Name: Lab Section:

CprE 288 Fall 2012 – Homework 10
Due Thu. Nov. 15 in the class

Notes:

- **Start early on homework.**
- Homework answers must be typed using a word editor. Hand in a hard copy in the class, before or at the end of the lecture.
- Late homework is accepted within three days from the due date. **E-mail late homework to both of the grading TAs, Min Sang Yoon (my222@iastate.edu) and Zhen Chen (zchen@iastate.edu)**. *Late penalty is 10% per day (counting from 10:45am of the due date if you are in the morning class or 2:10pm if you are in the afternoon class).*

Question 1: Read the following sections of the indicated documents located on the CPRE 288 course page and answer the following questions. Note: these documents will be useful throughout the homework [15 pts]

- Beginners Introduction to the Assembly Language of ATMEL-AVR-Microprocessors
  - Section 1 (1 page)
  - Section 2 (4 pages)
  - Section 5 (4 pages)
  - Section 6.7 (1 page)
  - Section 7 (3 pages)
- http://class.ee.iastate.edu/cpre288/resources/docs/doc2467.pdf (Atmega128 Datasheet)
  - Pages 9 – 13
- ATMega128 Starter Guide
  - Section 4.1 – 4.1.2.2
- AVR Assembler Guide
  - Section 4.4 "Instruction Mnemonics" (5 pages)
- http://class.ee.iastate.edu/cpre288/resources/docs/doc1234.pdf (Mixing C and Assembly Code)
  - Page 6
- Review lecture slides

a) Describe all registers of ATmega128 CPU (Note: CPU registers, not all registers in the ATmega128), including their name and usage. As for the status register, describe each bit [10 pts].

The ATmega128 CPU has 32 general purpose registers, R0-R31. They are 8-bit, and directly accessible by ALU. These registers could be used for accessing SRAM, for storing function parameters and for

Name: Lab Section:

instructions to execute operations on. R16-R31 are the only registers that can have immediate values load to them. Register pairs (R27,R26),(R29,R28),(R31,R30) can be used as 16-bit pointers, X, Y and Z.

Status register:

Bit 7-'I': Global interrupt enable, enable/disable interrupts to the CPU

Bit 6-'T': Bit copy storage, for moving a single bit between registers

Bit 5-'H': Half carry flag, to indicate a half carry in BCD arithmetic

Bit 4-'S': Sign bit, whether the result is negative or not with unsigned operands

Bit 3-'V': Two's complement overflow flag, whether an overflow happened or not with signed operands

Bit 2-'N': Negative flag, whether the result is negative or not with signed operands

Bit 1-'Z': Zero flag, whether the result is zero or not

Bit 0-'C': Carry flag, whether a carry is generated for unsigned operands

b) In the "Instruction Mnemonics" table, why is K defined to have a range of 0-255? (See end of table) [5 pts]

Question 2 (Status Register): What are the result and the values of the Z, C, H, N, V, S flags in the SREG after arithmetic operation a + b, assuming the following values of a and b? a and b are 8-bit signed integers (signed char in C). [10 pts]

a	b	a+b	Z	C	H	N	V	S
20 0b00010100	10 0b00001010	30 0b00011110	0	0	0	0	0	0
20 0b00010100	-20 0b11101100	0 0b00000000	0	0	1	0	0	0
-1 0b11111111	30 0b00011110	29 0b00011101	0	1	1	0	0	0
-10 0b11110110	-20 0b11101100	-30 0b11100010	0	1	1	1	0	1
-128 0b10000000	-1 0b11111111	127 0b01111111	0	1	0	0	1	1

Hint: Convert the values to 8-bit binary.

Name: Lab Section:

Note: H is 1 if there is a carry from adding the lower 4 bits.
Grading: It's not required to give those binary values.

Question 3 (Status Register): What are the result and the values of the Z, C, H, N, V, S flags in the SREG after arithmetic operation a - b, assuming the following values of a and b? a and b are 8-bit signed integers (signed char in C). [10 pts]

a	b	a-b	Z	C	H	N	V	S
20 0b00010100	10 0b00001010	10 0b00001010	0	0	1	0	0	0
20 0b00010100	-20 0b11101100	40 0x00101000	0	1	1	0	0	0
-1 0b11111111	30 0b00011110	-31 0x11100001	0	0	0	1	0	1
-10 0b11110110	-20 0b11101100	10 0x00001010	0	0	1	0	0	0
-128 0b10000000	-1 0b11111111	-127 0b10000001	0	1	1	1	0	1

Hint: Convert the values to 8-bit binary.
Note: H is 1 if there is a borrow into bit 3 (or if the lower 4 bits of b is greater than the lower 4 bits of a). C is 1 if there is a borrow into bit 7 (or the 8-bit absolute number of b is greater than the 8-bit absolute number of a).

Question 4 (Basic assembly practice): Based off of the instructions provided in the "AVR Assembler Guide" (Section 4.4) write a few lines of assembly (less than 5) to implement each the of the following tasks [10 pts].

a) R16 = R16 - 10 (Note: This can be done in 1 line of assembly) [1 pts]

```
subi    r16, 10
```

b) R17 = R16 + 20 (Note: This can be done in 2 lines of assembly) [3 pts]

```
ldi     r17, 20

add     r17, r16
```

Name: Lab Section:

c) R1 = R1 - 10 (Note: This has to be done in 2 lines of assembly.) [3 pts]

```
ldi     r16, 10
sub     r1, r16
```

Note: subi and ldi cannot be used with r1-r15.

d) value at memory address 0x0200 = 20 + value at memory address 0x0300 (Note: This can be done in 3 lines of assembly) [3 pts]

```
lds     r16, 0x0300
subi    r16, -20
sts     r16, 0x0200
```

Note: There is no addi in AVR assembly language. Use subi with negated immediate.

Question 5: AVR studio practice [15 pts]

Assume that you are writing the assembly code for an assembly function called "asm_func". It is called from the main function in the following C program file called "test-main.c":

```
#include <avr/io.h>
#include <stdio.h>

char ch1 = 0x30;
char ch2 = 0x40;
int a = 0x1010;

void asm_func();
```

Name: Lab Section:

```
int main()
{
      asm_func();
}
```

Function `asm_func()` should be in a separate assembly program file called "test-asm.S", whose template is as follows:

```
#include <avr/io.h>

.global asm_func

.extern ch1 ch2 a

asm_func:
      ;
      ; YOUR CODE SHOULD BE PUT HERE
      ;
      ret
```

Create a project and add the two files in AVR Studio 5. Test your assembly code on the AVR simulator with ATmega128 as the device. When you create the project, choose "AVR GCC" as the project type (if you choose AVR Assembly, the AVR assembler would be called, which is somewhat different from the GCC assembler).

Use complier optimization "-O0" (under menu "Project" => "Configuration Options") to help debug.

After you build the project, start "Debugging", open the watch window (from menu "view"), and add ch1, ch2, a, and b into the watch list. Watch how the values of those variables change to verify your assembly functions work.

Write a version of `asm_func()` to perform each of the following operations.

Name: Lab Section:

Grading: Note that students may write code in different ways but with the same, correct outcome.

a) Copy a 8-bit value [5 pts]

```
ch1 = ch2;
```

```
LDS     r16,   ch2
STS     ch1,   r16
```

b) Copy an 8-bit value into a 16-bit variable. The upper half of variable a should be cleared to zero, which can be done using the "CLR" instruction. [5 pts]

```
a = ch1;
```

```
LDS   r16, ch1
CLR   r17
STS   a, r16
STS   a+1, r17
```

c) Use the simulator environment to determine the address of the variables a, ch1, and ch2 (which are their locations in SRAM), and rewrite parts a) and b) using these addresses instead of using the C variable names. Verify in the Watch Window that when these addresses have a value stored to them that the corresponding variables are updated. [5 pts]

part a):

```
LDS   r16, 0x0100
STS   r17, 0x0101
```

Or

```
LDI   r26,  0x00
LDI   r27,  0x01        ; X register points to ch1
LD    r16,  X+          ; load ch1 to R16, X points to ch2
ST    X,    r16         ; store R16(ch1) to ch2
```

Part b)

```
LDS    r16, 0x0100
CLR    r17
```

Name: Lab Section:

```
      STS   0x0102, r16
      STS   0x0103, r17
```

Or

```
      LDI   r28,  0x00
      LDI   r29,  0x01        ; Y register points to ch1
      LD    r16,  Y           ; load ch1 to R16
      STD   Y+2,  r16         ; store R16 (ch1) to the low byte of a
      CLR   r17               ; the high byte of a =0
      STD   Y+3,  r17         ;  store 0 to the high byte of a
```

Grading: OK if the addresses of those variables are different.