COMPSCI 210
Computer Systems 2

Assignment 2 - Assembly Language

Worth 5%

final date 24th of April, 2018, 11:59pm

Introduction

This assignment requires you to write a number of LC-3 assembly language routines leading to being able to transform a string of digits (and + and - signs) into both the corresponding integer and the 16-bit floating point value (as in Assignment 1).

You submit this assignment using Canvas. Package all of your programs together into a zip file and submit the one file. If you submit more than once Canvas adds extra numbers to the submission. Don't worry, the markers will always use the most recent one.

Part 1 (1 mark)

This program finds the location (i.e. the bit number) corresponding to the location of the first "1" in a number.

Call the program `part1.asm`.

Because this program doesn't take data from the keyboard you must have a label `number` in the code as here:

```
number          .fill     b0100000000000001
```

This number must be loaded into a register in the first line of your program e.g.

```
                .orig     x3000
                ld        r0, number
```

The program then determines the location of the first "1" in that number and prints the result to the display. The result from the value above would be:

```
The first significant bit is 14
```

If you then modify the value of `number` to `b0000000000000001` the result would be:

```
The first significant bit is 0
```

The markers will never enter a value of zero into `number`.

Part 2 (3 marks)

This program accepts an integer typed in by the user, verifies that the number is valid, and if it is valid prints the binary version of the number to the display as on the following page.

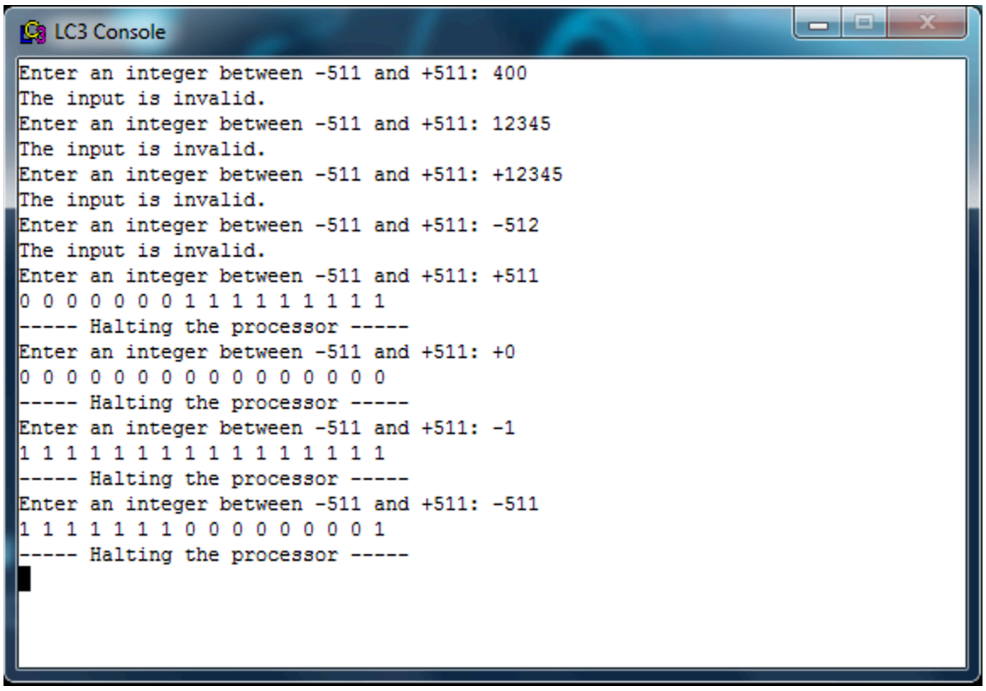
Call the program `part2.asm`.

As you can see the program rejects any input which doesn't start with a + or − sign and prints "`The input is invalid.`" to the display.

Also any integer values less than `-511` or greater than `+511` are rejected. Values like `+0123` are acceptable.

When a value is rejected the program loops back and asks for a new number.

The program stops after printing out the binary representation of the entered number, with a space between each pair of bits.



```
Enter an integer between -511 and +511: 400
The input is invalid.
Enter an integer between -511 and +511: 12345
The input is invalid.
Enter an integer between -511 and +511: +12345
The input is invalid.
Enter an integer between -511 and +511: -512
The input is invalid.
Enter an integer between -511 and +511: +511
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
----- Halting the processor -----
Enter an integer between -511 and +511: +0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
----- Halting the processor -----
Enter an integer between -511 and +511: -1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
----- Halting the processor -----
Enter an integer between -511 and +511: -511
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1
----- Halting the processor -----
```

Part 3 (1 mark)

This program shifts a number 11 places to the left, then ANDs the answer with `b0111100000000000`. This means the answer can only have 1s in bits 14 to 11. The answer is then written to the display as in Part 2.

Call the program `part3.asm`.

As in Part 1 the number to use must be stored at a label called `number` and the program begins with:

```
.orig   x3000
ld      r0, number
```

If the number is `b0000000000011011` the output should be:

```
0 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0
```

Part 4 (4 marks)

This program accepts an integer between `-511` to `+511` typed in by user. It works as in Part 2 until a valid number is entered, then the program prints the binary version of the number as an integer and follows it by printing the binary representation of the number as a 16-bit floating point number in the same format as in Assignment 1. Unlike in Assignment 1 the input numbers are only integers.

Bit 15 is the sign bit.

Bits 14 - 11 is the biased exponent (bias of 7).

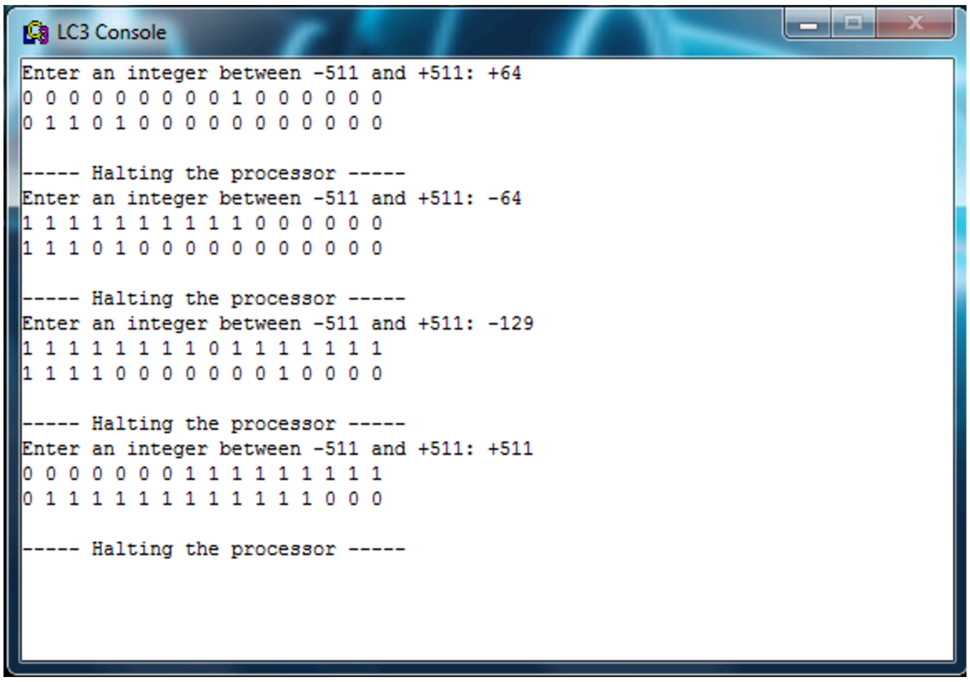
Bits 10 - 0 is the mantissa (fractional part).

Remember there is an implicit 1 for the fractional part.

On the next page are some example interactions with the program.

You must use subroutines for this program. To assist you to design your code there is a file called `part4.asm` which you must add to. You must not add any lines of code before the `halt` instruction.

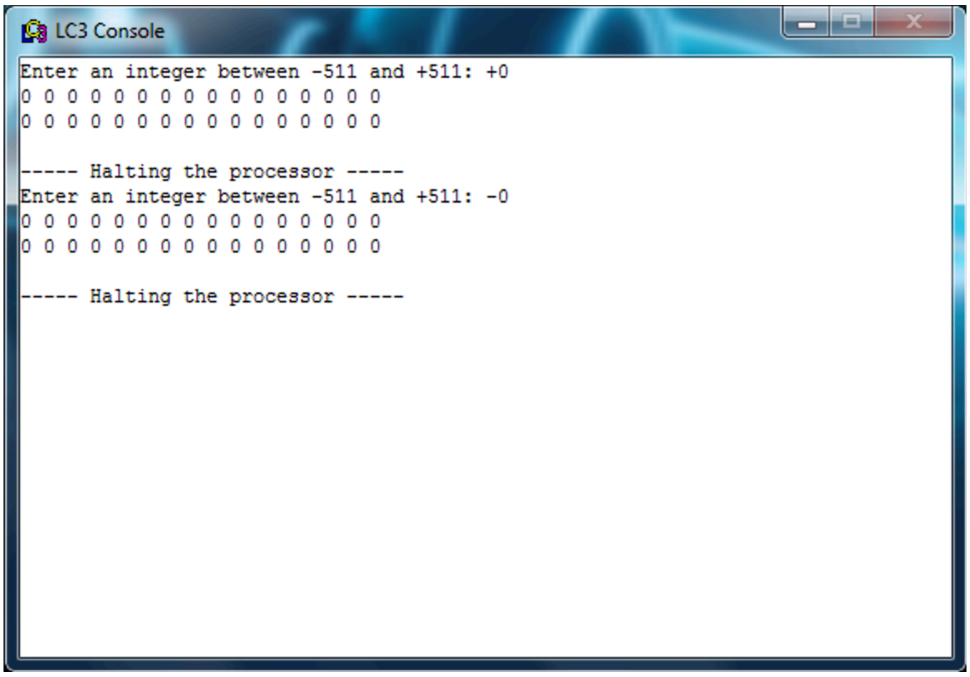
Example input and output.



LC3 Console

```
Enter an integer between -511 and +511: +64
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0

----- Halting the processor -----
Enter an integer between -511 and +511: -64
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0

----- Halting the processor -----
Enter an integer between -511 and +511: -129
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0

----- Halting the processor -----
Enter an integer between -511 and +511: +511
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

----- Halting the processor -----
```

Zero is a special case.



LC3 Console

```
Enter an integer between -511 and +511: +0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

----- Halting the processor -----
Enter an integer between -511 and +511: -0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

----- Halting the processor -----
```

When writing assembly language the temptation is to treat most values as globally accessible. The solution to this is to pass parameters and return values on the stack as in higher level languages. For this assignment you may use globally accessible values when that makes sense e.g. the input buffer used when the user types a number in can be accessed without having to pass its address to the subroutines which use it.

You should save all registers (except R0) when you enter a subroutine and restore them on exit. This is not strictly necessary in this assignment because the top-level program doesn't depend on registers maintaining their values. On a related point most programming languages keep all of the code separated from the data. In assembly language that is also a good idea but for this assignment it may be easier if you keep data associated with a subroutine near that subroutine. It makes it easier for you to read and fix your code. Also many of the LC-3 addressing modes limit offsets from the current value of PC to 256 words.

Style (1 mark)

The markers will be looking for nicely formatted code with good comments. Remember to record what you are using the registers for.

And

Include a comment with your login name (UPI) at the top of each program you submit. Markers will remove a mark if you have not done this on every program.

Any work you submit must be your work and your work alone.

To share assignment solutions and source code is not permitted under our academic integrity policy. Violation of this will result in your assignment submission attracting no marks, and you will face disciplinary actions in addition.