# Computer Systems Architecture

# Homework 5

**MIPS Simulator**

The implementation for this HW is divided into two parts:

1. MIPS simulator without Pipelining.
2. MIPS simulator with 3-stage Pipelining and Data Forwarding.

The MIPS simulator only support the following instructions:

add opr1, opr2, opr3 : opr1 = opr2 + opr3

addi opr1,opr2,value : opr1 = opr2 + value

sub opr1, opr2, opr3 : opr1 = opr2 - opr3

mul opr1, opr2, opr3 : opr1 = opr2 x opr3

div opr1, opr2, opr3 : opr1 = opr2 / opr3

b *label* : Branch/Jump to Label (Unconditional)

beq opr1, opr2, *label* : Branch if Equal (opr1 == opr2) to Label

bnq opr1, opr2, *label* : Branch if Not Equal (opr1 != opr2) to Label

end : End of the program

Each opcode (instruction) has certain number of input operands. Opcode 'end' signals the end of the program in which case instructions in the pipeline must be completed(if applicable) and desired output must be printed. The Operands can either be registers, an immediate value or a label. MIPS processor is expected to have 8 registers ($0, $2,...,$7). Labels signifies the target instruction for the branches. **The first lien is the final results of the registers that the program should produce. The second line in the input file is the initialization values for the 8 registers (in order $0-$7).**

An example input:

```
2,4,6,88,67,45,44,89
add $0,$1,$2
label1 sub $1,$0,$4
add $2,$3,$5
beq $0,$7,label1
end
```

There is single blank between the label and the opcodes, and the opcodes and their operands. All label start with the keyword 'label' and followed by a numeric value.

**Pipelining and Data Forwarding Rules:**
This is a 3 stage pipeline with Fetch, Execute and Write-Back as the only stages. Data forwarding happens ONLY after the execution of the preceding instruction to the execution stage of the current instruction. In case of Branches, **you must STALL the pipeline till the end of the execution stage for the branch to know if branch is taken or not.**

**Output:**
For Part 1: output at the end of the input program must be the values for all 8 registers (in order $0-$7), just like the first line of the input file. **This output should be return as a vector<int> pointer.**
For Part 2: when you are done with part one and your code is successfully running. Copy your solution.cpp into solution_pipeline.cpp and start implementing your pipeline. output at the end of the execution of all instructions will be the values for all 8 registers same as Part 1, the cycle of completion for each of the instruction in following format:
add $3,$2,$0
<cycle_of_completion>
For branches, the cycle of completion is the end of the execution stage. For Arithmetic instruction, completion is the end of write-back stage.

**Submission:**
Please clone the solution repository from :
https://github.com/hamidre13/cs250p_mips_simulator

**All your code should be written in solution.cpp solution_pipeline.cpp file. DO NOT CHANGE THE SOLUTION CLASS SIGNATURE otherwise we cannot run your code and YOU will receive 0.**You can add functions to these classes if you want for your internal use, but do not change the default ones.

1. Make sure that your code compiles and runs on the ics lab machines.
2. Generate a report with the screenshot of successful completion for each part.
3. **When you are ready to submit your code delete all files except solution.cpp and info.txt .** write your name and student id in that file as it shows.
4. Put these files in a folder called studentid_studentname
5. ZIP the folder and submit it
6. **There will be 10pts for correct submission**

Your application must be named **'mips'** and must take two command line input i.e. the input file with register initialization values and the instructions for the program. The second command will be clock cycle time.Also you can have a third optional command that when you provide it your app will go into debugging mode. The last command is not necessary but highly recommended.
$./mips <input_file> clock_time debug

Picture shows how your output should look like for a single cycle processor.