159.102 Instructions for Assignment 2

Assignment 2 starts in Week 5 and is due in Week 9 (26 September). Assignment 2 is longer than the other assignments and you have extra time to complete it. You should plan to complete the assignment about a week early. Then you still have a week if something unexpected happens to slow you down.

NOTE: Assignment 2 counts 20% towards your final result. The other assignments count 15% each.

It is a good idea to put your name and ID number in a comment at the top of your program.

Write a C++ program that simulates the MASSEY machine. The program must receive input in the form of a text file consisting of MASSEY machine code instructions. Your program then simulates the machine running, i.e. it works through the machine code instructions (in the correct sequence) changing values of registers and memory locations as required. You must design appropriate output that assists a machine code programmer to see what is going on with the machine code instructions. You should display items such as program counter, instructions register, values of recently changed registers, and values of recently changed memory locations. Ensure that you read through all the sections below.

Section A - input

The input to your program is a single text file containing a MASSEY machine code program.
For example, here is a typical input file where each line is a machine code instruction:

```
100A
1107
3209
6312
D307
43FF
E000
43EE
E000
0003
```

Notes about the input:

1. The programmer using the simulation can give the file whatever name they like. It is best to read in the file name when you start. A typical file name would be "prog1.txt".
2. Each line of the file is one machine code instruction. There is no other text in the file.
3. Make several (at least five) different files, each with a different machine code program. Test your program on a variety of machine code programs.
4. Do **not** attempt to check for invalid input. Assume every program contains correct machine code instructions although a program may have logic errors, e.g. instructions in the incorrect order.

Here is an incomplete piece of code that may be useful when reading a file of hexadecimal numbers. Note that you can **input a hex number directly** from the text file into the variable `hexnumber`.

```
string filename; int hexnumber, i;
cin >> filename;
inputfile.open(filename.c_str(), fstream::in);
// include code here to check if the file opened correctly

  i = 0;
  while (inputfile >> hex >> hexnumber) {
    memory[i] = hexnumber;

i++; }
```

(*Hint: Start off by writing a program that reads the file and displays each line on the screen.*
This is to check that your input is working correctly before you proceed to the rest of the program).

Section B – program design

Your program **must** use the following global variables:

```
int memory[256];
int reg[16];  // note: "register" is a reserved word
int pc, ir;
```

The array "`memory`" represents the memory of the MASSEY machine.
The array "`reg`" represents the registers of the MASSEY machine.
The variable "`pc`" is the Program Counter and "`ir`" is the Instruction Register.

Note that **there should be no strings in this program** (except the filename).

The basic algorithm for your program is:

- read instructions from the file and load them into memory
- run through the instructions and execute each instruction correctly

Notes about the program design:

5. Study the MASSEY machine code instructions in the notes.
6. Ensure that your input program correctly executes a valid machine code instruction.
7. You do **not** have to execute instruction 5 (floating point addition) – ignore instruction 5.
8. Do **not** check for invalid instructions. Only deal with valid instructions.
9. You must have **at least two** functions in your program.
10. **Test extensively.** Ensure that you have tested every instruction (except 5). Use machine code programs from the notes as test data.

(*Hint: get your program working for a few instructions, perhaps those in the input example. When these instructions are working correctly, expand the program to handle other instructions.*)

Section C - output

The output **must** meet the following requirements:

- display the full program (showing the memory locations) before executing the program
- identify the important items to display during execution of the instructions
- display one line for every machine code instruction (showing any changes)

Your display should look very similar to the following example:

```
Enter the file name of the MASSEY machine code: program1.txt
Memory[00] = 100A
Memory[01] = 1107
Memory[02] = 3209
Memory[03] = 6312
Memory[04] = D307
Memory[05] = 43FF
Memory[06] = E000
Memory[07] = 43EE
Memory[08] = E000
Memory[09] = 0003
```

```
PC: 00  IR: 100A  Register R0 = 000A
PC: 01  IR: 1107  Register R1 = 0007
PC: 02  IR: 3209  Register R2 = 0003
PC: 03  IR: 6312  Register R3 = 000A
PC: 04  IR: D307  TRUE - jump to location 07
PC: 07  IR: 43EE  Memory[EE] = 000A
PC: 08  IR: E000  Program halts
```

Notes about the output:

11. **Every number in the display is a hexadecimal number.** Do not display any decimal numbers.
12. Display all lines that have been input from the file and loaded into memory. Note that the location (the index) in memory is displayed as **2 digits** and the contents of memory is **4 digits**.
13. Display one line of output for each machine code instruction – just after it has been executed.
14. On each line, display the Program Counter (**2 digits**) and the Instruction Register (**4 digits**) – recall that the IR contains the current instruction.
15. On each line, display any register that has changed. E.g. the first instruction above loads R0 so the value in R0 is displayed. The register number is **1 digit** and the register contents are **4 digits**.
16. On each line, display any memory location that has changed. E.g. the instruction 43EE loads a value into memory location EE so the value in memory[EE] is displayed. Note that the location (the index) in memory is displayed as **2 digits** and the contents of memory is **4 digits**.
17. Study the display for D307 (a jump instruction). There is also an alternative display which is: `FALSE - do not jump`
18. Note the display for E000 (the halt instruction).
19. When displaying negative numbers, ignore the required number of digits stated above. For example, the display for -6 usually appears as FFFF FFFA. That is fine. While it is interesting to think about how to display negative numbers, don't waste time on this aspect.

(*Hint: to start with, get your program reading the file, loading instructions into the memory, and then displaying the memory locations and the instructions – first part of the output. Once this is working, then expand the output to display the results of the instructions.*)

Some general notes about the assignments in 159.102

- You can find the assignment instructions in a file under Assessments.
- You submit your assignments via Stream (under Assessments) before the due date and time
- The due date and time appear on the Assignment under Assessments (where you submit)
- **Submit only your .cpp file**
- Do not submit the .exe file or any data files or screen shots of the program running
- Staff are not available to check your assignment before you submit it.
- Do not rush into submitting an assignment. You may find useful information in the notes during the week after the assignment starts.
- Assignments may use C++ knowledge from 159.101, 159.102 and elsewhere. However, if you use knowledge from elsewhere, make sure you use it correctly.

IMPORTANT rules for assignments in 159.102

- You may get assistance when writing an assignment. Assistance includes asking questions and getting ideas from teaching staff and other students. Assistance also includes asking for help when your program is not working correctly and you cannot find the error.

- You may **NOT** get someone else to write your assignment for you. If you submit a program written by someone else, you will lose a significant amount of the marks for the assignment.
- You may **NOT** copy a program from the internet. If you submit a program copied from the internet you will receive **ZERO** marks for the assignment. It is very easy for markers to find the same program on the internet.
- The important thing is that you must show that you understand what is happening in the program you submit. Teaching staff will sometimes arrange zoom sessions with students to check that they understand their submission. If this happens to you, please do not be offended – it is something we have to do as part of the quality assurance for the course.

Working on your assignments in 159.102

- You need an editor/compiler to create and run your program. VSCode is provided (see notes to install VSCode under Week 1) but you can use any other IDE that supports C++
- Build up your program, for example: for example: start by reading the hex numbers from the file and displaying them (see the first part of the output). When this is working, include the execution of a few simple machine instructions (perhaps LOAD and STORE). Test these instructions with some simple test programs. Then move on to including the other instructions.
- Give yourself plenty of time. Do not start a few days before the deadline!
- Do not give up just because the deadline arrives. You will still get some marks for a partial solution. In a difficult situation, you can apply for an extension.

Marking criteria for assignments in 159.102

Assignments are usually marked out of 10 and marks can be lost for:

- programs not compiling or running
- errors in code
- programs that have not been tested for a variety of situations
- programs that do not follow the instructions that are provided
- programs that appear to be written by someone else
- programs that are copied from the internet