NOTE: If you decide to conduct this advanced project, please email me your decision by 3/4/2021 (any oral notification is not effective). This is one-time decision, once you decide, you cannot change anymore. By doing this advanced project, you will get 60% bonus on your Project. I will also provide you some hints about how to write the simulator in class around late February 2021.

In this advanced course project, you will simulate a simple MIPS integer pipeline. Your MIPS pipeline simulator will be capable of loading a specified file recording a sequence of MIPS instructions, and outputting their timing information in the MIPS pipeline. You are encouraged to use Unix/Linux operating system, C/C++/Python/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.
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, 8,..., 992. 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. We will use only five types of MIPS instructions: two data transfers: load double word (e.g., LD R2, 0(R1)), store double word (e.g., SD R4, 8(R3)); two arithmetic operations: add (e.g., DADD R1, R2, R3) and subtract (e.g., SUB R1, R2, R3); and one control-flow instruction: branch not equal to zero (e.g., BNEZ R4, Loop). You only need consider four addressing modes: register, register indirect, displacement, and immediate. All instructions are executed on an eight-stage pipeline: IF1, IF2, ID, EX, MEM1, MEM2, MEM3, and WB. The BNEZ instruction computes the address of target instruction in ID and checks the condition in EX. The load/store instructions complete the data cache access in MEM2. Instructions may write to and read from registers at the same clock cycle (first write, then read).
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 R0 is always equal to zero (loading of R0 has no effect).

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

Pipeline Simulation:

Your MIPS pipeline simulator should support the following mode with regard to the pipeline hazards:
There is forwarding and bypassing hardware, branches are predicted not taken for branch hazards.

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, e.g.,

```
c#1 I1-IF1
c#2 I1-IF2 I2-IF1
c#3 I1-ID I2-IF2 I3-IF1
c#4 I1-EX I2-ID I3-stall
...
```

For stall instructions, "stall" should be inserted in the file appropriately.

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

```
REGISTERS
R2 -8
R3 120

...
```

```
MEMORY
0 40
8 54
16 47
...
```

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

Sample test:

You may find a sample test in the Project assignment folder.

Submission

Project Deadline: May 11th, 2021, 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 your codes look similar to others), 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. There is forwarding and bypassing hardware, branches are predicted not taken for branch hazards (90%)
2. Correctly accept input file, has user interaction, output file follows the required format (10%)

The grading rubic is available in the Project assignment folder.

(note: final percentages are subject to change)