**CSCE 3301 – Computer Architecture**

**Spring 2020**

**Project: femtoRV32**

**RISC-V Implementation and Testing**

**Milestone 4**

Maha Shata 900170473

Marwah Sulaiman 900172284

Sarah Ali 900172348

Dina Wahdan 900142379

**Objective:**

The objective of this milestone is the implementation of the pipelined RISC V, entailing the hazard detection and testing all instructions.

**Steps for Implementation:**

1. Hazard Detection unit: stall the pipeline for one clock cycle in the case of load-use hazards.
2. Forwarding unit: solve read after write hazards
3. Flushing pipeline: it happens in the case of branching when the instructions that entered are wrong so we needed to flush the pipeline for three clock cycles, a multiplexer is used to either take the normal value or take zero (if the signal is wrongly predicted / branches to be taken and the next instruction will not be the one after it immediately, we will flush the pipeline). Therefore, we made the selection line to take nop instruction in the IF/ID stage and also the control signals to be equal to zero.
4. Testing: created different instructions to make sure of the functionality of all the components and that the whole datapath is working.
5. Bonus: First

* Compressed Instructions: convert the 16-bits instructions to 32-bit equivalent for RV32I, and also the way to differentiate between both instructions the 2 least significant bits
* Increment the PC by either 2 ,for the compressed instructions in order not miss half of the next instruction if it isn’t compressed or all of it if it’s compressed. If not it will be incremented normally by 4.
* The RISC-V Instruction Set Manual document was used to determine whether an instruction is supported by the RISC V 32I set. ( tables include all opcodes).
* We did a decoder, depending on the input instruction. We decoded the 16 bits into 32 bits and we didn’t change anything in the datapath.
* Module Idea: check the first 2 bits of the opcode that will come out from the instruction memory. If the first 2 bits are not 11, this means it is a compressed instruction and then check if they are of opcodes 00, 01 or 10. Then case statements and conditional statements were implemented inside the module, to differentiate between the instructions having the same opcode. Based on this we will make the right output instruction.

*Second:* Creating a comparator module in order to calculate the branch output in the ID stage and not in the execution state, that would cause the hazard detection unit to need to stall for one or two clock cycles depending on the instruction used, however, that would save the processor from the two clock cycles flushing need to be done when the branch was calculated in the execution state, so the processor should be more accurate and efficient.

Another forwarding unit was implemented in the ID stage in order to pass the values in the execution stage and the memory stage to the decoding stage.

The code has been implemented (The comparator and the forwarding unit). However, due to lack of time, the Hazard detection unit wasn’t updated to handle the new issues.

The below figures show the compressed instructions supported by RISC V:

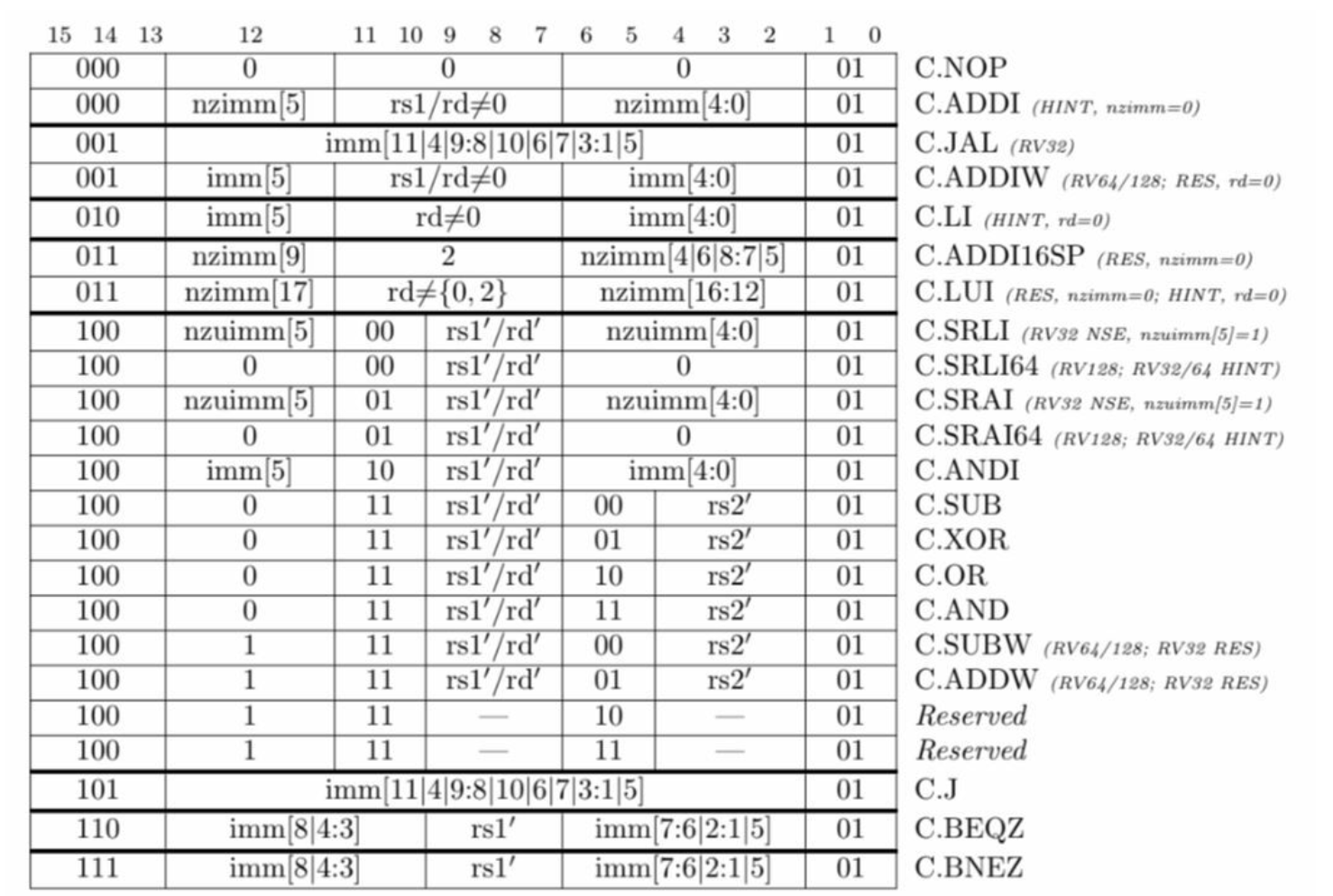


Figure 1

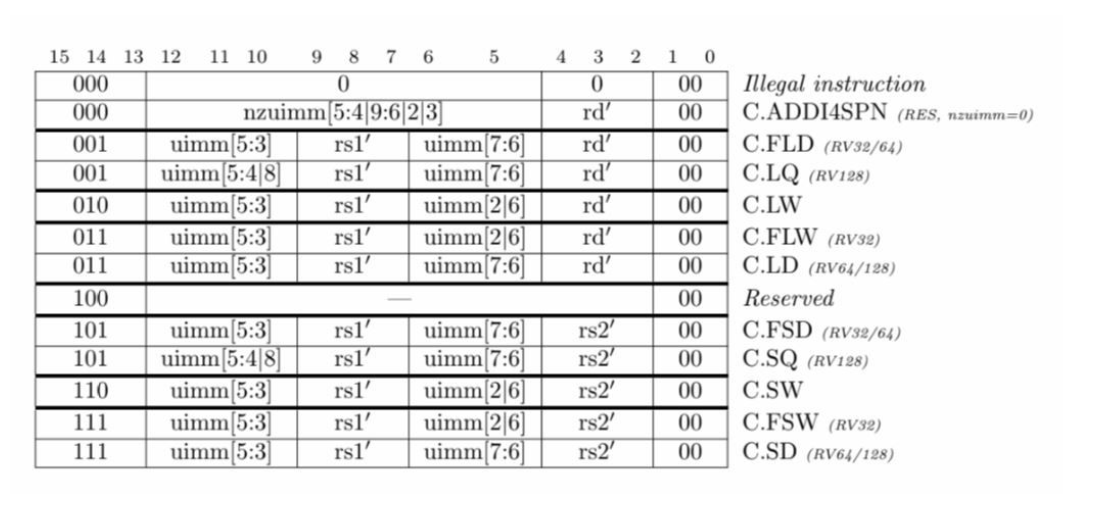


Figure 2

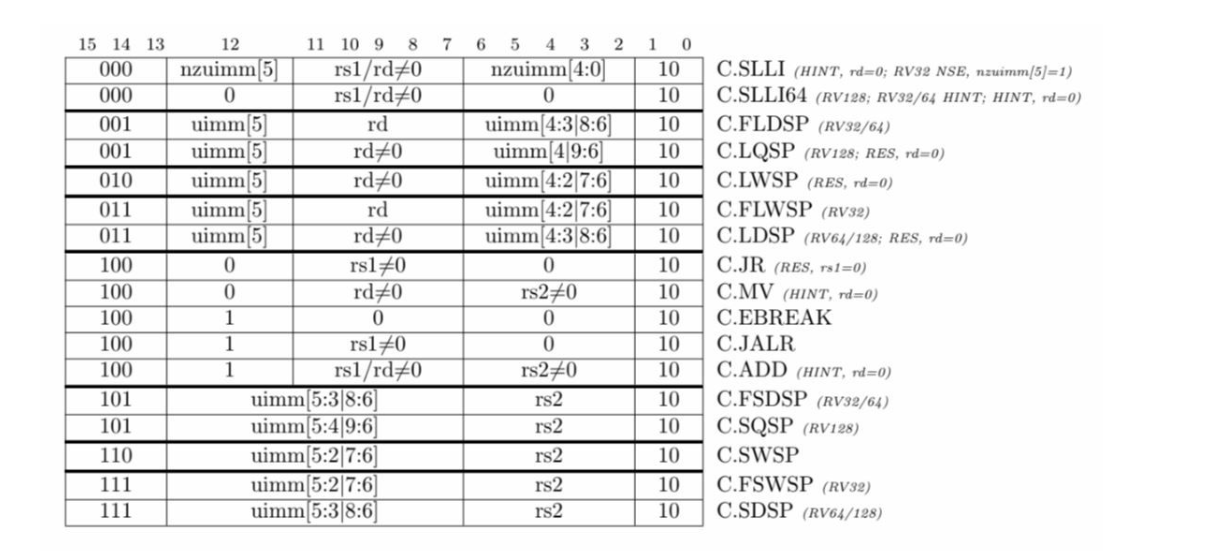


Figure 3

**Datapath:**

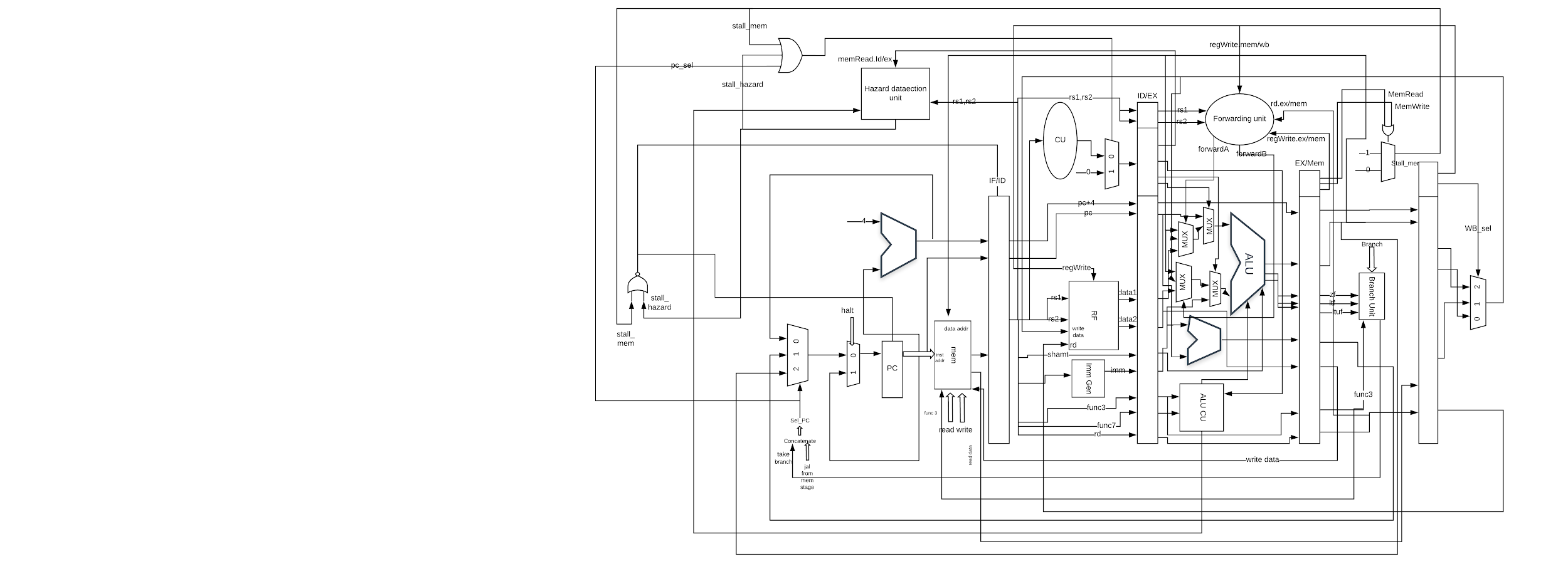


Figure 4: Datapath