32-bit RISC-V Pipelined Processor Report

- Abhishek Dhital

Introduction

A 5-stage pipelined RISCV-32 processor was implemented on AMD Xilinx FPGA using system Verilog. This was done as a part of the CSE 5392 (RISC Processor Design) course at University of Texas at Arlington. A brief description of each of the modules and stages of the pipeline in the design are as follows:

1. IF Stage (Instruction Fetch)

The IF stage is responsible for fetching instructions from memory. The instruction port of dual port RAM is connected to the program counter, and the instruction word at that address is read in the next clock cycle. This fetched instruction is registered for use in the next stages. The IF stage is also responsible for the updating the program counter (PC) as follows:

- If a branch instruction is decoded in the next stage, a jump enable signal is forwarded back from ID with a jump address and in this case the PC is updated to the jump address.
- If the pipeline needs to stall, then a stall signal is sent back from ID, and the PC is not incremented.
- If a reset is asserted, the PC is reset back to 0x0000.0000
- In all other cases, the PC is incremented by 4 to point at the next instruction word in the memory.

The PC is then registered for use in the next stages of the pipeline.

2. ID Stage (Instruction Decode)

The ID stage is where the instruction is decoded, and the control signals are generated for the next stages. This stage is responsible for the following:

- Instruction Decode: The instruction is decoded to determine the operation type (ALU operation, memory operation, etc.).

- Register File Read: The rv32i_regs module is used to read registers for source operands (rs1, rs2). The values are then registered for use in later stages.

- Control Signals: Generates control signals for instruction execution (e.g., ALU operation type, memory read/write, branch control, register read/write). For J-type instructions, jump enable and jump address are sent back to IF. For Branch instructions, they are sent only if the condition is evaluated to be true. In both cases, if a branch is taken, ID is responsible for sending a NOP for the instruction that is currently in IF, effectively flushing it out of the pipeline.

- Hazard Detection: Ensures that data dependencies are resolved by stalling or forwarding data as needed. If a hazard is detected in any of the further stages, the values read from the register file are replaced with the new values before being registered or being used for evaluation of branch condition.

- Stall: If a LD instruction is detected in EX or MEM stages, then the pipeline is stalled for one or two clock cycles depending on what the current instruction in this stage is. If it is an ALU instruction, then it is sent to EX after one stall, with the incorrect register values, which will be corrected in EX using the forwarded data from WB. This let us move the pipeline with just one stall. However, for branching instructions, the pipeline needs to be stalled for two clock cycles so that we get correct forwarded values from WB before evaluating the branch condition. When the pipeline is stalled, the PC and IW values are saved for use after the stall ends. During the stall, ID sends NOP to later stages to flush the pipeline.

3. EX Stage (Execution)

The EX-stage executes the instruction, performing ALU operations and calculating addresses for memory operations. The ALU module is instantiated in this stage. In short, EX is responsible for the following:
- ALU Operations: The ALU performs arithmetic, logical, or shift operations based on the decoded instruction.
- Address Calculation: For load/store operations, the effective memory address is calculated using the base register and immediate values.

4. MEM Stage (Memory Access)

The MEM stage is where data memory and io memory accesses (loads and stores) occur. For LD instructions, the address calculated by ALU is used to read the data port of the memory or the IO module. This data will be read in the next clock, right in time for WB to shift, sign-extend and write the value to register file.

For STR instructions, the address calculated by ALU is used as write address for the memory or the IO module. The register value is shifted depending on the address calculated and the width of data to be written. This shifted data is then written to the memory at the calculated address.
MEM also generates a signal that tells WB about the source of register writeback. For LD instructions, the source will be set to memory-read if the address is in the range of the memory module. For addresses with the MSB set (i.e. starting at 0x8000.0000), the read and write will be done on the IO module instead, so the source for register write is the data read from the IO module. For ALU instructions, the writeback source is alu output

5. WB Stage (Write-back)

The WB stage writes the result of an instruction back to the register file. Once the memory or IO data is read, it is shifted and sign-extended before being written back to the register file. The shift and sign extension are done based on the width and the type of LD instruction (i.e. LBU, LB, LHU, LH, etc.).

6. Dual Port RAM Module

The dual_port_ram module implements a 128KiB, dual-port RAM, which provides separate ports for instruction and data accesses. It is initialized with a .hex file containing the 32-bit instruction words of the program to be run on the processor. For reads, the instruction word at read address is sent to the output, synchronously. For writes, data is synchronously written to the write address based on the byte enable signals, only when the write enable signal is asserted.

7. RV32i_Regs Module

The register file (rv32i_regs) holds the 32 general-purpose registers (x0 - x31), providing:

- Asynchronous Register Reads: Data from two source registers (rs1 and rs2) are output asynchronously.
- Synchronous Register Write: Writing data back to the register file during the WB stage, based on control signals, on the rising edge of the clock.

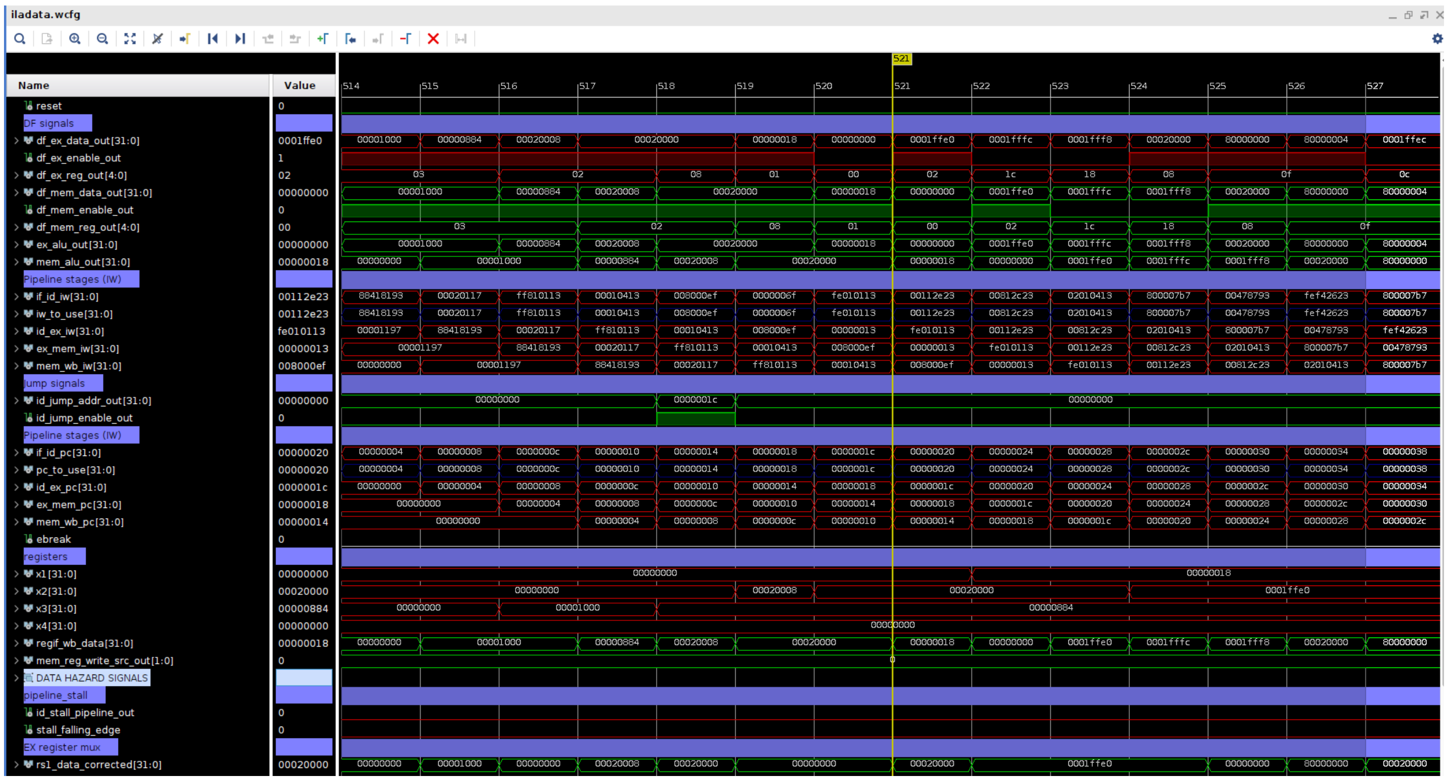
8. IO Interface Module

The io_interface module provides the interface to memory-mapped I/O for interacting with peripherals like LEDs and a push button:

- LED Control: Writing data to address 0x8000_0000 controls the LEDs. Reading from the address provides the current state of the LEDs.
- Push Button: The push button state is read from address 0x8000_0004. Writes to this address are ignored.

9. SystemTop Module

The SystemTop module integrates all the discussed components of this RISC-V pipeline-based processor.



Name	Value
reset	0
DF signals	
df_ex_data_out[31:0]	0001ffe0
df_ex_enable_out	1
df_ex_reg_out[4:0]	02
df_mem_data_out[31:0]	00000000
df_mem_enable_out	0
df_mem_reg_out[4:0]	00
ex_alu_out[31:0]	00000000
mem_alu_out[31:0]	00000018
Pipeline stages (IW)	
if_id_iw[31:0]	00112e23
iw_to_use[31:0]	00112e23
id_ex_iw[31:0]	fe010113
ex_mem_iw[31:0]	00000013
mem_wb_iw[31:0]	008000ef
Jump signals	
id_jump_addr_out[31:0]	00000000
id_jump_enable_out	0
Pipeline stages (IW)	
if_id_pc[31:0]	00000020
pc_to_use[31:0]	00000020
id_ex_pc[31:0]	0000001c
ex_mem_pc[31:0]	00000018
mem_wb_pc[31:0]	00000014
ebreak	0
registers	
x1[31:0]	00000000
x2[31:0]	00020000
x3[31:0]	00000884
x4[31:0]	00000000
regif_wb_data[31:0]	00000018
mem_reg_write_src_out[1:0]	0
DATA HAZARD SIGNALS	
pipeline_stall	
id_stall_pipeline_out	0
stall_falling_edge	0
EX register mux	
rs1_data_corrected[31:0]	00020000

Figure 1 ILA Capture showing the pipeline progression and control signals along the way