In the course project, you will simulate a simple MIPS integer processor. Your MIPS simulator will be capable of loading a specified file recording a sequence of MIPS binaries, and outputting the program computation outputs. You are encouraged to use Unix/Linux operating system, C/C++/Java to finish the project.

Input file:

The first part of the input file starts from a list of initial contents of registers. This list starts from the word REGISTERS, then comes a sequence of lines containing the pairs of register names and integers, separated by spaces, e.g., R5 42. All possible register names are: R0, R1,..., R31. The list of the pairs of register names and integers may be empty, but the file should contain the word REGISTERS. Instead of showing register notations, the input file simply uses register numbers (e.g., R8 instead of $t0) to denote registers.

The second part of the input file starts from the word MEMORY, then comes a sequence of lines containing the integer pairs separated by spaces, e.g., 40 25. The first integer is the number of the memory location, the second is its contents. All possible memory location numbers are: 0, 4,..., 996. Again, the list of integer pairs may be empty, but the file should contain the word MEMORY.
The following part starts from the word CODE, followed by a sequence of lines, each line containing a single instruction in MIPS binary. We will use only nine types of MIPS instructions: two data transfers: load word (e.g., LW R1, 0(R2)), store word (e.g., SW R1, 8(R2)); five arithmetic logic operations: add (e.g., ADD R1, R2, R3) and add immediate (e.g., ADDI R1, R2, 6), subtract (e.g., SUB R1, R2, R3), set less than (e.g., SLT R1,R2, R3) ; and two control-flow instruction: branch if equal (e.g., BEQ R1, R2, Loop), and branch not equal (e.g., BNE R1, R2, Loop). You should simulate the multi-cycle un-pipelined MIPS processor. The control-flow instruction computes the address of target instruction in ID and checks the condition in EX.

Spaces should be understood not only as ordinary spaces but also as white space characters, such as the end of line, tab, etc. Any line of the input data file may start from one or more spaces and one or more spaces may end it. For simplicity, initial contents of all registers, not listed after the word REGISTERS, and initial contents of all memory locations, not listed after the word MEMORY, are equal to zero. Additionally, the contents of the register Zero is always equal to zero (loading of

$zero has no effect).

You may assume that the input data file does not contain any errors.

Processor Simulation:

When running your simulator, it should first ask the user for the name of the input and output files. The expected response of the user is the input file name, and output file name followed by pressing the <ENTER> key. After finishing one round of simulation, your simulator should ask the user if he/she would like to continue with another round of simulation and accept the corresponding response.

Output file:

The output file should include the timing of the instruction sequence and final contents of all registers and memory locations appearing in the CODE.
The timing of the instruction sequence should show cycle numbers in rows, while columns should be labeled with instruction numbers , following the format as

C#1 I1-IF
C#2 I1-ID
C#3 I1-EX
C#4 I1-MEM
C#5 I1-WB
C#6 I2-IF
C#7 I2-ID
C#8 I2-EX
C#9 I2-WB
……

The final contents of all registers and memory locations should follow the format as

REGISTERS
R2 -8
R3 120

...

MEMORY
0 40
4 54
16 47
...

You don't need to show the registers and memory with content of zero

Instead of showing register notations, you can simply put register numbers (e.g., R8 instead of $t0) in your output file.

Sample test:

You may find a sample test in the Project assignment folder

Submission

Project Deadline: May 2nd, 2018, 11:59PM CST

When you are ready to submit the project, include all necessary source files with comments (this is important, you will be asked to explain your code if necessary), makefile (if any), and readme (if any) in a single file named as your_last_name.your_UHID# (e.g., if your last name is Woods, your UHID is 12435, the file name would be Woods.12435), and **upload to the Blackboard.** Do not submit object files, executable files, and test data files. Late projects will be accepted with 15% penalty per day up to two days.

Grading

1. Instructions are correctly decoded, executed, and memory and registers are correctly updated (90%)
2. Correctly accept input file, has user interaction, output file follows the required format (10%)

The grading rubric is in the Project assignment folder

(Note: final percentages are subject to change)