Lab 1 Q&A Refinement

FAQ for part 1

(Q) How do I run a specific test file?
(A) Change `IDMEMINITFILE` in `define.vh` (line 21) to point to the desired `.mem` file (for example, `test/part1/test3.mem`), then run `make` in the `lab1` folder. Both `imem` and `dmem` use this same path, and the resulting `trace.vcd` corresponds to that test.

(Q) My code does not load any instructions / all GTKWave signals are zero. How do I fix that?
(A) This is usually caused by an incorrect `IDMEMINITFILE` or environment setup. First verify that `IDMEMINITFILE` points to a valid `.mem` file, then run `source /storage/ice-shared/cs3220/labs_setup.sh` in the same terminal before running `make`. If an unmodified reference clone with only `IDMEMINITFILE` changed works, the issue lies in your edits.

(Q) Debugging takes so much time. Any tips?
(A) See below:

1. Carefully review your code and the ISA behavior.
2. If `make` fails, read the error messages and fix syntax/width issues (e.g., LHS/RHS bit-width mismatches).
3. Use GTKWave to inspect `clk`, `PC`, and key stage signals (e.g., `regval1_AGEX`, `regval2_AGEX`, `aluout_AGEX`) and compare them to the `.asm` or a RISC-V emulator.

(Q) How do I know whether my implementation is correct?
(A) A correct implementation prints "Passed!" when you run `make` on the PACE environment. The exit code used to decide pass/fail is the final value of register `x3` (`gp`), as checked in `sim_main.cpp`.

(Q) Is lab 1 code hand-graded, and are there hidden test cases?
(A) There are no hidden tests. If your code passes all provided tests for a part on Gradescope, you receive full coding credit for that part.

(Q) Are Part 1 tests independent?
(A) Yes. Each test is independent; passing or failing one does not affect the others.

(Q) Do we need to implement a branch predictor?
(A) No. Branches in Lab 1 are always predicted not-taken; a branch predictor is part of Lab 2.

(Q) Do we need a hardware stack for nested `JAL`?
(A) No. The hardware does not manage call stacks; treating nested calls is the software's responsibility.

(Q) For `BEQ t1, t1, imm`, is the new PC `PC + imm` or `PC + 4 + imm`?
(A) Use `PC = PC + offset` where `offset` is the properly sign-extended branch offset. Be careful when converting the immediate to an offset.

(Q) Do I need to prevent writes to `x0` in hardware?
(A) No. Hardware does not need to enforce `x0` being zero for this lab; software conventions handle that.

(Q) Is the immediate in the assembly decimal or hex?
(A) If it starts with `0x`, it is hexadecimal; otherwise, treat it as decimal.

(Q) When accessing memory, why are the lowest 2 bits of the address dropped?
(A) The ISA is byte-addressable, but `imem`/`dmem` are word-addressable and assume aligned accesses. The provided code uses `PC[IMEMADDRBITS-1:IMEMWORDBITS]` / `memaddr_MEM[DMEMADDRBITS-1:DMEMWORDBITS]` to ignore the lower two bits; you do not need to change this.

(Q) What does `assign inst_FE = imem[PC_FE_latch[IMEMADDRBITS-1:IMEMWORDBITS]];` mean?
(A) It uses the (word-addressed) PC value, excluding the least significant 2 bits, to index into the instruction memory, which has size 2^{14} words.

(Q) How do I know the correct instruction behavior?
(A) Cross-check with the Tiny RISC-V ISA document (`tinyrv-isa.txt`) and/or a RISC-V emulator such as the Cornell interpreter linked from the README.

(Q) The assembly for Part 1 Test 5 uses `add` where it seems `addi` is intended. Will this break grading?
(A) No. That is a typo in the human-readable assembly; the `.mem` file contains the correct instruction sequence with `addi`, and grading uses the `.mem` code.

(Q) I am confused by `sxt_imm_AGEX` for tests 4 and 5. Should I modify it in Part 1?
(A) No. For Part 1 you only modify `agex_stage.v`. Test 4 and 5 differ in the actual distances encoded in the compiled `.mem` files, even if the original `.asm` looks similar. `sxt_imm_AGEX` reflecting `0x10` vs `0x8` is expected.

(Q) How detailed should my Part 1 written explanations be?
(A) Briefly but clearly describe the behavior of the key signals (e.g., stage inputs/outputs, hazard handling, branch behavior) and support your explanation with waveform screenshots. Concise, correct explanations are sufficient.

(Q) I feel lost on how to start Lab 1. Any suggestions?
(A) Review the pipeline lectures, skim all five pipeline stage files to understand what is already provided, then focus on the `TODO` items in `agex_stage.v`. Reading the other stages helps you see how AGEX fits into the full pipeline.

FAQ for part 2

(Q) Which instructions must be implemented for Part 2?
(A) You must support enough of the RV32I subset to pass all Part 2 tests. This includes arithmetic (e.g., `add`, `addi`, `sub`), branches (`beq`, `bne`, `blt`, `bge`, `bltu`, `bgeu`), jumps (`jal`, `jalr`, `jr`), and immediates (`auipc`, `lui`). If a specific test fails, check its `.asm`/`.dump` to see which instruction you are missing or mis-implementing.

(Q) My Part 2 tests partially pass (e.g., 8/10) even with only Part 1 logic. Is that expected?
(A) Yes. Some tests rely only on the instructions already implemented for Part 1, so they may pass even without full Part 2 support. You still need to debug the failing tests to get full credit.

(Q) Do I need to implement bypassing for Part 2?
(A) No. You may rely on pipeline stalls as long as you correctly detect all hazard conditions (including those involving `JAL` and `JALR`) and maintain correct behavior.

(Q) I am confused by macros and the structure of Part 2 tests. How should I debug them?
(A) The tests reuse RISC-V test-suite code that uses macros. To debug:

1. Set `IDMEMINITFILE` to the failing test's `.mem` file.
2. Run `make` and open the resulting `trace.vcd` in GTKWave.
3. Track the PC to find the sub-test where behavior diverges.
4. Compare against the `.dump`/`.asm` version of that sub-test and optionally run it in the online RISC-V interpreter to see expected behavior.

(Q) In signed vs unsigned comparisons (`bge` vs `bgeu`, etc.), how should I treat operands and immediates?

(A) In Verilog, declare wires as `signed` when you want signed interpretation; the underlying bits are unchanged. Use signed comparisons for `bge`/`blt` and unsigned for `bgeu`/`bltu`. Immediate values are always sign-extended in RISC-V, but you may still need `signed` on wires when comparing them as two's complement.

(Q) My Part 2 test 7 starts at PC `0x200`. Is that correct?
(A) Yes. The first instruction is placed at address `0x200` (for example, line 128 in the `.mem` file corresponds to `128 × 4 = 512 = 0x200`). The online simulator may display a different starting address, but that does not affect your design's correctness.

(Q) There seems to be a strange `jal` offset convention in `.dump` files. Does that affect my implementation?
(A) No. Some `.dump` files annotate `jal` with the target address instead of the raw immediate value, but the encoded hex in the `.mem` files is correct; implement `jal` as `PC + sext(imm)` and your design will be graded against the correct hex code.

(Q) I see comments like "expand this always block" or "complete the rest of the pipeline" in `de_stage.v` or `fe_stage.v`. Are these required for Parts 2–3?
(A) No. For the current lab, you can pass Parts 1–3 by focusing on the items marked with `TODO`. The older "expand" comments are historical and can be ignored; the repo has been updated to clarify this.

FAQ for part 3

(Q) How do I debug Part 3 tests that use many macros?
(A) The Part 3 tests consist of multiple sub-tests built from macros. Use the same approach as for Part 2: set `IDMEMINITFILE` to the failing test's `.mem`, generate `trace.vcd`, track PC to identify the failing sub-test, then compare against the `.dump`/`.asm` and the RISC-V interpreter's behavior.

(Q) How is Part 3 extra credit applied to my grade?
(A) Part 3 points are proportional to the fraction of Part 3 tests you pass (e.g., 24/30 tests yield $20 \times 24/30$ bonus points), but your overall Lab 1 score is capped at 100%. Extra credit cannot push Lab 1 above full credit.

(Q) I passed 30/30 Part 3 tests earlier but now see fewer passing. What might cause this?
(A) This usually indicates that some files were changed after your earlier run. Re-run the tests and compare your current code to the version that previously achieved 30/30.

(Q) For instructions like `slti` and `sltiu`, what should the result be?

(A) The result is either 0 or 1, depending on whether $R[rs1] < \text{sext}(imm)$ using signed comparisons for `slti` and unsigned for `sltiu`.

General FAQ (all parts)

(Q) The RISC-V RV32I manual link in the README is inaccessible. What should I use instead?
(A) The external manual is optional. Your primary reference is the included Tiny RISC-V ISA file `tinyrv-isa.txt`. If needed, you can access archived versions of the original manual via the Web Archive.

(Q) I get errors like `verilator: command not found`, Makefile Error 127, or `ccache: No such file or directory`.
(A) These errors almost always mean the lab environment is not initialized. Run
`source /storage/ice-shared/cs3220/labs_setup.sh`
in your terminal (with a space after `source`) before `make` or `./run_tests.sh`. On local machines, you can temporarily disable CCache with `export CCACHE_DISABLE=1` if needed.

(Q) Is it okay to develop locally instead of using the PACE interactive desktop?
(A) Yes. You may develop on WSL, a Linux VM, or via VS Code tunnels (with local installs of Verilator/GTKWave). However, grading is done on the PACE environment, so you must verify that your final code passes the tests there.

(Q) Why do `run_tests.sh all` and `run_tests.sh part4` fail?
(A) Those options are deprecated and should be ignored; they are not used for Lab 1 grading.

(Q) The log shows “Total instructions = 0, cycles = 0, IPC = -nan” but the test says “Passed!”. Is that okay?
(A) Yes. As long as “Passed!” appears, the test is considered successful; the instruction and cycle counters are not used for grading.

(Q) How is grading computed across parts, and what does my test coverage imply?
(A) Coding points for each part are proportional to the fraction of that part’s tests you pass (for example, 9/10 Part 2 tests → 90% of Part 2 coding points). There are no hidden tests; grading uses exactly the provided test sets.

(Q) How do I submit my work on Canvas?
(A) Submit your Part 1 PDF writeup as a separate file on the same assignment.

(Q) Can I request extensions or late writeup submissions for Lab 1?
(A) Extensions are generally not granted once the lab schedule is fixed. Late writeup submissions after the posted deadline are not accepted, but later labs may offer bonus opportunities to offset missed written points.