 5 seconds and store the nibble as the last four bits of location $4EH$.

```
readNibble :
; Routine to read a nibble and confirm from user

; First configure switches as input and LED's as Output.
; To configure port as Output clear it
; To configure port as input, set it.

; Logic to read a 4 bit number (nibble) and get confirmation from  user
loop:
  ;turn on all 4 leds (routine is ready to accept input)
  ;wait for 5 sec during which user can give input through switches
  ;turn off all LEDS
  ;read the input from switches (nibble)
  ;wait for one sec
  ;show the read value on LEDs
  ;wait for 5 sec ( during this time delay User can put all switches to OFF
```

```
    position to signal that the read value is correct and routine can proceed to
    next step)
    ;clear leds
    ;read the input from switches
    ;if  read value  <> 0Fh go to loop
; return to caller with previously read nibble in location 4EH (lower 4 bits).
```

Note: you should push / pop all registers being used in the algorithm

This algorithm provides a visual handshake and is to be used for taking nibble inputs from slider switches in the lab work problems. The user is to setup the slider switches to specific value of interest during the 5 sec period when all LEDs are ON to confirm the previous nibble entered.

2. Write a subroutine `packNibbles`. Two successive 4 bit values read using `readNibble` should be combined to form a byte (with most significant nibble being read first followed by least significant nibble), which should be stored at location $4FH$.

3. Write a subroutine `readValues` that will call `packNibbles` to read in k bytes ($0 < K < 10$, with K stored in 50H) this way, storing them in an array in memory starting at location P. Location $51H$ holds the value of P. Remember that `packNibbles` will always place the byte at $4FH$ but `readValues` should copy the values to different locations.

4. Write another subroutine `displayValues` that should read a pointer to an array from $51H$ and value of K from 50H. It should then read a 4 bit value from the port. If this value is greater than K (invalid input), the subroutine should turn OFF all LEDs and stop. Otherwise, this value should be used as an index in the stored array. The corresponding byte nibbles should be toggled and displayed on LEDs every 2 seconds. Higher nibble should be displayed first. The subroutine must be able to read the switches and update continuously.

2.3 Problem 3: Array manipulation

As in the previous problem, store k elements of an array by reading the port. You should use the same subroutines `readNibble`, `packNibbles` and `readValues` from Problem 2. Use location $60H$ to store the values of A.

Now, write another subroutine `shuffleBits` which does the following task: if the original array is $A[0]$, $A[1]$, ..., $A[K-1]$, generate the array B, such that it contains

$$A[0]\ XOR\ A[1], A[1]\ XOR\ A[2], \ldots, A[8]\ XOR\ A[K-1], A[K-1]\ XOR\ A[0].$$

Location for B is $70H$. Now, call `displayValues` from Problem 2 to display the elements of B by reading the index from the port as in the previous part and displaying continuously. Use the template code provided next.

```
;=====================MAIN===================
ORG 0XXXH
MAIN:

MOV SP,#0CFH;-----------------------Initialize STACK POINTER
MOV 50H,#XX;------------------------Set value of K
MOV 51H,#XX;------------------------Array A start location
MOV 4FH,#XX;------------------------Clear location 4FH
LCALL readValues
```

```
MOV 50H,#XX;------------------------Value of K
MOV 51H,#XX;------------------------Array A start location
MOV 52H,#XX;------------------------Array B start location
LCALL shuffleBits

MOV 50H,#XX;------------------------Value of K
MOV 51H,#XX;------------------------Array B start Location
LCALL displayValues;----------------Display the last four bits of elements on LEDs

here:SJMP here;---------------------WHILE loop(Infinite Loop)
END
; -----------------------------------END MAIN-----------------------------------
```

3 Appendix

The code segment below can be used to get a delay of $\sim 50ms$:

```
MOV R2,#200
BACK1:
    MOV R1,#0FFH
    BACK :
         DJNZ R1, BACK
    DJNZ R2, BACK1
```

The delay calculations are as follows. `MOV` and `DJNZ` instructions take 1 and 2 machine cycles respectively. Therefore, the number of machine cycles required for the for loop is 1+255*2 = 511 machine cycles. Since this is inside a loop where R2 = 200, the outer loop at `BACK1` takes 200*511 + 200*2 = 102,600 machine cycles. Each machine cycle is 12 clock cycles making the total number of clock cycles 1, 231, 200 clock cycles. Dividing this by the crystal frequency (24 MHz), we get a delay of $\sim 50ms$.