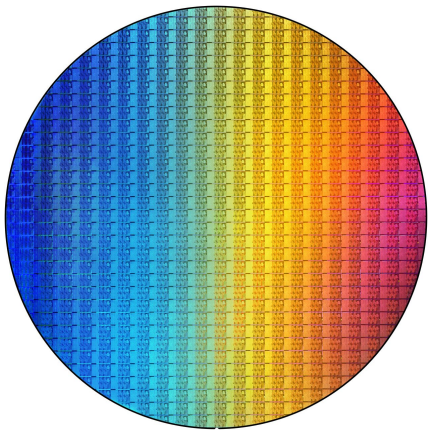
Portland State University

ECE571 - SystemVerilog

Winter 2019



MIPS16 ISA Verification Plan

R. Ignacio Genovese
Chenyang Li
Shouvik Rakshit
Aishwarya Doosa

V1.0

Introduction

As digital designs get bigger, the effort in verification can scale up to 70% or even 80% of the total design effort, thereby making it the most expensive step in the entire IC design flow. Besides, any behavioral or functional bugs escaping this phase will surface only after the silicon is integrated into the target system, resulting in even more costly design and silicon iterations. For this reason, nowadays it is essential for engineers to learn different approaches to overcome these bugs, getting to successfully implement verification environments aimed to reduce these costs.

Therefore, as final project for the course *ECE571 - Introduction to SystemVerilog for Design and Verification* at Portland State University, the team will develop and implement a Verification Plan for a MIPS16 ISA.

Verification Approach

Instead of using a set of functional specifications as base for our Verification Plan, we will learn the architecture and requirements from a completely functional MIPS 16 ISA design. This will be used to define a set of **unit tests** (by assembly code) that will serve as stimulus for the processor core. After running these tests, the QuestaSim **coverage tool** will be used to determine if this set of testcases is enough to get a good (branch, statement, fsm, toggle) coverage. Based on the results from the coverage tool, we will develop new testcases to get a better coverage in case it's necessary.

To assure the functional correctness of these testcases (besides taking it for granted as the design is supposed to be correct), the final state of the register file and memory will be checked, as each unit test should be simple enough to know this result.

Once we get a good set of tests, we will proceed to save into files the inputs and outputs of each of the 5 pipeline stages (Instruction Fetch, Instruction Decode, Execute, Memory and Writeback), in order to carry on with the verification of each of these stages independently. These files will then be used to stimulate each stage and compare its outputs.

Along with the unit tests, a set of **assertions** for each pipeline stage will be defined in a separate file and bound to the design.

Finally, this coverage and assertion based verification environment will be used to verify a "broken" design, in order to prove its effectiveness and implementation to find bugs.

Unit Tests

After reviewing and understanding the implemented MIPS16 ISA, we proceeded to consider each stage inputs to be stimulated and defined the following basic unit tests to implement:

- IF:
  - Implement branch taken and branch not taken.
  - Try the instruction memory boundaries using different offsets.
  - Produce stalls in the hazard detection unit.
- ID:
  - Implement all type of instructions using every register as destination, source operand 1 and source operand 2. There are 13 types of instructions:
    - OP_NOP
    - OP_ADD
    - OP_SUB
    - OP_AND
    - OP_OR
    - OP_XOR
    - OP_SL
    - OP_SR
    - OP_SRU
    - OP_ADDI
    - OP_LD
    - OP_ST
    - OP_BZ
- EX:
  - Test all ALU operations using every register as destination, source operand 1 and source operand 2. There are 8 ALU operations:
    - ADD
    - SUB
    - AND
    - OR
    - XOR
    - SL (shift left)
    - SR (shift right, preserving sign bit)
    - SRU (shift right unsigned)

- MEM:
    - Implement load and store operations to every memory position, using every register as source operand 1 (base), source operand 2 (offset) and destination register.
    - Consider that writes to register 0 (R0) will always produce a zero.
- WB:
    - Write back every register with results from the ALU and values read from the memory.
- Hazard detection unit:
    - Produce every possible stall using every register as destination, source operand 1 and source operand 2. There are 3 possible stalls:
        - When a source operand for the current instruction is the destination operand for the instruction in the EX stage.
        - When a source operand for the current instruction is the destination operand for the instruction in the MEM stage.
        - When a source operand for the current instruction is the destination operand for the instruction in the WB stage.

Testcases

Name	Stage	Description	Owner
hazard_r0.asm	Hazard Detection	Produce every possible stall using R0 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r1.asm	Hazard Detection	Produce every possible stall using R1 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r2.asm	Hazard Detection	Produce every possible stall using R2 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r3.asm	Hazard Detection	Produce every possible stall using R3 as destination, source operand 1 and source operand 2.	Ignacio Genovese

hazard_r4.asm	Hazard Detection	Produce every possible stall using R4 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r5.asm	Hazard Detection	Produce every possible stall using R5 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r6.asm	Hazard Detection	Produce every possible stall using R6 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_r7.asm	Hazard Detection	Produce every possible stall using R7 as destination, source operand 1 and source operand 2.	Ignacio Genovese
hazard_detection.asm	Hazard Detection	Normal Operation test, producing every possible hazard.	Ignacio Genovese
R0_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R0 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R1_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R1 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R2_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R2 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R3_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R3 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R4_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R4 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R5_load_store	Memory, Write Back & Instruction	Perform memory store from register R5 to 256 locations of the memory. Performs a load back to the register from	Shouvik Rakshit

	Decode	memory.	
R6_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R6 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
R7_load_store	Memory, Write Back & Instruction Decode	Perform memory store from register R7 to 256 locations of the memory. Performs a load back to the register from memory.	Shouvik Rakshit
add.asm	Execution	Performs addition of two operands using every register as a destination and source.	Aishwarya Doosa
sub.asm	Execution	Performs subtraction of two registers using every register as a destination and source.	Aishwarya Doosa
and.asm	Execution	Performs bitwise and operation between two registers using every register as a destination and source.	Aishwarya Doosa
or.asm	Execution	Performs bitwise or operation between two registers using every register as a destination and source.	Aishwarya Doosa
xor.asm	Execution	Performs bitwise exor operation between two registers using every register as a destination and source.	Aishwarya Doosa
sl.asm	Execution	Performs logical shift left operation using every register as a destination and source.	Aishwarya Doosa
sr.asm	Execution	Performs logical shift right operation using every register as a destination and source.	Aishwarya Doosa
sru.asm	Execution	Performs unsigned shift right operation using every register as a destination and source.	Aishwarya Doosa

branch_taken.asm	Instruction Fetch & Instruction Decode	Branch operation test, test branch taken	Chenyang Li
branch_not_taken.asm	Instruction Fetch & Instruction Decode	Branch operation test, test branch not taken	Chenyang Li
nop.asm	Instruction Fetch & Instruction Decode	NOP operation test, test nop instruction	Chenyang Li
add_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
add_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
add_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
add_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
add_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
add_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
add_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for addition of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li

sub_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sub_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sub_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sub_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sub_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sub_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sub_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for subtraction of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
and_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
and_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
and_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li

and_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
and_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
and_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
and_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for AND of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
or_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
or_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
or_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
or_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
or_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
or_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li

or_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for OR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
xor_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sl_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sl_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li

sl_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sl_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sl_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sl_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sl_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for SL of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sr_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sr_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sr_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sr_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sr_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li

sr_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sr_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for SR of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r1.asm	Instruction Fetch & Instruction Decode	Single instruction test for XOR of two operands using r1 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r2.asm	Instruction Fetch & Instruction Decode	Single instruction test for SRU of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r3.asm	Instruction Fetch & Instruction Decode	Single instruction test for SRU of two operands using r3 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r4.asm	Instruction Fetch & Instruction Decode	Single instruction test for SRU of two operands using r4 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r5.asm	Instruction Fetch & Instruction Decode	Single instruction test for SRU of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r6.asm	Instruction Fetch & Instruction Decode	Single instruction test for SRU of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
sru_r7.asm	Instruction Fetch & Instruction Decode	Single instruction test for SRU of two operands using r2 register as a destination and source operand 1 and source operand 2.	Chenyang Li
addi.asm	Instruction Fetch & Instruction Decode	Single instruction test for ADDI of two operands using r1 register as a destination and r2 as operand 1 and an immediate number.	Chenyang Li

Assertions

Property Name	Stage	Description	Owner
stall_length	Hazard Detection	Stall shouldn't last more than three cycles	Ignacio Genovese
stall_operation	Hazard Detection	Stall should be produced if a source operand is the destination operand for the current instruction in the EX, MEM or WB stage	Ignacio Genovese
R0_operation	Register File	Every time register 0 is read, the output should be 0	Ignacio Genovese
RF_write	Register File	If register write is enabled, then the reg_write_data value should be written to the register destination	Ignacio Genovese
RF_read	Register File	For each input register address, the output should be the content of that register	Ignacio Genovese
Mem_write	Memory	If write mem is enabled, then the mem_write_data value should be written to the memory address	Ignacio Genovese
Mem_read	Memory	The output of the memory should be the value stored in the indicated memory address	Ignacio Genovese
pc_increment	Instruction Fetch	If there's no stall and no branch is taken, the PC should increment on each cycle	Chenyang Li
pc_stall	Instruction Fetch	If there's a stall, the PC should remain stable	Chenyang Li
pc_branch	Instruction Fetch	If there's no stall and a branch is taken, the PC should change to the correct offset	Chenyang Li
id_stall	Instruction Decode	If there's a stall, the instruction register shouldn't change	Chenyang Li
id_op_stall	Instruction Decode	If there's a stall, the opcode should be 0	Chenyang Li
id_dest_stall	Instruction Decode	If there's a stall, the destination register should be 0	Chenyang Li
id_wb_en	Instruction Decode	Check that the correct value is given to the write_back_en according to the current opcode	Chenyang Li
id_wb_mux	Instruction Decode	Check that the correct value is given to the write_back_result_mux according to the current opcode	Chenyang Li
id_alu_cmd	Instruction	Check that the correct value is given to the	Chenyang

	Decode	ex_alu_cmd according to the current opcode	Li
id_alu_src2_mux	Instruction Decode	Check that the correct value is given to the alu_src2_mux according to the current opcode	Chenyang Li
id_op_src2	Instruction Decode	Check that the correct value is given to the decoding_op_src2 according to the current opcode	Chenyang Li
alu_add	Execution	Check that the output corresponds to the sum of the inputs	Aishwayra Doosa
alu_sub	Execution	Check that the output corresponds to the subtraction of the inputs	Aishwayra Doosa
alu_and	Execution	Check that the output corresponds to the and of the inputs	Aishwayra Doosa
alu_or	Execution	Check that the output corresponds to the or of the inputs	Aishwayra Doosa
alu_xor	Execution	Check that the output corresponds to the xor of the inputs	Aishwayra Doosa
alu_sl	Execution	Check that the output corresponds to the correct shift left of the input	Aishwayra Doosa
alu_sr	Execution	Check that the output corresponds to the correct shift right of the input	Aishwayra Doosa
alu_sru	Execution	Check that the output corresponds to the correct unsigned shift right of the input	Aishwayra Doosa
wb_data	Writeback	Check that the correct data is selected to be written back	Shouvik Rakshit

Verification Environment

For the first phase of the verification flow (top level verification) there will be a testcase folder containing each stage testcases (bench\top\testcases). Some of these testcases will be shared between stages, as they are the same (for example, every instruction decoding will include ALU instructions, or stalls for the IF stage will be produced in the hazard detection unit testcases). A top level testbench will be used, which will have a string array containing every testcase file path and the expected results (register file and memory contents) for every unit test (used at the end of each test for proving functional correctness).

For the second phase (testing each pipeline stage separately), there will be a folder for each pipeline stage containing the input/output files for stimulating and comparing results (results\saved_regs). Each independent stage will have its own separate testbench (inside the bench folder).

For both of these phases there will be assertions that run along each testbench.

Also, a Makefile will be used for compiling and running each testbench (in the sim folder).

Along with this, there will be a results folder containing the coverage, simulation and assertions results for each testbench, and the saved inputs and outputs of each stage.

Finally, for the third stage, this whole structure will be reproduce to test the “broken” design.

File structure

Folder	Contents
├── bench	
│ ├── EX	Execution stage testbench
│ ├── ID	Instruction Decode stage tetbench
│ ├── IF	Instruction Fetch stage testbench
│ ├── MEM	Memory stage testbench
│ ├── WB	Writeback stage testbench
│ ├── hazard_detection	Hazard Detection Unit testbench
│ ├── register_file	Register File testbench

│ └── top	Top level testbench
│ └── testcases	Assembly testcases and expected outputs
├── docs	Documents folder
├── mips_16	
│ ├── bench	Example testbenches
│ ├── doc	MIPS 16 documentation
│ ├── rtl	DUT source code
│ └── sw	Software tools, like java assembler
├── results	
│ ├── EX	Execution stage coverage reports
│ ├── ID	Instruction Decode stage coverage reports
│ ├── IF	Instruction Fetch stage coverage reports
│ ├── MEM	Memory stage coverage reports
│ ├── WB	Writeback stage coverage reports
│ ├── hazard_detection	Hazard Detection Unit coverage reports
│ ├── saved_regs	Files containing the outpus of each stage for ever testcase
│ └── top	Top level coverage reports
└── sim	
├── EX	Execution stage Makefile and simulation scrits
├── ID	Instruction Decode stage Makefile and simulation scripts
├── IF	Instruction Fetch Makefile and simulation scripts
├── MEM	Memory stage Makefile and simulation scripts
├── WB	Writeback stage Makefile and simulation scripts
├── hazard_detection	Hazard Detection Unit Makefile and simulation scripts
├── register_file	Register File Makefile and simulation scripts

├── top	Top level Makefile and simulation scripts
└── veloce	Veloce Makefile and simulation scripts

Testbenches

- *Top Level Testbench:* this testbench runs all of the testcases (loading the output of the java assembler into the instruction memory), compares the final state of the register file and memory with the expected results for each test and saves the inputs and the outputs of each stage for every test. *Owner*: All team members
- *Instruction Fetch Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Chenyang Li
- *Instruction Decode Testbench*: this testbench uses the saved inputs of the ID testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Chenyang Li
- *Execution Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Aishwayra Doosa
- *Memory Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Shouvik Rakshit
- *Writeback Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Shouvik Rakshit
- *Hazard Detection Unit Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Ignacio Genovese
- *Register File Testbench*: this testbench uses the saved inputs of the IF testcases to stimulate the instruction fetch stage and compares the obtained outputs with the saved outputs of the IF testcases. *Owner*: Ignacio Genovese

Team members responsibilities

For each stage, the designated member will create the assembly code, produce the prog file using the java assembler, create the expected register file and memory results and create and run each independent testbench.

- ID: Chenyang Li
- IF: Chenyang Li
- EX: Aishwarya Doosa

- MEM: Shouvik Rakshit
- WB: Shouvik Rakshit
- Hazard Detection Unit: R. Ignacio Genovese
- Register File: R. Ignacio Genovese
- Top level integration, makefiles, coverage and assertions results: R. Ignacio Genovese
- Veloce (set environment, run examples, run MIPS16 ISA): Aishwarya Doosa/R. Ignacio Genovese