University of Windsor
Faculty of Science
School of Computer Science

COMP-2650
Computer Architecture I: Digital Design
Winter 2022

Lab#	Date	Title	Due Date	Grade Release Date
Lab03	Week 03	**Number Systems (Complement)**	Feb. 09, 2022, Wednesday 4 AM EDT	Feb. 14, 2022

The objectives of the second lab will be for you to master the topics in number systems by implementing the algorithms with a programing language, herein, C/C++.

Step 1. Environment Setup

Our programing environment is the same as the previous lab (Lab02). In this lab, we want to extend our program to support larger binary numbers and more operations. For instance, we want to support binary numbers of *n* digits. Also, we want to calculate its 1's or 2's complement.

Step2. Writing Modular Programs

Now, let's open our previous program that accepts two Boolean values and returns the AND of them. We used the standard type `int` for our program and used 0 for `false` and 1 for `true` values.

```
01 #include <stdio.h>
02 int main(void) {
03
04     int x;
05     int y;
06     scanf("%d", &x);
07     scanf("%d", &y);
08     printf("%d AND %d is %d", x, y, x & y);
09     return 0;
10 }
```

As seen, this program can accept one **bi**nary digi**t**, (bit). We want to extend the program to accept binary numbers with more than a single digit. We know that C/C++ has arrays to store multiple items with the same data type. In C++, we can have arrays of `bool` like any other data type to store a binary number with more than one bit. In the following program, we create an array of 8 bits, also called Byte, as follows:

```
01 #include <stdio.h>
02 int main(void) {
03     setbuf(stdout, NULL);
04     bool x[8]; //Byte = 8 bits
05     printf("Enter a binary number:\n");
06     for(int i=0; i < MAX; i = i + 1){
07           scanf("%d", &x[i]);
08     }
09     printf("The binary number is:\n");
10     for(int i=0; i < MAX; i = i + 1){
11         printf("%d", x[i]);
12     }
13     return 0;
14 }
```

However, there is no Boolean data type in C. We can use an array of integer for our purpose, but we must check the integer digits are only 0 and 1. In the following program, we simulate a Byte by an array of integers. We accept 8 digits (assuming all digits are in {0,1}) and apply AND digitwise:

```
01 #include <stdio.h>
02 #define MAX 8//Byte = 8 bits
03 int main(void) {
04     setbuf(stdout, NULL);
05
```

```
06      int x[MAX];
07      int y[MAX];
08
09      printf("Enter the first binary number:\n");
10      for(int i=0; i < MAX; i = i + 1){
11          scanf("%d", &x[i]);
12      }
13      printf("Enter the second binary number:\n");
14      for(int i=0; i < MAX; i = i + 1){
15          scanf("%d", &y[i]);
16      }
17      int z[MAX];
18      for(int i=0; i < MAX; i = i + 1){
19          z[i] = x[i] & y[i];
20      }
21      printf("The first number AND second binary yield:\n");
22      for(int i=0; i < MAX; i = i + 1){
23          printf("%d", z[i]);
24      }
25
26      return 0;
27  }
```

The best practice in C is to write modular programs, that is, to implement any individual task in a function. For instance, for applying AND, we can write a function that accepts two arrays of binary digits in its argument. Unfortunately, in C, it is impossible to return an array (there is a workaround for this, though. We'll explain it in future assignments). So, we can put the result in another array in the third argument as follows:

```
01  #include <stdio.h>
02  #define MAX 8//Byte = 8 bits
03
04  void func_and(int a[], int b[], int result[]){
05      for(int i=0; i < MAX; i = i + 1){
06          result[i] = a[i] & b[i];
07      }
08  }
09  int main(void) {
10      setbuf(stdout, NULL);
11
12      int x[MAX];
13      int y[MAX];
14
15      printf("Enter the first binary number:\n");
16      for(int i=0; i < MAX; i = i + 1){
17          scanf("%d", &x[i]);
18      }
19      printf("Enter the second binary number:\n");
20      for(int i=0; i < MAX; i = i + 1){
21          scanf("%d", &y[i]);
22      }
23
24      int z[MAX];
25      func_and(x, y, z);
26      printf("The first number AND second binary yield:\n");
27      for(int i=0; i < MAX; i = i + 1){
28           printf("%d", z[i]);
29      }
30
31     return 0;
32  }
```

Lab Assignment

You should complete the above program that firstly outputs a menu of commands as follows:

```
Enter the command number:
0) Exit
1) AND
2) OR
3) NOT
4) 1's complement
5) 2's complement
6) 2's complement*
```

Based on the chosen number of commands by the user, the program should then ask for the input(s). For instance, if a user selects (1), the program should accept two inputs as follows:

```
Enter the first binary number:
x0 =
x1 =
...
x7 =

Enter the second binary number:
y0 =
y1 =
...
y7 =
```

When the user enters the two binary numbers, the program should apply the AND command on the input x and y and print the result and come back to the main menu. However, if the user selects (4), the program should ask for one input only:

```
Enter the binary number:
x0 =
x1 =
...
x7 =
```

When the user enters the binary number, the program then applies the 1's complement operation on the input and prints out the result and comes back to the main menu. If the user selects (0), the program ends. Please allow the user to enter inputs within the range {0,1} only. For instance, if the user enters 2, -1, …, print out an error message and come back to ask for correct inputs.

It is required to write a *modular* program according to the following instructions:

1. For each command in the menu, write a function and call the function to do the command:

```
void func_and(int a[], int b[], int result[]){}
void func_or(int a[], int b[], int result[]){}
void func_not(int a[], int result[]){}
void func_1s_comp(int a[], int result[]){}
void func_2s_comp(int a[], int result[]){}
void func_2s_comp_star(int a[], int result[]){}
```

2. To apply 1's complement, use the function for NOT (`func_not`).
3. To apply 2's complement, use the function for 1's complement (`func_1s_comp`).
4. There is another algorithm to compute the 2's complement of a binary number as follows:

- Start from the right to left until you see digit 1, then pass it and NOT the digits after that. For instance, 2's complement of 110100 is 001100. For the 2's complement* (`func_2s_comp_star`), use this algorithm.

Deliverables

You will prepare and submit the program in one single zip file `lab03_{UWinID}.zip` containing the following items:

1. The code file (`main.c` or `main.cpp`) and executable file (`main.exe` in windows or `main` in unix/mac)
2. The result of the four commands in the file `results.png/jpg`. Simply make a screenshot of the results.
3. [Optional and if necessary] A readme document in a txt file `readme.txt`. It explains how to build and run the program as well as any prerequisites that are needed. *Please note that if your program cannot be built and run on our computer systems, you will lose marks.*

(90%) `lab03_hfani.zip`

- (80%)
  - `main.c or main.cpp` => Must be compiled and built with no error!
  - `main.exe or main`
- (10%) `results.jpg/png`
- (Optional) `readme.txt`

(10%) `Files Naming and Formats`

Please follow the naming convention as you lose marks otherwise. Instead of UWinID, use your own UWindsor account name, e.g., mine is hfani@uwindsor.ca, so, `lab03_hfani.zip`