CS209P Project Phase 2

February 23, 2025

- Commit your phase 1 with a tag `phase1` and push it to your repository. Phase 1 evaluation will be based on the code till the tag `phase1`. The release should have been created before the deadline of Phase 1.
  `https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases`
  `https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a`
- Extend the simulator that you developed in Phase 1 to incorporate pipelining (with data forwarding).
- The user should be given an option to enable or disable forwarding.
- The arithmetic instructions can have variable latencies. The user should be able to specify the latencies for each instruction. For example, the user should be able to specify that the latency of `ADD` is 1, `MUL` is 3. You can design the interface for this as you see fit.
- At the end of the execution the simulator should output the number of stalls, and the IPC (Instructions per Cycle).
- Important: In the previous phase you had four compute units. Now is the time to merge them! How? There will be only one fetch unit for all the compute units. But each compute unit will have its own decode/register fetch, execute, memory and writeback stages. The compute units will share the instruction memory and data memory. How does this work? Remember that each compute unit had a special purpose register, let us refer to it CID. Assume the following code snippet:

```
    ADD x2, x3, x4 # all compute units will execute this
       instruction
    BNE cid, 1, Label # all compute units will execute this
       instruction, but only the compute unit with CID=1 will
       take the branch
    ADD x5, x6, x7 # all compute units will fetch this
       instruction because there is only one fetch unit. But CID
       =1 will not execute this instruction. The instruction is
       simply ignored after the fetch stage in CID=1. The other
       compute units will execute this instruction.
Label: ADD x8, x9, x10 # all compute units will execute this
    instruction
```

- One of the test cases for this phase will be array addition. The below example is given in C code, but a direct RISC-V assembly code can be used as a test case.

```
for (i = 0; i < 100; i++) {
  sum += a[i];
}
print(sum);
```

All the four compute units should be able to execute this code. But compute unit 1 will compute the sum of the first 25 elements, compute unit 2 will compute the sum of the next 25 elements, and so on. The final sum should be the sum of all the elements in the array. Only compute unit 1 will print the sum. You can convert the above C code to RISC-V assembly code as you see fit.

- If you are unable to complete the project, do prepare a document detailing what you tried, and what did not work etc. The document along with the incomplete code will be evaluated.
- **Deadline**: March 10^{th} 11:59PM.
- Please note that there will be no extensions this time.