Undergraduate Lab Report of Jinan University

Course Title Computer Organization Evaluation

Lab Name Lab 1: Manipulating Bits Instructor SUN Heng

Lab Address N116

Student Name TSUI, Hon Fai Student No 2018058445

College International School

Department Major CST

Date 12 / 10 / 2019 ~~Morning~~ / Afternoon

1. Introduction

1) Objective

The goal of the assignment is to become familiar with the basics of bit-level representations of integers numbers by solving a series of programming puzzles.

2) Logistics

Elementary functions are implemented only under a limited subset of operands, somewhat faithful to the electronic diode configuration.

2. Instructions

Start by copying bits.c to a directory on a Linux machine in which to plan to work.

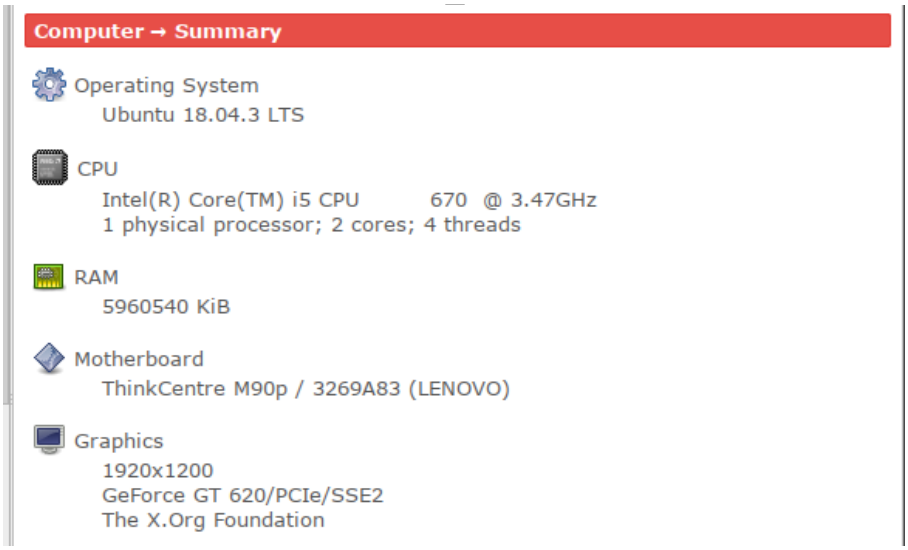
The bits.c file contains a skeleton for each of the 3 programming puzzles. The assignment is to complete each function skeleton using only straight line code for the integer puzzles (i.e., no loops or conditionals) and a limited number of C arithmetic and logical operators.

Specifically, only the following eight operators are allowed to use:

! ~ & ^ | + << >>

3. Lab Devices

Gcc 7.4 on Ubuntu 18.04 x86_64



4. Results

```
Enter x
1
Enter y
3
Enter n
3
x pow2plus4
6
x y BitAnd
1
getByte N from x
0
Negate x
-1
is x Positive?
TRUE
BitCount
1
x is ⩽ y?
TRUE
```

```
Enter x
1
Enter y
3
Enter n
0
x pow2plus4
6
x y BitAnd
1
getByte N from x
1
Negate x
-1
is x Positive?
TRUE
BitCount
1
x is ⩽ y?
TRUE
```

5. Appendix（Program Code）

```
#include <stdio.h>
#include <stdlib.h>

/* pow2plus4 - returns 2^x + 4, where 0 <= x <= 31
 *     Legal ops: + << */
int pow2plus4(int x) {
    return (1 << x) + 4; //1
}

/* bitAnd - x&y using only ~ and |
 *     Example: bitAnd(6, 5) = 4
 *     Legal ops: ~ | */
int bitAnd(int x, int y) {
    return ~(~x | ~y); //deMorgan's Law 2
}

/* getByte - Extract byte n from word x
 *     Bytes numbered from 0 (LSB) to 3 (MSB)
 *     Examples: getByte(0x12345678,1) = 0x56
 *     Legal ops: & << >> */
int getByte(int x, int n) {
    return (((0xFF << n * 8) & x) >> n * 8); //3
}

/* * negate - return -x
 *     Example: negate(1) = -1.
 *     Legal ops: ~ + */
int negate(int x) {
    return ~x + 1; //4
}

/* isPositive - return 1 if x > 0, return 0 otherwise
 *     Example: isPositive(-1) = 0.
 *     Legal ops: ! | >> */
int isPositive(int x) {
    return !(x >> 31); //5 -ve = 1000 0000 0000 0000
}

/* bitCount - returns count of number of 1's in word(2Byte in 32bit)
 *     Examples: bitCount(5) = 2, bitCount(7) = 3
 *     Legal ops: ! ~ & ^ | + << >> */
int bitCount(int x) {
    int mask = 0x11111111; //255
    int sum = x & mask;
    sum += x >> 1 & mask;
    sum += x >> 2 & mask; //calculating number of 1's in each group
    sum += x >> 3 & mask;
    sum = sum + (sum >> 16);
 //now combine high and low order bytes; now, low order 16bits consists of 4 sums, each
between between 0 and 8*1
    sum = ((sum & 0xF0F) + ((sum >> 4) & 0xF0E));
    return (sum + (sum >> 8)) & 0x3f; //remain the last 6 bits
```

```
/* the reason to remain last 6 bits(using mask 0x3f) instead of the last 4 bit only is
because if a word has maximum amount of 1 ie. 32, 2^4=16 cannot hold it, 2^5=32 but
can only hold any int ranging from 0-31.   Therefore, '6 bit' is the minumum amount of
bits required to obtain correct result. */
}

/* isLessOrEqual - if x <= y   then return 1, else return 0
 *    Example: isLessOrEqual(4,5) = 1.
 *    Legal ops: ! ~ & ^ | + << >> */
int isLessOrEqual(int x, int y) {
    int x_sign = x >> 31, y_sign = y >> 31;
    return !(((!x_sign) & y_sign) | ((!(x_sign ^ y_sign)) & (y + ~x + 1) >> 31)); //7
    //    sign bits are different?1:0 or [ sign bit are the same? and (y-x) is positive? ]
}


int main() {
    int x, y, n;

    puts("Enter x");
    scanf("%d", &x);
    puts("Enter y");
    scanf("%d", &y);
    puts("Enter n");
    scanf("%d", &n);

    puts("x pow2plus4");
    printf("%d\n", pow2plus4(x));

    puts("x y BitAnd");
    printf("%d\n", bitAnd(x, y));

    puts("getByte N from x");
    printf("%d\n", getByte(x, n));

    puts("Negate x");
    printf("%d\n", negate(x)); //4

    puts("is x Positive?");
    printf("%s\n", isPositive(x)?"TRUE":"FALSE");

    puts("BitCount");
    printf("%d\n", bitCount(x));

    puts("x is <= y?");
    printf("%s\n", isLessOrEqual(x, y)?"TRUE":"FALSE");

    return 0;
}
```