

# NPTEL-WEL Summer Workshop on Micro Controllers

## Lab-1: Intoduction to Assembly Programming

This set of experiments has the following objectives:

- Familiarization with the development environment for 89C5131A.
- Familiarization with the instruction set of 8051 family.

You should browse through the manual for Keil software and the data sheet for 89C5131A. 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 the memory, because arrays are also easily accessed using a pointer. The address range from 80H to 0CFH is available for this (since memory beyond this is reserved for the stack).

Direct addressing in the address range 80H to 0FFH accesses special function registers, which control IO ports, timers, interrupts etc. In particular, the micro-controller board that we shall use has push button switches and LEDs connected to a port. The switches and the LEDs will be our primary medium for input/output during this set of experiments. Since we do not have a monitor program running on the board, most of the debugging should be carried out using simulation at this stage.

To end the program, you put it in an endless loop.

### Part-1

1. Write an assembly language program to generate programmable software delay. It should accept a 16 bit value as a count, stored at address 81H. The delay should be proportional to this count. The delay is generated by a count down loop, which uses the instruction DJNZ (Decrement and Jump if NonZero).  
Write another version which uses a count up loop, using CJNE (compare and jump if not equal). In both cases, you will have to use a loop within a loop in order to manage a 16 bit count.  
You should assemble and debug these programs using Keil software on a PC or laptop. You should check that the program is operating correctly using single stepping and break points provided by Keil Software.
2. Convert the above program to a subroutine. The count value will be placed in the upper memory by the calling program and R0 will be pointing to this location when the subroutine is called.

## Part-2

The part of the lab work involves three simple exercises.

### 1: Software delay for 1 second

Use the subroutine developed as homework to make a 1 second timer. The program should set the pins P1.4 to P1.7 of Port-1 for about 1 second and then clear these off.

Write the main routine which should:

- Set the pins P1.4 to P1.7 of Port-1,
- enter a loop which calls the delay routine in each iteration and
- when the loop terminates, Clear the pins P1.4 to P1.7 of Port-1.

Adjust the number of iterations in the main routine loop and the delay count to produce a total delay of nearly 1 second.

### 2: Port I-O and use of arrays

1. Write an assembly program to toggle port pin P1.7 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 toggle port pin P1.7 by setting it ON for *D/2* seconds and clearing it OFF for *D/2* seconds. *D* will satisfy the following constraint:  $1 \leq D \leq 10$ .
2. Write a subroutine which converts a byte in binary to two ASCII characters representing its Hexadecimal representation. The byte will be placed in A by the calling program. The subroutine should return the two characters in registers A and B. For example if 0x4A is passed through subroutine then it should return 0x34 and 0x41. Where 0x34 and 0x41 are the ASCII character representation of '4' and 'A' respectively.

Debug this using the assembler/debugger on a PC or laptop.

Note:-This part will be used in subsequent lab experiment

3. Write a subroutine **readNibble** (as per the algorithm template given next) which will read the binary value which is set on the port using pins of port 1 (P1.3-P1.0). The subroutine should display this value on the pins of port 1 (P1.4 to P1.7) for 5 seconds and store the nibble as the last four bits of location *4EH*.  
Once it is displayed, the program should clear the pins P1.7-P1.4 for one second and call **readNibble** again to read the new value from the pins (P1.0 to P1.3). If the read value equals to *0FH*, the program should display the value stored at *4EH* (the previously read nibble), otherwise, display the new value. Here **readNibble** is the subroutine, which is to be called only once for reading the nibble.

```

readNibble :
;Routine to read a nibble and confirm from user
;First configure P1.0 to P1.3 as input and P1.4 to P1.7 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:
    ;set all 4 pins P1.4 to P1.7 (routine is ready to accept input from the user)
    ;wait for 5 sec during which user can give input through switches
    ;clear all 4 pins P1.4 to P1.7
    ;wait for one sec
    ;read the input from P1.0 to P1.3 (nibble)
value_changed:
    ;show the read value on pins P1.4 to P1.7
    ;wait for 5 sec
    ;clear pins (pin P1.7 - p1.4)
    ;read the input from switches
    ;if read value != 0Fh go to loop
display_old:
    ;otherwise display previously stored nibble from location 4EH
    ;wait for 5 seconds
    ;read the input from pins (P1.0 to P1.3)
    ;if it is still 0FH display_old value
    :else go to value_changed

```

**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 the pins P1.0 to P1.3 in the lab work problems. The user is to setup the pins P1.0 to P1.3 to specific value of interest during the 5 sec period when the pins P1.4 to P1.7 are ON to confirm the previous nibble entered. textbfNote: you should push / pop all registers being used in the algorithm

4. 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**.
5. Write a program which will read the binary value which is set on the port pins P1.0 to P1.3 . The program should display this value on the Port Pins P1.4 to P1.7 for 5 seconds. Two successive 4 bit values read like this should be combined to form a byte, which should be stored as an element of an array.

The program should read in 10 bytes this way, storing them in an array in memory starting from 60H. After doing this, the program should read another 4 bit value from the port pins P1.0 to P1.3. If this value is greater than 09, the program should clear the pins P1.4 to P1.7 and stop. Otherwise, this value should be used as an index in the stored array. The corresponding byte value should be displayed on Pins P1.4 to P1.7 with higher nibble first followed by lower nibble with a delay of 2 seconds.

**3: Additional problems: Array Manipulation**

Store 10 elements of an array by reading the port. If the original array is  $A[0], A[1], \dots A[9]$ , Generate the array B, such that it contains

$$A[0] \text{ XOR } A[1], A[1] \text{ XOR } A[2], \dots A[8] \text{ XOR } A[9], A[9] \text{ XOR } A[0]$$

Now display the elements of this new array by reading the index from the port-1 as in the previous part, stopping when the given index is  $> 9$ . Location of first element of array A starts from 60H and Location of first element of array B starts from 70H.

