EE 337: Advanced Assembly Programming
Lab 2

19^{th} August, 2019

This set of experiments has the following objectives:

- Familiarization with advanced instructions of 8051 family.
- Familiarization of breaking down a problem into subproblems and using subroutines to solve a problem.

You should browse through the manual for Keil software and the data sheet for 89C5131A uploaded on moodle site. Specifically, refer to these to understand the memory-map.

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. Understand the sample (`sample_led.asm`) program provided (in `lab1.zip` file) before getting started on these exercises. Specifically, understand how the code is written down as subroutines to make the code readable. Also, the subroutines use `PUSH` and `POP` instructions for storing and restoring the callers' registers. (Viva 5 points)

   You will be using a board in the later weeks which uses a 24MHz crystal. In order for your simulation to match design running on the board, you have to setup the crystal frequency in the simulator to be 24MHz.

2. Write an assembly program to blink an LED at port 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 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 $50ms$. See the Appendix to know how the delay is calculated.

You have to nest this inside another loop with appropriate counters to get the required delay.(5 points)

You should assemble and debug these programs using Keil software on a PC or laptop. You should check that the program is operating correctly. If necessary, use single stepping and breakpoints provided by Keil Software.

2 Lab work

2.1 Problem1:Clear Memory ZeroOut

Write a subroutine `zeroOut` which will read a number N from location $50H$ 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: $1 \leq N \leq 16$. This routine can be used to initialize memory before usage.(5 points)

2.2 Problem2:Nibble display on Port P1

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. Note that the simulator supports display of 8 output ports but only 4 LEDs are present in the board. Later, you will be able to use this subroutine as is when we try designs on the board.(5 points)

2.3 Problem3:ASCII Conversion

Write a subroutine `bin2ascii_checksumbyte` that will read a number N from location $50H$, a read pointer P1 from $51H$, write pointer P2 from $52H$. The subroutine should compute the checksum byte (as done in lab-1). The checksum byte should be at the next location after the N bytes. The lower and higher nibbles of the checksum byte should then be converted into a pair of ascii (8-bit) format numbers with higher nibble in first and lower nibble in next memory location specified by pointer P2.(10 points)

For example: if checksum byte is $1FH$ then the two ascii bytes should be $31H$ and $46H$ representing characters 1 and F respectively. If the write pointer stored at $52H$ is $60H$, then these two bytes $31H$ and $46H$ should respectively be stored in locations $60H$ and $61H$.

2.4 Problem4:Memory Block Copy

There are many circumstances where a series of memory locations are copied to another location. For example, a network packet captured in a certain memory location by a network device may get copied by the OS to another location where an application processes the packet.

Write a subroutine `memcpy` which will read a number N from location $50H$, a pointer A from $51H$ and another pointer B from $52H$. It should then copy N locations from address A to address B. Assume that the N is smaller than the difference between the addresses A and B so that overlap issues do not arise.(5 points)

3 Learning Checkpoints

Following tasks have to be shown to TA to get full credit for this experiment.

1. Initialize an array to any number other than zero using the subroutine `zeroOut` and clear it using the same subroutine.

2. Explain the purpose of initializing stack pointer to `0CFH`. What is the content of SP when 8051 is powered up?

4 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 $\sim 50ms$.