Due: Friday, October 14 at 5:30 p.m.

Submission: Please hand me a paper copy or slide it under my office door (SCI 3809).

Problems

For problems 1-4, translate the C code segment into AVR assembly language. Assume that all variables are 8-bit integers that they reside in memory. For clarity, I'll specify the location of each variable in memory for each problem below. Each of your program segments should include statements to read the necessary values from memory and to store the results back to memory. If any variable's previous value is used, it should be read from memory first, and if any variable's value is changed, it should be written back to memory afterward.

Note that all of these program segments can be written using the assembly instructions we learned in class. You don’t need to look up additional instructions from the AVR instruction set documentation.

1. Assume $n$, $i$, and $sum$ are located in memory at addresses $0060$, $0061$, and $0062$, respectively.

   ```
   sum = 0;
   for (i = 1; i < n; i++) {
       sum += i;
   }
   ```

2. Assume $n$ and $m$ are located in memory at addresses $0060$ and $0061$, respectively.

   ```
   while (n < 20) {
       m--;
       if (n >= m) {
           n = n + 2;
       } else {
           n++;
       }
   }
   ```

3. Assume $a$, $b$, and $c$ are located in memory at addresses $0060$, $0061$, and $0062$, respectively.

   ```
   if (a >= 5 && a < 15)
       b = c;
   else
       b = d;
   ```
4. Assume $a$ and $b$ are located in memory at addresses $0060$ and $0061$, respectively.

Notice that the conditional operator is greater than, not greater than or equal to. The AVR assembly language does not have an instruction to test for greater than; however, you can write this code with only the instructions we learned in class.

```c
if (a > b) {
    a = 0;
} else {
    a--;  
}
```

5. Show the machine language code (in binary) for the following AVR instructions. Refer to the AVR Instruction Set documentation on the course web site for instruction formats.

```c
CLR r5  
LDI r16, 0x0A  
ADD r5, r16
```

6. Refer to the attached AVR assembly program. Show the contents of the stack during the innermost call to the `termial` procedure. You may write RAH for the high order byte of the return address and RAL for the low order byte of the return address. (Note that this version is somewhat different from the one we wrote in class last week.)

**Optional extra problems**

You may submit a solution to these optional extra problems at any time before the beginning of Finals Week. Submit your solution via email to [davisjan] with the subject line “Reset Counter [lastnames]”.

Build a controllable counter in Logisim as follows.

1. Build a four-bit counter using D flip-flops. The counter should have two inputs: a clock signal and an INC (increment) signal. The counter should increment only when both the clock and INC are high. When INC is low, the counter should hold its value regardless of the clock.

   The outputs $B_3$, $B_2$, $B_1$, and $B_0$ should be exactly equal to the values stored in the four D flip-flops.

   Please label each input and output.

2. Add a RESET input to your counter. When the RESET input is high, all four bits of the counter should be set to 0.

   Note: The behavior of the counter is undefined when both INC and RESET are high.
.include "m8515def.inc" ; Architecture-specific constants

; set up Port B for output
ldi r16, 0xff
out DDRB, r16

; initialize the stack pointer
ldi r16, high(RAMEND)
out SPH, r16
ldi r16, low(RAMEND)
out SPL, r16

; call termial procedure
ldi r16, 0x04
push r16
rcall termial
pop r16

; display results on LEDs (on=0, off=1)
COM r0
out PORTB, r0

; delay forever so result can be seen
forever:
rjmp forever

; termial
; parameter: unsigned byte n, stored on stack
; produces: unsigned byte termial-n, stored in r0
; postconditions:
; if n is 0, then r0 = 0
; otherwise, r0 = n + termial (n-1)
termial:
push YH ; Save Y pointer
push YL
push r16 ; Save registers
push r17

in YH, SPH ; Get stack pointer
in YL, SPL
ldd r16, Y+7 ; Get parameter n

tst r16 ; base case: n=0
brne termialrecur
clr r0
rjmp termialreturn
termialrecur: ; recursive case
    mov r17, r16 ; Compute n-1
    dec r17
    push r17 ; Call termial(n-1)
    rcall termial
    pop r17
    add r0, r16 ; Add n to the result

termialreturn:
    ; Restore registers and Y pointer (in LIFO order)
    pop r17
    pop r16
    pop YL
    pop YH
    ret ; Return to caller

; END OF PROGRAM