Pipelined Processor Project Report

Cherif Mikhail

Jacques Jean

Kirollos Mounir

CPU Project Report

Introduction

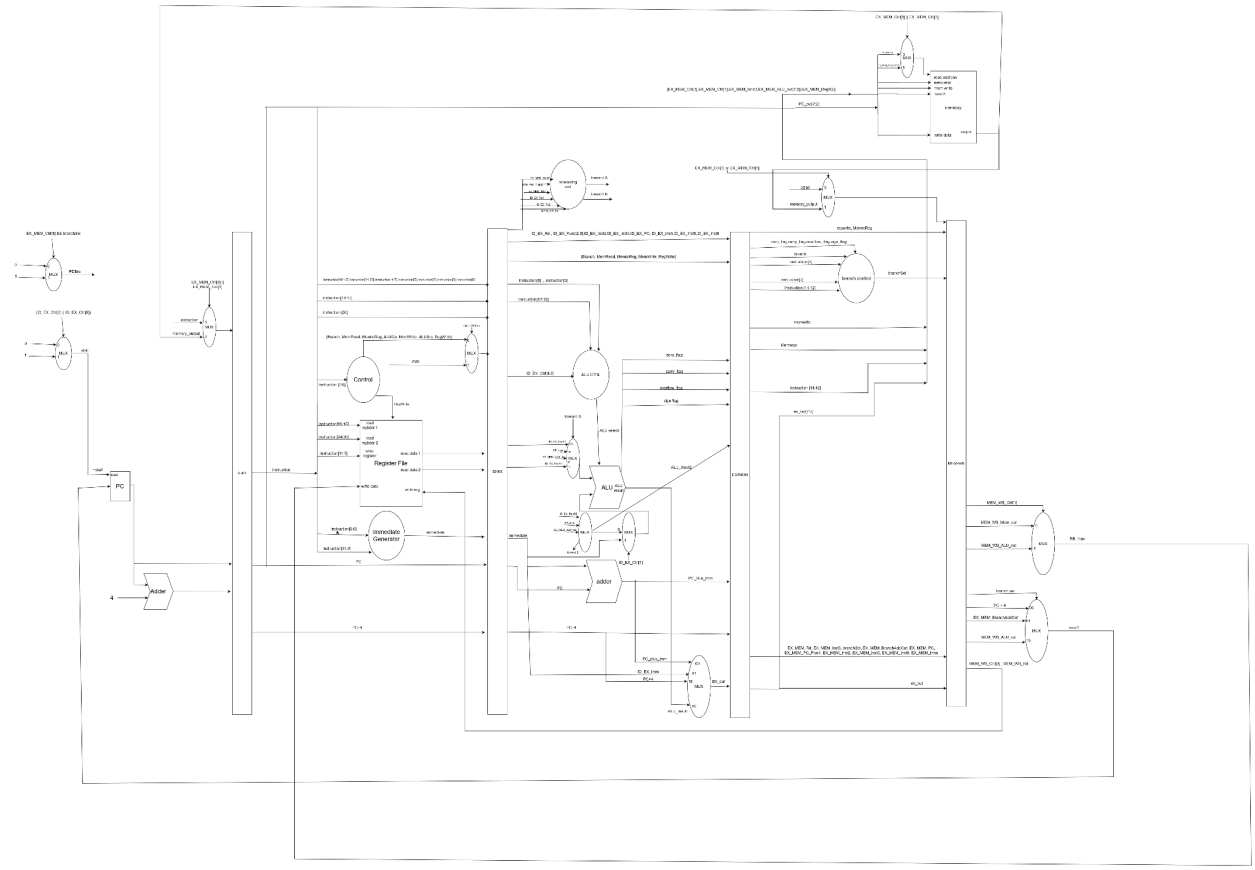
This report documents the design and implementation of a CPU modeled in Verilog; it follows a fully pipelined design. The project implements a functional processor based on the RISC-V Instruction Set Architecture (ISA).

Team Members: Cherif Mikhai, Jacques Jean, Kirollos Mounir

The CPU supports the RV32I Base Integer Instruction Set. All forty-two user-level instructions are implemented.

BLOCK DIAGRAM LINK FOR CLARITY

Block diagram:



Pipeline Stages Description

The CPU_pipelined module is designed with a traditional five-stage pipeline structure, with pipeline registers IF/ID, ID/EX, EX/MEM, MEM/WB between stages for passing data and control signals.

1. Instruction Fetch (IF)

- The next PC address is calculated and the instruction is fetched from memory.
- It uses the RCA component to compute the next sequential address, PC_plus4.
- The last next address, mux1, is chosen based on branch results or PC+4.
- Register PC updates only when there is no stall and no memory conflict signal.
- In case of a detected branch or memory conflict, the instruction input to the IF/ID register is replaced by a NOP.

2. Instruction Decode (ID)

- Reads operands from the register file, generates control signals, and calculates the immediate value.
- The Control Unit decodes the instruction opcode and generates control signals.
- The Register File reads two operands according to readReg1 and readReg2.
- The Immediate Generator calculates the immediate value depending on the instruction type.

3. Execute (EX)

- Performs all arithmetic and logical operations and calculates the branch target address.
- ALU Control selects the ALU operation based on ALUOp and funct3.
- ALU inputs (ALU_input1, ALU_input2) are selected by forwarding logic.
- The ALU outputs ALU_result and status flags (zero_flag, sign_flag, etc.).

- PC_plus_imm is computed using the immediate value and PC.

4. Memory (MEM)

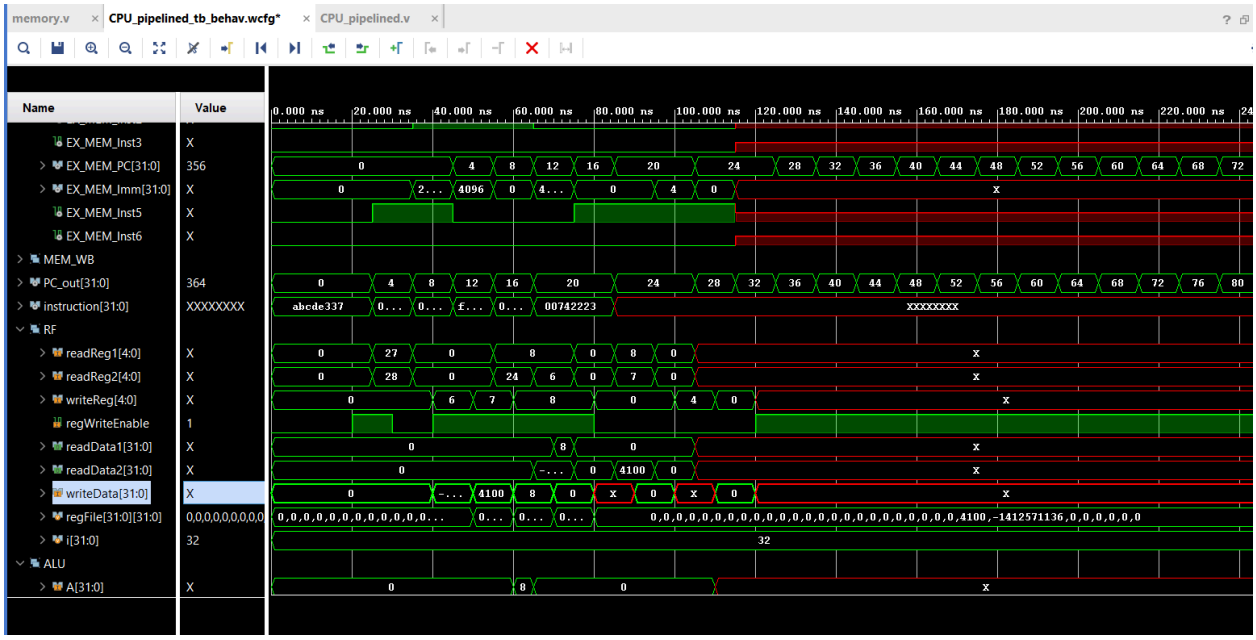
- Performs data memory access (read/write) and determines branch outcomes.
- The unified memory is accessed using the ALU result as the address; MEM stage has priority during conflicts.
- branch_control module outputs the branch selection signal (branchSel) based on ALU flags.
- Memory outputs read data or performs a write.

5. Write Back (WB)

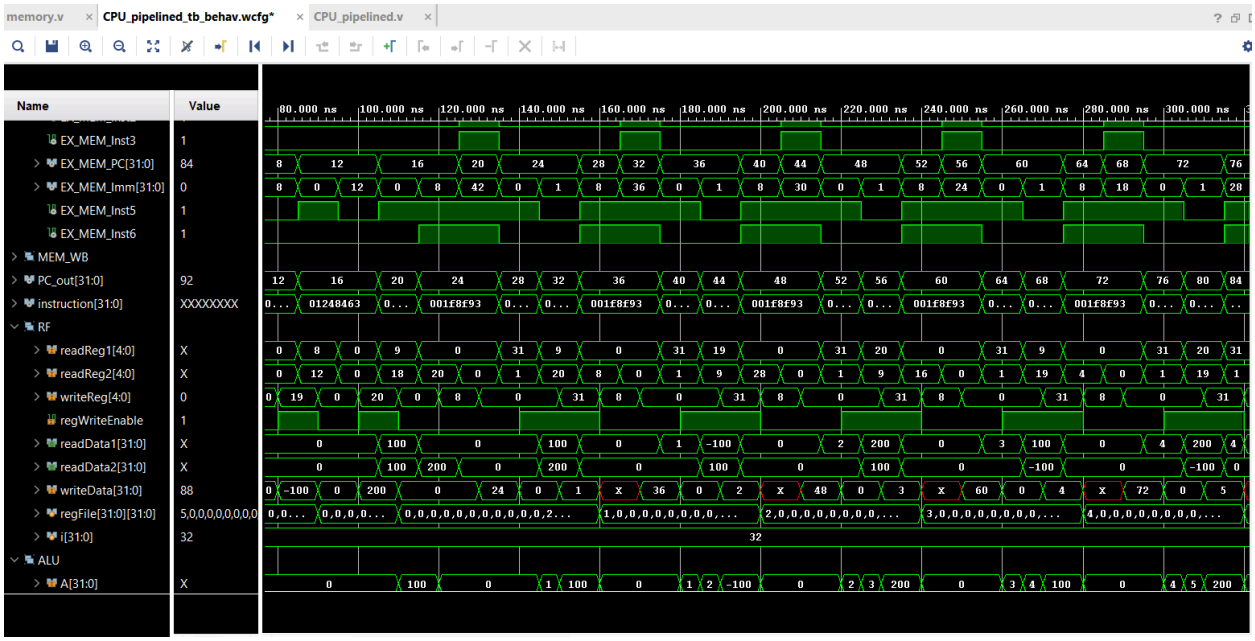
- Writes final results back into the register file.
- RB_mux selects the write-back value (either memory output or ALU result).
- Some instructions (auipc, lui, jal, jalr) overwrite write_data to ensure PC+4 or correct immediate.
- Data is written into MEM_WB_Rd when RegWrite is high.

WAVE FORMS AND TESTING:

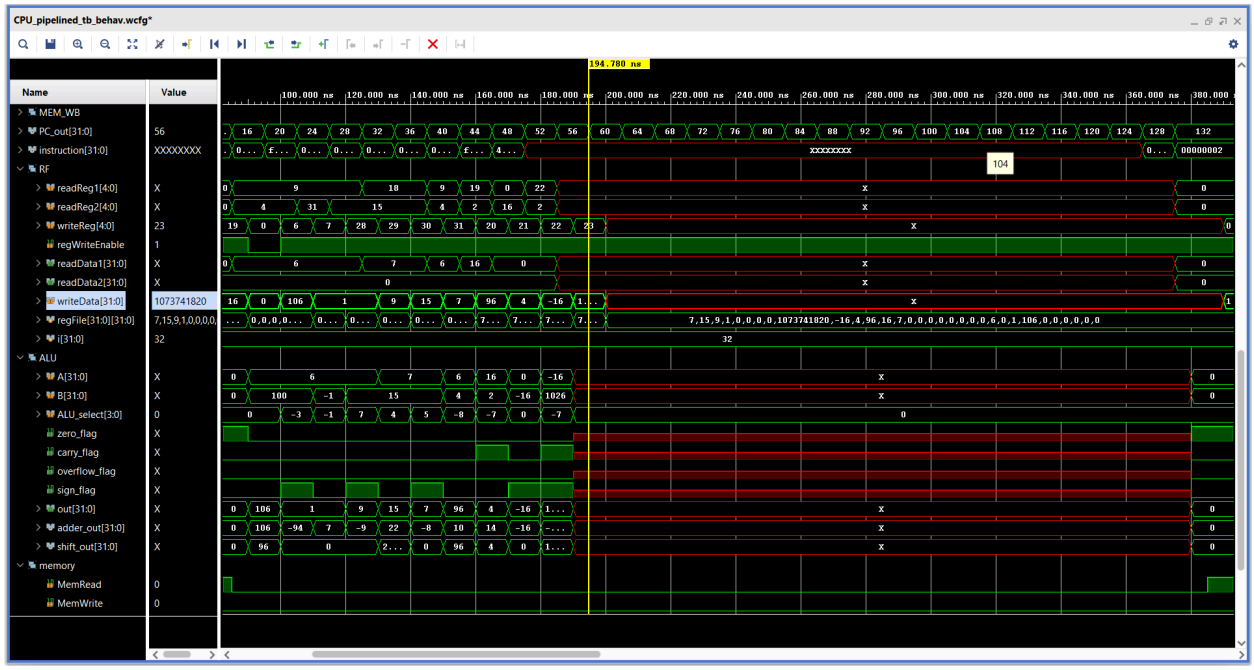
1. auipc & lui:



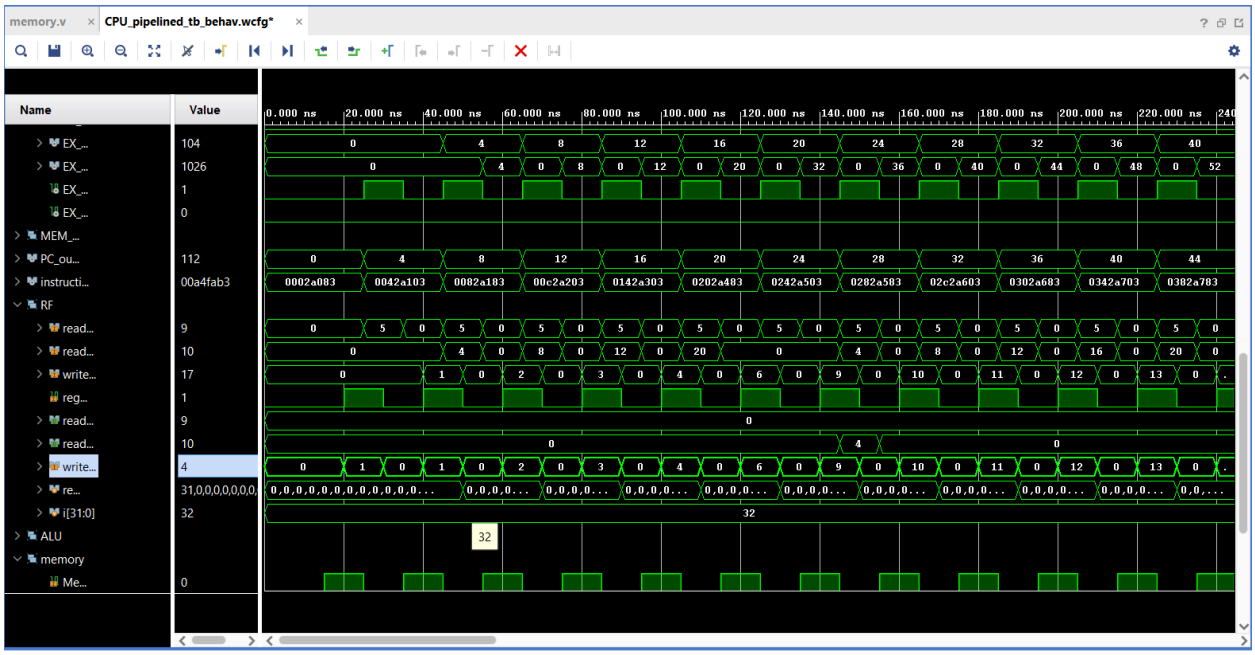
2. B-type:



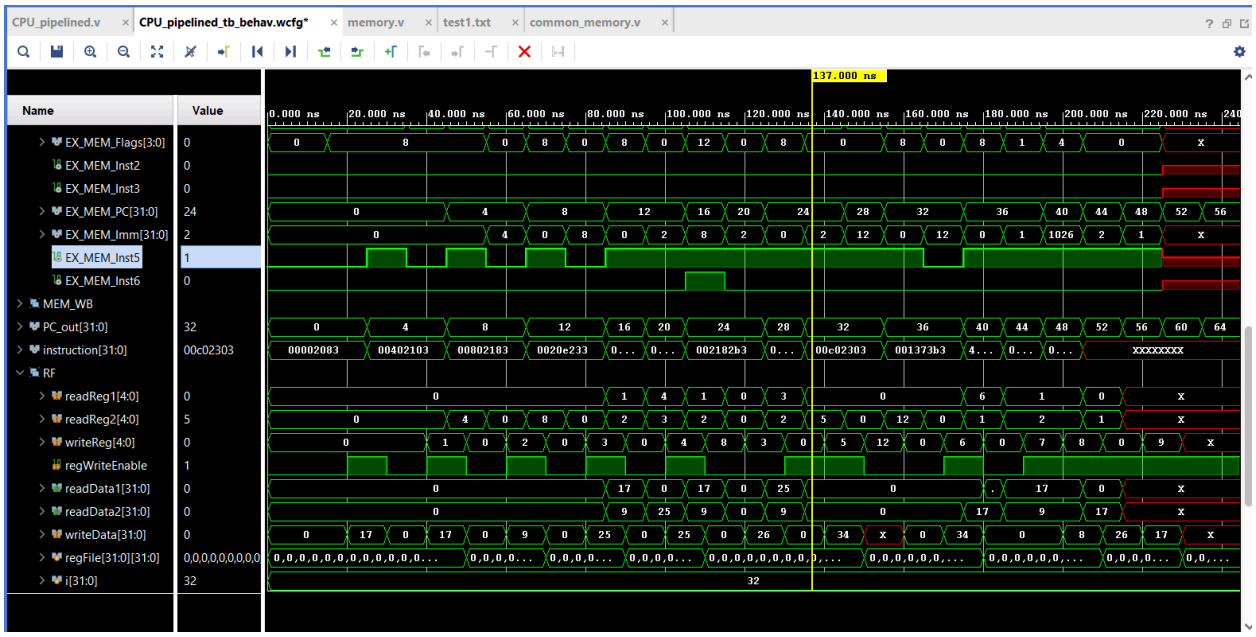
3. I-type:


CPU_pipelined_tb_behav.wcfg*
194.780 ns
Name
Value
MEM_WB
PC_out[31:0]
56
instruction[31:0]
XXXXXXXX
RF
readReg1[4:0]
readReg2[4:0]
writeReg[4:0]
23
regWriteEnable
1
readData1[31:0]
readData2[31:0]
writeData[31:0]
1073741820
regFile[31:0][31:0]
7,15,9,1,0,0,0,0,
i[31:0]
32
ALU
A[31:0]
B[31:0]
ALU_select[3:0]
zero_flag
carry_flag
overflow_flag
sign_flag
out[31:0]
adder_out[31:0]
shift_out[31:0]
memory
MemRead
MemWrite
104

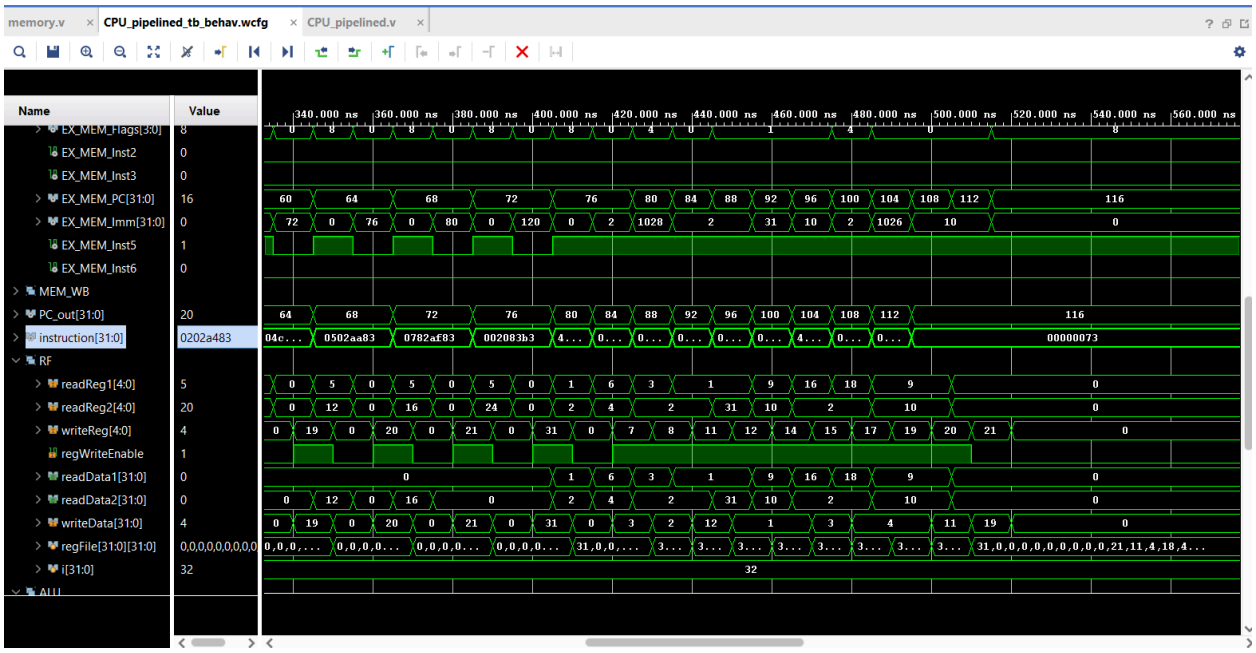
4. R-type:


memory.v
CPU_pipelined_tb_behav.wcfg*
Name
Value
EX_...
104
EX_...
1026
EX_...
1
EX_...
0
MEM_...
PC_ou...
112
instructi...
00a4fab3
RF
read...
9
read...
10
write...
17
reg...
1
read...
9
read...
10
write...
4
re...
31,0,0,0,0,0,0,
i[31:0]
32
ALU
memory
Me...
0
32

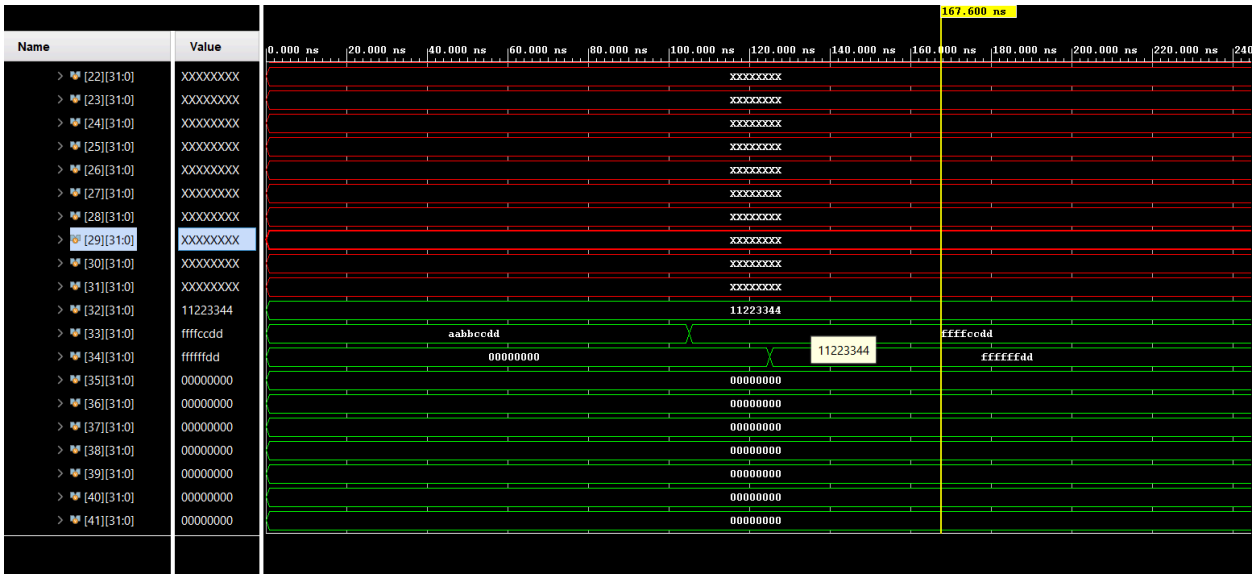
5. final syscalls:



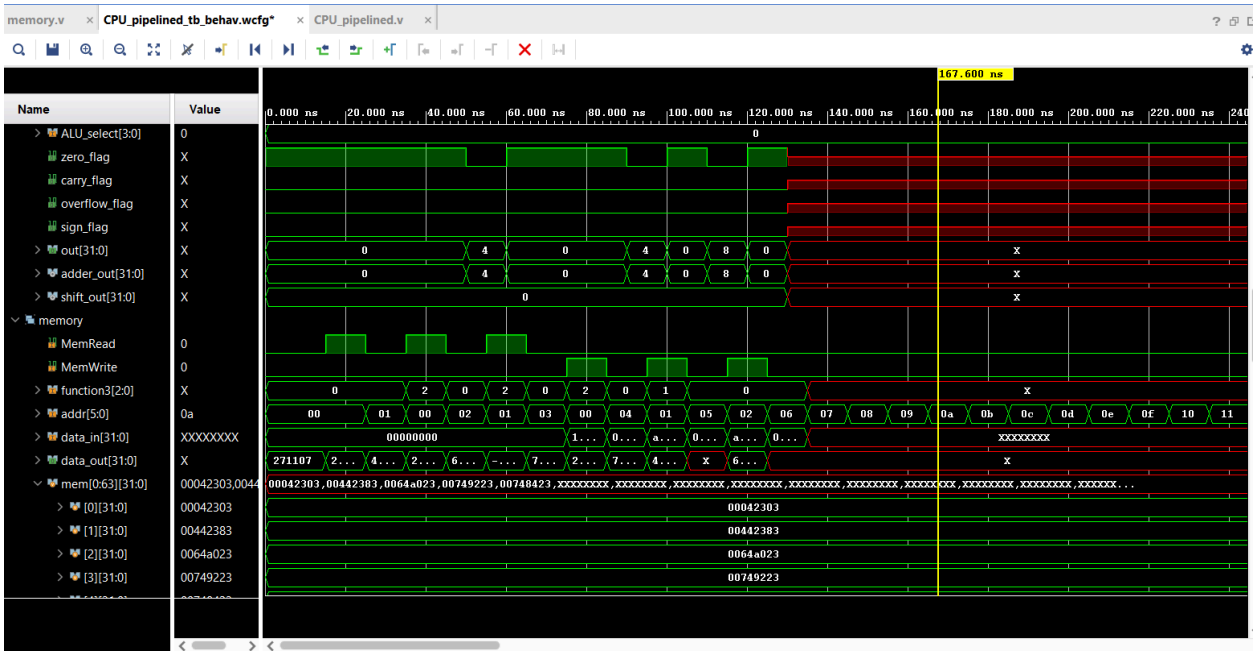
6. syscall:



7.


167.600 ns
Name
Value
[22][31:0] XXXXXXXX
[23][31:0] XXXXXXXX
[24][31:0] XXXXXXXX
[25][31:0] XXXXXXXX
[26][31:0] XXXXXXXX
[27][31:0] XXXXXXXX
[28][31:0] XXXXXXXX
[29][31:0] XXXXXXXX
[30][31:0] XXXXXXXX
[31][31:0] XXXXXXXX
[32][31:0] 11223344
[33][31:0] ffffccdd
[34][31:0] ffffffdd
[35][31:0] 00000000
[36][31:0] 00000000
[37][31:0] 00000000
[38][31:0] 00000000
[39][31:0] 00000000
[40][31:0] 00000000
[41][31:0] 00000000
aabbccdd
11223344
ffffccdd
00000000
ffffffdd

8.


memory.v
CPU_pipelined_tb_behav.wcfg*
CPU_pipelined.v
167.600 ns
Name
Value
ALU_select[3:0] 0
zero_flag X
carry_flag X
overflow_flag X
sign_flag X
out[31:0] X
adder_out[31:0] X
shift_out[31:0] X
memory
MemRead 0
MemWrite 0
function3[2:0] X
addr[5:0] 0a
data_in[31:0] XXXXXXXX
data_out[31:0] X
mem[0:63][31:0] 00042303,0044
[0][31:0] 00042303
[1][31:0] 00442383
[2][31:0] 0064a023
[3][31:0] 00749223
00042303,00442383,0064a023,00749223,00748423,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxxxx,xxxxxx...

Hazard Handling

Data Hazards

- fw_unit handles RAW hazards by forwarding EX/MEM and MEM/WB results directly to EX stage inputs.

- Prevents unnecessary pipeline delays.

Control Hazards (Flushing and Stalling)

- No branch prediction; sequential path assumed by default.

- If a branch is taken (PCSrc = 1), the instruction in IF/ID is flushed with a NOP.

- Load-use hazard triggers a stall, specifically when: The next instruction needs a register produced by the previous instruction in ID/EX or the destination register is not x0.

Structural Hazards

the processor uses a single module called common_memory for both storing instructions and holding data. This module has an internal array, mem [0:63], where instructions are placed from address mem up to mem and data is placed starting from address mem up to mem [32].
Since the memory is single-ported, a potential structural hazard arises when the Instruction Fetch (IF) stage needs to read an instruction and the Memory (MEM) stage needs to read or write data simultaneously.
To resolve this conflict without issuing instructions only every two clock cycles, the design prioritizes the data access required by the MEM stage.
This priority is managed as follows:

1. Conflict Detection: A wire named conflict is used to signal simultaneous memory demand. It is asserted if the MEM stage requires a read or a write operation, which is checked by looking at the control signals EX_MEM_Ctrl (MemRead) or EX_MEM_Ctrl (MemWrite).

    assign conflict = EX_MEM_Ctrl || EX_MEM_Ctrl.

2. Memory Access Muxing: The common_memory module is always connected to the CPU. It needs an address (addr) and control signals (MemRead_temp, MemWrite_temp) to know if it is serving an instruction fetch or a data access.

If the current instruction in the MEM stage is not accessing memory (ismem is false), the memory is dedicated to the IF stage, using the Program Counter (PC_out) for the address.

If the current instruction is accessing memory (ismem is true), the memory is dedicated to the MEM stage, using the calculated ALU output (offset by 32,EX_MEM_ALU_out[7:2]+6'd32) as the address.

3. IF Stage Stall: When conflict is asserted, the IF stage is stalled by preventing the Program Counter (PC) from updating .

The PC register loads its new value only if both the general stall logic and the conflict signal are inactive: .load(~stall && ~conflict).

By using this approach, if a structural hazard occurs, the Instruction Fetch stage is temporarily halted for one cycle, allowing the Memory stage to complete its data access before the IF stage proceeds, thus ensuring correct operation at the cost of a delay

Development Issues and Challenges

The project involved several challenges with pipelined design and verification:

- Waveform Complexity debugging: One important issue was the verification of program execution's correctness through a waveform viewer. Since pipeline stages IF, ID, EX, MEM, WB are performing different components of an instruction in each cycle, close monitoring was required to confirm that all stages were operating correctly for each instruction flowing through.

- Signal Index Management: Control and data signals passing into successive stages were difficult because each time they had to be packaged into the next pipeline register, the indices for the signals changed . That meant rigorous control of signal width and ordering in those large pipeline registers such as ID_EX_reg and EX_MEM. Recognition that a needed signal was available only in an early stage but required in a late stage like MEM/WB forced the team to continuously pass that signal through several sequential registers.

- Structural Hazard Resolution: As mentioned previously, it was a challenge to design the logic to correctly multiplex access to the single unified memory and stall the IF stage only when the MEM stage required access (conflict=EX_MEM_Ctrl || EX_MEM_Ctrl).

- Testing Environment Limitation: The major external challenge was the inability to work from outside the campus because access to Vivado was strictly on-campus. Efficiency of Testing: The testing of the design using only the waveform viewer was far more efficient than using only FPGA testing since the use of the waveform viewer precluded the long waiting times involved in generating the bitstream used to deploy the design on the FPGA.