==========

A. 5 Stage Basic MIPS pipeline without bypassing [2 marks]

==========

Implement in C++ the basic 5 stage MIPS pipeline — IF (Instruction Fetch), ID (Instruction Decode), EX (ALU execute), MEM (memory read/write) and WB (write back to register).

Your program should have 32 registers as a data structure. The control signals, input and output for each pipeline stage can be implemented as a struct, with a separate function to implement the working of the stage. The function for EX stage can have support for some basic ALU operations like add, sub, addi, sll, srl, and, or, slt etc. assuming instructions have only those ALU operations.

WB should happen in the first half cycle, and the rest of the stages in the second half cycle. Cycles can be implemented as loop iterations. Total cycles will depend on data dependencies among different instructions.

Output should print cycle number, first vs. second half cycle in that cycle number and what each pipeline stage is doing in that cycle.

Implement this program in a C++ file with the name **5stage.cpp**. You can test this with instruction pairs, as discussed in class. Check total cycles taken by the simulator, vs. calculated with pen and paper. Some examples of instruction pairs are given below:
(i) add $1, $2, $3; lw $4, 8($1)
(ii) lw $1, 8($2); lw $4, 8($1)
(iii) lw $1, 8($2); sw $1, 8($3)
(iv) lw $1, 8($2); add $4, $1, $3

==========

B. 5 Stage Basic MIPS pipeline with bypassing [3 marks]

==========

Implement **5stage_bypass.cpp,** where you forward input to the ID stage, not just from L3, but from L4 and L5. Test this program also with instruction pairs as above, and note the improvements in total cycles compared to no bypassing.

==========

C. 7-9 Stage pipeline without and with bypassing [3 marks]

==========

Implement the 7-9 stage pipeline taught in class, with 7 stages for ALU operations and 9 stages for load-store operations. Without bypassing source file should be named **79stage.cpp** and with bypassing source file should be named **79stage_bypass.cpp**. Test this program also with instruction pairs as above, and note the improvements/degradations in total cycles compared to 5 stage pipeline.

==========

D. Report with cycle comparison plots and code quality [3 marks]

==========

Test files will be shared soon. You are expected to run the 4 cpp files on the test files. Create a PDF report with 4 bar-plots, one for each .cpp file. Y-axis should denote total cycles, and x-axis the instruction set number. Compare across the plots and list your observations.

You may use the following 4 sets of instructions for initial testing:
(i) lw $10, 20($1); sub $11, $2, $3; add $12, $3, $4; lw $13, 24($1); add $14, $5, $6
(ii) sub $2, $1, $3; and $12, $2, $5; or $13, $6, $2; add $14, $2, $2; sw $15, 100($2)
(iii) add $1, $1, $2; add $1, $1, $3; add $1, $1, $4
(iv) lw $2, 20($1); and $4, $2, $5; or $8, $2, $6; add $9, $4, $2; slt $1, $6, $7

Your project should use object oriented design, such that the four files **5stage.cpp, 5stage_bypass.cpp, 79stage.cpp** and **79stage_bypass.cpp** reuse as much code as possible.

==========

E. Branch prediction for control hazard [4 marks]

==========

Implement a branch predictor with 2-bit saturating counters and branch history registers (BHR). Use 2^14 2-bit saturating counters indexed using 14 LSBs of each branch instruction and a 2 bit Branch History Register (BHR) that records the outcomes of the previous 2 branches.

branchtrace is a trace of branch instructions consisting of the PC at which each branch occurs, and whether the branch is Taken(1) or Not Taken(0).

Predict varying the start state of (a) each 2-bit saturating counter between 00,01,10,11 and (b) each BHR also as 00,01,10,11. Predict using only saturating counters, only BHRs and some ways of combining the two. For each of these variations, your output file should indicate, for each branch in the trace, whether it was predicted as Taken(1) or Not Taken(0). Compute branch prediction accuracy of each output file, as (#correct predictions/#total predictions).

Include these branch prediction accuracies in a table in the PDF report above. Each row in the table should mention the prediction strategy and the corresponding accuracy.

==========

What to submit

==========

Name your folder as entrynum1_entrynum2. Include all cpp and header files, a Makefile and a readme (with instructions on how to compile and run your program). Include the PDF report. Mention in the report %work split between the project partners. Zip and upload on Moodle.

==========

Starter Code Resource

==========

You are suggested to refer to the MIPS Processor GitHub repository to help you with the parsing of the MIPS instructions. You are free to directly use and modify the code given in this repository **only**.