

**School of Computer Science**

**Faculty of Science**

**COMP-2650: Computer Architecture I: Digital Design**

**Winter 2021**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Lab# | Date | Title | Due Date | Grade Release Date |
| Lab04 | Week 04 | **Number Systems (Conversion)** | Feb. 02, 2021  Tuesday Midnight [AoE](https://www.timeanddate.com/time/zones/aoe)  Wednesday 7 AM EDT | Feb. 08, 2021 |

The third lab's objectives will be for you to master the topics in number systems, esp., conversion, and arithmetic, 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 extend our Lab03 to support binary numbers conversion to other bases and more arithmetic operations. For instance, we want to convert binary numbers to octal or hexadecimal. Also, we want to calculate the addition or subtraction of two binary systems.

As we discussed in the lectures, if we increase the base in a number system, there would be less position 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");

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", x); //Alphabet in capital letters

16 **return** 0;

17 }

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 have to write the conversion functions for octal, hexadecimal, and decimal.

**Step2. 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 to different files.

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

**logic.cpp**

**#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.cpp**

**#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++, in order 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.cpp as we used func\_not for doing 1’s complement). Let’s create the header files for each of our new file 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:

01 **#include** <stdio.h>

02 **#include** "logic.h"

03 **#include** "complement.h"

04 **#define** MAX 8//Byte = 8 bits

05 **int** **main**(**void**) {

06 **setbuf**(stdout, NULL);

07

08 **int** x[MAX];

09 **int** y[MAX];

10

11 **printf**("Enter the first binary number:\n");

12 **for**(**int** i=0; i < MAX; i = i + 1){

13 **scanf**("%d", &x[i]);

14 }

15 **printf**("Enter the second binary number:\n");

16 **for**(**int** i=0; i < MAX; i = i + 1){

17 **scanf**("%d", &y[i]);

18 }

19

20 **int** z[MAX];

21 //func\_and(x, y, z);

22 //func\_not(x, z);

23 func\_1s\_comp(x, z);

24 **printf**("The first number AND second binary yield:\n");

25 **for**(**int** i=0; i < MAX; i = i + 1){

26 **printf**("%d", z[i]);

27 }

28

29 **return** 0;

30}

**Lab Assignment**

You should complete the above program under the name of a project COMP2650\_Lab04\_UWinID that firstly outputs a menu of commands as follows:

Enter the command number:

1. Exit
2. AND
3. OR
4. NOT
5. 1’s complement
6. 2’s complement
7. 2’s complement\*

Based on the user's chosen number of commands, the program should then ask for the input(s). After that, the program asks to what base the user wants to see the results. Then, it applies the command and prints out the result in the requested base. 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 asks for a base number to print out the result:

Enter the output base:

1. Binary
2. Octal
3. Decimal
4. Hexadecimal

Then the program applies the AND command on the input x and y and prints the result on the selected base and comes back to the main menu. Other commands should follow the same flow. 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. Reorganized the previous functions in Lab03 in separate files as I did in this manual.
2. For the base conversion, create conversion.cpp and conversion.h.
3. Write functions in conversion.cpp to output the result according to the selected base by the user by calling the functions from main.cpp file:

**void** to\_octal(**int** a[]){}

**void** to\_decimal(**int** a[]){}

**void** to\_hexadecimal(**int** a[]){}

For converting binary numbers to octal or hexadecimal, 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**

You will prepare and submit the program in one single zip file COMP2650\_Lab04\_UWinID.zip containing the following two items:

1. The entire project folder COMP2650\_Lab04\_UWinID including the code file (main.c or main.cpp) and executable file (main.exe in windows or main in mac)

2. The result of the commands in the file COMP2650\_Lab04\_Results\_UWinID.pdf. Simply make a screenshots of the results and save (print) them into a single pdf.

1. [Optional and if necessary] A readme document in a txt file COMP2650\_Lab04\_ReadMe\_UWinID.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.*

In sum, your final COMP2650\_Lab04\_UWinID.zip file for the submission includes 1 folder (entire project folder), 1 image (results snapshot) and 1 txt (report). *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](mailto:hfani@uwindsor.ca), so,

COMP2650\_Lab04\_hfani.zip

* (40%) COMP2650\_Lab04\_hfani
  + logic.cpp
  + logic.h
  + complement.cpp
  + complement.h
  + conversion.cpp
  + conversion.h
  + main.c or main.cpp => Must be compiled and built with no error!
  + main.exe or main
* (10%) COMP2650\_Lab04\_Results\_hfani.pdf
* (Optional) COMP2650\_Lab04\_ReadMe\_hfani.txt

(30%) Modular Programming (using separate header and source files)

(10%) Files Naming and Formats

(10%) Folder Structure