

Please read this document carefully. If you have any questions about the conditions of this assignment please ask for clarification as soon as possible. i.e. Don't leave it until the day before the assignment is due :).

N.B. Submit all files in Windows line ending format. The markers will be using the assembler and simulator on Windows in order to mark your work.

Include your name and login in a comment at the start of each file you submit. You get **1 mark** if you do this.

Exercise 1 - 8-bit addition (2 marks)

Take the signed4bit.asm program from class and turn it into an 8-bit version. The program has to work exactly the same as the original in the sense that op1 and op2 are filled in before the program runs and the result of adding op1 and op2, the carry, and overflow are correct when the program has finished. The difference is that the signed values are now 8-bits long. The source file should be called ex1.asm.

The markers will put test values directly into memory at the op1 and op2 locations.

Exercise 2 (2 marks)

The same as exercise 1 except the 8-bit operands have to be printed to the display along with the 8-bit number result and the carry (c) and overflow (v) bits as in the figure below. The source file should be called ex2.asm.

The markers will put test values directly into memory at the op1 and op2 locations.

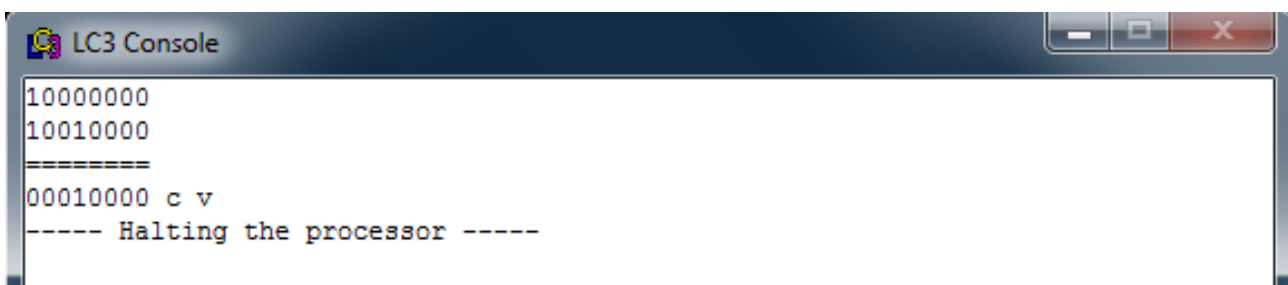
The program must start with the following lines of code. The reason for this is that program will be reasonably long and this way the markers can find your operand labels easily.

```
.orig    x3000

        lea    r0, start
        jmp    r0

op1      .fill   b100000000      ; must be < 256, treating as -128 to 127
op2      .fill   b10010000      ; must be < 256, treating as -128 to 127

start
```



Exercise 3 - 30-bit addition (5 marks)

This is very similar to exercise 2 except each number is stored in memory as 2 words of 15-bits each. We will call the two words the LSW (least significant word) and the MSW (most significant word) of the number.

The 16th bit of each word (bit 15) is used for the carry. When we add two 30-bit numbers we first add the LSWs of the two numbers and then the MSWs (including adding an extra 1 if the carry was set on the LSWs).

See the figure below for a picture of how the output should appear when the program is run several times with different 30-bit values. The 30-bit values are printed out MSW first and there is a space between the MSW and the LSW of each value. Figure 2 also shows the values you should be getting for the carry and overflow values.

The markers will put test values directly into memory at the op1_lsw, op1_msw and op2_lsw, op2_msw locations.

The program must start with the following lines of code. Same reason as in Exercise 2.

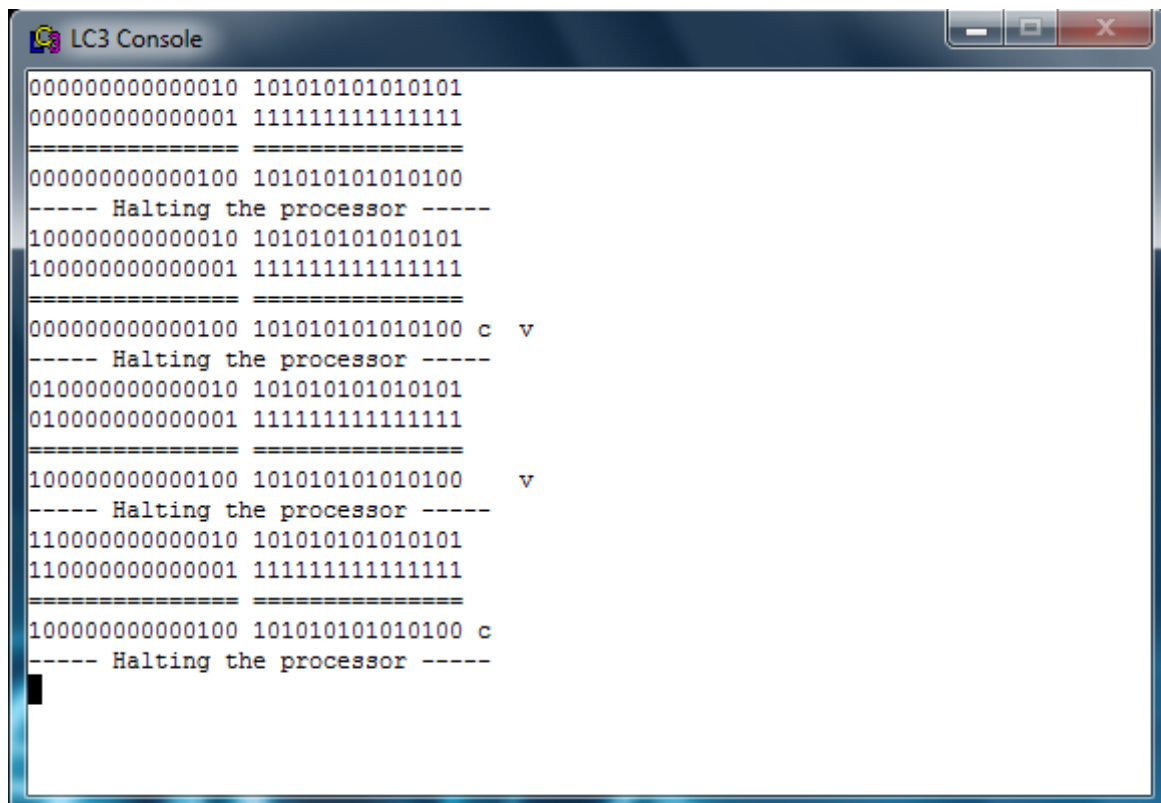
```
.orig    x3000

        lea    r0, start
        jmp    r0

op1_lsw    .fill b10101010101010101
op1_msw    .fill b00000000000000010
op2_lsw    .fill b11111111111111111
op2_msw    .fill b00000000000000001

start      ; this is where your real program begins
```

This exercise has **1 mark** for style. The markers will be looking for a good use of subroutines, good labels and useful comments.



```
LC3 Console
0000000000000010 10101010101010101
0000000000000001 11111111111111111
=====
0000000000000100 10101010101010100
----- Halting the processor -----
1000000000000010 10101010101010101
1000000000000001 11111111111111111
=====
0000000000000100 10101010101010100 c v
----- Halting the processor -----
0100000000000010 10101010101010101
0100000000000001 11111111111111111
=====
1000000000000100 10101010101010100 v
----- Halting the processor -----
1100000000000010 10101010101010101
1100000000000001 11111111111111111
=====
1000000000000100 10101010101010100 c
----- Halting the processor -----
```

Really important things

Develop your code in small sections, carefully testing each section as you go.

If you find your subroutines are getting longer than about 40 instructions they are too long. You need to refactor them to be smaller.

If you get assembler error messages because your labels are too far away, that means you need to refactor your code. There are ways around this but for this assignment just moving things closer will work. Normally keep data associated with a subroutine close to the subroutine.

Remember all of the hints about keeping track of the contents of your registers, and how to remember which registers get modified inside each of your subroutines. Store things into memory when you need to.

And of course when you jump to a subroutine you must save the return address and restore it before returning. We will see a better way to do this but the obvious approach is suitable for this assignment.

Submission

The assignment is to be submitted via Canvas.

And

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.