WESTERN
WASHINGTON UNIVERSITY

Computer Science Department

CSCI 247 Computer Systems I
Laboratory Exercise 3

Objectives

1) Get familiar with bit manipulation in C

Submitting Your Work

Submit your C program files as the Lab Exercise 3 Submission item on the course web site. You must submit your program by 11:59pm on the Friday following your scheduled lab session.

Lab Tasks

Note: After testing each function from your main program you can comment out the call in the main program (but leave the function as is) and move to the next function. This should allow you to test each function independently by passing arguments in the command line.

Task 1: Count the number of set bits in an integer

Implement a function with the following function signature:

```
int CountSetBits(unsigned int var);
```

This function should return the number of set bits in argument.

Call this function from your main routine with values gotten from stdin to confirm it is working correctly.

Task 2: Given an array of integers where all but one occurs in pairs, identify the unique element

Implement a function with the following function signature:

```
int IdentifyUniqueInteger(int arrayCount, int* intArray);
```

Call this function by passing the array as command line parameters to the main program.

Hint: Know that XOR operation has the following properties…

a^b = b^a
a^(b^c) = (a^b)^c
a^a = 0
a^0=a

Task 3: Reverse the Bits in an Integer

Implement a function with the following function signature:

```
int ReverseBits(int var);
```

This function should return an integer that has all the bits (including the sign bit) of the argument reversed.

Call this function by passing an integer as a command line parameters to the main program.

Task 4: Find out if only one bit is set in an Integer

Implement a function with the following function signature:

```
bool OnlyOneBitSet(int);
```

This function should return "true" if only one bit is set in the argument passed in.

Call this function by passing an integer as command line parameters to the main program.

Task 5: Find out if an integer with a single bit set has the set bit at an odd or even position

Implement a function with the following function signature:

```
bool OnlyOneBitSetInEvenPostion(int);
```

You can use your previous function to check if only one bit is set and then call this function to see if the bit is set in an odd or even position. If Even, return true, else return false.

Call this function by passing an integer as command line parameters to the main program.

Task 6: Perform Modulus operation when denominator only has 1 bit set without using "%"

Implement a function with the following function signature:

```
int ModWithoutUsingModOperator(int numerator, int denominator);
```

Call this function by passing the numerator and denominator as command line parameters to the main program.

Task 7: Swap nibbles within each byte of an integer

Implement a function with the following function signature:

```
int SwapNibbles(int);
```

Call this function by passing an integer as command line parameters to the main program.

Task 8: Counter Game

Given a preset counter that is greater than 1, you are allowed 2 types of operations:

1) If the counter is not a power of 2, reduce the counter by the largest power of 2 that is less than the counter value.
2) If the counter is a power of 2, reduce the counter by half.
3) Repeat this until counter equals 1

The problem is to find out how many operations are required to get the counter to "1".

Implement a function with the following function signature:

```
int NumberOfOperationsRequired(int);
```

Call this function by passing an integer as command line parameter to the main program.

Hints:

1) Write a function to check if an unsigned integer is a power of 2. [x && (x & (x-1)) == 0]
2) Read the value of the preset counter from the command line
3) While preset counter is greater than "1", if preset counter is power of 2 divide preset counter by 2, else subtract the largest power of 2 from it.
    a) Note that dividing by 2 is the same as a right shift
    b) Subtracting the largest power of 2 is the same as turning the most significant 1 in the preset counter to 0.
4) Count the number of times you have to go through the While loop until you reduce the preset Counter to 1.

Task 9: Counter Game Optimization

Try and optimize your algorithm in task eight above by studying what the bit operations are actually doing and see if you can reduce the algorithm to a single subtraction and a set bit count operation.

Hints:

1. If a number is a power of 2, it has “1” followed by a number of trailing zeros. The number of times you will end up doing the right shift operation will be a function of the number of trailing zeros. What happens to the trailing zeros, if you subtract “1” from the number?
2. If a number is not a power of 2, you end up setting the most significant “1” to a zero. So the number of “1’s” dictate how many times you have to perform the operation of setting the most significant “1” to a “0”.