Submitted by: Raghav Taneja
(Student #) 20058783

CISC-221 Computer Architecture
Data Representation Assignment 1: Manipulating Bits
Fall, 2020

1 Introduction

In studying computer architecture, we are often required to examine things at a much lower level that at the level of a high-level language. The purpose of this assignment is to become more familiar with bit-level representations and operations in the C programming language and with issues regarding how data is stored within a computer (including endianness). The lab also includes manual conversions of numeric data from one numeric base (radix) to another.

2 Learning Objectives and Component Skills

From CISC221 Syllabus:

1. Foundation Knowledge

I. Apply your knowledge and understanding of how data is represented, manipulated and stored in a computer system to create data structures that make the most effective use of a computer's resources.

By doing this lab, students will demonstrate their ability to:

1. manually trace the operational flow of a C program that includes conditional statements, function calls and looping structures.
2. identify and evaluate changes to variables during the execution of a C program.
3. apply bit-level transformations to C data structures, including radix conversions, masking, shifting and other transformations.
4. Evaluate the use of shifting operations and C pointers to manipulate data in memory in both big-endian and little-endian architectures.

3 References:

You will need a basic understanding of the structure and syntax of the C programming language in order to successfully complete this assignment. You will not be writing C code but will need to have a basic understanding of the flow and syntax of the C program that is provided.
For those of you that are new to C programming, there are numerous helpful links in the Extras folder, including a tutorial in both text and video format. You might also visit this site for specific tutorials and examples in both text and video formats:

https://www.tutorialspoint.com/cprogramming/index.htm

4 Logistics

You must do this assignment individually and submit your solution to the associated drop-box in OnQ under your own name, before the scheduled due date and time.

Clarifications and corrections will be posted to the course onQ site.

5 Handout Instructions

ANY COMPILATION OF CODE IN THIS COURSE SHOULD BE RUN ON THE RASPBERRY PI ALLOTTED TO YOUR TEAM (pi1.caslab.queensu.ca through pi30.caslab.queensu.ca). Use *putty* or some other terminal program to connect to these systems, either from the labs or from home.
Although there is no inherent requirement in this assignment to do so, the file *as1.c* referred to in this assignment is available online (without the comments) for you to compile and validate the operation of the program and your answers using gdb or some other form of automatically generating the answers. The as1.c file is available in the CASLAB domain, on your Ydrive in the cisc221/assignmentsF2020 folder. If you are outside of the CASLAB domain, log into your allotted team pi and cd to the ydrive folder at /cas/course/ydrive.

A word of warning: if you use gdb or some other method of validating the results for you, make sure you do so after you have manually come up with the answers yourself. You will be tested in the in-class exercises and in quizzes that you have acquired the skills that you are intended to develop in this lab. Remember – seeing a solution is a very poor substitute to coming up with it *yourself*! You may ultimately lose much more than you gain!

6 What to do:

Fill in the blanks to the questions scattered within the following as1.c file.

```
/*as1.  The purpose of this assignment is to familiarize you with bit-level operations in C, radix conversions, and endianness.
The conversion code in this lab mimics the way you would manually perform conversions.
You are to complete the comment sections in the program below.
Note that no built-in or library functions are called within the code. Conversions are done with simple arithmetic and logical operators. The constants referred to in the printranges() function are embedded in the limits.h library.
*/
#include <stdio.h>
#include <limits.h

/*function dec2bin converts an unsigned 32-bit decimal value to binary value using the manual procedure explained in the text on page 36 (adapted for base 2) and also on slide 10 of the archived lecture notes on Data Representation.
No built-in or library C conversion functions are used.
*/
```

```
unsigned dec2bin(unsigned char x){
	unsigned char d;
	unsigned  rslt, newrslt;
	int i = 0;;
	rslt = 0;
	d = x;
	while (d != 0){

		// Enter one value for each iteration of the loop in the below

				//d = 0xAF, 0x57, 0x2B, 0x15, 0xA, 0x5, 0x2, 0x1

		newrslt = d%2;

		//newrslt = 0x1, 0x1, 0x1, 0x1, 0x0, 0x1, 0x0, 0x1

		d = d/2;
		rslt = rslt | (newrslt << i);

		//  i  = 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7

		//rslt = 0x1, 0x3, 0x7, 0xF, 0xF, 0x2F, 0x2F, 0xAF

	 i++;
	}

	return rslt;
}

/*function bin2dec converts an unsigned 32-bit binary value to decimal by expanding the polynomial as
depicted on slide 5 of the archived lecture notes on Data Representation. No built-in or library C
conversion functions are used.
*/
unsigned bin2dec(unsigned x){
	unsigned rslt = 0;
	unsigned digitval = 0;
	int i;
	for (i=0;i<32;i++){
//for the I values shown, fill in the values of digitval and rslt in the spaces provided
	  digitval = x & (0x1 <<i);
	  rslt = rslt + digitval;

/*	   i=0; digitvalue = 0x1, rslt = 0x1

		i=1; digitvalue = 0x2, rslt = 0x3

		i=2; digitvalue = 0x4, rslt = 0x7
```

```
        i=3; digitvalue = 0x8, rslt = 0xF

        I think you'll get the idea without doing all 32 iterations!
*/
    }
    return rslt;
}

void printranges(){

/*this function prints the possible range of values (in decimal) for char, unsigned char, int, and unsigned
on the current machine.  Use the constant values in library limits.h
*/
    printf("Range of unsigned char is 0 to %u\n",UCHAR_MAX);
    printf("Range of  char is %d to %d\n",SCHAR_MIN,SCHAR_MAX);
    printf("Range of unsigned (int)is 0 to %u\n",UINT_MAX);
    printf("Range of int is %d to %d\n",INT_MIN,INT_MAX);
return;
}

/*Endianness
```

The table, below, represents the binary contents of four consecutive locations in memory in a **little-endian architecture**.

Memory Address	Memory Data
0000	1100 0011
0001	1010 1100
0002	0101 1100
0003	0110 1011

Treat the data stored at these addresses as:

(a) A single 32-bit (int) hexadecimal value: **6B5CACC3**

(b) Two unsigned 16-bit values. Express these values in hexadecimal.

ACC3	**6B5C**
0000,0001	0002,0003

(c) Four signed (two's complement) byte values. Express these values in **decimal**.

-61	**-84**	**92**	**107**
0000	0001	0002	0003

```
    */
```

```
/*function chkendian determines if the current machine has a little endian or a big endian architecture.
No built-in or library functions are used. The code stores a multi-byte value into memory and then
examines it byte-by-byte to determine endianness. The function should print out "I'm Big Endian" or
"I'm Little Endian" as appropriate before returning.
*/
void chkendian(){
      union
      {char Array[4];
       int Chars;
      }TestUnion;
      char c = 'a';
      int x;

/*why wouldn't the following work?
           int x = 0x12345678;  //set x to a recognizable value
           int y = x&0xff;    // read x back and check lower 8 bits
           if (y == 0x78){
                    printf("I'm Little Endian\n");
          }else{
                   printf("I'm Big Endian\n");
          }
          Answer: The conditional, if statement, compares the number values which are the same in Big
and Little Endian, therefore this fragment of code would always evaluate to true.
*/
/*What is the purpose of a union between a char array and an int? (What does it allow me to do?)

          Answer: The int value can be used to track elements in the char array, this allows for easier
reuse of memory. Additionally, this union allows storage of a number and letter, which gives the
ability to store hexadecimal numbers. */

      /*Test platform Endianness*/
      for(x=0;x<4;x++)
        TestUnion.Array[x] = c++;

/*what did the for loop above do?

Answer: The loop above creates an array with the following elements: a, b, c, d.*/

     if (TestUnion.Chars == "abcd")
           printf("I'm Big Endian\n");
      else
           printf("I'm Little Endian\n"

      return;
}
/*function le2be converts a little endian int and return it as a big endian int.
The code does this using shift and logical operations only.
No built-in or library conversion functions are used.*/
```

```
int le2be(int x){
      int d;
/*                ((x >>24) & 0x000000FF) =0x1
                  ((x >> 8) & 0x0000FF00) = 0x2300
                  ((x << 8) & 0x00FF0000) = 0x450000
                  ((x <<24) & 0xFF000000) = 0x67000000
*/
      d= ( ((x >>24) & 0x000000FF) | ((x >> 8) & 0x0000FF00) | \
           ((x << 8) & 0x00FF0000) | ((x <<24) & 0xFF000000) );

                  //d = 0x67452301
return d;
}

/*function le2bep converts a little endian int and return it as a big endian int.
The code does this using pointers. No built-in or library conversion functions are used.
Assume that the address of int d is 0x10000, and the address of int bed is 0x11000 for the purpose of
filling in the blanks*/
int le2bep(int x){
     int d = x;                          //d = 0x1234567
      char *dptr = &d;                   //dptr = 0x1C442C20
      int bed;
      char *bedptr = &bed;               //bedptr = 0x1C442C24
      *(bedptr + 3) = *dptr;             //*(bedptr + 3) = 0x67
     *(bedptr + 2) = *(dptr + 1);        //*(bedptr + 2) = 0x45
      *(bedptr + 1) = *(dptr + 2);       //*(bedptr + 1) = 0x23
      *bedptr = *(dptr + 3);             //  *bedptr = 0x1
      return bed;                        //bed = 0x67452301
}
int main(){
      unsigned char  h = 175;
      unsigned  i = 0x0000ffff;
      unsigned   hrslt;
      int irslt;
      hrslt = dec2bin(h);
      printf("calculated hex value of %d is: 0x%x\n",h,hrslt);
      printf("actual hex value of %d is: 0x%x\n", h,h);
      irslt = bin2dec(i);
      printf("calculated decimal value of 0x%x is: %d\n",i,irslt);
      printf("actual decimal value of 0x%x is: %d\n",i,i)
      printranges();
      chkendian();
      unsigned le = 0x01234567;
      printf("Using logical ops:\n");
      printf("Little Endian = 0x%x,Big Endian = 0x%x\n", le,le2be(le));
      printf("Using pointers:\n");
      printf("Little Endian = 0x%x,Big Endian = 0x%x\n",le,le2bep(le));
}
```

The program should print out the following:

```
calculated hex value of 175 is: 0xaf
actual hex value of 175 is: 0xaf
calculated decimal value of 0xffff is: 65535
actual decimal value of 0xffff is: 65535
Range of unsigned char is 0 to 255
Range of  char is -128 to 127
Range of unsigned (int)is 0 to 4294967295
Range of int is -2147483648 to 2147483647
I'm Little Endian
Using logical ops:
Little Endian = 0x1234567,Big Endian = 0x67452301
Using pointers:
Little Endian = 0x1234567,Big Endian = 0x67452301
```