University of Windsor
Faculty of Science
School of Computer Science

COMP-2650
Computer Architecture I: Digital Design
Fall 2024

Lab#	Date	Title	Due Date	Grade Release Date
Lab04	Week 04	**Number Systems (Conversion)**	Oct. 08, Tuesday Midnight	Oct. 14

The lab's objectives will be to master the topics in number systems, esp., conversion, by implementing the algorithms with a programing language, herein, C/C++.

Step 1. Environment Setup

Our programing environment is the same as the first lab (Lab02). In this lab, we want to convert binary numbers to other bases. For instance, we want to convert binary numbers to octal or hexadecimal.

As we discussed in the lectures, if we increase the base in a number system, there would be fewer positions required to represent the same numbers. For instance, the number $(1111)_2$ is $(17)_8$ and $(F)_{16}$. As seen, although we are writing programs to work with binary numbers, it would be preferable for the user to input numbers faster using a smaller number of digits or seeing a shorter representation of numbers instead of seeing a long stream of 0 and 1. In C/C++, the octal and hexadecimal numbering systems are already available for printing output:

```
01 #include <stdio.h>
02 int main(void) {
03
04     setbuf(stdout, NULL);
05     int x;
06
07     printf("Enter an integer number:\n");
08     scanf("%d", &x);
09
10     printf("The number is: \n");
11     //printf("Binary: %b \n",x); There is no option for binary!
12     printf("Octal: %o \n",x);
13     printf("Decimal: %d \n",x);
14     printf("Hexadecimal: %x \n",x); //Alphabet in small letters
15     printf("HEXAdecimal: %X \n", x); //Alphabet in capital letters
16     return 0;
17 }
```

As shown in lines# 12 to 15, there are *format specifiers* that output the value of x in the base-{8,10,16}. Unfortunately, there is no format specifier for base-2 or binary. An example run would be:

```
Enter an integer number:
15
The number is:
Octal: 17
Decimal: 15
Hexadecimal: f
HEXAdecimal: F
```

Also, you can ask the user to input a value in octal, hexadecimal, or decimal in line# 7.

```
01 #include <stdio.h>
02 int main(void) {
03
04      setbuf(stdout, NULL);
05      int x;
06
07      printf("Enter an integer number:\n");
08      scanf("%x", &x);
09
10      printf("The number is: \n");
```

```
    //printf("Binary: %b \n",x); There is no option for binary!
    printf("Octal: %o \n",x);
    printf("Decimal: %d \n",x);
    printf("Hexadecimal: %x \n",x); //Alphabet in small letters
    printf("HEXAdecimal: %X", x); //Alphabet in capital letters
    return 0;
}
```

An example run would be:

```
Enter an integer number:
ff01
The number is:
Octal: 177401
Decimal: 65281
Hexadecimal: ff01
HEXAdecimal: FF01
```

Unfortunately, the format specifiers in C/C++ cannot be used for our program since we store the binary digits in an integer array. So, we must write the conversion functions for octal, hexadecimal, and decimal.

Step 2. Writing Modular Programs

Before adding new functionalities to our program, let's organize it better. In our previous lab (Lab03), all the functions for operations such as AND, OR, 1's complement, etc., were supposed to be in the same file as the `main()` function. As we add more functions, this file will become bigger and bigger and hard to maintain. So, it's better to put related functions in different files.

For instance, let's put all logical operations such as AND, OR, NOT in a separate file, named `logic.c`. Also, we can put all functions related to calculating complement in another file, named `complement.c`. Further, we will put all functions related to conversion in another file, named `conversion.c`.

logic.c

```
#define MAX 8//Byte = 8 bits
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[]){...}
```

complement.c

```
#include "logic.h"//Required as we use func_not for doing 1's comp!

#define MAX 8//Byte = 8 bits
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[]){...}
```

In C/C++, to call the functions in other files, we have to add *header* files in the main file of our program or any other files that use the functions (e.g., we added `logic.h` in `complement.c` as we used `func_not` for doing 1’s complement.) Let’s create the header files for each of our new files first:

logic.h

```
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[]);
```

complement.h

```
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[]);
```

As seen, header files contain only the signatures of the functions and *not* the bodies. Please look at the ‘;’

in the end of each function. Now, we are ready to add the headers to our main program and use the functions in each separate file:

```
#include <stdio.h>
#include "logic.h"
#include "complement.h"
#define MAX 8//Byte = 8 bits
int main(void) {
    setbuf(stdout, NULL);

    int x[MAX];
    int y[MAX];

    printf("Enter the first binary number:\n");
    for(int i=0; i < MAX; i = i + 1){
        scanf("%d", &x[i]);
    }
    printf("Enter the second binary number:\n");
    for(int i=0; i < MAX; i = i + 1){
        scanf("%d", &y[i]);
    }

    int z[MAX];
    //func_and(x, y, z);
    //func_not(x, z);
    func_1s_comp(x, z);
    printf("The first number AND second binary yield:\n");
    for(int i=0; i < MAX; i = i + 1){
         printf("%d", z[i]);
    }

   return 0;
}
```

Step 3. 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 user's chosen number of commands, 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 =
```

After that, the program asks to what base the user wants to see the results:

```
Enter the output base:
1) Binary
2) Octal
3) Decimal
4) Hexadecimal
```

Then the program applies the selected command (e.g., AND) on the input(s) and prints the result on the selected base and comes back to the main menu. If the user selects (0), the program ends. Please restrict the user to enter inputs within the range {0,1}. 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 the logic operations, create `logic.h` and `logic.c` and write functions to do AND, OR, NOT:
   ```
   logic.c
   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[]){...}
   ```

2. For the complement operations, create `complement.h` and `complement.c` and write functions to do 1's and 2's complements:
   ```
   Complement.c
   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[]){...} #look at lab03's manual
   ```

3. For the base conversion, create `conversion.h` and `conversion.c` and write functions in `conversion.c` to output the result according to the selected:
   ```
   conversion.c
   void to_octal(int a[]){...}
   void to_decimal(int a[]){...}
   void to_hexadecimal(int a[]){...}
   ```

For converting binary numbers to octal (base-8) or hexadecimal (base-16), you can use either the steps explained in the class or the fast method explained in the lecture assignment Lec02. For converting to decimal, you can use the sum of powers of 2, as described in the class.

Deliverables

Prepare and submit the program in one single zip file lab04_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/macOS)
2. The result of the four commands in the file results.pdf/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.

`lab04_hfani.zip`

- (10%) `logic.c, logic.h`
- (10%) `complement.c, complement.h`
- (30%) `conversion.c, conversion.h`
- (20%) `main.c` → Must be compiled and built with no error!
- (05%) `main.exe or main`
- (10%) `results.jpg/png` → Must match with the program output!
- (Optional) `readme.txt`

(10%) `Modular Programming (using separate header and source files)`
(5%) `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, lab04_hfani.zip