Union College
ECE352
Assignment 3 Solution

Due Date: Tuesday April 29th

Problem 1.

Write a sequence of instructions in assembler to convert a 5-bit number in A to its BCD (Binary Coded Decimal) equivalent, i.e., the result in A should have the first 4 bits as the tens and last 4 bits as the units (example: A=18d=00010010 (binary) = 0001 1000 (BCD) [Final result in A])

```
; divide by 10
 mov A, #26
 mov B, #10
 div AB

; Take the lower nibble of A and the lower nibble of B
 swap A

 ; Note: to exchange lower nibbles, we must use indirect addressing
 mov 30h, B
 mov R0, #30h
 xchd A,@R0
```

Problem 2.

Write an assembly program, by following steps a to e below, which will read the **eight (8)** values in code memory given at the location given my myvals (in the code segment given below), and compute the average. Be very careful to account for the size of the result of the additions. Also, because we have 8 values, **division should be avoided** to make the calculation faster. Put the result of your calculation in the register R7.

```
            $include (c8051F020.inc)
            cseg at 0
            ljmp Main

            cseg at 2000h
myvals:     db 23h,58h,04h,99h,AEh,1Ch,D3h,86h

            cseg at 100h
Main:       mov  WDTCN, #0DEh
            mov  WDTCN, #0ADh
```

a. Write an algorithm to implement the program.
b. Draw box diagrams to indicate what memory and registers will be used.
c. Translate your algorithm into 8051 assembler code.

```
; This is essentially a 16b addition problem.
; I will store the total in R1 (high) and R0 (low)
mov R1, #00h
mov R0, #00h

; Initialize the DPTR to the start of the table
    mov DPTR, #2000h

loop:       clr A
    movc A, @A+DPTR

     add A, R0   ; add to the low byte total
    mov R0, A   ; write back to the low

            mov A, R1
    addc A, #00h ; there is no high byte to the data
                 ; so just account for the carry
            mov R1, A    ; write back to the high

    inc DPTR
    mov R7, DPL
    cjne R7, #08, loop

; Now we have the totals in R1 and R0, we need to divide by 8
; by shifting both to the right by 3 (similar to homework problem)

mov R7, #03h
dv:
    clr C
    mov A, R1
    rrc A       ; lsb is shifted into C
    mov R1, A
    mov A, R0
    rrc A       ; C bit is made msb of A
    mov R0, A
 djnz R7, dv

finish: sjmp finish

end
```

d. Implement your code on your development board and test.
e. Briefly comment on how accurate your solution will be to this problem.

Problem 3.

How long does the following software delay subroutine take? (Note that the answer will be a function of R0). (Hint: First calculate the number of cycles.)

```
delay:   mov R7, #02h     ; 2 cycles
Loop1:   mov R6, #00h     ; 2 cycles
Loop0:   mov R5, #00h     ; 2 cycles
         djnz R5, $       ; line takes 255*3(for jumps)+2 (for no jump) = 767 cycles
         djnz R6, Loop0   ; Loop0 takes 255*(767+3+2) + 2 = 196862 cycles
         djnz R7, Loop1   ; Loop1 takes 2*(196862 +2) +3 +2 = 393733 cycles
         djnz R0, delay   ; delay takes (R0)*(396733+2) + (R0-1)*3 +2 cycles
                          ; provided that R0 is in the range 0 to FFh
         ret
```

a) Assuming a 2MHz internal clock?

Approximately R0.198 seconds (divide number of cycles by 2000000)*

b) Assuming a 22.1184MHz external oscillator?

Approximately R0.0179 seconds (divide number of cycles by 22118400)*

Problem 4.

We are going to use Timer 1 in interrupt mode to blink the LED on the development board.

a). Assume that timer 1 is set up to be in 16-bit mode, and sysclock is set to be the internal 2MHz oscillator (NOT divided by 12). Write the instructions that would configure Timer 1 to these specifications and enable the interrupt.

```
mov   IE, #088h        ; T1 and Global Interrupt Enable
mov   TMOD, #010h      ; Timer Mode 16-bit
mov   CKCON, #010h     ; Clock Control - use sysclk
mov   TCON, #040h      ; Run Timer
```

b) Below is the interrupt service routine (ISR) for blinking the light. Note the write to the Timer 1 register. Calculate the delay between toggles of the LED.

```
mr1_isr:
            cpl LED
             mov TH1, #80h    ;reinitialize timer
             mov TL1, #0
             reti             ; no need to clear timer interrupt flag, it is reset by hardware
```

Since the timer is reinitialized to 8000h, it will run 8000h before it overflows. This corresponds to 32768 cycles. With a 2 MHz clock, the time required is 16.4ms.

Problem 5.

Revise the interrupt service routine (ISR) in problem 4 so that the delay between toggles is 1 ms.

1ms on a 2 MHz clock means that the number of cycles is 2000. In hex, this corresponds to 0x07D0 and subtracting this from 0x10000 gives 0xF830. So we just need to load this value into the timer registers:

```
tmr1_isr:   cpl LED
            mov TH1, #0F8h    ;reinitialize timer
            mov TL1, #30h
            reti
```

Problem 6.

Write a C function to solve Problem 1.

```
#pragma CODE          //generate assembly code in the LST file
//-----------------------------------------------------------------------------
// Program:DCD.c
// Author:
// Purpose: Solve the assembler proble 1 in C
// Convert a 5 bit number to BCD and store both bytes in 1 memorl location
//
//-----------------------------------------------------------------------------

#include <c8051f020.h>                  // SFR declarations

//-----------------------------------------------------------------------------
// Global CONSTANTS
//-----------------------------------------------------------------------------

sbit  LED = P1^6;
sbit  SW = P3^7;
```

```
//-----------------------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------------------

// create an absolute address to save result
unsigned char idata BCD _at_ 0x80;


void PORT_Init (void);

//-----------------------------------------------------------------------------
// MAIN Routine
//-----------------------------------------------------------------------------
void main (void) {

unsigned char Number;
unsigned char LoNib;
        WDTCN = 0xde;                   // disable watchdog timer
        WDTCN = 0xad;



   Number = 26;
        // get the high nibble by dividing by 10 (whole part) and shifting 4 bits to the left
   BCD  = (Number / 10) << 4 ;
   // the low nibble is the remainder
   LoNib = Number % 10;  // get low nibble from remainder
  // combine low nibbel with BCD
  BCD = BCD | LoNib;
  for(;;);




}
```

Spring 2014