EE 337: Advanced Assembly Programming
Lab 2 Homework

Rajbabu Velmurugan and Shankar Balachandran

30th July, 2015

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.

   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 P1.4 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.

3. 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.

4. 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. Write a subroutine `sumOfSquares` that will read a number N from location $50H$ and a pointer P from $51H$. The subroutine must generate sum of squares i.e. $\sum_{j=1}^{j=i} i^2$ for $i = 1 \cdots N$ and store them in locations P to $P + N - 1$.

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

The lab work involves two exercises. To perform the second exercise, make use of the template code provided.

2.1 Problem1: Block copy of memory locations

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. Note that A and B may overlap in which case parts of A may get overwritten. N satisfies the constraint that $1 <= N <= 16$.

Example:

If $A = 60H$ and $B = 65H$ and if the values 1 to 10 were in $60 - 69H$, then after calling `memcpy`, locations $60H - 6EH$ must contain 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10.

If $A = 65H$ and $B = 60H$ and if the values 1 to 10 were in $60 - 69H$, then after calling `memcpy`, locations $60 - 6EH$ must contain 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 6, 7, 8, 9 and 10.

2.2 Problem 2: Putting it all together

In this problem, you will use the template of the main program below which calls various subroutines that you have developed earlier. The program is written in such a way that the following steps are carried out:

- N memory locations of an array A are cleared.

- N memory locations of an array B are cleared.
- Array A is filled up with sum of squares.
- The values are then copied from A to B.
- The values at B are displayed on the portview of the simulator.

```
;=======================================MAIN============================================
ORG 0XXXH
MAIN:
MOV SP,#0CFH;-----------------------Initialize STACK POINTER

MOV 50H,#XX;------------------------N memory locations of Array A
MOV 51H,#XX;------------------------Array A start location
LCALL zeroOut;----------------------Clear memory

MOV 50H,#XX;------------------------N memory locations of Array B
MOV 51H,#XX;------------------------Array B start location
LCALL zeroOut;----------------------Clear memory

MOV 50H,#XX;------------------------N memory locations of Array A
MOV 51H,#XX;------------------------Array A start location
LCALL sumOfSquares;-----------------Write at memory location

MOV 50H,#XX;------------------------N elements of Array A to be copied in Array B
MOV 51H,#XX;------------------------Array A start location
MOV 52H,#XX;------------------------Array B start location
LCALL memcpy;-----------------------Copy block of memory to other location

MOV 50H,#XX;------------------------N memory locations of Array B
MOV 51H,#XX;------------------------Array B start location
MOV 4FH,#XX;------------------------User defined delay value
LCALL display;----------------------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*501 + 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 $\sim 50ms$.