Design and Implementation Of 5 Stage 32-bit Pipelined MIPS Processor

Jiatao Kuang, Pooja Songadkar and Honghao Zhu

Seidenberg School of Computer Science and Information Systems - Pace University

**Abstract.** The main objective of this project is to understand the basics of pipelining and the implementation of the MIPS pipeline. In this project we will design a 5-stage pipelined MIPS processor, using Verilog VHDL. The 5 stages of the implementation are Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Write Back (WB). The instruction set being used is of 32-bits. The various modules being used are Instruction Memory, Data Memory, ALU, Registers etc. The aim is also to solve a data hazard by implementing a forwarding or stalling mechanism.

**Introduction**

The hardware component of a computer system that executes commands from a computer program is known as the Central Processing Unit, or CPU. Popular processors come in two primary categories: RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computer) processors. There are numerous unique multi-clock complicated instructions available in CISC. In contrast to RISC, this type of CPU is slower, but it also employs fewer instructions. The RISC processor employs one-size, generally simpler and faster instructions. As a result, theoretically, it consumes fewer transistors, making the design of RISC processors simpler and less expensive.

The R-type (Register type), I-type (Immediate type), and J-type instructions are the three different types of instructions that can be used in a MIPS (Microprocessor without Interlocked Pipeline Stages) architecture (Jump type). R-type instructions are classified as those that perform mathematical or logical operations between two registers. I-type instructions work with a register and an immediate number to generate a memory address for a load or store instruction and an arithmetic result for operations that need to be completed right away. I-type instructions also include conditional branch instructions. The J-type instruction accepts a pseudo target address and unconditionally branches.

**Pipelining**

Pipelining is an implementation technique where in multiple instructions are overlapped during execution; it takes advantage of parallelism that exists among the steps needed to execute an instruction. All the latest processors incorporate pipelining as a key implementation technique. MIPS is a five-stage pipelining structure; where each stage is responsible for completing a part of an instruction. These five stages are connected through the pipelining registers. The throughput of the pipeline is determined by how much time an instruction takes to be executed. The time required to move an instruction one step down to another stage among five stages sequentially is known as ‘processor cycle’. The slowest pipeline stage decides the length of the processor cycle. It is the designer’s responsibility to balance the length of processor cycle of each stage.

**Implementation**

In MIPS there are 3 different types of instructions, R-Type (Register Type), I-Type (Immediate Type), JType (Jump Type). R-type instructions take 3 different arguments: RT and RS are source registers and RD is the destination register. I-type instructions take two arguments, RS, RT and a 16-bit immediate value. J-type instructions are written with labels and it is the linker or assembler’s duty to convert the label into numerical target address.

In order to update the PC (Program Counter), a new instruction must be fetched from the instruction memory and stored in both the pipeline register and the PC register. The register values are read from the register file, the immediate value's 16 bits are sign extended, and control signals are generated using the opcode and function section. The ALU is where all ALU operations are carried out. For R-type instructions, arithmetic or logic operations, addition to determine the memory address for Load/Store instructions, and comparison for Branch instructions can all be used. For store instructions, data is written to memory, and for load instructions, data is read from memory; this is where a new program counter value can be created. Data is written back to register file from ALU output for R-type instructions or from memory for Load instructions.

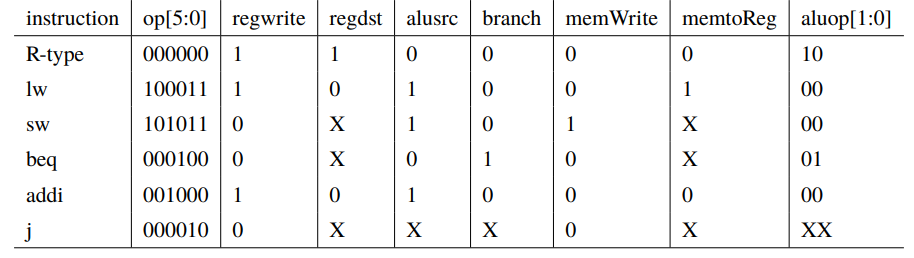
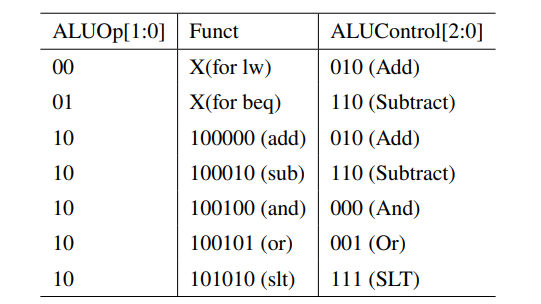


Figure 1 MIPS Instructions



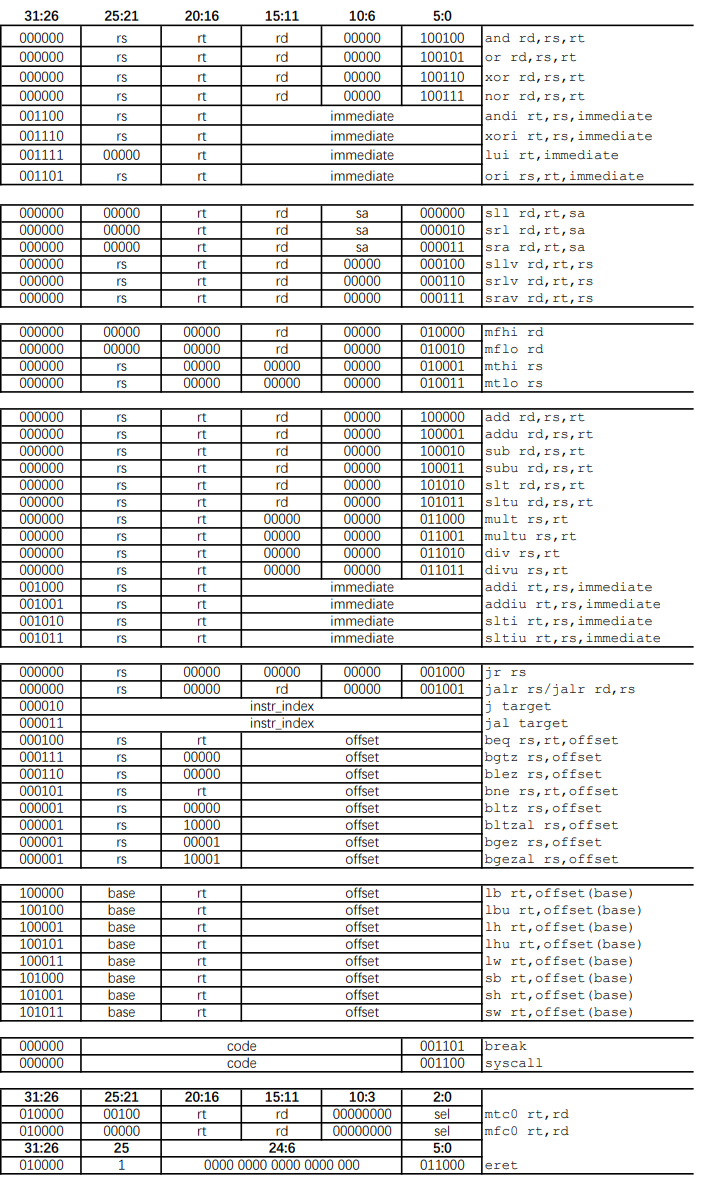
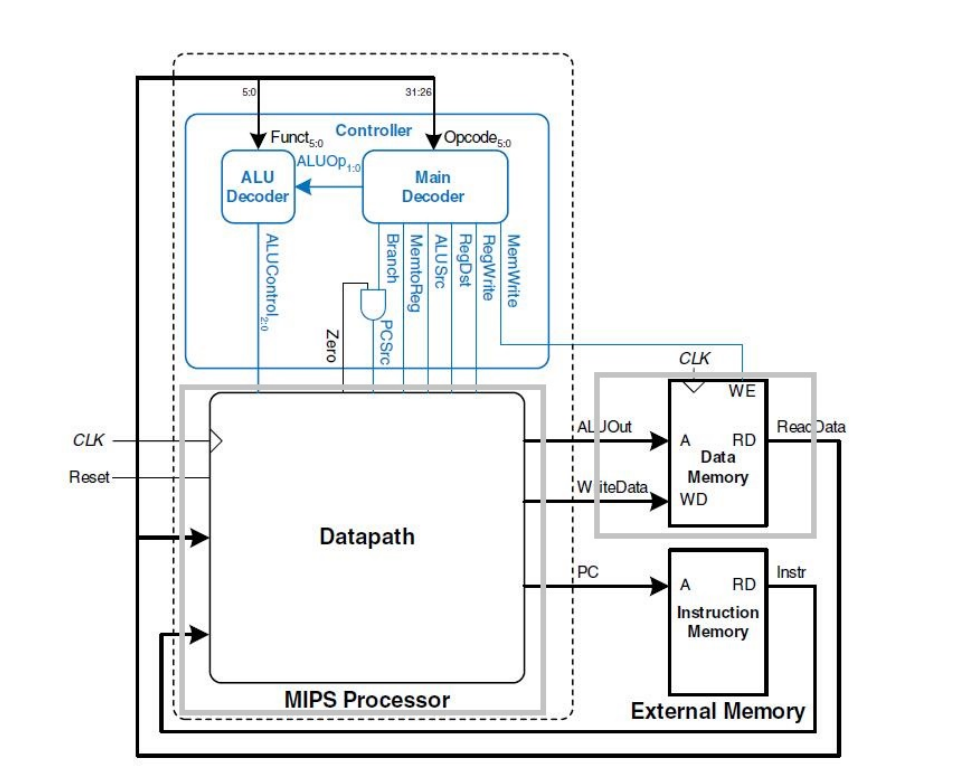
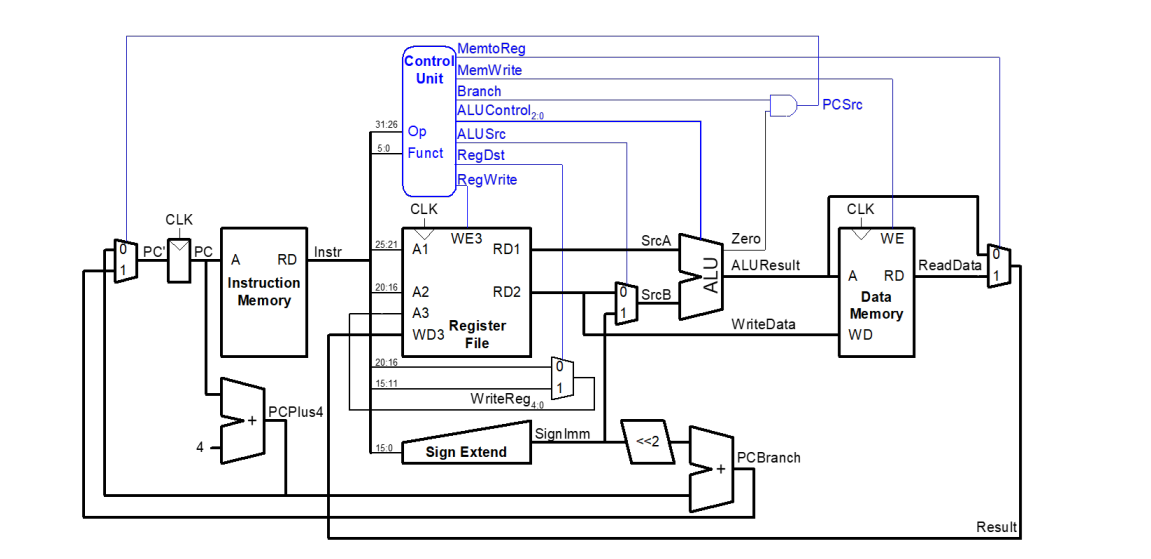


Figure 2 MIPS Instruction Machine Code

**Single-Cycle Implementation**

A single cycle Implementation means that all the operations take equal amount of time. A CPU has a large number of instructions. Depending on the instruction, the time required may vary. However, in a single cycle, all operations take the same amount of time and are finished in a single clock cycle. Consequently, the question of how the clock cycle is determined arises. The number of instructions could be 'n'. The time needed to complete the slowest instruction, however, establishes the clock cycle. Usually, the slowest instruction is load word. Before the clock tick, other instructions might be executed before the clock tick. In which case the instruction will have to wait for the next clock to begin execution.





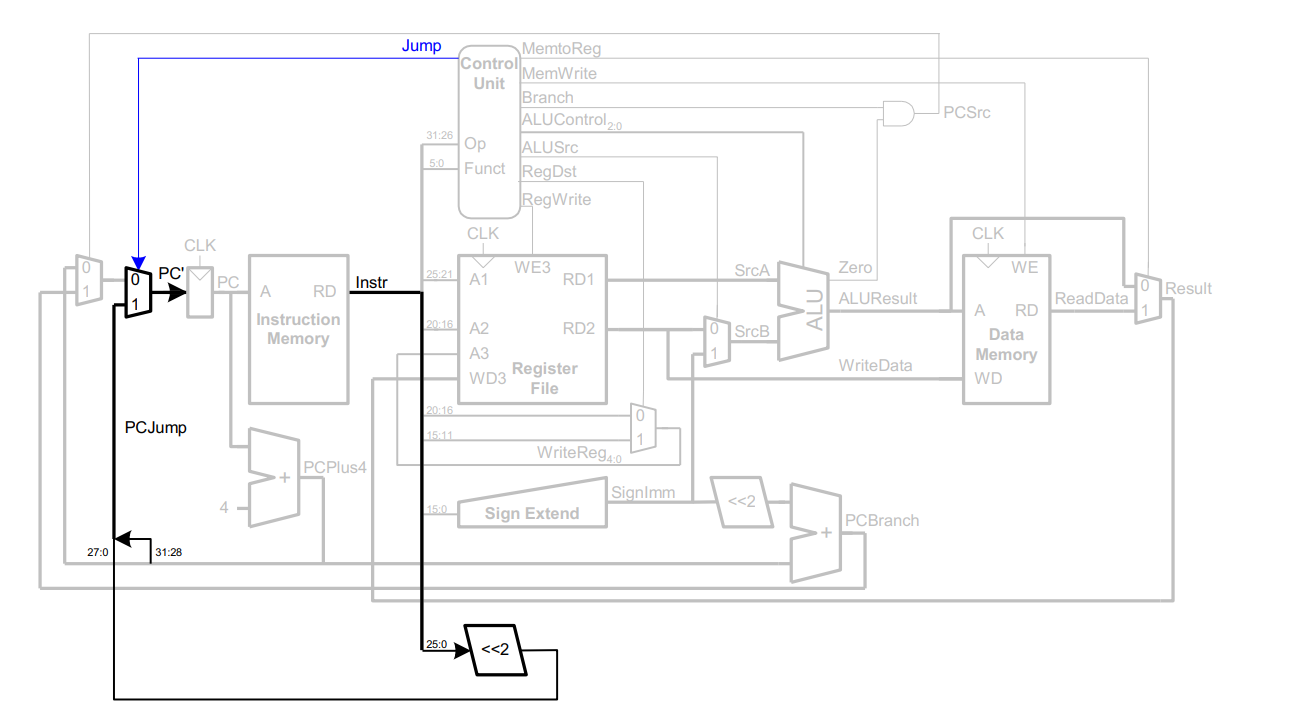
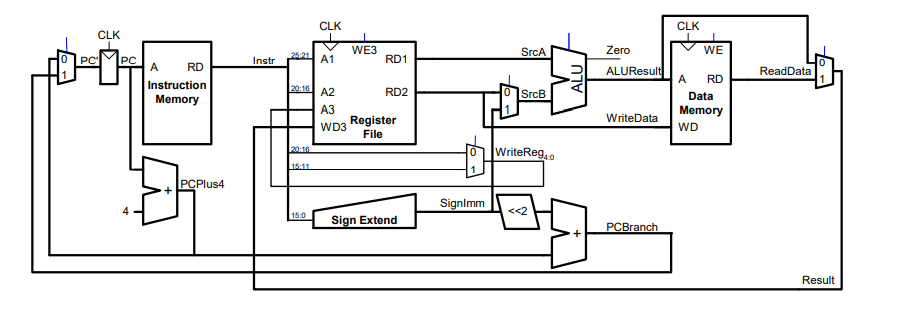
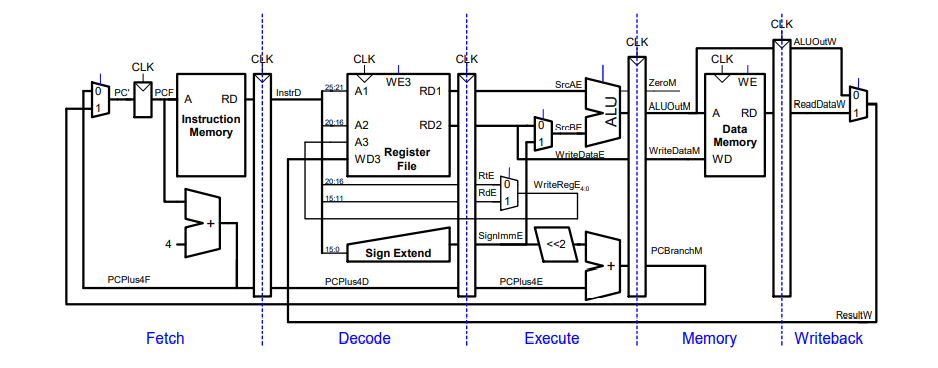


Figure 3 Single Cycle Implementation

**Single cycle to Pipeline**





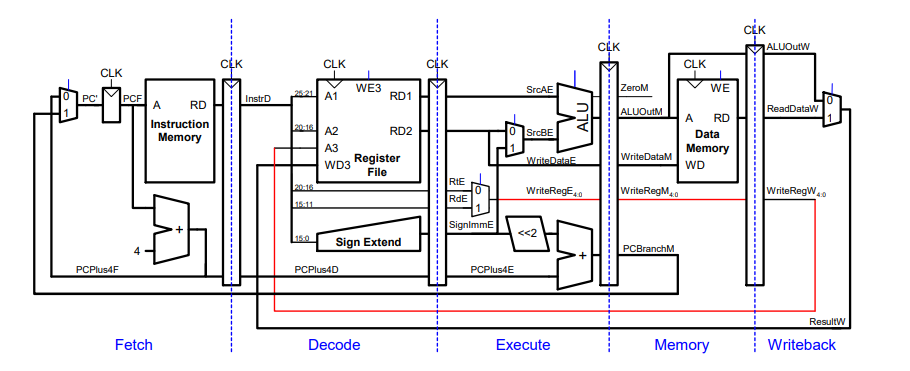
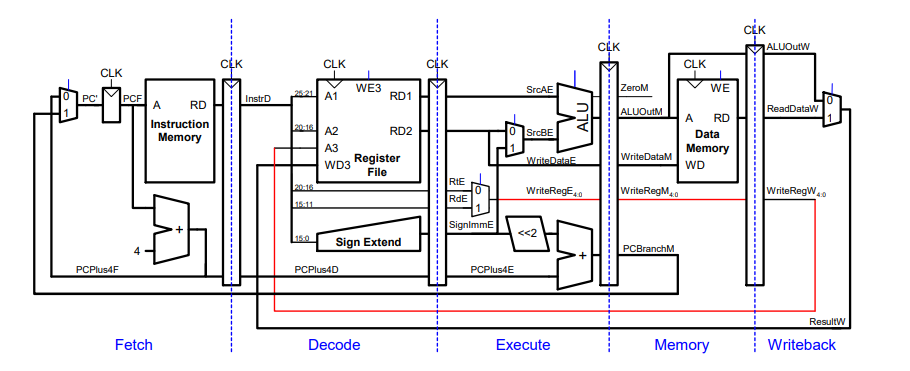


Figure 4 The 5-stage pipeline

**Pipeline Hazards**

As we know, the CPU’s speed is limited by memory Few instructions are at some stage of execution in a pipelined design. There is a chance that these sets of instructions will become dependent on one another, reducing the pipeline’s pace. Dependencies arise for a variety of reasons the dependencies in the pipeline are referred to as hazards since they put the execution at risk.  A hazard, in essence, prevents an instruction present in the pipe from being performed during the specified clock cycle. Since each of the instructions may be in a separate machine cycle, we use the term clock cycle. The three different types of hazards in computer architecture are: Structural, Data and Control Hazard.

**Structural Hazard**



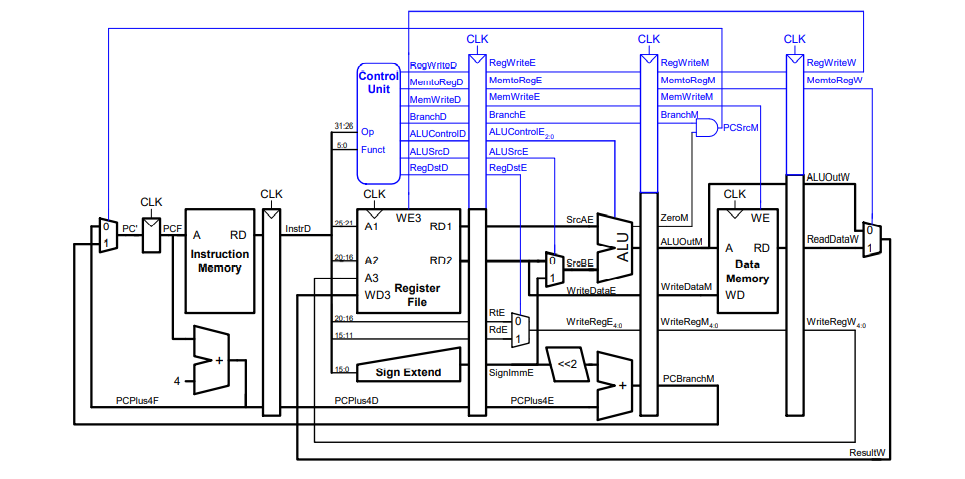
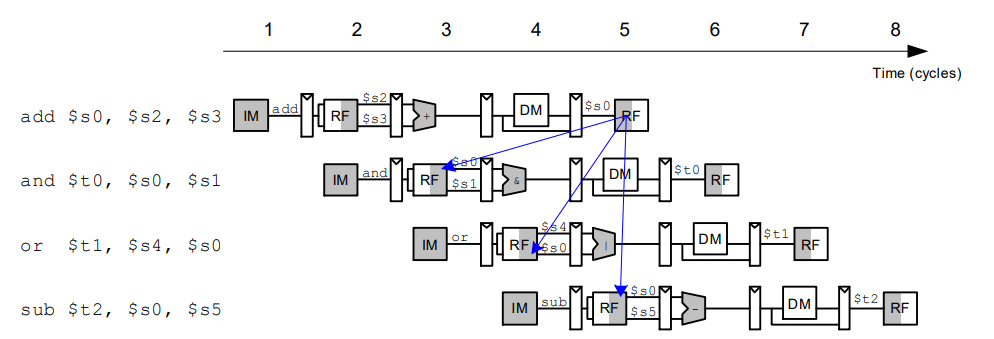
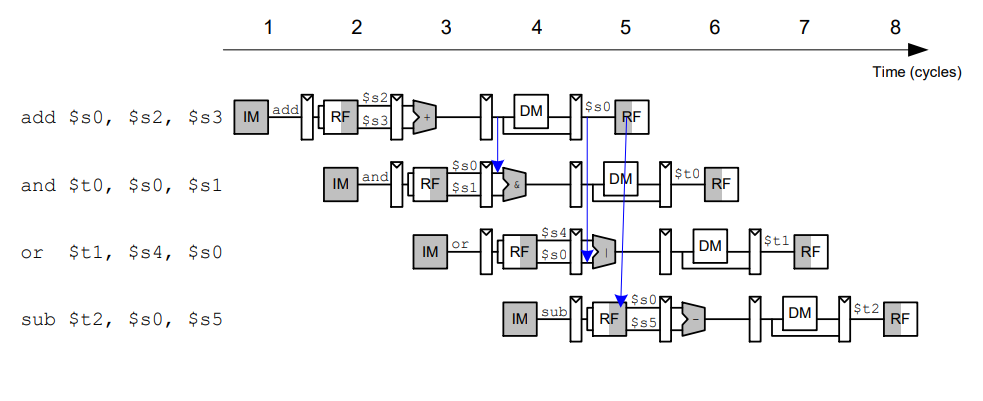


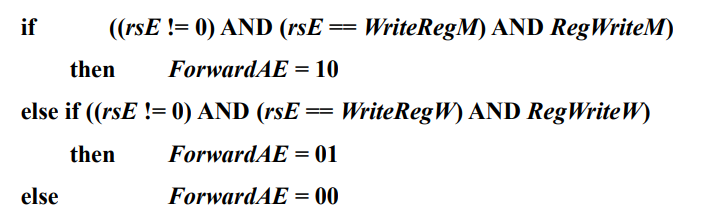
Figure 5 Structural Hazard is solved by Harvard Architecture

**Data Hazard**





When the rd in the EX/MEM pipeline register and the MEM/WB pipeline register is the same as the source operand in the EX stage, the rd in the EX/MEM stage is relatively new and should be used preferentially:



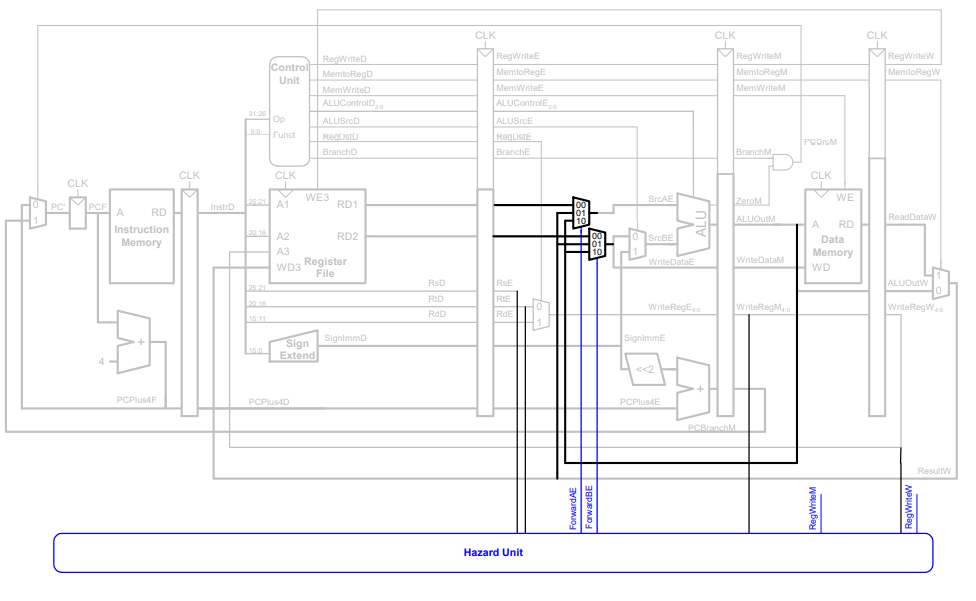
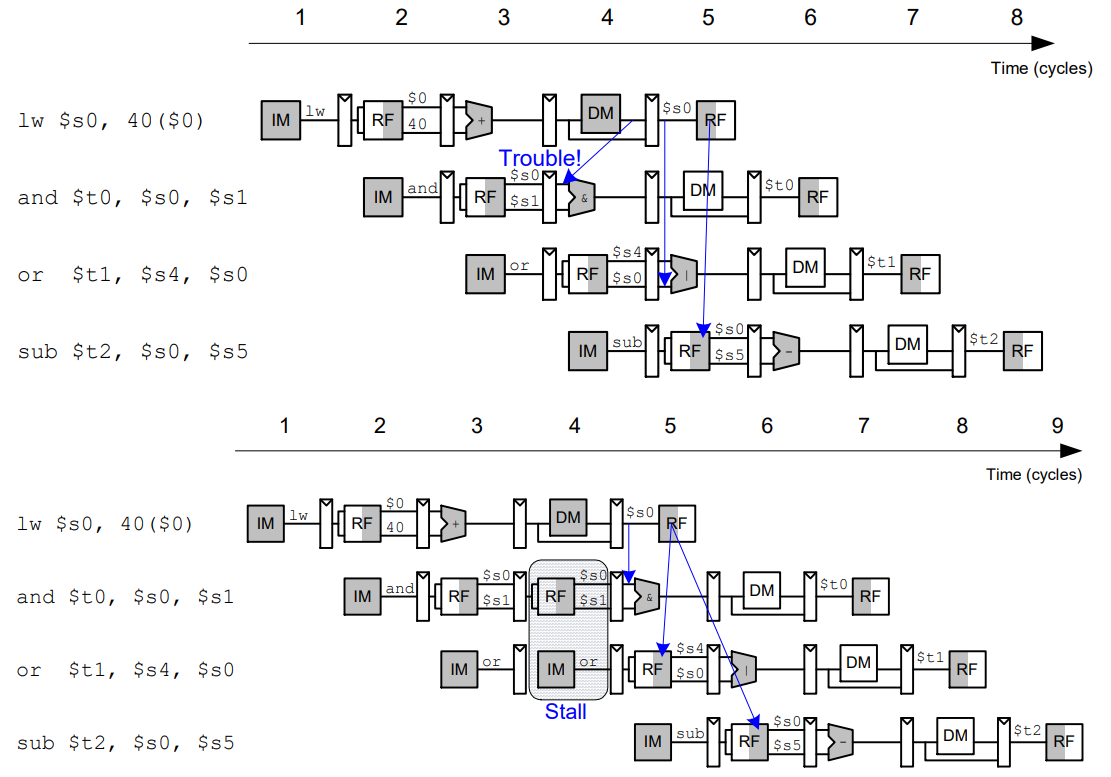


Figure 6 Pipeline depicting hazard unit

**Pipeline Stall**



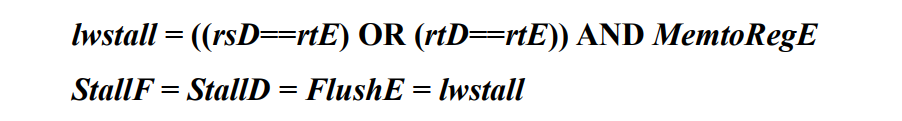
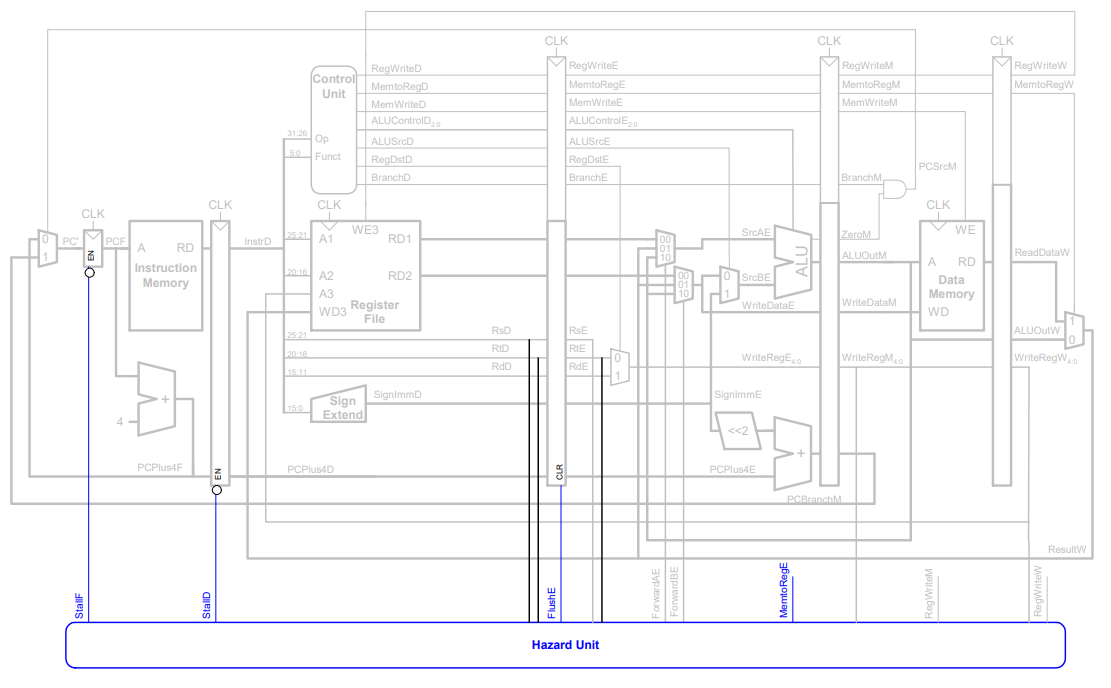


Figure 7 Solving Data Hazard



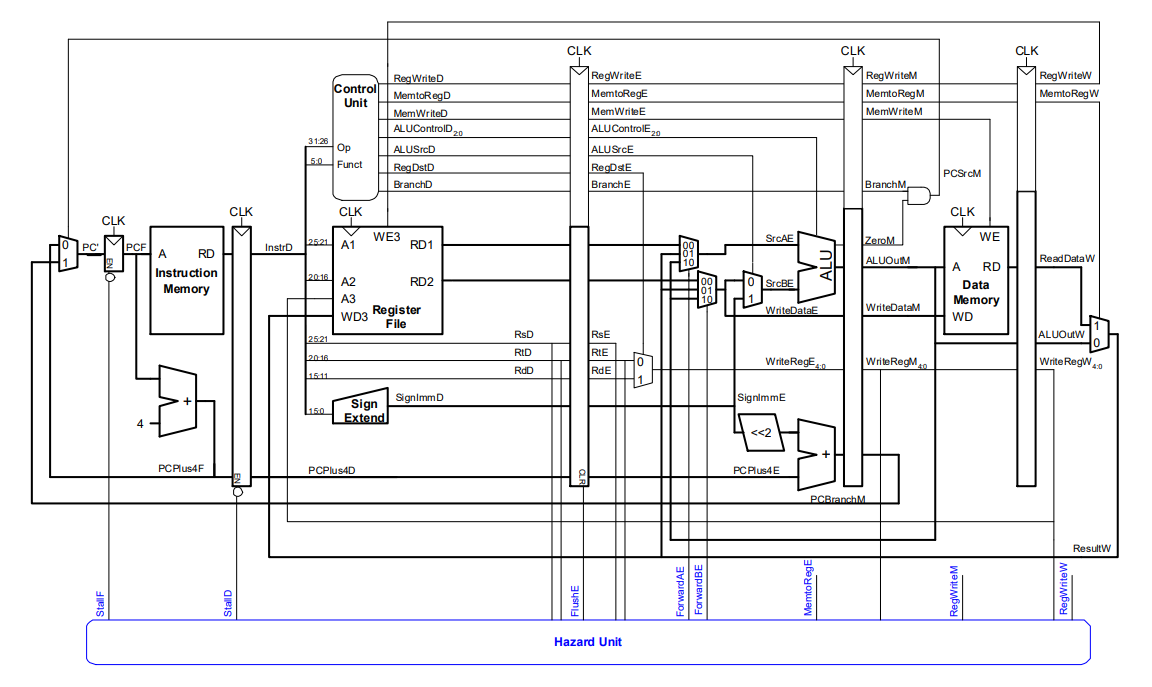
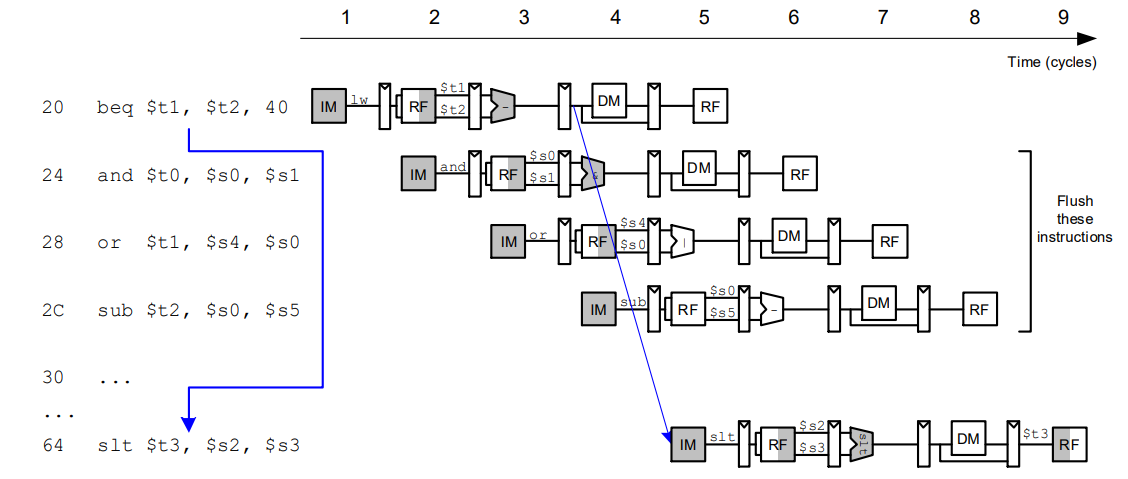
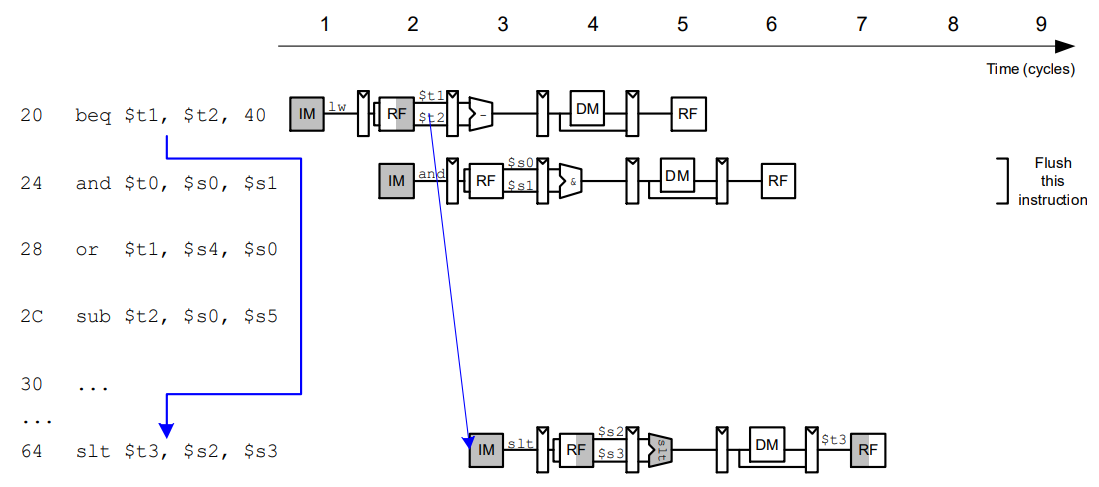


Figure 8 Pipeline after solving Data Hazard

**Control Hazard**





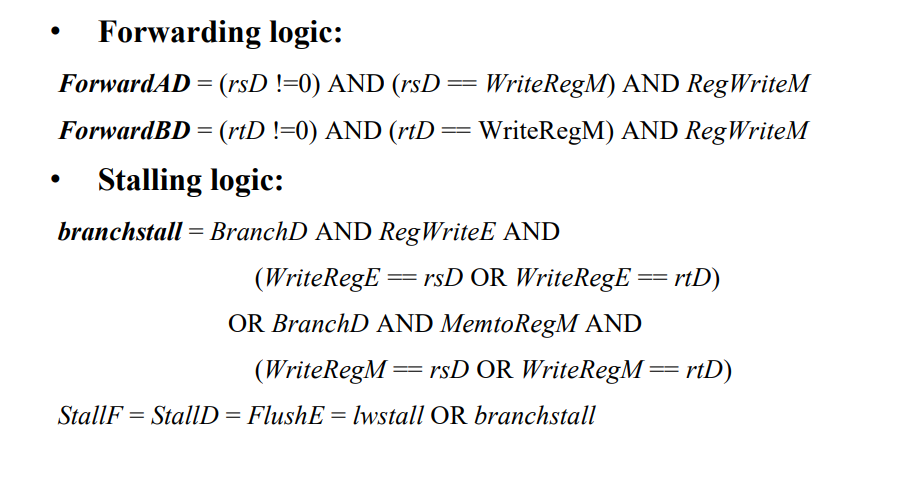
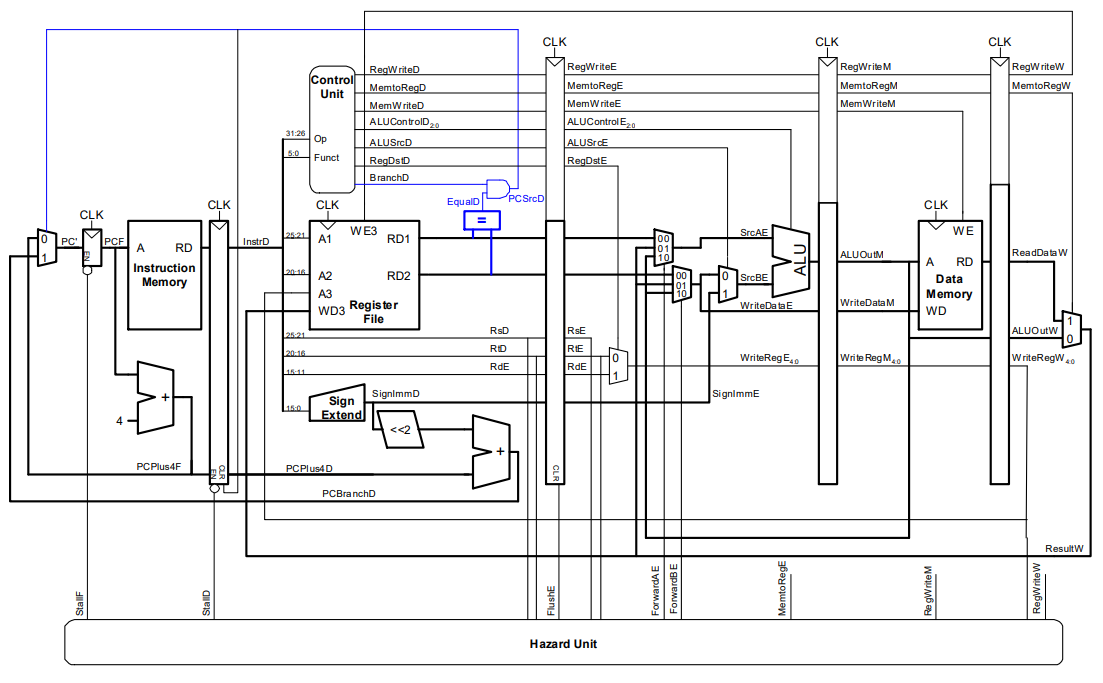
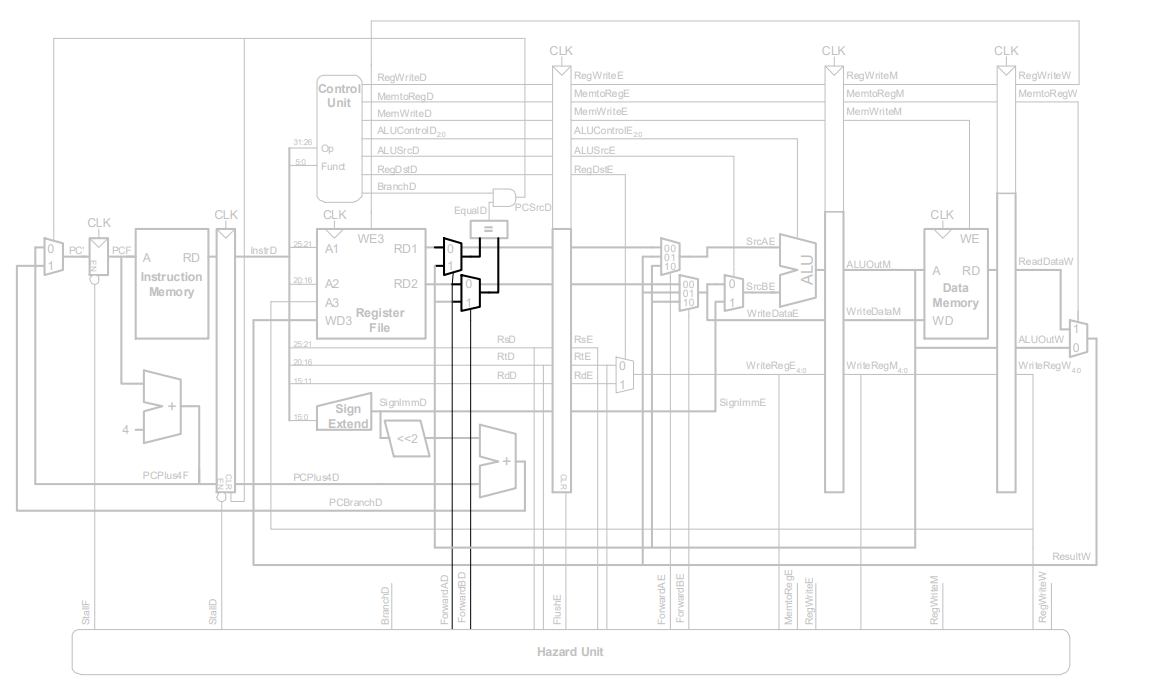


Figure 9 Solving Control Hazard





The design above does not contain the Jump instruction pipeline data path, in fact, Jump instructions and Beq are carried out in ID phase, through pcsrcD\_flush (= pcsrcD | jump) as IF/ID pipeline register the clear signal, it can be as normal execution as Beq instructions. This is the end of the design, but it has many drawbacks, such as when the instructions are as follows:

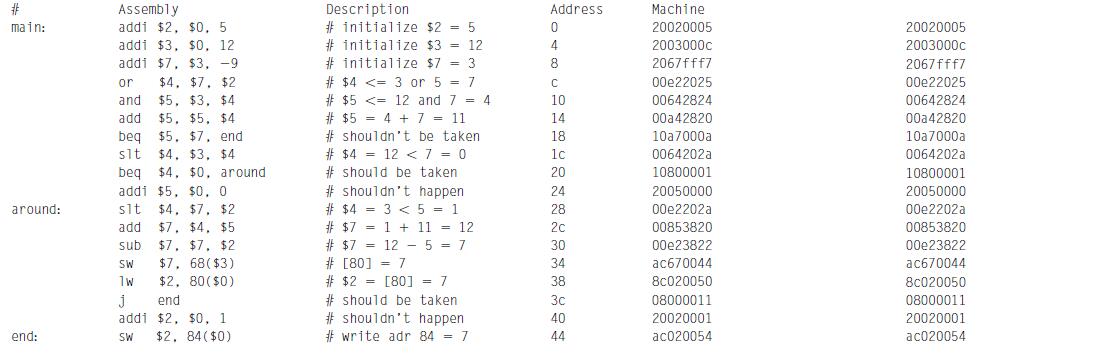
lw $2, 80($0)

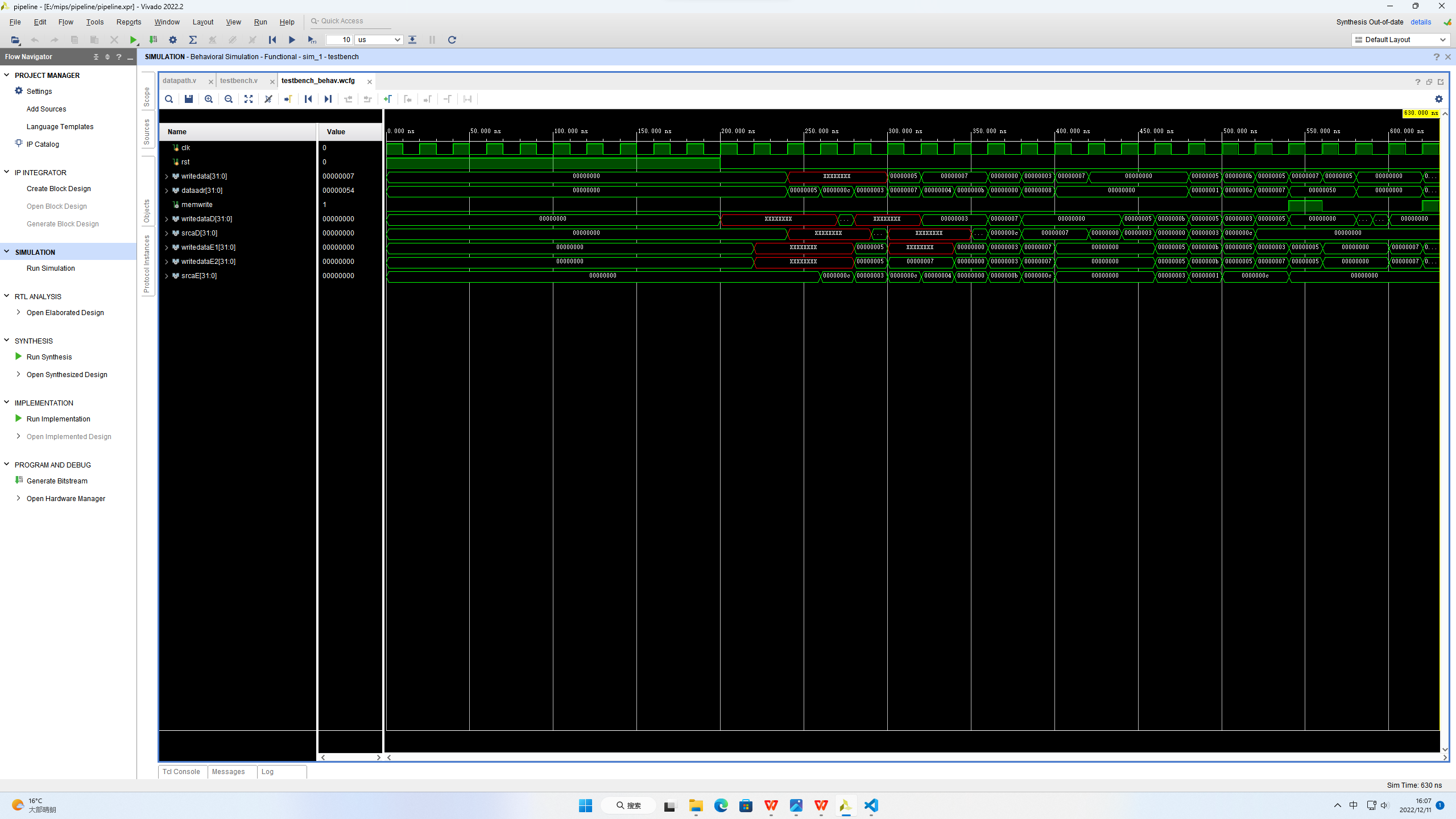
beq $2, $0, end

For normal execution, the CPU should generate two stalls until the lw instruction is in the MEM stage and the beq instruction is executed. This CPU cannot generate the above stall correctly.

**Results**

* **Simulation**





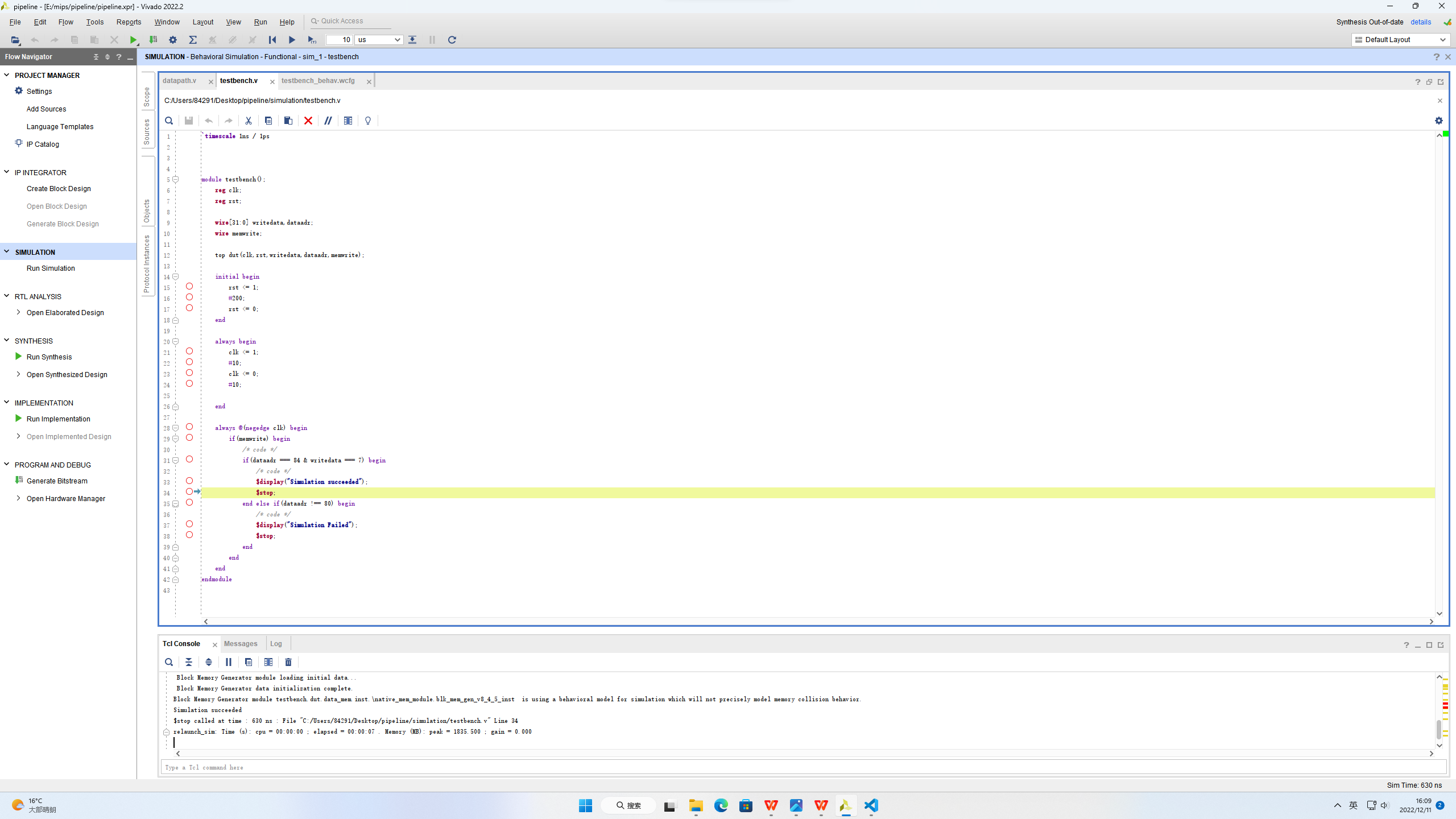
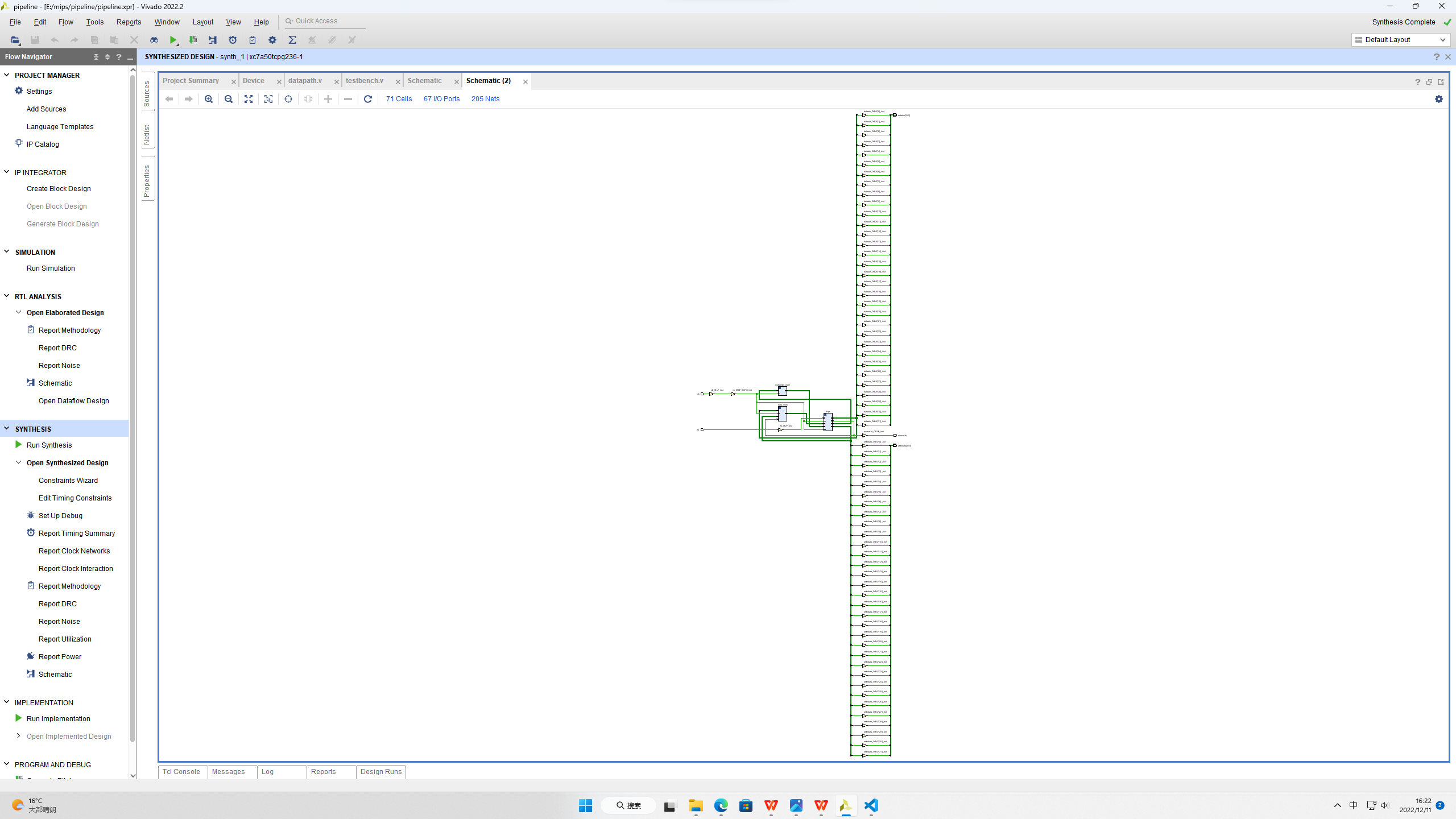


Figure 10 Simulation Succeeded

* **Synthesis**



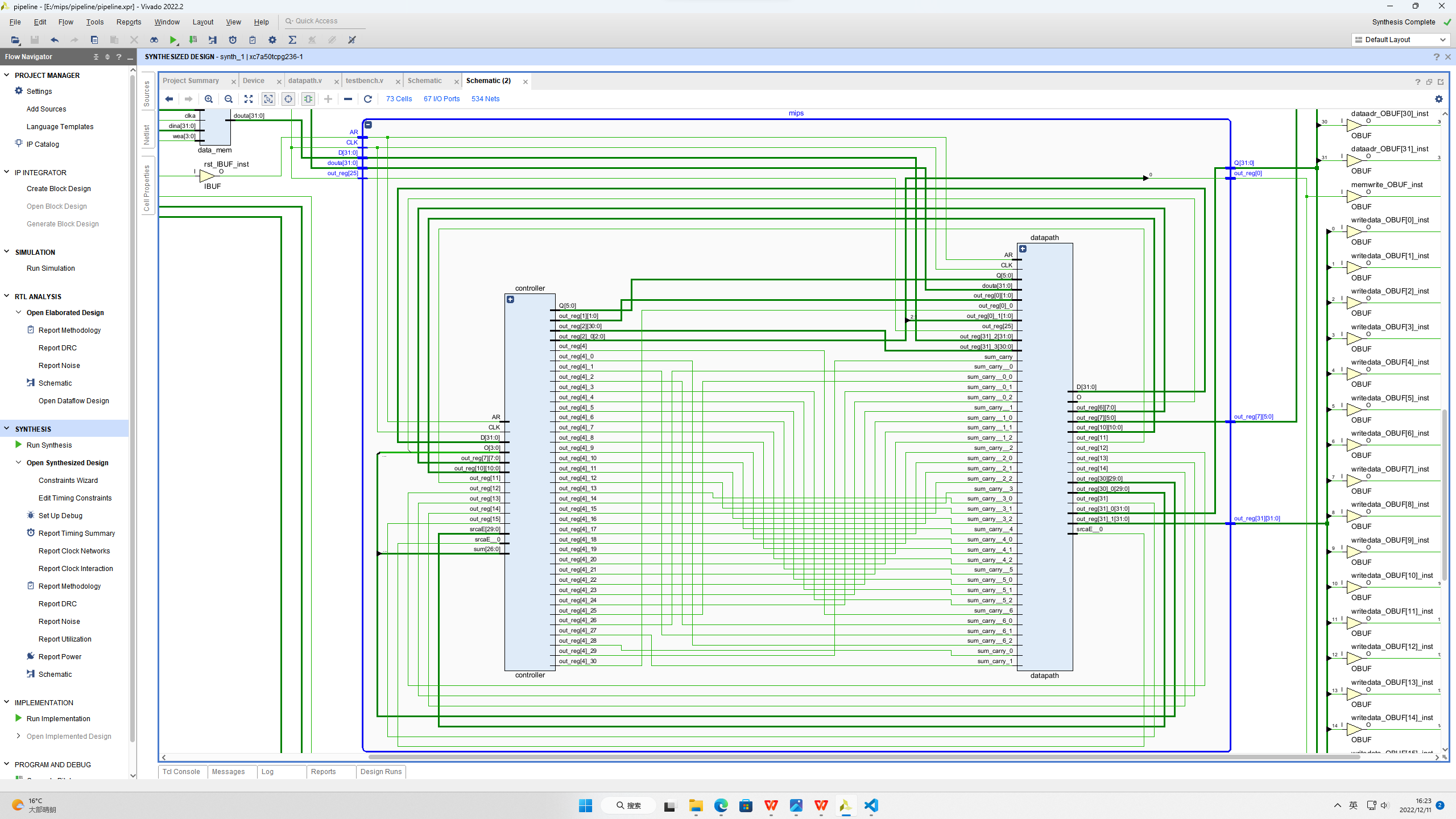


Figure 11 Synthesis Completed

**Conclusion**

The project has been successfully implemented. Starting with the CISC processor, RISC Processor, Single cycle implementation, Pipelines, MIPS architecture and the implementation of 5 stage MIPS processor was understood. The 5 stage 32-bit pipeline was successfully designed. We studied about various pipeline hazards and tried to solve them in the pipeline design. The end results i.e., Simulation and Synthesis were conducted successfully.

**References**

[1] David A. Patterson, John L. Hennessy: Computer Organization and Design – The Hardware/Software Interface ARM Edition Morgan Kaufmann Publisher, Inc.

[2] D. Kumar, & K. Singh. Design of High-Performance MIPS-32 Pipeline Processor. 2012.

[3] Pantazi-Mytarelli. The history and use of pipelining computer architecture: MIPs pipelining implementation Systems. IEEE Proc. of Applications and Technology Conference (LISAT), 2013, pp. 1-7.

[4] Nan Wang+ , Ranjith Kumar and Rajath M. Basavaraj. Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering, pp. 4 05-4 09

[5] Internet Resource: https://www.cs.umd.edu/CA-online/chapter/pipelining-mips-implementation.html