Assignment 1: Computer Number Systems

Objectives

Write C programs that perform the followings:

1. Converts any **decimal** number into a **binary** number.
2. Converts any **binary** number into a **decimal** number.
3. Converts any **decimal** number into a **hexadecimal** number.
4. Converts any **hexadecimal** number into a **decimal** number.
5. Converts any **binary** number into a **hexadecimal** number.
6. Converts any **hexadecimal** number into a **binary** number.
7. Converts any **real** number into a **fixed point** number.
8. Converts any **fixed point** number into a **real** number.

Specifications

A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner. The value of any digit in a number can be determined by a digit, its position in the number, and the base of the number system. The numbers are represented in a unique manner and allow us to operate arithmetic operations like addition, subtraction, multiplication, and division.

This assignment consists of designing C programs to convert numbers from one number system to another number system. Although many number systems (decimal, binary, octal, hexadecimal, real, fixed point, and floating point) exist in the literature, this assignment focuses only on decimal, binary, hexadecimal, real, and fixed point numbers. A number can be either signed or unsigned. Unsigned numbers are always positive, while signed numbers are either positive or negative. In this assignment, we are focusing on the unsigned numbers only. Signed numbers will be dealt with the next assignment.

The decimal number system uses ten digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 and the base 10. The decimal number system is the system that we generally use to represent numbers in real life.

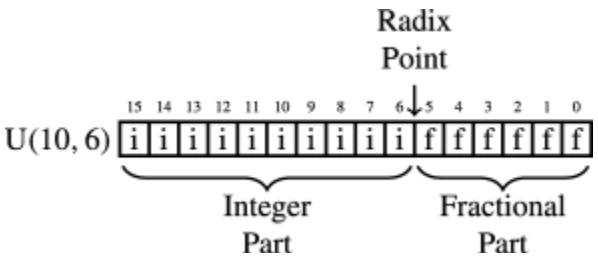
The binary number system uses only two digits: 0 and 1 and the base 2. Digits 0 and 1 are called bits. Binary numbers are easy to process and store in the computers and all computers present in the world now use binary numbers as its internal number system.

The hexadecimal number system uses sixteen digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F and the base 16. Here, A-F of the hexadecimal system means the numbers 10-15 of the decimal

number system respectively. Hexadecimal numbers are more human readable than binary numbers and easier to convert to and from binary numbers. In order to present binary numbers in human readable format computer literature often presents binary numbers after converting them into corresponding hexadecimal equivalents.

Any number that can be found in the real world is a real number. We find numbers everywhere around us. Natural numbers are used for counting objects, rational numbers are used for representing fractions, and irrational numbers are used for calculating the square root of a number, integers for measuring temperature, and so on. These different types of numbers make a collection of real numbers.

In computer systems, real numbers are stored as either fixed point or floating point binary numbers. As mentioned before, we are focussing on unsigned real numbers and unsigned fixed point numbers. Each unsigned fixed point binary number has two important parameters that describe it; the position of the radix point in relation to the most significant bit and the number of fractional bits. For example, a 16-bit fixed point number system can be represented as U(10,6) to indicate that there are six bits of precision in the fractional part of the number, and the radix point is ten bits to the right of the most significant bit stored. The layout for this number is shown graphically as:



In this assignment, we are using a 32-bit unsigned fixed point number system U(16,16) to store an unsigned real number, i.e., there are sixteen bits of precision in the fractional part of the number, and the radix point is sixteen bits to the right of the most significant bit stored.

Tasks

1. You will submit this assignment using **GIT submission system**. A central repository named '**assignment1**" has been created for this assignment.
2. Create your own fork of **assignment1** on the central GIT repository using following command.

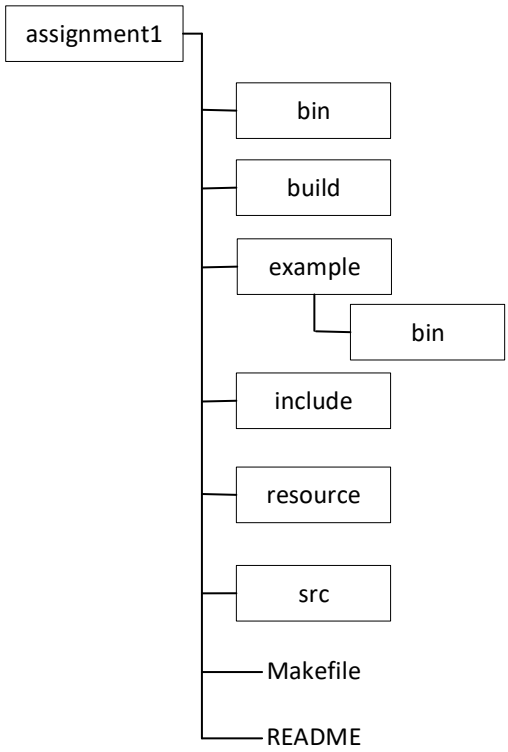
   ssh csci fork csci261/assignment1 csci261/$USER/assignment1

3. Create a folder named **csci261** in your home folder.

4. Go into your **csci261** folder that you have created in your home folder and create a clone of your forked **assignment1** repository using following command.

   git clone csci:csci261/$USER/assignment1

5. Repository **assignment1** has been organized as follows:



A **README** file template has been placed in the root of the application development folder. The README file gives a general idea of the application, technologies used in developing the application, how to build and install the application, how to use the application, list of contributors to the application, and what type of license is given to the users of the application. You will need to complete the **README** file before your final submission.

The **specifications** or **header** files (**convert.h** and **utility.h**) have been placed in **include** sub folder.

All **source codes** (**convert.c**, **utility.c**, and **main.c)** go into **src** sub folder**.** You need to implement the functions defined in the header file **utility.h** in your source code file **utility.c**. You need to implement the functions defined in the header file **convert.h** in your source code file **convert.c**. The **main.c** file in the **src** folder is complete, it uses the functions from your source code in **convert.c** and **utility.c.** You don't need to must not modify **main.c** file.

All **object** files will be placed in **build** sub folder and **executable** files in **bin** sub folder by **make**.

Data files are usually placed in **resource** folder. In this assignment there is no data file in resource folder.

A **Makefile** has been placed in the root of the application development folder. The **Makefile** does the followings and you don't need to and must not modify **Makefile**:

a) Defines and uses **macros** for each **GCC flag** that you are going to use to compile the codes and objects.
b) Defines and uses **macros** for each **sub folder** of the application, e.g., **src**, **include**, **resource**, **build**, and **bin**.
c) Uses GCC **debug flag** to facilitate debugging using **gdb**.
d) Uses GCC **include flag** to specify the path of application's custom header files so that the code does not need to specify relative path of these header files in **#include** pre-processor macro.
e) Creates individual object file (***.o**) into **build** folder from each source (***.c)** file of **src** folder.
f) Links the necessary object files from the **build** folder into a single executable file in **bin** folder.
g) Runs the main executable of the application from **bin** folder.
h) Cleans or removes files from both **build** and **bin** folders using **PHONY target** named **clean**.

6. One example executable (**convert**) of this application has been placed in the **example/bin** folder. You can run this example executable to get an idea what is expected from you in this assignment. To run the example executable type the following from the assignment root folder:

   make run-example

   This example executable has been built and tested in Linux Debian machines available in the labs. Run this executable in other kind of machines at your own risks. Make clean command will not delete this example executable and you should not delete it either.

7. Type following at the command prompt to clean previously built artefacts (binary and object files) of the application:

   make clean

8. Type following at the command prompt to build the application from your own source code:

   make

9. Type following at the command prompt to run your own application:

   make run

10. Make sure you can compile, link, run, and test this application error and warning free.

11. Complete the **README** file. Therefore, you need to give the general description of the application, technologies that are used in the application, how a user can build (compile and link) and install the application, how a user can run the application after the installation. Mention instructor's name and your name in the list of contributors. Give GPL license to the users to use the application. You can google to find README examples if you are not sure how to write one.

12. Organize and comment your code to make it easy to understand. Make sure you have typed **your name** and **student number** in the top comment section in each **.c** file. Make sure you have deleted all debugging print codes that you were using to debug your code during your development time but not necessary in the final code. Also, make sure you have deleted all commented out codes from your final submission.

13. Continue your work in your cloned or local **assignment1** repository and **commit** and **push** your work to your central **assignment1** repository as it progresses.

Deadline and Submission

The **deadline** to **demonstrate** this assignment **in the lab** is **January 24, 2022** for lab section **S22N01** and **January 25, 2022** for lab sections **S22N03** and **S22N04.** The assignment will be evaluated to zero without lab demonstration.

The **deadline** to **submit the code** of this assignment is **11:00 PM** on **January 25, 2022** for all sections. The assignment will be evaluated to zero without code submission.

Commit and **push** your work from your local repository to your remote repository regularly. Use following git commands to commit and push your work from your local repository to your remote repository from your project's root folder:

git add --all

git commit –am"Commit message"

git push origin master

Remember '**git add --all**' has double dashes before 'all'. You can also use '**git add .**' dot option instead of 'all' option. Both options do the same, add all the new and the modified files into the commit index. If you want to add only a specific file into the commit index, you can specify the relative path of the file instead of dot or 'all' option. For example, if you want to add only **prog.cpp** file of the src folder into the commit index, you can use '**git add src/prog.cpp**' command. You can also include multiple files or paths separated by space in the same '**git add**' command instead of using multiple 'git add' commands. Command '**git add**' is necessary before each '**git commit**' command. If you skip 'git add' command before a 'git commit' command, it does not perform the actual commit of the new and modified files into the repository. Always type a meaningful message in your '**git commit**' command's **'-m**' option. For example, if you are committing after adding all the necessary comments into your 'Makefile', use '**git commit –m"Added Makefile comments**"'. Remember there is a single dash before m in 'git commit' command. It is not recommended to make a huge commit instead of a number of small commits. It is also recommended to avoid unnecessary commits. Commits should reflect the check points of your software development milestones. Command 'git push' is necessary to push the local commit to the remote repository. If you skip '**git push**' after a '**git commit'**, your local and remote repository will become unsynchronized. You must keep your local and remote repositories synchronized with each other using '**git push**' after each '**git commit'**.

You will find most useful git commands in this **git cheat sheet** from GitLab. You will be allowed to commit and push until the deadline is over. Incremental and frequent commits and pushes are highly expected and recommended in this assignment.

Evaluation

Module	Functions	Marks
utility.c	*reverse*	**2**
	add_prefix	**2**
	extend_to_word_size	**3**
	are_binary_digits	**2**
	are_hex_digits	**2**
	are_decimal_digits	**2**
	is_binary	**2**
	is_hex	**2**
	is_decimal	**2**
	is_real_part	**2**
	is_fixed_point_part	**2**
	is_real	**2**

	is_fixed_point	**2**
	get_integer_part	**3**
	get_fraction_part	**3**
convert.c	*decimal_to_binary*	**2**
	binary_to_decimal	**2**
	decimal_to_hex	**3**
	hex_to_decimal	**4**
	nibble_to_hex_digit	**4**
	binary_to_hex	**4**
	hex_to_binary	**4**
	to_decimal	**3**
	to_binary	**3**
	to_hex	**3**
	integer_part_to_binary	**2**
	binary_to_integer_part	**2**
	fraction_part_to_binary	**4**
	binary_to_fraction_part	**4**
	to_fixed_point	**4**
	from_fixed_point	**4**
README		**05**
Code Quality and Comments		**10**
Total		**100**