Lab-1: Introduction to 8051 Instructions

8 Aug. 2019

This set of experiments has the following objectives:

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

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.

- Data sheet: Section 12 (On-chip Expanded RAM) to understand the memory-map of 89C5131A.
- Reference slides or text relevant to internal RAM of 8051, addressing modes, stack pointer etc.

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 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 sliding 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.

1 Homework

1. Write an assembly program that adds two 8-bit numbers in memory locations 50H and 60H and writes the result to memory location 70H and carry should be at 71H. (4 points)
2. Write an assembly program to perform addition of two 16-bit numbers given in 2's complement form. The two numbers are stored in memory locations 60H and 70H. Storage is big-endian i.e., most significant byte is in the smallest address. The result can be 17 bits long, so it needs three bytes for storage. The result is to be stored in memory locations starting from 62H. (62H will contain the sign bit, 63H and 64H will contain the sum.) Make use of the template code `template_adder.asm` provided in the `lab1.zip` file.(6 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 using single stepping and break points provided by Keil Software.

2 Lab work

The lab work involves one exercise. To perform this exercise, make use of the template code `template_checksum.asm` provided in the `lab1.zip` file.

2.1 Obtain checksum of 40 numbers

There are different kinds of checksum mechanisms. Here is a simple and weak scheme, presented here as your exercise.

Load the values from 1 to 40 in memory locations 40H to 67H respectively. Find the checksum for the stored 40 numbers. To obtain the checksum add all 40 numbers (ignoring the carries) i.e. modulo 256 addition. Then perform 2's compliment of the resultant sum. This is the checksum byte. Store this checksum byte to memory location 68H.

Now we have the original array of bytes, appended with their checksum byte.

To validate integrity of this, add up (ignoring carries) these 41 bytes (the original 40 bytes lying at locations 40h to 67h and the checksum lying in the next location 68h). The result should be zero.

3 Learning Checkpoints:

You have to show correct result to the testcases provided to the TA to get the full credit for this experiment.

4 Testcases for TAs

1. {1.1}
   (a) Number1=FF, Number2=FF, Answer=FE , carry=1
   (b) Number1=09, Number2=F0, Answer=F9 , carry=0
   (c) Number1=0E, Number2=0F, Answer=1D , carry=0
2. {1.2}
   (a) Number1=3678(+13944), Number2=3A79(+14969), Answer=70F1(+28913) , carry=0
   (b) Number1=FFF9(-7), Number2=7FFF(+32767), Answer=7FF8(32760) , carry=0
   (c) Number1=FEBE(-322), Number2=FFFF(-1), Answer=FEBD(-323) , carry=1
3. {2.1} The addition (ignoring carries) of 41 bytes from 40h to 68h should be zero.