**Final Project: MIPS Five-stage Pipelined CPU with Forwarding**

**Max Guthrie**

**CMPEN331**

**December 16, 2020**

**Abstract.**

The five-stage CPU built with Verilog handles MIPS instruction formatting in a pipelined fashion. The five stages of the processor consists of the instruction fetch (IF) stage, the instruction decode (ID) stage, the execution (EXE) stage, the data memory (DM) stage, and the write-back (WB) stage. Each stage is separated by pipeline registers to store and prepare the instructions for the next stage. 32-bit instructions are stored in and fetched from an instruction memory located in the IF stage, a register memory for quick data storage is located in the ID stage, and a data memory unit resides in the DM stage. This CPU also takes advantage of a forwarding unit to handle data hazards, allowing EXE or DM results to bypass for use in future dependencies. With forwarding implemented, the CPU is able to handle data hazards such as the ones depicted below.

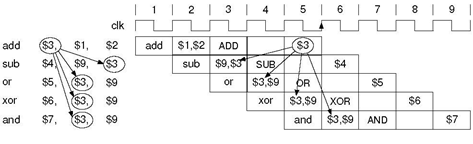


Figure 1: Instruction set with data hazards.

The instruction set shown above resides in the instruction memory and is used to validate the CPU’s forwarding efficacy. The results of this are shown in the waveform section.

**Introduction.**

Starting with the IF stage, the program counter (PC) register is initialized at the first address of the program’s first instruction. At every clock cycle, which is when a new instruction can be fetched, 4 is added to the current PC value and restored in the register. As seen in figure 2 below, updating the PC requires using an adder, and the current PC value is used to retrieve instructions. The 32-bit instruction is then held in the IF/ID register, preparing for decode.

In the decode stage, retrieving register values from the register file (RF) and control unit (CU) signals are the most important aspects. The control unit sends a specific combination of control signals depending on the instruction’s opcode and function code. These signals are instrumental in later stages for choosing ALU inputs, ALU operations, and whether a register needs to be rewritten from the EXE stage or MEM stage. In the case of load/store instructions, the immediate value from the instruction must be extended to 32 bits in order to properly work in the ALU. Otherwise, the CU’s ealuimm will be low and choose register data qb for the ALU input. The CU signals are stored in the EXE/MEM register, along with the ALU result, the destination register, which depends on the regrt signal, and the qb data for the case of store instructions.

As seen in the MEM stage of figure 2, the data memory (DM) can either be read or written depending on the mwmem (wmem from CU) signal. Store instructions, for example, cause a high mwmem signal, whereas load instructions would only read the DM. If the instruction requires writing back to the register file, the WB stage is crucial.

As shown in figure 2, the WB stage invovles a mux to choose if either read data or the ALU result should be written back. In the case of figure 1, which are R-type instructions, wm2reg (mem to reg) will be low and choose the ALU result to be written to RF. For simultaneous read-writes in the RF, which will happen quite frequently with the CPU and necessary for proper forwarding, the RF was designed to write values in the first half of the clock cycle and read values in the second half. This allows for the same data to be written and then read in the same cycle.

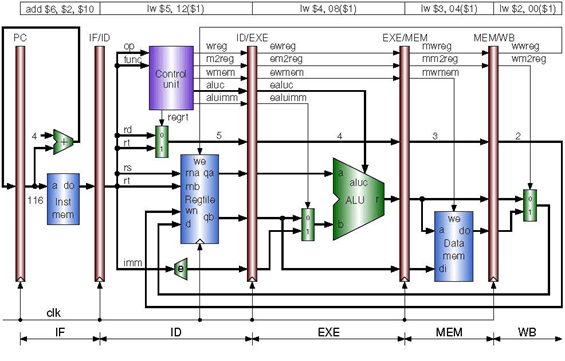


Figure 2: MIPS pipelined CPU implementation (No forwarding)

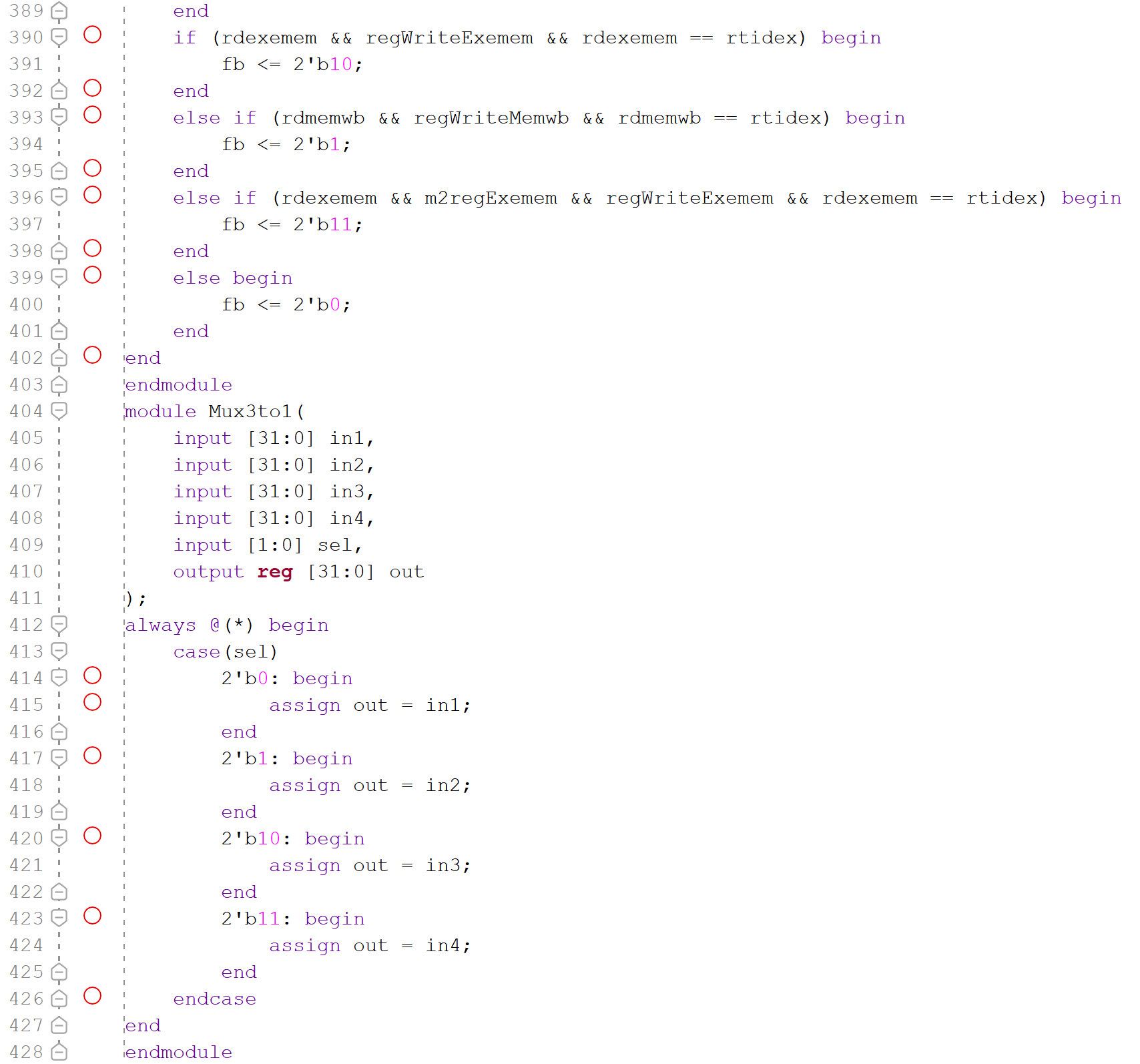
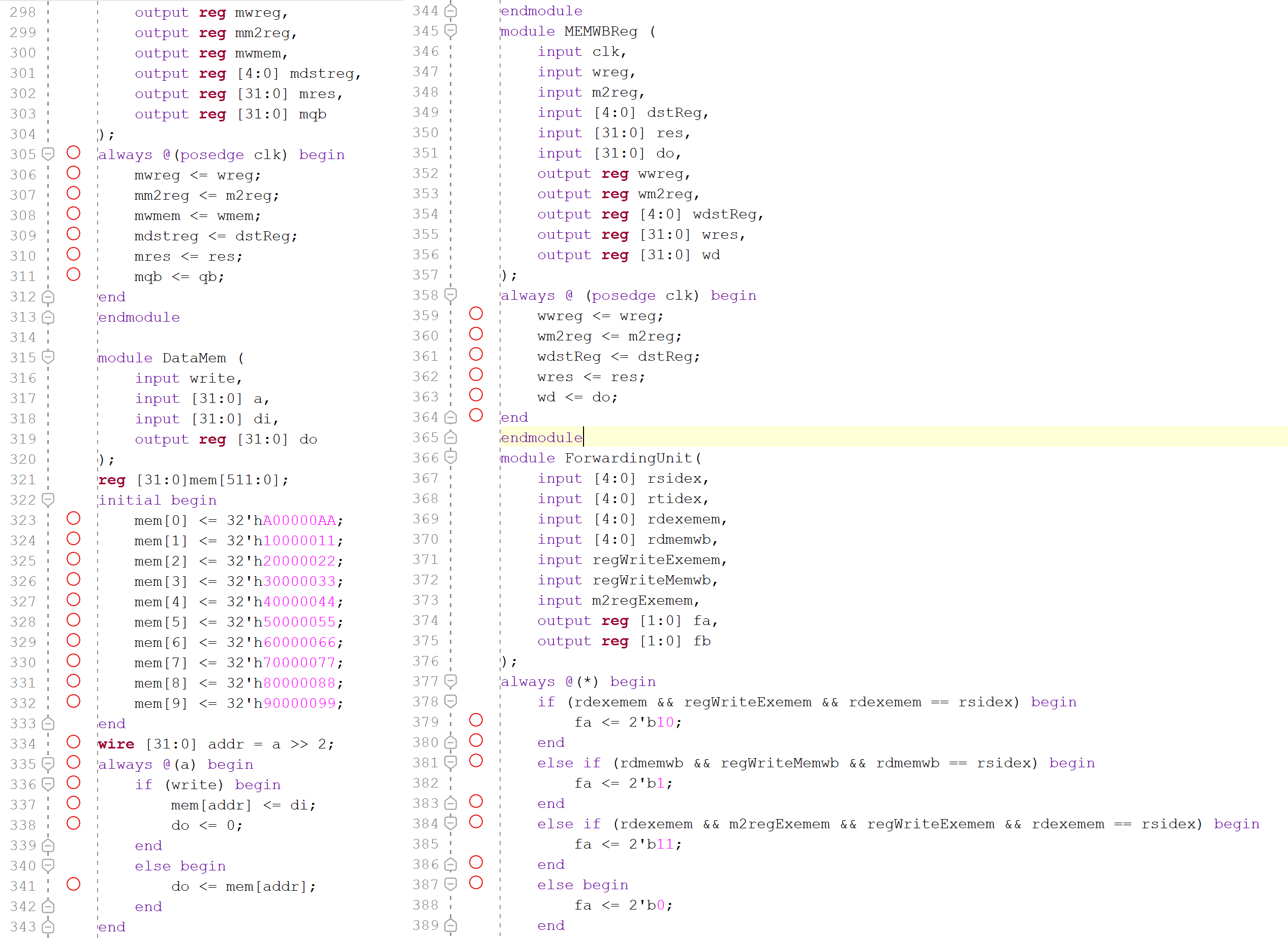
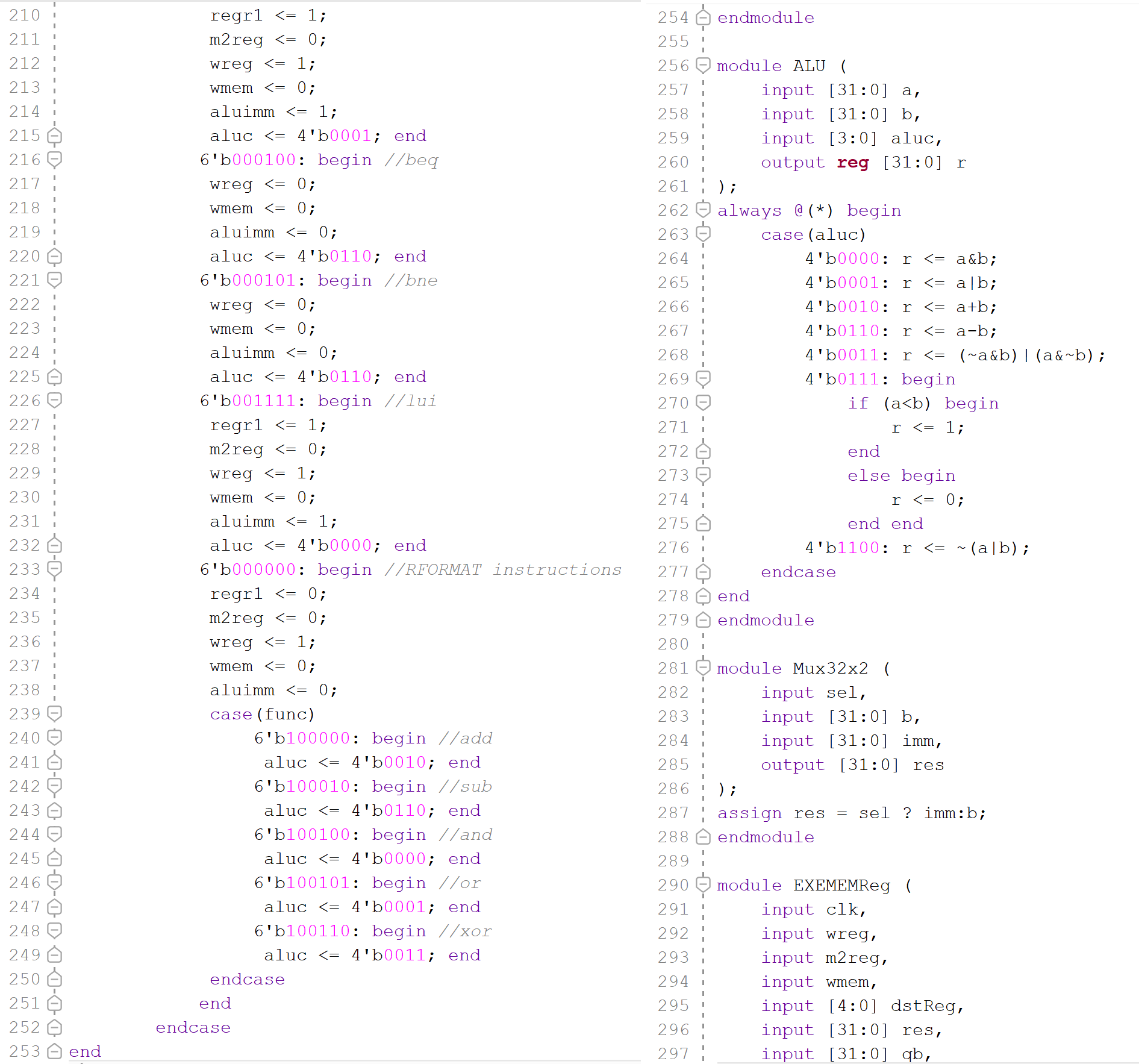
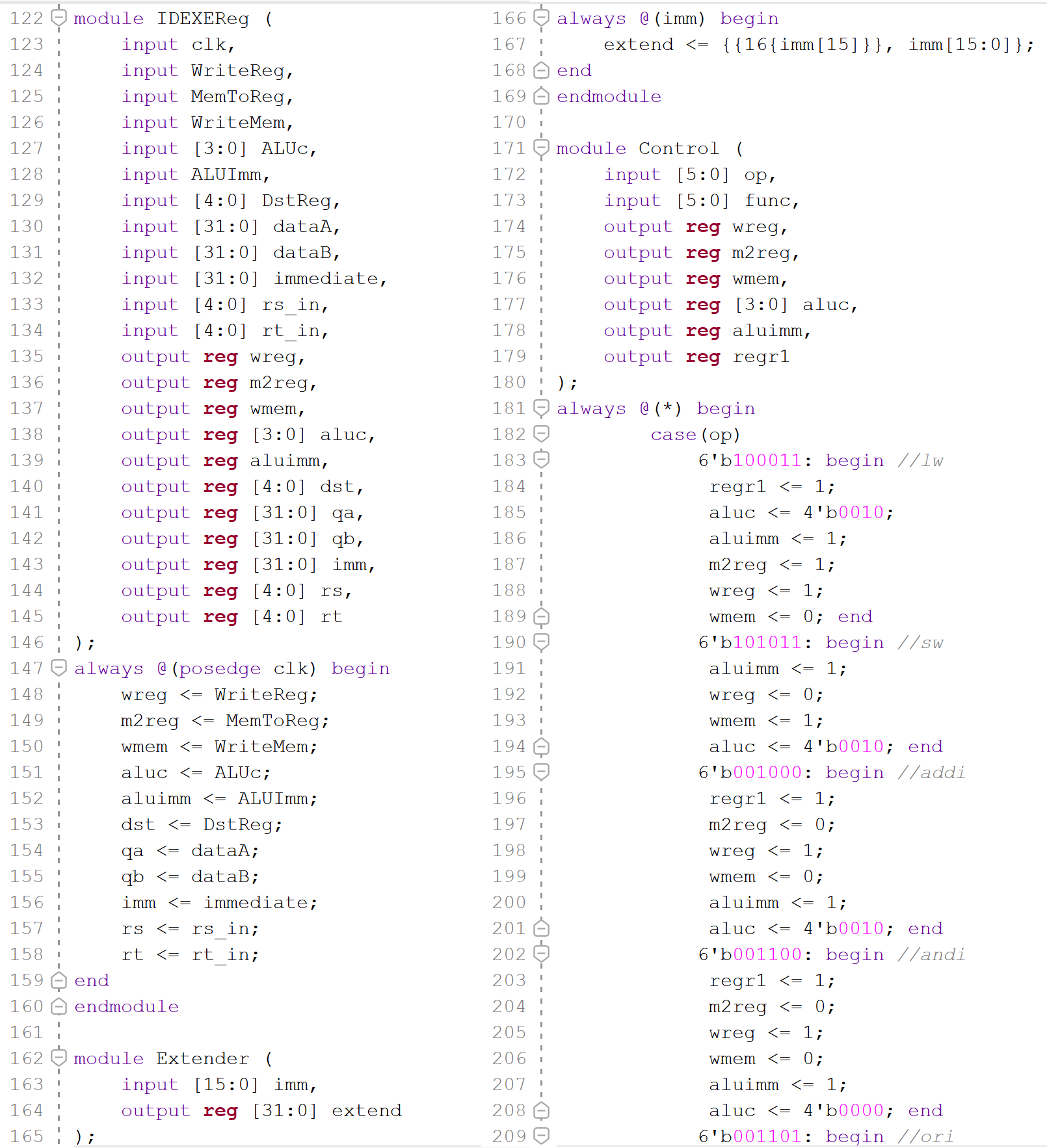
The above CPU design will handle a large chunk of R-type and I-type instructions. Data hazard handling was also implemented with the use of a forwarding unit (FU), which compares the destination registers in the EXE/MEM and MEM/WB to the source registers used by instructions in the ID/EXE. When either of these equalities are reached (and given that mwreg or wwreg is high), a data hazard has been detected, indicating the need to forward. Shown in figure 3 below, forwardA and forwardB signals come from the FU and use two mux’s to choose between the ID/EX qa and qb values, the previous cycle’s ALU results, or the writeback result. This design alows for immediate register use after they have been changed and eliminates the need for stalling.

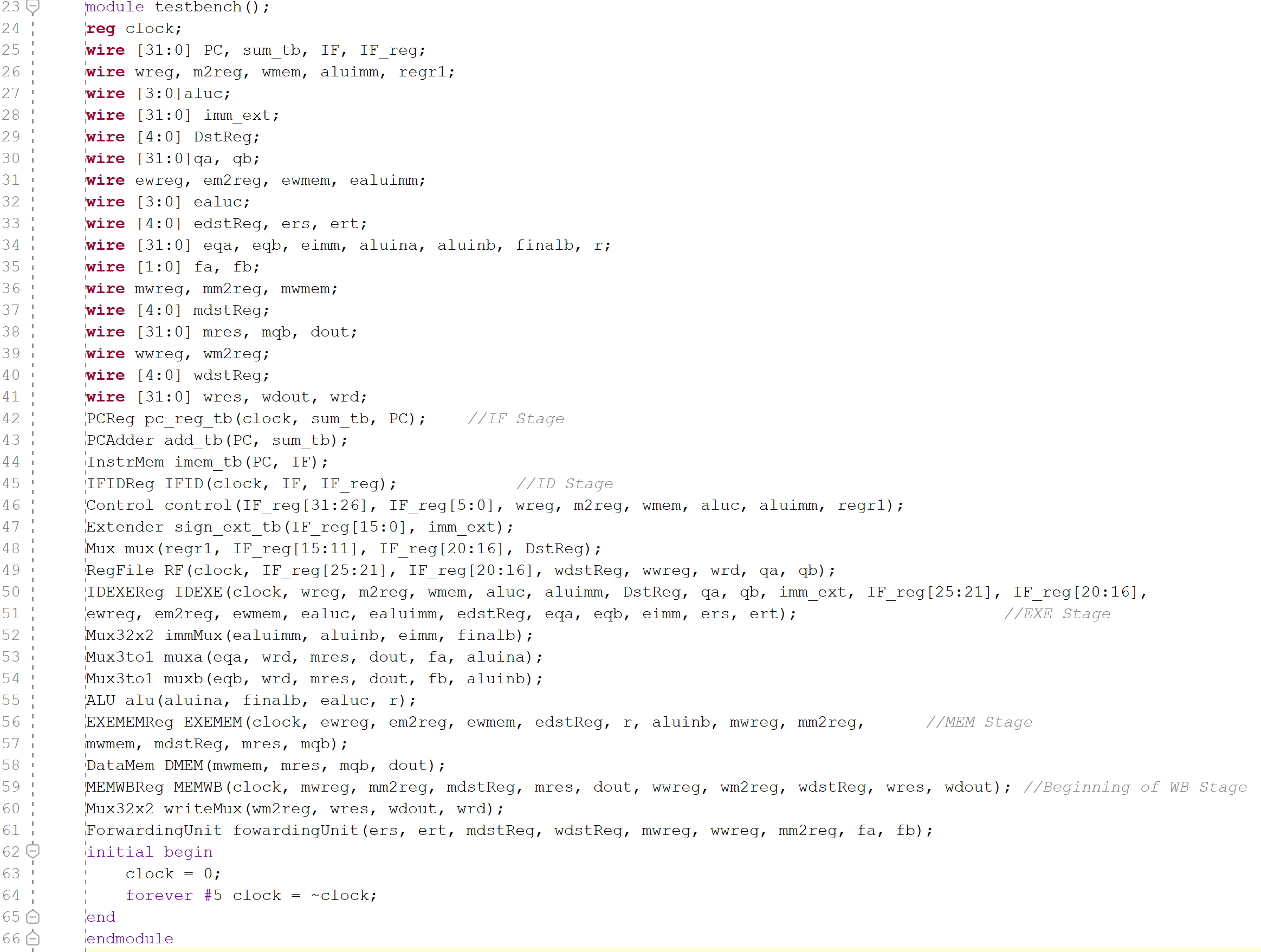
Diagrams of a pipeline before and pipeline after the forwarding unit is added. In the figure, item a is a diagram of pipeline registers with no forwarding. There is a registers unit, ID/EX register, ALU, EX/MEM register, Data memory unit, MEM/WB register, and a multiplexor. The Registers unit has four input lines, one of which comes from the multiplexor output. The Registers unit has two output lines that are inputs to the ID/EX register. The ID/EX register has two output lines that are inputs to the ALU. The output line of the ALU is the upper input to the EX/MEM register. The lower output of the ID/EX register is the lower input to the EX/MEM register. The output lines of the EX/MEM register are inputs for data memory. The output line of data memory is the upper input to the MEM/WB register. The upper output of the EX/MEM register is the lower input of MEM/WB. The two output lines of the MEM/WB register are inputs to the multiplexor. Item b of the figure is a diagram of pipeline registers with forwarding. Item b is similar to item a with the addition of three multiplexors, a forwarding unit, and more data lines. The upper output line of the ID/EX register is an input to the top multiplexor between the ID/EX register and the ALU. The next output line of the ID/EX register is an input to the bottom multiplexor between the ID/EX register and the ALU. The ID/EX register now has four additional outputs labeled Rs, Rt, Rt, and Rd that are located at the bottom. The Rt and Rd outputs are inputs to the multiplexor between the ID/EX and EX/MEM registers. The output of this multiplexor is an input the EX/MEM register. The output of the top multiplexor, between the ID/EX register and the ALU, is the upper input of the ALU. The output of the bottom multiplexor, between the ID/EX register and the ALU, is the lower input of the ALU. The output of the bottom multiplexor is also an input of the EX/MEM register. The output line of the ALU is the top input of the EX/MEM register. The upper two outputs of the EX/MEM register are inputs of the data memory. The top output of the EX/MEM register is an input of the MEM/WB register and an input of both the top and bottom multiplexors between the ID/EX register and the ALU. The bottom output of the EX/MEM register is labeled EX/MEM.RegisterRd and is the bottom input of the MEM/WB register as well as an input of the forwarding unit. The data memory output is still the top input of the MEM/WB register. The upper two outputs of the MEM/WB register are still inputs to the multiplexor that was also in item a. The MEM/WB register has an additional output at the bottom labeled MEM/WB.RegisterRd, which is an input of the forwarding unit. The output of the multiplexor which was also in item a is an input to the registers unit and an input of both the top and bottom multiplexors between the ID/EX register and the ALU. The output of the forwarding unit labeled ForwardA is the select line for the top multiplexor between the ID/EX register and the ALU. The output of the forwarding unit labeled ForwardB is the select line for the bottom register between the ID/EX register and the ALU.

Figure 3: Fowarding implementation with forwarding unit (Zybooks 4.7)

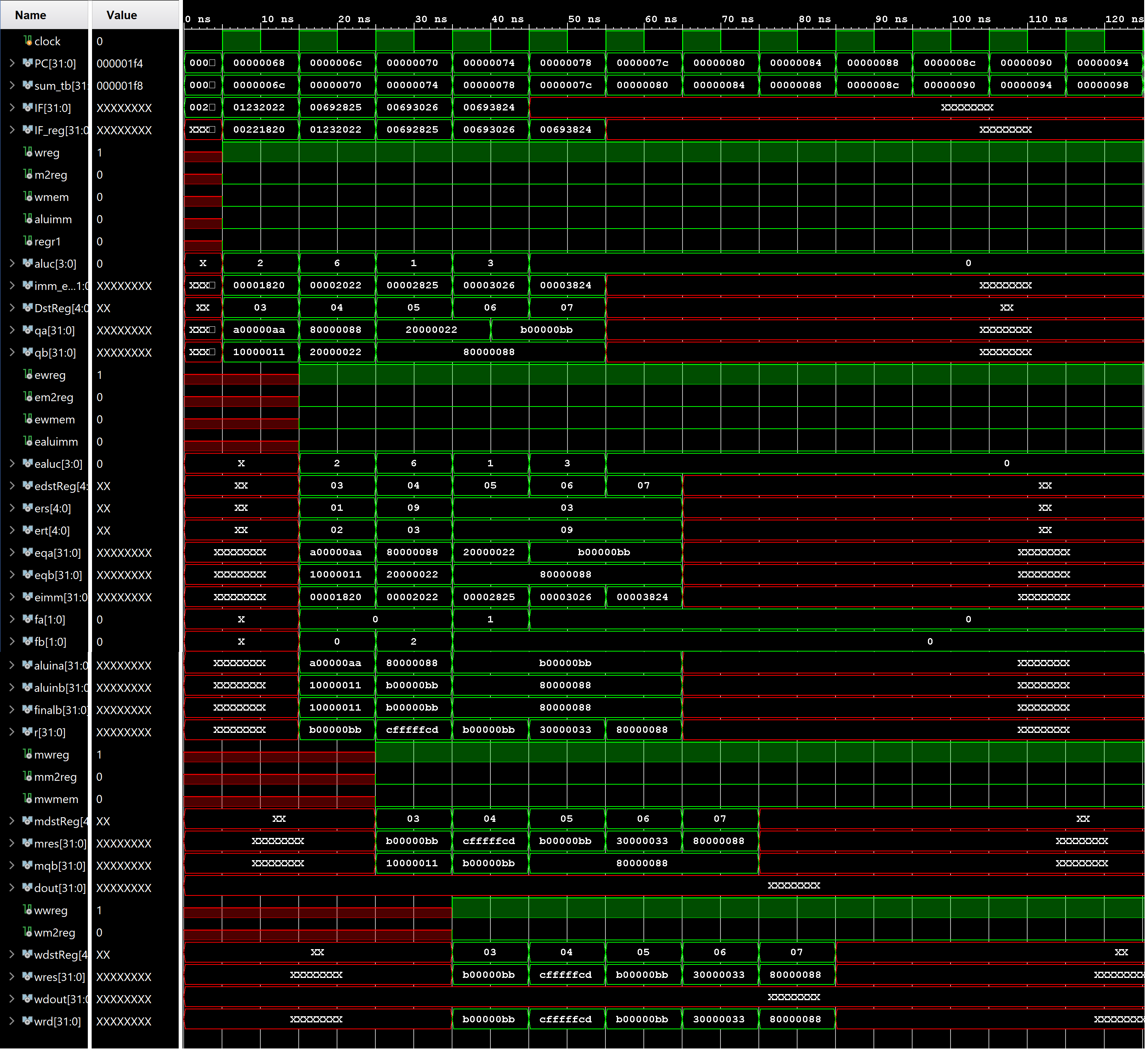
The combined design of figure 3 and figure 2 bring many benefits to CPU performance. The MIPS ISA has a simple decoding process, which keeps the CPU design simple and thus less prone for errors and hazards. The pipelining structure allows for a faster throughput of instructions without having to decrease the execution time of an individual instruction. If the program has many instructions, the CPU can effectively execute one instruction per cycle. Although pipelining introduces new hazards, implementing controls for those hazards does not significantly decrease throughput and thus increases the performance of the CPU.

**Verilog Design Code for all Stages.**

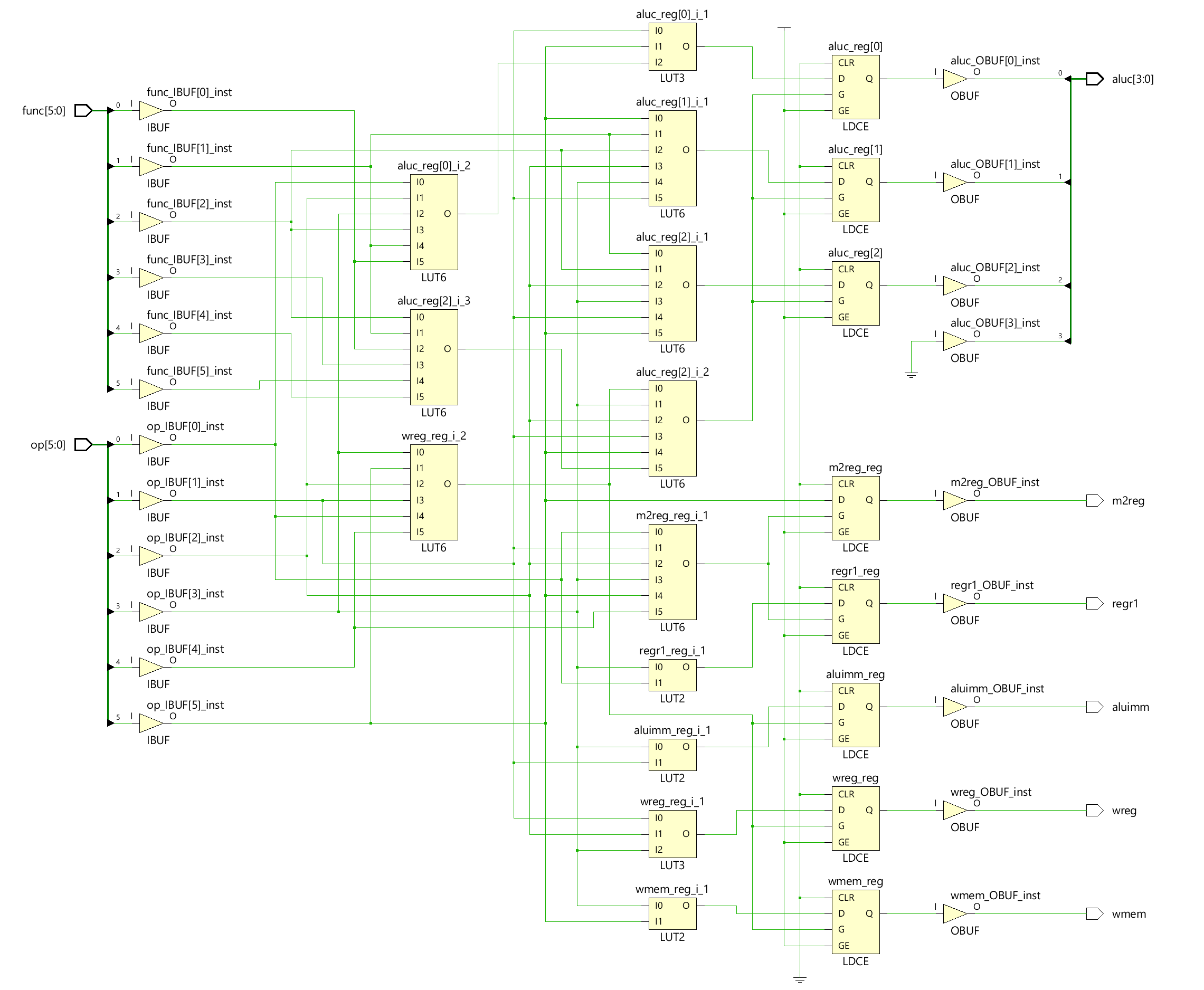


**Verilog Code for Testbench.**

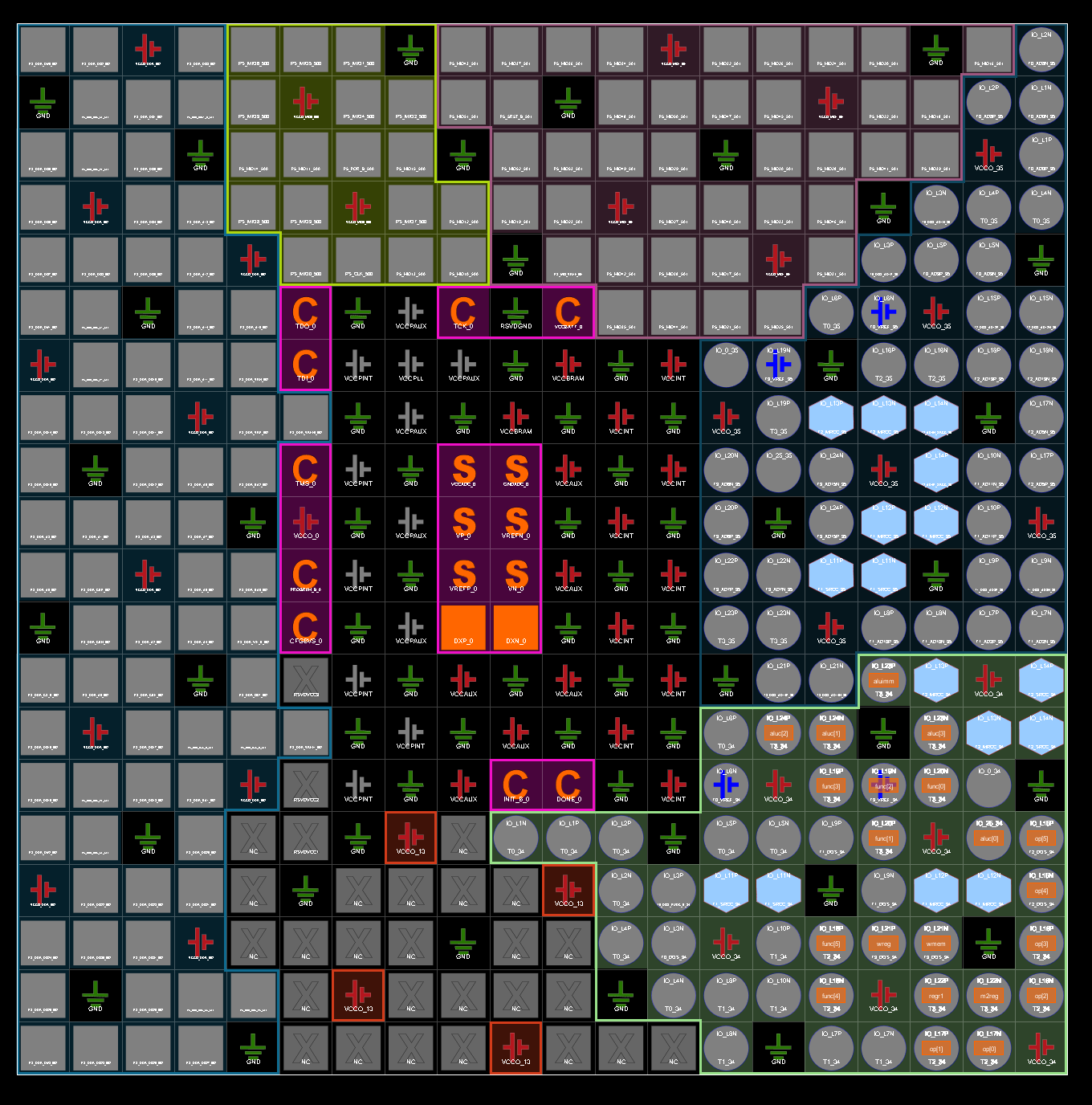
**Waveform.**



**Design Schematics.**



**I/O Planning.**



**Floor Planning.**

