CP1

Goal: basic instructions

Difficulties encountered:

1. Everything from scratch. As there is no state diagram will be allowed, we have to implement a completely new pipelined CPU using a control word which is passed down the pipeline. To start a new project means a lot of codes to write…but we get it done.
2. The control word. In the beginning, we did not quite understand how the control word work and especially what info needs to be present in the word and how can it be set. The solution comes out when we decided to make each stage of pipeline, IF, DE, EX, MEM and WB, a module itself and use a higher level data path to connect all of them. This comes handy because we can simply put the *control_rom* module inside the decode_stage module with inputs buffered from the fetch stage.
3. Synchronization. Although we are provided with the *magic_dual_port_mem* module which does help a lot, we still have some synchronization problems. We found this problem when doing *op_load* that the pc value is not correct. The solution is to make stage register between the fetch and decode stage use a load signal of *resp_a* all together.
4. Testing. For the given testcode, *mp3-cp1.s,* our design seems not to understand the %lo(label)(x0) command. So we modified into a new test called *test2.s* which is basically the same as given testcode by removing %lo and %hi.

Timeline:

Mar. 12: basic file added
Mar. 15: CPU datapath updated
Mar. 16: test1.s and test2.s tested
Mar. 17: roadmap, progress report, and cache design

CP2

Goal: L1 split cache, arbiter, all RISCV instructions

Difficulties encountered:

1. Iteration limit reached. In our *IF_stage*, we previously used combinational logic between *i_resp*, *load_pc*, and *i_read*. This is the place we found that *i_read* will depend on *i_resp* which is output and input respectively so that a combinational loop has occurred.
2. Stage inconsistency. This has always been a problem since the beginning. This checkpoint we developed a new strategy by making every load signal of each stage register dependent on the next state's load signal.
   Mar.24 added: when *stage_wb* is stalling, added a *stall_all* signal in datapath to stall all stages.
3. AUIPC. This opcode comes to a problem when we are doing stores. As all stores begin with *op_auipc* while the *ir[11:7]* for stores are not *rd* but *imm* so that immediate value will be written to source register. A temporary solution is to turn of *load_regfile* signal in *op_auipc.*
4. Cache pipeline. Unfortunately, we are unable to implement a pipelined cache in this checkpoint. The temporary solution is to set *i-read* <= *i_resp* with one clock cycle delay so that the cache won't be responding more than one addresses.

Timeline:

Mar. 18: arbiter & cache modification
Mar. 19: fixed combinational loop logic and instruction cache works
Mar. 21: d cache works with self-written testcode.
Mar. 22-25: Branch operation debugging & pipeline stage registers synchronization.