Name: Lab Section:

CprE 288 Fall 2012 – Homework 12
This homework will NOT be graded

Notes:

- Start on the homework as soon as possible. In general, it's the best to do exercise right after the lecture.
- The solutions have been posted. Check the solutions **after** you solve the problems by yourself.

For programming exercises, points may be deducted (if the homework were graded) for following reasons:

- The program may cause compiler error
- The program will not produce the correct output.
- The program does not follow good programming style, including commenting, indentation and variable naming.
- The program is obviously more complex than necessary.

You may use any notation declared in avr/io.h.

Questions 1-6: 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>

int a = 0x1010;
int b = 0x1011;
int max;
unsigned c = 0x8000;
unsigned d = 0x7000;
char flag;
int X[8] = {1, 2, 3, 4, 8, 6, 7, 5};
int Y[8];

void asm_func();

int main()
{
  asm_func();

  // a loop that helps trace the debugger
  while (1)
```

Name: Lab Section:

```
    {
      count++;
    }
}
```

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 a b max c d flag X Y

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

You may want to test your code. 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" to help debug. After you build the project, start "Debugging", open the watch window, 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.

The `asm_func()` should perform equivalently as the following C code OR described functionality. All questions are independent with each other.

1. [6] If-else statement: get the maximum of two signed integers

```
if (a > b)
   max = a;
else
   max = b;
```

```
     ; a in r25:r24, b in r23:r22
asm_func:
     ; test for (b < a)
```

Name: Lab Section:

```
      LDS   r24, a              ; load a to r25/r24
      LDS   r25, a+1
      LDS   r22, b              ; load b to r23/r22
      LDS   r23, b+1
      CP    r22, r24            ; cmp b and a
      CPC   r23, r25
      BRGE  else                ; br if b>=a

; if-body: max = a
      STS   max, r24
      STS   max+1, r25
      RJMP  endif

; else-body: max = b
else: STS   max,      r22
      STS   max+1,    r23
endif:
```

2. [6] Compare an unsigned integer number with an immediate number.

```
   if(a <= 20)
   {
     flag = 0;
   }
   else
   {
     flag = 1;
   }
```

```
      ; a in r25/r24
asm_func:
      ; test (a < 21)
      LDS   r24,  a
      LDS   r25,  a+1
      CPI   r24,  21
      CPC   r25,  r1    ; AVR-GCC call convention: r1 holds 0
      BRGE  else

      ; if-body: flag = 0
      STS   flag, r1
```

Name: Lab Section:

```
      RJMP  endif

      ; else-body: flag = 1
else: LDI   r0,   1
      STS   flag, r0
endif:
```

3. [6] If statement with complex condition.

```
   if ((a < b) || (a > c))
     flag = 1;
```

```
      ; a in r25:r24, b in r23:r22, c in r21:r20
asm_func:
      ; test (a < b) || (c < a)
      LDS   r24,  a          ; load a
      LDS   r25,  a+1
      LDS   r22,  b          ; load b
      LDS   r23,  b+1
      CP    r24, r22         ; cmp a, b
      CPC   r25,  r23
      BRLT  if_body
      LDS   r20,  c          ; load c
      LDS   r21,  c+1
      CP    r20,  r24        ; cmp c, a
      CP    r21,  r25
      BRGE  endif

      ; if-body: flag = 1;
   if_body:
      LDI   r0,   1
      STS   flag, r0
```

4. [6] Copy the contents of the X[] array to the Y[] array. *You must use a loop*.

```
      ; &X[i] in X-reg, &Y[i] in Z-reg, X[i] in r24:r25, count in r22
asm_func:
      LDI   r26,  lo8(X)     ; set up X-reg
      LDI   r27,  hi8(X)
      LDI   r30,  lo8(Y)     ; set up Y-reg
```

Name: Lab Section:

```
      LDI   r31,  hi8(Y)
      LDI   r22,  8           ; count = 8
loop:
      LD    r24,  X+          ; load X[i]
      LD    r25,  X+
      ST    Z+,   r24         ; store to Y[i]
      ST    Z+,   r25
      DEC   r22               ; count--
      BRNE  loop
```

5. [6] Find out the maximum value among all elements in the X[] array, store the value into "max." *You must use a loop*.

```
; Z-reg holds &X[i]
; r24 is loop index
; r23:r22 is X[i]
; r21:r20 is max
ams_func
      LDI   r30, lo8(X)           ; set up Z-reg
      LDI    r31, hi8(X)
      LD     r20, Z+              ; max = X[0]
      LD     r21, Z+
      LDI    r24, 7               ; loop count is 7
loop:
      LD     r22, Z+              ; load another X[i]
      LD     r23, Z+
      CP     r20, r22             ; cmp max, new element
      CPC    r21, r23
      BRGE   else                 ; br if max is greater or equal
      MOVW   r20, r22         ; copy the new element to max
else:
      DEC    r24              ; reduce loop count
      BRNE   loop
```

6. [6] Start ADC, wait for conversion to finish, and store the value into a.

```
   while (ADCSRA & (1<<ADSC))
       ;
   a = ADCW;
```

```
asm_func:
```

Name: Lab Section:

```
      ; loop if (ADCSRA & (1<<ADSC)) is true
      ; the names are defined from <io.h> and can be used in Assembly
loop: LDI   r24,  ADCSRA
      ANDI  r24,  (1<<ADSC)
      BRNE  loop

      ; a = ADCW
      LDS   r24,  ADCW
      STS   a,    r24
      LDS   r24,  ADCW+1
      STS   a+1,  r24
```

Questions 7-10 (Function Calls): 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>

// Return the max of two integers
unsigned max(unsigned a, unsigned b);

// Copy the contents of array X into array Y.
// Both arrays have N elements.
void copyArray(int X[], int Y[], int N);

// Find out the maximum value of an array, return
// the value. The array has N elements
int maxOfArray(int X[], int N);

// Calculate Fibonacci number
unsigned int fibonacci (unsigned char n);

int main()
{
     // YOUR TESTING CODE
}
```

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 max
.global copyArray
.global maxOfArray
```

Name: Lab Section:

```
; YOUR ASSEMBLY CODE
```

You may want to test your code. 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).

Each question is independent. **Your code should be as efficient as possible.** After your testing, cut & paste the code for each function in the following space. *Note that the prototypes of those functions have been declared in the above code.*

7. [6] Which registers must be persevered by the assembly that implements an ATMega128 C function call. Why?

   All non-volatile (or callee-save) registers. The rules of function call convention states that the values of those registers must be preserved across a function call, i.e. the caller function can assumes their values won't change before and after a function call.

   If the callee function has to use any of those registers, it should push the register values into the stack at the beginning of the function execution, and pop them back into the same registers right before the function returns.

8. [6] The assembly code for function max().

```
; parameters: a in r25/r24, b in r23/r22
; return value in r25/24
max:      CP    r24, r22      ; cmp a, b
          CPC   r25, r23
          BRSH  endif         ; note it's unsigned type
          MOVW  r24, r22      ; result is b
endif:    RET
```

9. [6] The assembly code for function copyArray().

```
; parameter: X in r25:r24, Y in r23:r22, N in r21:r20
; Use Z-reg for &X[i], Y-reg for &Y[i]
copyArray:
          MOVW  r30, r24      ; Z-reg = X
```

Name: Lab Section:

```
          MOVW  r26, r22      ; X-reg = Y

loop:     LD    r24, X+       ; load *X
          LD    r25, X+
          ST    Y+,  r24      ; store to *Y
          ST    Y+,  r25
          SUBI  r20, 1        ; N--
          SBC   r21, r1       ; note r1 holds zero
          BRNE  loop
          RET
```

10. [6] The assembly code for function maxOfArray().

```
; parameters: X in r25/r24, N in r23/r22
; return value: r25/r24
; reg usage: Z-reg(r31/r30) for X, r25/r24 for max
;   r21/r20 for *X
maxOfArray:
          MOVW  r30, r24      ; Z-reg = X
          LD    r24, Z+       ; load X[0]
          LD    r24, Z+
          SUBI  r22, 1        ; N--
          SBC   r23, r1       ; note r1 holds zero

loop:     LD    r20, Z+       ; load *X
          LD    r21, Z+
          CP    r24, r20      ; cmp max, *X
          CPC   r25, r21
          BRGE  endif         ; skip if max>=*X

          MOVW  r24, r20      ; max = *X

endif:    SUBI  r22, 1        ; N--
          SBC   r22, r1
          BRNE  loop
          RET                 ; max value in r25/r24
```