Lab 1 Report
ZedBoard Linux Continued

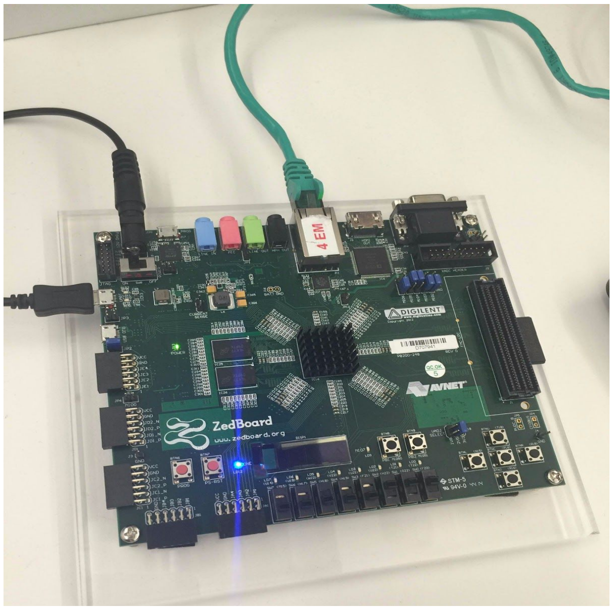
Hao Jiang
Shiyu Wang

14 Sept 2016

Introduction

This lab is a succession of lab 0 as an introductory lesson of ZedBoard Linux. Moreover, we practiced writing more sophisticated c programs in this class, and learnt how the compiler managed different data formats. We first log into our computer with myneu credentials and then connected ZedBoard to our computer. We then connect the ZedBoard to the PC, and make an SSH connection to the ZedBoard using appropriate IP address and port. We compiled IntegerMath.c which we wrote as pre-lab, and also wrote two other c codes in class. In the process, we explored the compiler option -s, and printed out values in different data formats. At last, we explored pointers by adding comments to every line in the last code.

Connecting to ZedBoard:

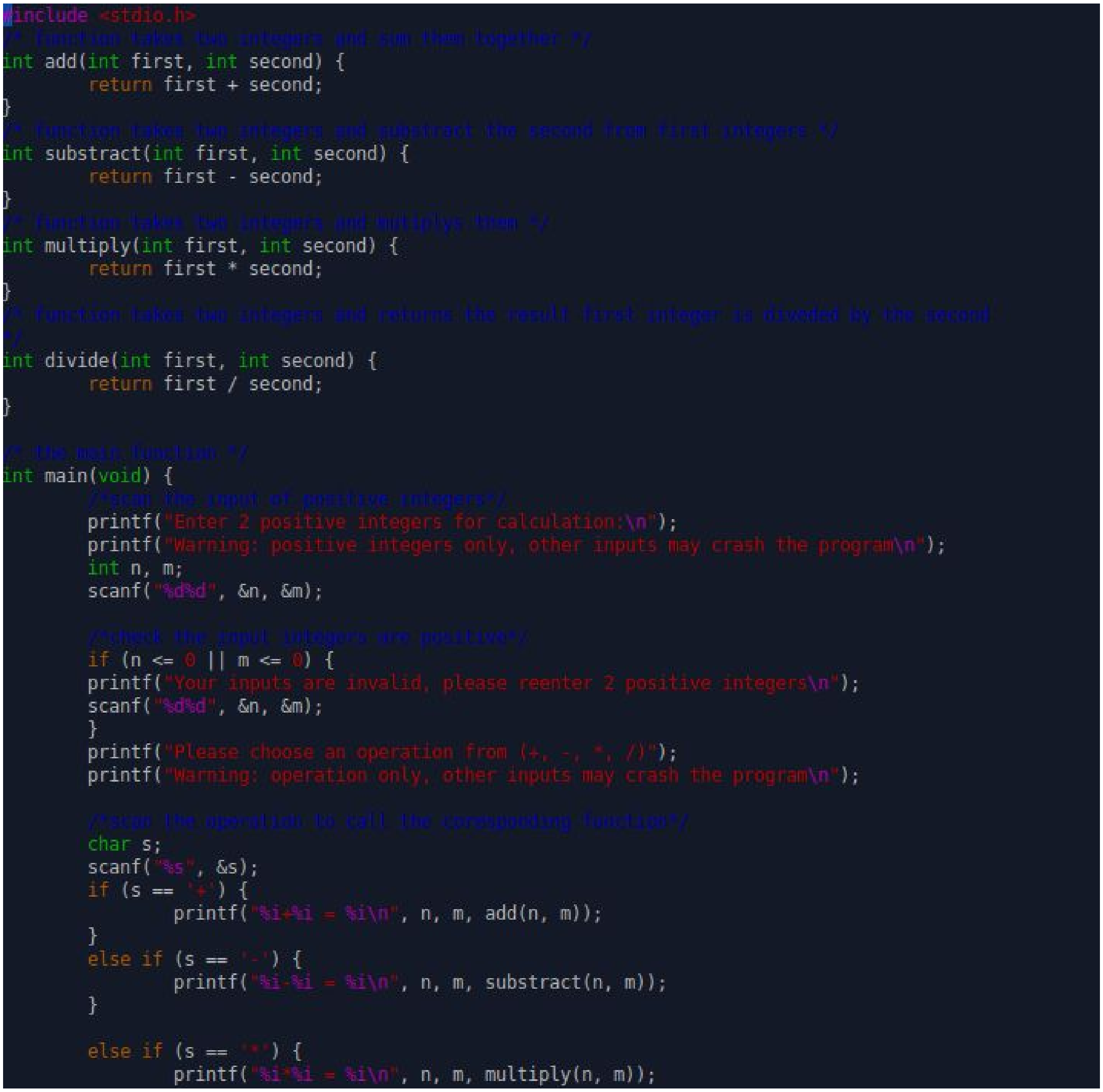


We first logged into the Windows desktop PC using myneu credentials. Then we powered the ZedBoard. The ZedBoard we connected to our computer host via Ethernet to the RJ45 connector when the light was on. We logged into MobaXterm, and made an SSH connection to the ZedBoard. The IP address is 192.168.1.10, our username is user 550 and port number is 22.

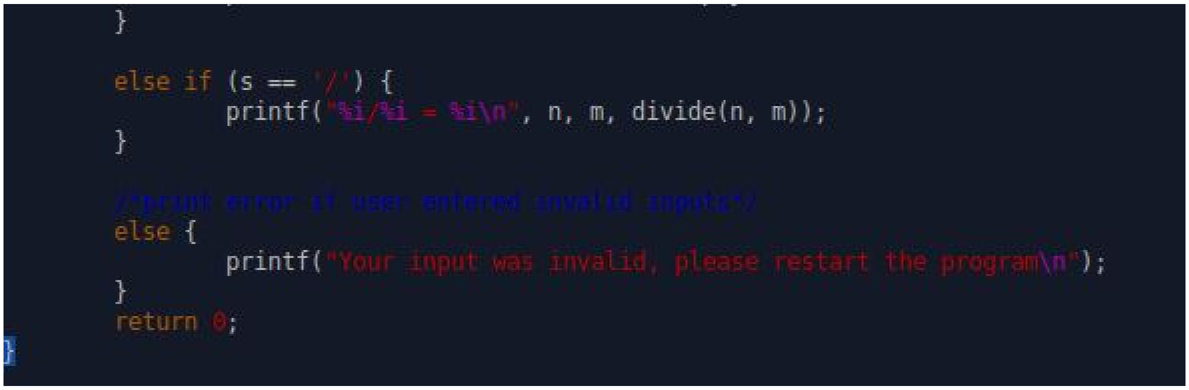
Compile IntergerMath on ZedBoard

This c program was written as a pre-lab, but was not tested until we attend this class. This c file is listed in Appendix. We tried one code first. We used an SFTP connection to securely transfer the files from the host to the ZedBoard. We then typed “gcc IntergerMath.c” to compile it, but the first one we tested proved to have a lot of errors. We transferred the other one instead which had less errors. After we fixed these errors, we run it by typing “./a.out”

IntegerMath.c:

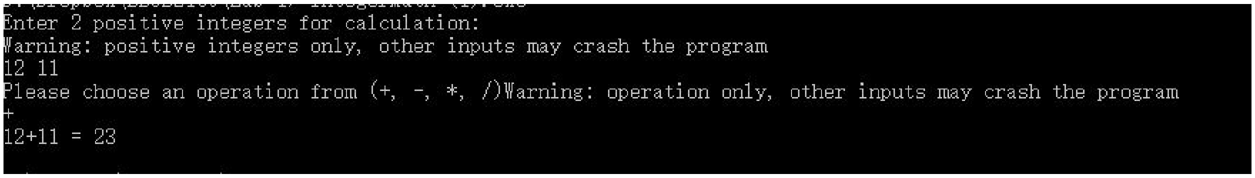


```c
#include <stdio.h>
/* function takes two integers and sum them together */
int add(int first, int second) {
        return first + second;
}
/* function takes two integers and substract the second from first integers */
int substract(int first, int second) {
        return first - second;
}
/* function takes two integers and multiplys them */
int multiply(int first, int second) {
        return first * second;
}
/* function takes two integers and returns the result first integer is diveded by the second
*/
int divide(int first, int second) {
        return first / second;
}

/* the main function */
int main(void) {
        /*scan the input of positive integers*/
        printf("Enter 2 positive integers for calculation:\n");
        printf("Warning: positive integers only, other inputs may crash the program\n");
        int n, m;
        scanf("%d%d", &n, &m);

        /*check the input integers are positive*/
        if (n <= 0 || m <= 0) {
        printf("Your inputs are invalid, please reenter 2 positive integers\n");
        scanf("%d%d", &n, &m);
        }
        printf("Please choose an operation from (+, -, *, /)");
        printf("Warning: operation only, other inputs may crash the program\n");

        /*scan the operation to call the corresponding function*/
        char s;
        scanf("%s", &s);
        if (s == '+') {
                printf("%i+%i = %i\n", n, m, add(n, m));
        }
        else if (s == '-') {
                printf("%i-%i = %i\n", n, m, substract(n, m));
        }

        else if (s == '*') {
                printf("%i*%i = %i\n", n, m, multiply(n, m));
```



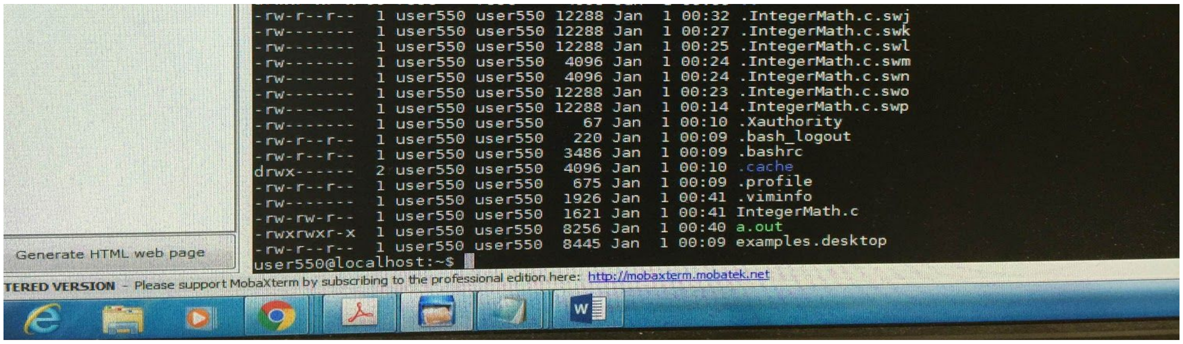
```
        }

        else if (s == '/') {
                printf("%i/%i = %i\n", n, m, divide(n, m));
        }

        /*print error if user entered invalid inputs*/
        else {
                printf("Your input was invalid, please restart the program\n");
        }
        return 0;
}
```

Output:



```
Enter 2 positive integers for calculation:
Warning: positive integers only, other inputs may crash the program
12 11
Please choose an operation from (+, -, *, /)Warning: operation only, other inputs may crash the program
+
12+11 = 23
```

In order to know permissions of the files, we typed the command: “ ls -la ” where l stands for “long listing” and a stands for “all”. Details are shown as in the picture:



```
-rw-r--r--  1 user550 user550 12288 Jan  1 00:32 .IntegerMath.c.swj
-rw-------  1 user550 user550 12288 Jan  1 00:27 .IntegerMath.c.swk
-rw-------  1 user550 user550 12288 Jan  1 00:25 .IntegerMath.c.swl
-rw-------  1 user550 user550  4096 Jan  1 00:24 .IntegerMath.c.swm
-rw-------  1 user550 user550  4096 Jan  1 00:24 .IntegerMath.c.swn
-rw-------  1 user550 user550 12288 Jan  1 00:23 .IntegerMath.c.swo
-rw-------  1 user550 user550 12288 Jan  1 00:14 .IntegerMath.c.swp
-rw-------  1 user550 user550    67 Jan  1 00:10 .Xauthority
-rw-r--r--  1 user550 user550   220 Jan  1 00:09 .bash_logout
-rw-r--r--  1 user550 user550  3486 Jan  1 00:09 .bashrc
drwx------  2 user550 user550  4096 Jan  1 00:10 .cache
-rw-r--r--  1 user550 user550   675 Jan  1 00:09 .profile
-rw-------  1 user550 user550  1926 Jan  1 00:41 .viminfo
-rw-rw-r--  1 user550 user550  1621 Jan  1 00:41 IntegerMath.c
-rwxrwxr-x  1 user550 user550  8256 Jan  1 00:40 a.out
-rw-r--r--  1 user550 user550  8445 Jan  1 00:09 examples.desktop
user550@localhost:~$
```

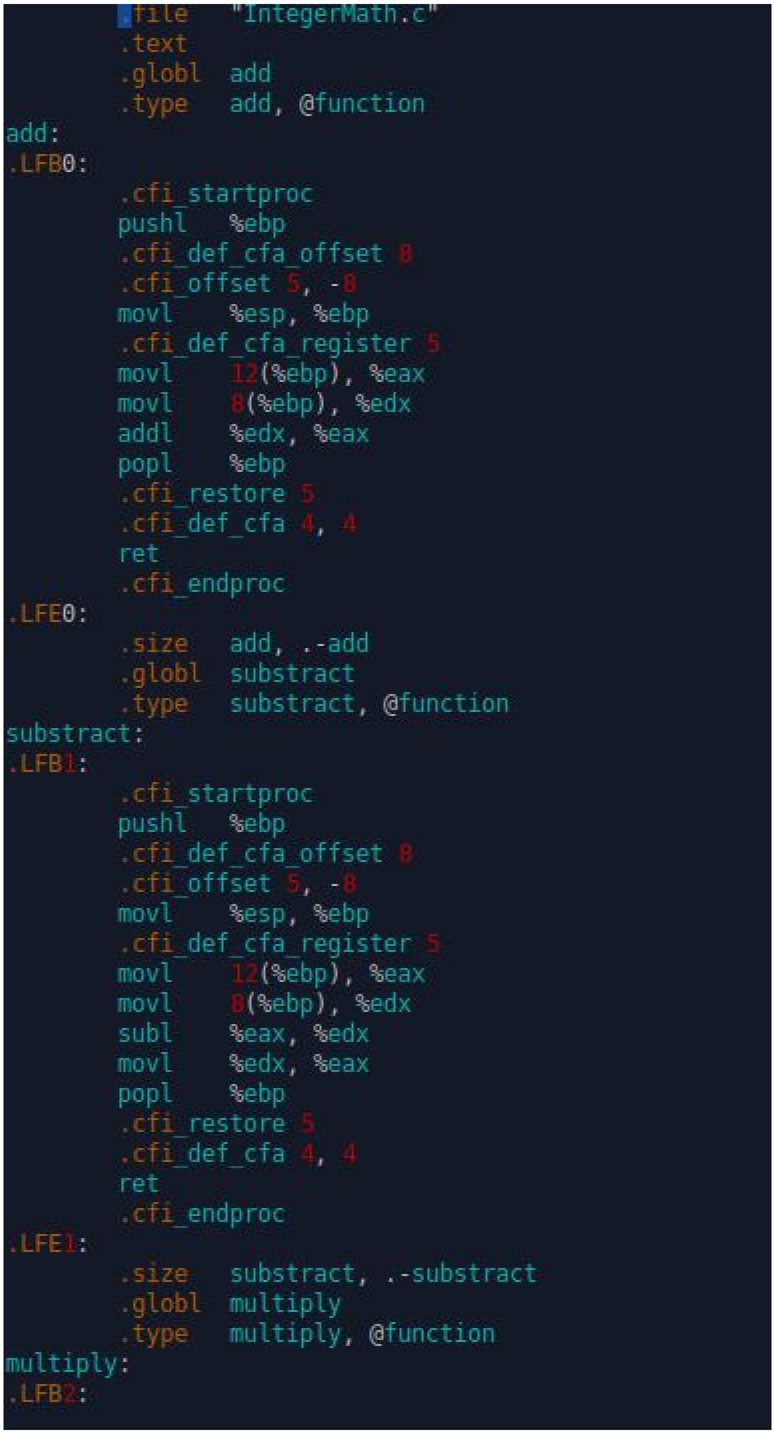
Generate HTML web page
TERED VERSION - Please support MobaXterm by subscribing to the professional edition here: http://mobaxterm.mobatek.net

The first 10 digits stand for permissions associated with the source and executable files where r stands for read permission, w stands for write permission and x means execution permission. Size of each file is that after the second “user550” for each file.

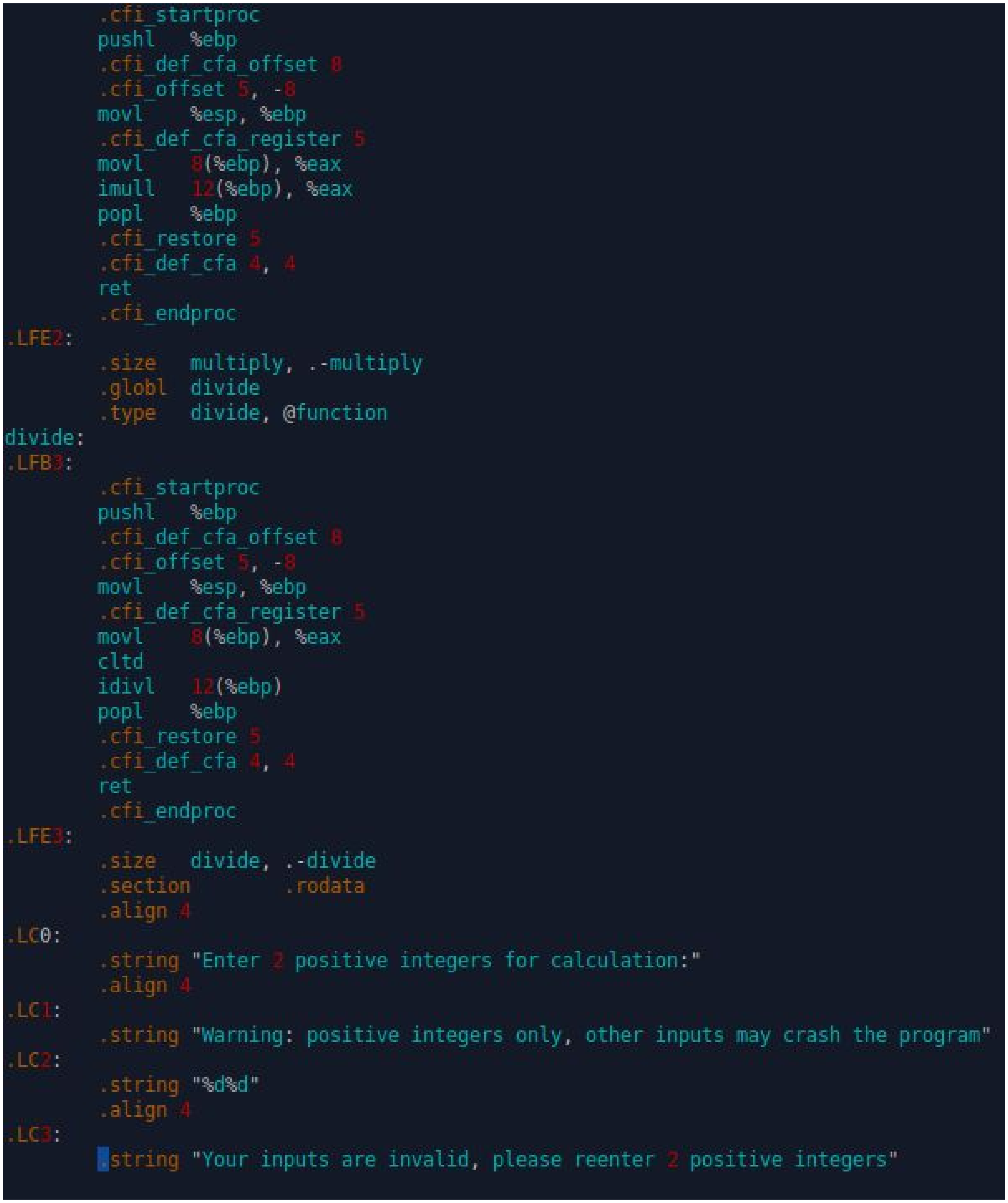
Explore Compiler Option -S

We compiled the file again but this time with a -s as in “gcc -s IntergerMath.c”. However, we found it gave no difference from “gcc IntergerMath.c”. Then we found it is case sensitive, so we compiled it with “gcc -S IntergerMath.c”. This time, after we “ls” this file a line appear as “ IntergerMath.c   IntergerMath.s   a.out   examples. Desktop”. We didn’t have the part: “ IntergerMath.s” last time we did “gcc IntergerMath.c”. We opened the IntegerMath.s file, and found it is the assembly code of the IntegerMath.

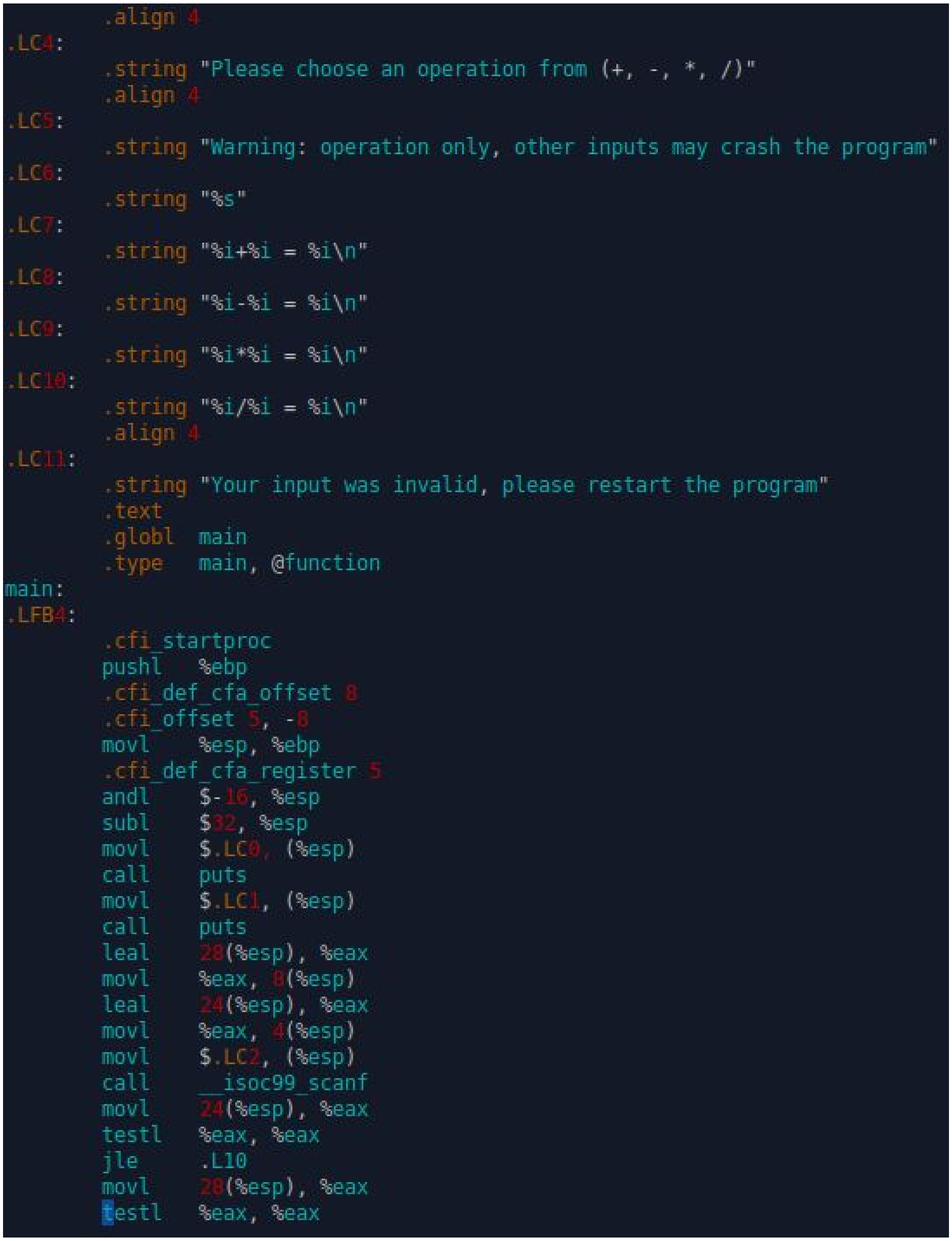
IntegerMath.s:



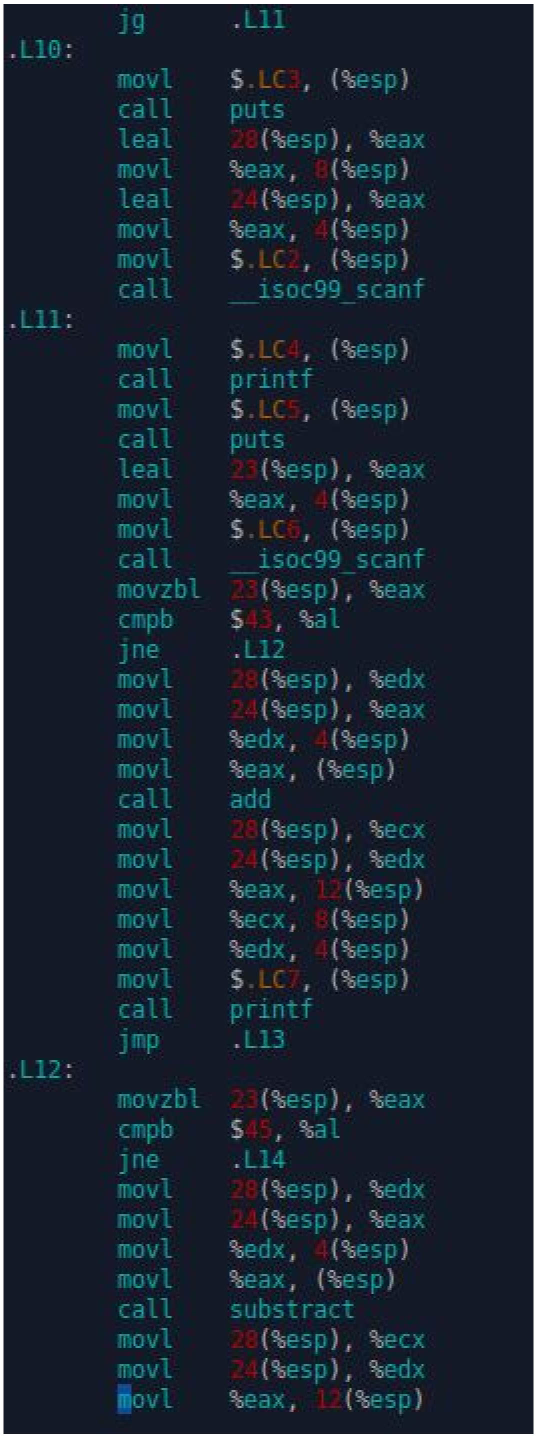
```
        .file   "IntegerMath.c"
        .text
        .globl  add
        .type   add, @function
add:
.LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        movl    12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   add, .-add
        .globl  substract
        .type   substract, @function
substract:
.LFB1:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        movl    12(%ebp), %eax
        movl    8(%ebp), %edx
        subl    %eax, %edx
        movl    %edx, %eax
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE1:
        .size   substract, .-substract
        .globl  multiply
        .type   multiply, @function
multiply:
.LFB2:
```



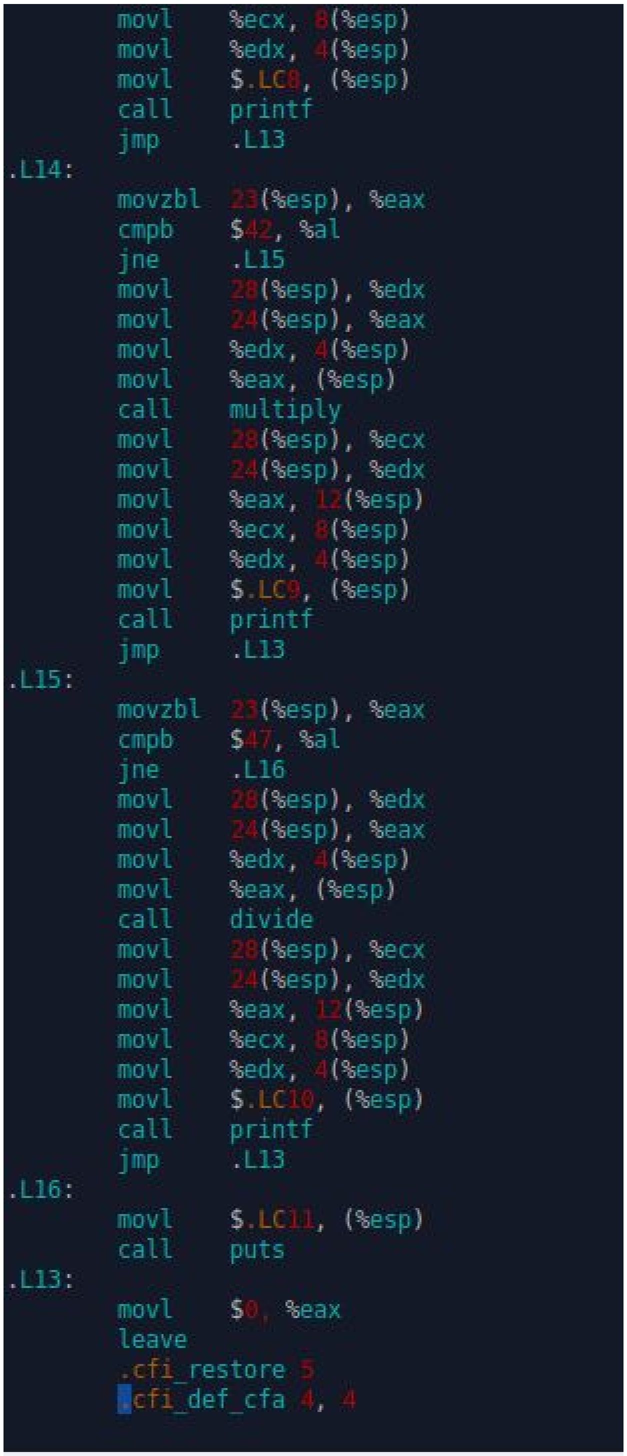
```
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        movl    8(%ebp), %eax
        imull   12(%ebp), %eax
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE2:
        .size   multiply, .-multiply
        .globl  divide
        .type   divide, @function
divide:
.LFB3:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        movl    8(%ebp), %eax
        cltd
        idivl   12(%ebp)
        popl    %ebp
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE3:
        .size   divide, .-divide
        .section        .rodata
        .align 4
.LC0:
        .string "Enter 2 positive integers for calculation:"
        .align 4
.LC1:
        .string "Warning: positive integers only, other inputs may crash the program"
.LC2:
        .string "%d%d"
        .align 4
.LC3:
        .string "Your inputs are invalid, please reenter 2 positive integers"
```



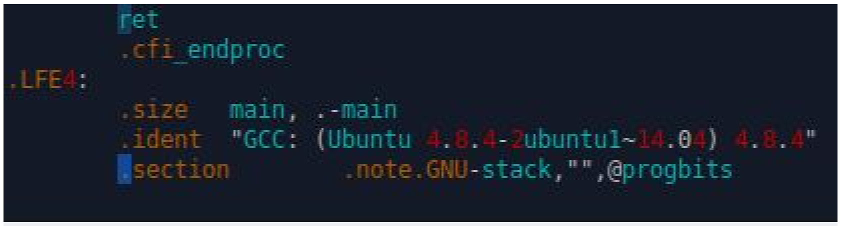
```
        .align 4
.LC4:
        .string "Please choose an operation from (+, -, *, /)"
        .align 4
.LC5:
        .string "Warning: operation only, other inputs may crash the program"
.LC6:
        .string "%s"
.LC7:
        .string "%i+%i = %i\n"
.LC8:
        .string "%i-%i = %i\n"
.LC9:
        .string "%i*%i = %i\n"
.LC10:
        .string "%i/%i = %i\n"
        .align 4
.LC11:
        .string "Your input was invalid, please restart the program"
        .text
        .globl  main
        .type   main, @function
main:
.LFB4:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
        subl    $32, %esp
        movl    $.LC0, (%esp)
        call    puts
        movl    $.LC1, (%esp)
        call    puts
        leal    28(%esp), %eax
        movl    %eax, 8(%esp)
        leal    24(%esp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC2, (%esp)
        call    __isoc99_scanf
        movl    24(%esp), %eax
        testl   %eax, %eax
        jle     .L10
        movl    28(%esp), %eax
        testl   %eax, %eax
```



```
        jg      .L11
.L10:
        movl    $.LC3, (%esp)
        call    puts
        leal    28(%esp), %eax
        movl    %eax, 8(%esp)
        leal    24(%esp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC2, (%esp)
        call    __isoc99_scanf
.L11:
        movl    $.LC4, (%esp)
        call    printf
        movl    $.LC5, (%esp)
        call    puts
        leal    23(%esp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC6, (%esp)
        call    __isoc99_scanf
        movzbl  23(%esp), %eax
        cmpb    $43, %al
        jne     .L12
        movl    28(%esp), %edx
        movl    24(%esp), %eax
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    add
        movl    28(%esp), %ecx
        movl    24(%esp), %edx
        movl    %eax, 12(%esp)
        movl    %ecx, 8(%esp)
        movl    %edx, 4(%esp)
        movl    $.LC7, (%esp)
        call    printf
        jmp     .L13
.L12:
        movzbl  23(%esp), %eax
        cmpb    $45, %al
        jne     .L14
        movl    28(%esp), %edx
        movl    24(%esp), %eax
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    substract
        movl    28(%esp), %ecx
        movl    24(%esp), %edx
        movl    %eax, 12(%esp)
```



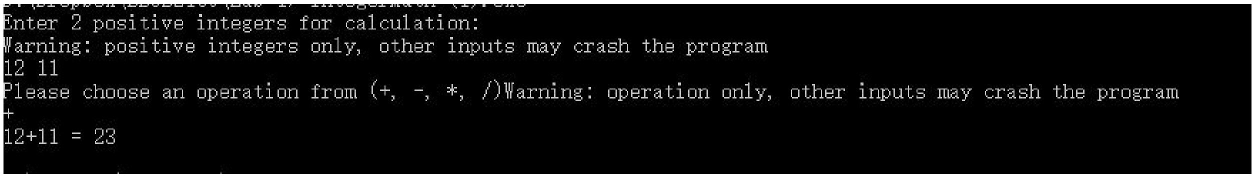
```
        movl    %ecx, 8(%esp)
        movl    %edx, 4(%esp)
        movl    $.LC8, (%esp)
        call    printf
        jmp     .L13
.L14:
        movzbl  23(%esp), %eax
        cmpb    $42, %al
        jne     .L15
        movl    28(%esp), %edx
        movl    24(%esp), %eax
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    multiply
        movl    28(%esp), %ecx
        movl    24(%esp), %edx
        movl    %eax, 12(%esp)
        movl    %ecx, 8(%esp)
        movl    %edx, 4(%esp)
        movl    $.LC9, (%esp)
        call    printf
        jmp     .L13
.L15:
        movzbl  23(%esp), %eax
        cmpb    $47, %al
        jne     .L16
        movl    28(%esp), %edx
        movl    24(%esp), %eax
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    divide
        movl    28(%esp), %ecx
        movl    24(%esp), %edx
        movl    %eax, 12(%esp)
        movl    %ecx, 8(%esp)
        movl    %edx, 4(%esp)
        movl    $.LC10, (%esp)
        call    printf
        jmp     .L13
.L16:
        movl    $.LC11, (%esp)
        call    puts
.L13:
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
```



```
        ret
        .cfi_endproc
.LFE4:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04) 4.8.4"
        .section        .note.GNU-stack,"",@progbits
```

Then we compiled the Integermath.s using command “gcc -c Integer.s” and and run the ./a.out. We found the IntegerMath.s gave the same result of IntegerMath.c.

<u>Output</u>:

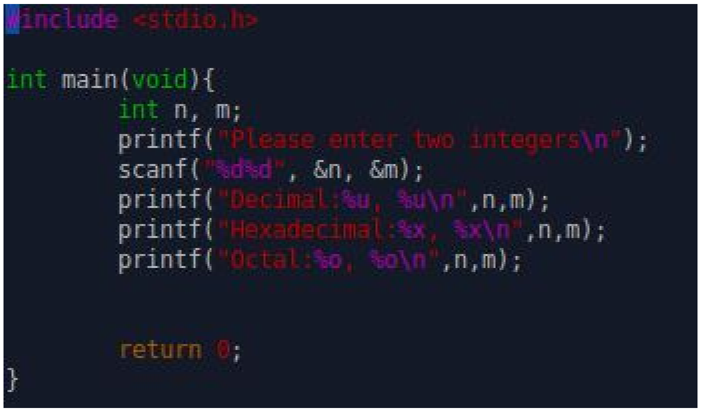


```
Enter 2 positive integers for calculation:
Warning: positive integers only, other inputs may crash the program
12 11
Please choose an operation from (+, -, *, /)Warning: operation only, other inputs may crash the program
+
12+11 = 23
```

<u>Exploring different data formats</u>

We didn’t spend too much time on this question. The code is really simple, which just scan two input integers and print them out by options.

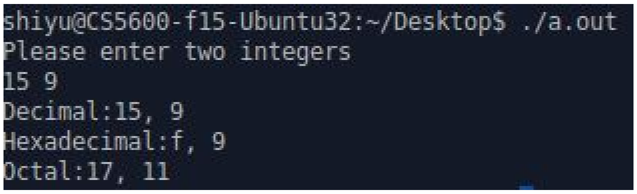
convert.c:



```
#include <stdio.h>

int main(void){
        int n, m;
        printf("Please enter two integers\n");
        scanf("%d%d", &n, &m);
        printf("Decimal:%u, %u\n",n,m);
        printf("Hexadecimal:%x, %x\n",n,m);
        printf("Octal:%o, %o\n",n,m);


        return 0;
}
```

This program will read the value in decimal number, and uses printf()’s ability to print number in different bases. Similar to computers store value in binary form, computer can also convert them into values based on different bases.

Output:



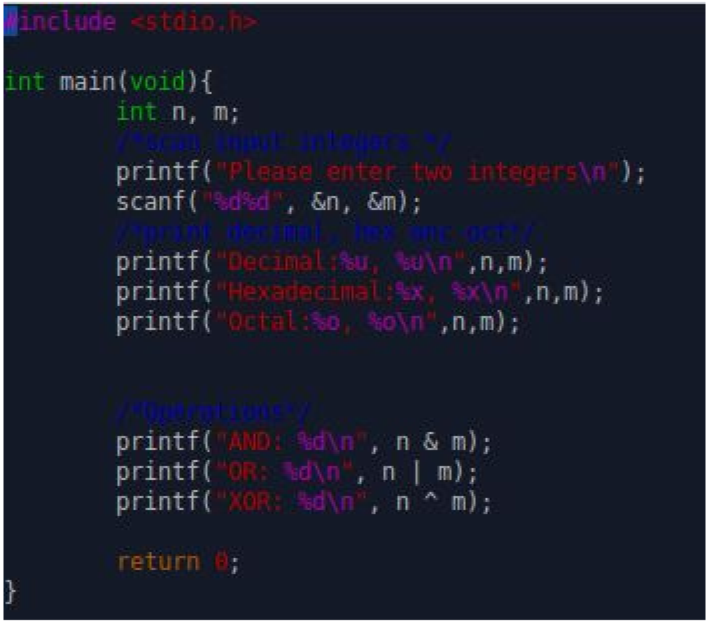
```
shiyu@CS5600-f15-Ubuntu32:~/Desktop$ ./a.out
Please enter two integers
15 9
Decimal:15, 9
Hexadecimal:f, 9
Octal:17, 11
```

From my point of view, the printf is converting and storing the input decimal in binary form and then convert them into different base when printing.

<u>Logical operations</u>

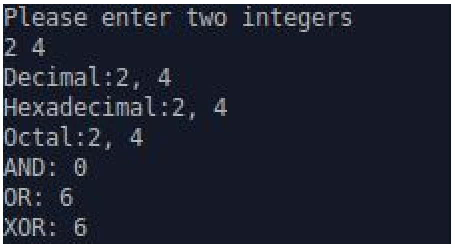
This program LogicOp prompts user to type in two integers, and returns these two integers in binary form. It will also generate bitwise AND, OR and XOR in binary form. After running this program with two positive inputs, we found that the results are correct.

logicOp.c:



```
#include <stdio.h>

int main(void){
        int n, m;
        /*scan input integers */
        printf("Please enter two integers\n");
        scanf("%d%d", &n, &m);
        /*print decimal, hex and oct*/
        printf("Decimal:%u, %u\n",n,m);
        printf("Hexadecimal:%x, %x\n",n,m);
        printf("Octal:%o, %o\n",n,m);


        /*Operations*/
        printf("AND: %d\n", n & m);
        printf("OR: %d\n", n | m);
        printf("XOR: %d\n", n ^ m);

        return 0;
}
```

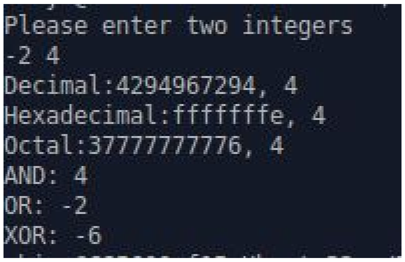
We firstly tried two positive integers:



```
Please enter two integers
2 4
Decimal:2, 4
Hexadecimal:2, 4
Octal:2, 4
AND: 0
OR: 6
XOR: 6
```

It works correctly.

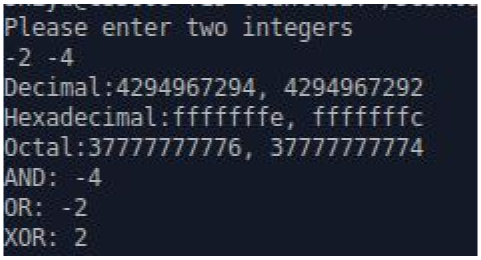
Then we tried it with one negative integer:



```
Please enter two integers
-2 4
Decimal:4294967294, 4
Hexadecimal:fffffffe, 4
Octal:37777777776, 4
AND: 4
OR: -2
XOR: -6
```

Considering the negative int into two’s complement, -2 is ‘1110’, 4 is ‘0100’, therefore AND is ‘0100’, 4. OR is ‘0100’, -2. And XOR is ‘1010’, -6. So the result is still correct.

After that, we tried it with two negative integers:

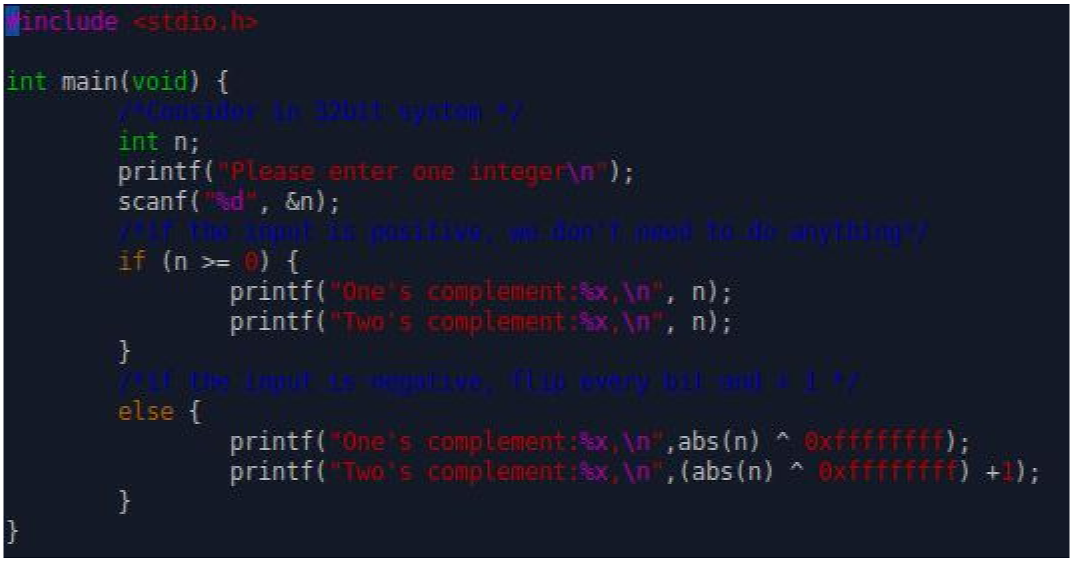


```
Please enter two integers
-2 -4
Decimal:4294967294, 4294967292
Hexadecimal:fffffffe, fffffffc
Octal:37777777776, 37777777774
AND: -4
OR: -2
XOR: 2
```

Again, we consider them into two's complement, -2 is '1110' -4 is '1100', so AND is '1100', -4. OR is '1110', -2 and XOR is '0010'. 2. Which is totally correct.

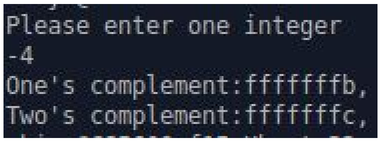
Two's Complement

This program consider the system as 32-bit. Finds one's and two's complements of a number in binary form. If the number entered is positive, then it's one's and two's complements are itself. If it is negative, it's one's complement is to flip it, and it's two's complement is its one's complement plus one.



```c
#include <stdio.h>

int main(void) {
        /*Consider in 32bit system */
        int n;
        printf("Please enter one integer\n");
        scanf("%d", &n);
        /*If the input is positive, we don't need to do anything*/
        if (n >= 0) {
                printf("One's complement:%x,\n", n);
                printf("Two's complement:%x,\n", n);
        }
        /*If the input is negative, flip every bit and + 1 */
        else {
                printf("One's complement:%x,\n",abs(n) ^ 0xffffffff);
                printf("Two's complement:%x,\n",(abs(n) ^ 0xffffffff) +1);
        }
}
```

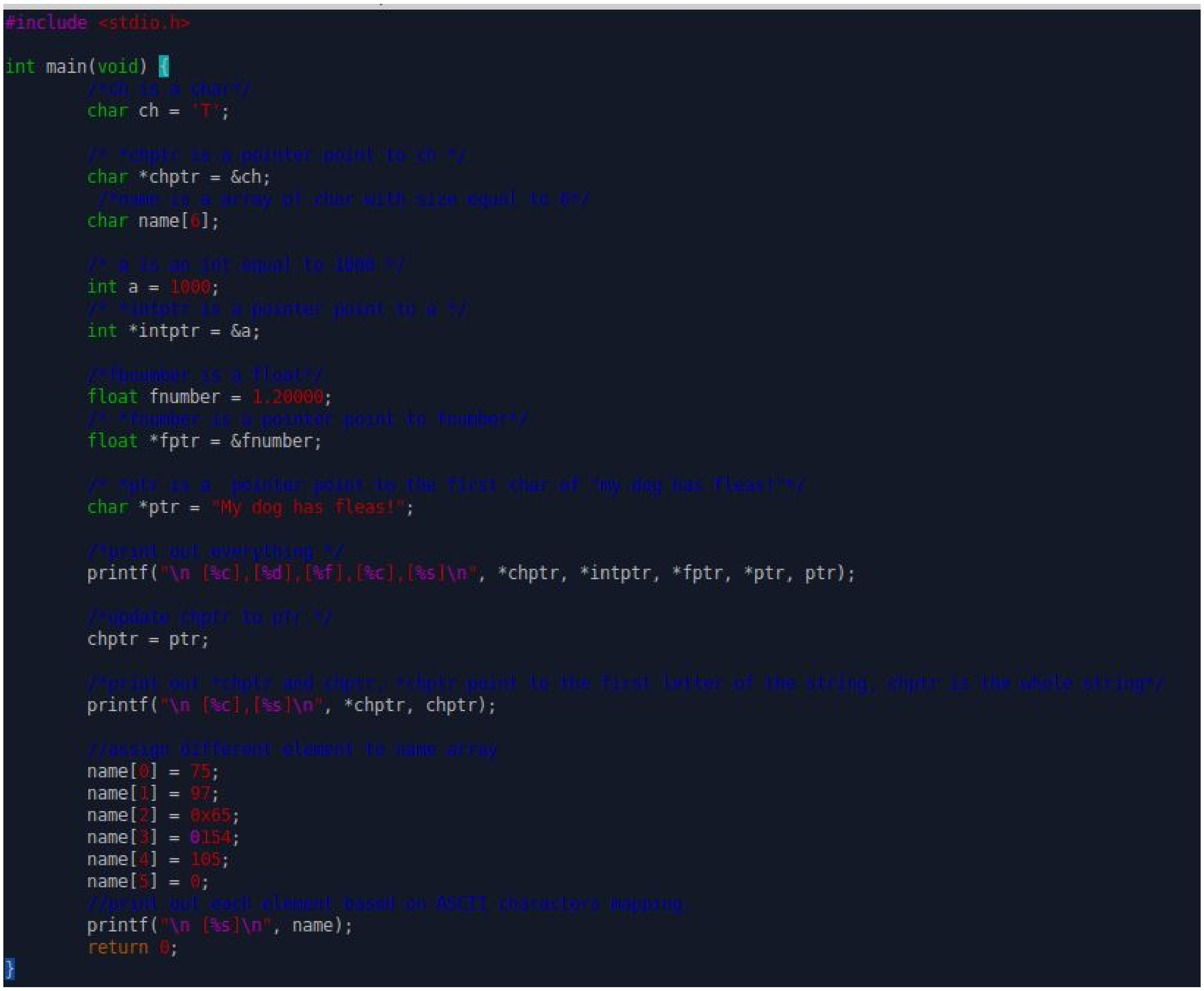
Output:



```
Please enter one integer
-4
One's complement:fffffffb,
Two's complement:fffffffc,
```

Explore Pointers

We fully read and understand the program. Then we add comments on each line to explain what it is doing.



```c
#include <stdio.h>

int main(void) {
        /*ch is a char*/
        char ch = 'T';

        /* *chptr is a pointer point to ch */
        char *chptr = &ch;
         /*name is a array of char with size equal to 6*/
        char name[6];

        /* a is an int equal to 1000 */
        int a = 1000;
        /* *intptr is a pointer point to a */
        int *intptr = &a;

        /*fnumber is a float*/
        float fnumber = 1.20000;
        /* *fnumber is a pointer point to fnumber*/
        float *fptr = &fnumber;

        /* *ptr is a  pointer point to the first char of "my dog has fleas!"*/
        char *ptr = "My dog has fleas!";

        /*print out everything */
        printf("\n [%c],[%d],[%f],[%c],[%s]\n", *chptr, *intptr, *fptr, *ptr, ptr);

        /*update chptr to ptr */
        chptr = ptr;

        /*print out *chptr and chptr, *chptr point to the first letter of the string, chptr is the whole string*/
        printf("\n [%c],[%s]\n", *chptr, chptr);

        //assign different element to name array
        name[0] = 75;
        name[1] = 97;
        name[2] = 0x65;
        name[3] = 0154;
        name[4] = 105;
        name[5] = 0;
        //print out each element based on ASCII characters mapping
        printf("\n [%s]\n", name);
        return 0;
}
```

Conclusion

Overall, we became more skilled at using Linux system. We learnt to write more difficult C programs. Also, we got a better understanding of different data formats and their conversion between each other. We found this lab very challenging yet very useful.