# Build a MARIE emulator

Due: Mar 13. Friday before spring break. Spring break 14th to 22nd will count as a single day for any late homework. I will be available for appointments over spring break - but I am working some days.

Pts: 400

Use the fetch-execute cycle and build an emulator for the MARIE language. Pick a language to implement this in. C, C++, F#, Kotlin, Go, Python, Haskell. (No Java, JavaScrip(node.js), Ruby).

Command Line: Process the command line for `--in <fn>` and `--out <fn>` and get the file names. They can be in either order.

C Example:

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

int main ( int argc, char *argv[] ) {

    int i;
    char *in, *out;

    in = "";
    out = "";

    for ( i = 1; i < argc; i++ ) {
        if ( strcmp ( argv[i], "--in" ) == 0 ) {
            if ( (i+1) < argc ) {
                in = argv[i+1];
                i++;
            } else {
                fprintf ( stderr, "Error: --in missing file name\n" );
                exit(1);
            }
        } else if ( strcmp ( argv[i], "--out" ) == 0 ) {
            if ( (i+1) < argc ) {
                out = argv[i+1];
                i++;
            } else {
                fprintf ( stderr, "Error: --out missing file name\n" );
                exit(1);
            }
        } else {
            fprintf ( stderr, "Error: invalid paramter >%s<\n", argv[i] );
            exit(1);
```

```
            }
        }

        if ( strcmp(in,"") == 0 ) {
            fprintf ( stderr, "Error: missing --in <fn> parameter\n" );
            exit(1);
        }
        if ( strcmp(out,"") == 0 ) {
            fprintf ( stderr, "Error: missing --out <fn> parameter\n" );
            exit(1);
        }

        /* TODO: Add more code */

        /* Read in Input file into `memory` */

        /* Loop until 'Halt' instrcution */

            /* Fetch */

            /* Execute */

    }
```

## Input

Read in a file with hex values - one on each line. If the line starts with '#' then it is a comment and should be skipped. Skip empty lines. Each line should have a 4 digit hex value on it. Read it in and store it in the "memory" of the machine. The addresses for reading start with 0 and increment.

## Process

Start the `pc` at 0 and fetch an instruction into the `ir`. Increment the `pc`. Decode the instruction into it's individual processing. Run some stuff to implement each instruction. Loop until you get a `Halt` ( 0x7000 ) instruction. On Halt, then exit the loop. Put in a counter that limits the number of instructions to 5000 - if exceeded then exit with an error. Print out the `pc` and the current instruction being run. Make the `Output` instruction print the hex and decimal value of the output from the `ac` register and if the value is between (inclusive) a space and 127 (the end of the ASCII table) then print it out as a character also.

The registers in the system are, `pc`, `ir`, `ac`, `mar`, `mdr`. These should be declared as integer variables. In the book `mdr` is the memory buffer register (MBR). In von Naumann's original paper `ir` is current instruction register (CIR).

## Testing

Write (at least 5) programs of your choice that demonstrate that all of the instructions in the MARIE instruction set work. Run the multiply example from the midterm and verify that it works.

## Turn in (via upload)

Your Code. A 1 page outline of what the files are that you are turning in called `outline.txt` as a *TEXT* file (no word documents). The outline says in 1 line what each file is. Put all the files into a .zip or .tar file. So, for example:

Put you name in the overview and in the comments in code.

Put in a comment in each source file as to what is in that file. If it is a test then specify what is getting tested and what the expected output is.

```
Overview of Homework 4
By Philip Schlump

Files           Description
-------------   ------------------------------------------------
overview.txt    This file
main.c            main code
lib.c            library functions

test1.mas        assembler source for multiply
test1.hex        hex file for test1.mas
test1.out        output from run.

test2.mas        test of LoadI, Jump, JumpI instruction
test2.hex        hex of test2.mas
test2.out        output from run.

test3.mas        test of AddI instruction
test3.hex        hex of test3.mas
test3.out        output from run.

test4.mas        test of StoreI instruction
test4.hex        hex of test4.mas
test4.out        output from run.

test5.mas        test of Halt instruction
test5.hex        hex of test5.mas
test5.out        output from run.
```

## Grading

1. Did you test all the instructions?
2. Do all the instructions work?
3. Are your tests complete?
4. Did you include an ‘overview.txt’ file?
5. If I run a testX.hex file that works will I get the correct output?
6. Have you included your name in the comments?

# Copyright