CprE 288 Spring 2016 – Homework 4
Due Mon. Feb. 15 (on BlackBoard-midnight)

Notes:
- Homework is individual work. Adhere to the University’s policy relating to the integrity of scholarship. See http://catalog.iastate.edu/academiclife/regulations/, “Academic Dishonesty”.
- Homework must be typed and uploaded to BlackBoard as a PDF or Word Document (i.e. .doc or .docx) only.
- Late homework is accepted within three days from the due date. Late penalty is 10% per day (for each 24 hours following your class).

Note: Unless otherwise specified, all problems assume the ATMega128 is being used

Question 1 (10 pts): Binary operations, 2pts each

Write a single line of C code to implement each of the following. Assume the following declarations

```c
unsigned char ch;
unsigned int n;
```

Example: For ch if, bit 0 of is set

```c
if (ch & 0x01)
```

a. Set bits 5, 3, 0 of ch to 1, and preserve the remaining bits.

```c
ch = ch | 0010 1001;  // 0x29
```

b. For ch if, any of bits 6, 4, 2, 0 are Set to 1.

```c
if( ch & 0101 0101)  // 0x55
```

c. For n, Set bits 13, 12 to 1, Clear bits 3, 2 to 0, Toggle bits 11, 7, and preserve the remaining bits.

```c
n=((n | 0011 0000 0000 0000)& 1111 1111 1111 0011^0000 1000 1000 0000;
```

d. For n if, all bits 14, 12 are Set to 1, all bits 10, 9, 8 are Cleared to 0, any of bits 3, 2, 1, 0 are Set to 1, any of bits 15, 11 are Cleared to 0.

```c
if( (n & 0101 0000 0000 0000) == 0101 0000 0000 0000 && // Check ALL 1
    (~n & 0000 0111 0000 0000)== 0000 0111 0000 0000 && // Check ALL 0
    (n & 0000 0000 0000 1111) &&  // Check ANY 1
    (~n & 1000 1000 0000 0000) ) // Check ANY 0
```
e. For ch, rotate the bits to the left one position (bit 7 should end up at bit position 0), and then clear the upper 4 bits of the result, while preserve the remaining bits.

\[
ch = ((ch << 1) \mid (ch >> 7)) \& 0000\,1111; \quad // \,0x0F
\]

**Question 2 (10 pts): Code evaluation, 2 pts each**

For each C code fragment, give the final value of the indicated variable after the code fragment is run. If the final value cannot be determined, then give N/A.

a. Final value of ch is **0xA0** (in hex)

```c
char ch = 0x55;
ch = ch << 5;
```

b. Final value of int_ptr is **0xFC02** (in hex)

```c
int my_int = 0x55;  // Assume my_int is located at address 0xFC00
int *int_ptr;

my_int = my_int >> 3;
int_ptr = &my_int;
int_ptr++;  // points to an int so increases by 2
```

c. Final value of my_int is **0x0033** (in hex)

```c
int my_int = 0x55;  // Assume my_int is located at address 0xFFFE
int *int_ptr;

int_ptr = &my_int;
int_ptr++;  // point to an int so increase by 2, but addresses are
my_int++;   // only 16-bit in size so adding 2 sets int_ptr to 0

if(int_ptr){ // int_ptr == 0
   my_int = 0x44;
}
else {
   my_int = 0x33;
}
```

d. Final value of flag is **0** (in decimal)

```c
signed char ch = -128;
signed char flag = 0;

ch--;  // under flows the signed char, ch becomes 127

if(50 > ch){  // 50 is not > 127
   flag = 1;
}
```
e. Final value of \( ch \) is \( 5 \) (in decimal)

```c
char ch = 5;
char *ch_ptr = &ch;

ch_ptr = ch_ptr << 16; // ch_ptr is 16-bit, left shift of 16 will 0 out
if(ch_ptr > 0x7FC0){ // 0 is less than 0x7FC0
    ch = 1;
}
```

**Question 3 (10 pts): Coding practice**

Complete the following function, `max_consecutive_1s`, so that it returns the maximum number of consecutive 1’s in the variable that is passed to it.

For example, for \( 0x98FF5588 \) the max number of consecutive 1’s is 8.

Another example, for \( 0xF007FE05 \) the max number of consecutive 1’s is 10.

```c
char max_consecutive_1s(unsigned long x)
{
    int i = 0; // loop count
    char count = 0;  // temp max consecutive 1’s
    char max_count = 0;  // max consecutive 1’s

    // check each bit of x
    for(i=0; i < 64; i++)
    {
        if( (x >> i) & 0x1 ) // check if bit 0 is a 1 after shifting
        {
            count++;
        }
        else // reset max consecutive 1 count, since we detected a 0
        {
            count = 0;
        }

        // update max consecutive 1 count
        if(count > max_count)
        {
            max_count = count;
        }
    }

    return max_count;
}
```
Question 4: Pointers (10 pts)

Complete the table below (i.e. fill in the memory map) to show the state of memory after the following C fragment has been executed. Assume the ATmega128 platform, and all the variables are in the stack (and thus the storage is allocated from high address to low address). Remember the ATmega128 is a little endian architecture (i.e. least significant byte of a variable is stored in the lowest address).

Note: TYPO updated: num_ptr intended to be: int *, and p_ptr intended to be: int **

Be very careful with this question. (.5pts for each incorrect value)

typedef struct coord {
    char x;
    char y;
} coord_t;

coord_t  *coord_ptr;
int  *num_ptr;
int  **p_ptr = &num_ptr;
char a = 0x07;
coord_t my_coord[2];
int num_array[2]={1,4};

int main()
{
    coord_ptr = my_coord;
    num_ptr = num_array;
    my_coord[1].x = 0x33;
    coord_ptr++;
    coord_ptr->y = 0x44;
    num_ptr = num_ptr + 2;
    *num_ptr = 0x5040;
    p_ptr++;
    *p_ptr = 0xFEC0;
}
Question 5 (10 pts): I/O Ports, 5pts each. Note each part is independent.

a. Fill in the figure with arrows to indicate the direction (i.e. input or output) for each wire.

DDRA = 0xCC;  // 1100 1100
DDRB = 0xF3;  // 1111 0011
DDRC = 0xF0;  // 1111 0000
DDRD = 0x0F;  // 0000 1111
DDRE = 0x00;  // 0000 0000
b. Write C code that will read in button values, and send values to LEDs, based on the figure below. Make sure that your code first sets up the directions of the wires as shown in the figure. Add any variables that you think you may need.

There are 4 buttons (B0, B1, B2, B3), whose values should be stored in elements 0 to 3 of buttons (i.e. button B0 should be stored in button[0]). There are four LEDs (L0, L1, L2, L3) that can have a value of 1 or 0 written (i.e. sent) to them. All wires that do not have an LED or button connected should have their values persevered (i.e. you code should not change their value).

```c
void main(void)
{
    unsigned char buttons[4];

    // Set direction of Port wires
    DDRC = 0xBE;  // 1011 1110
    DDRD = 0x0F;  // 0000 1111

    // Read in Buttons into button array (assuming active high button)
    // fine if student assumes active low button also since not specified
    buttons[0] = (PINC >> 6) & 0x01;
    buttons[1] = (PINC >> 0) & 0x01;
    buttons[2] = (PIND >> 7) & 0x01;
    buttons[3] = (PIND >> 4) & 0x01;

    // Write LEDs (preserves all other bits)
    PORTC = (PINC & 0xEF) ^ 0x04;  // Clear bit 4, toggle bit 2
    PORTD = (PIND & 0xFB) | 0x01;  // Clear bit 2, Set bit 0
}
```