IITB-RISC-23: Pipelined Processor

11.05.2024

EE309: Microprocessors

Instructor: Prof. Virendra Singh

Team ID: 40

Harshil Singla (22b1260)

Saumya Shah (22b1238)

Himanshi Shende (22b1235)

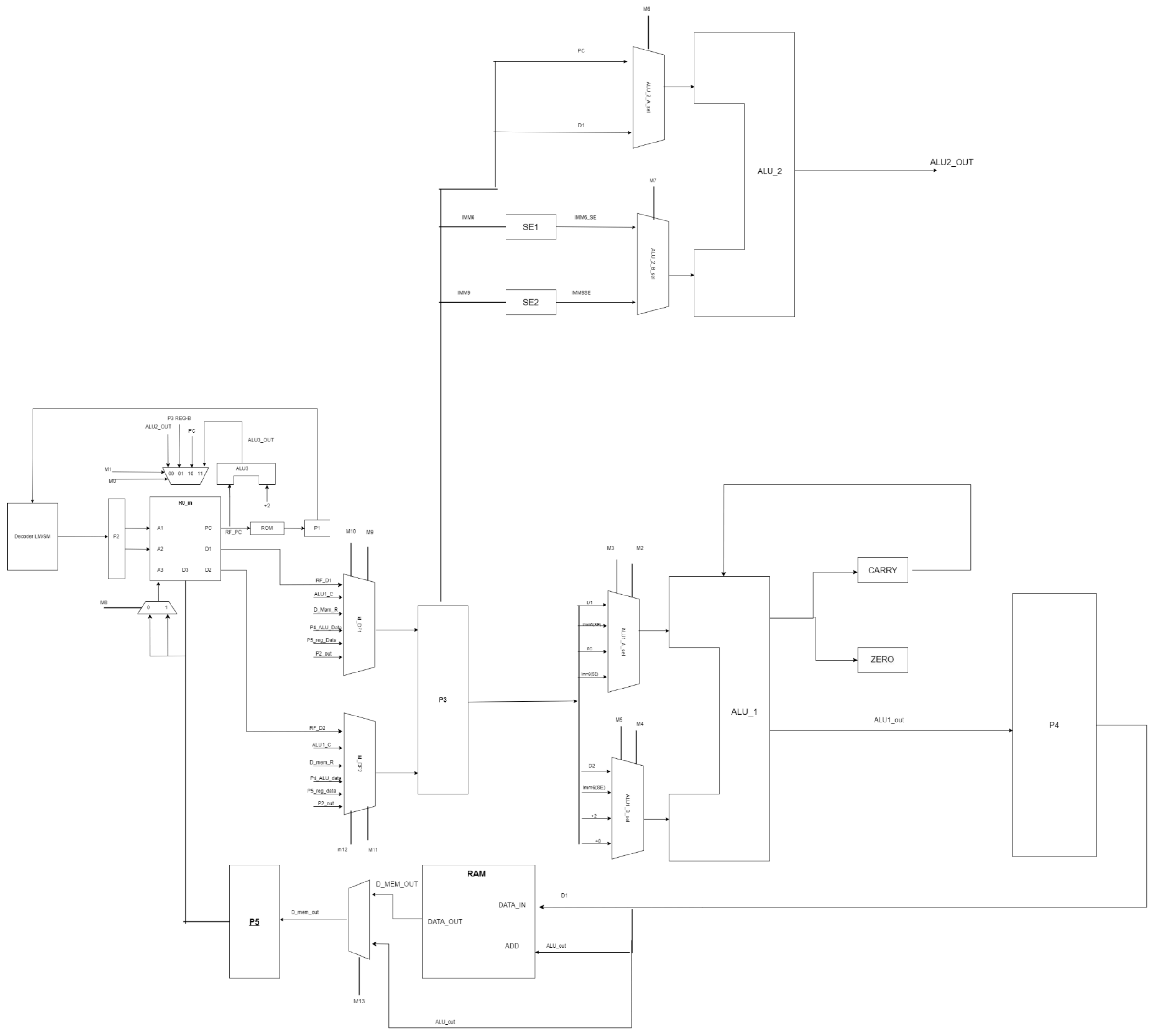
Sachin Awasthi (22b1264)

Overview

The IITB-RISC-23 is a 16-bit computer system with 8 registers. It follows a 6-stage pipeline (Instruction fetch, instruction decode, register read, execute, memory access, and write back). The program counter (PC) is stored inside Register 0. The ISA consists of 14 instructions with further possible variations in ADD and NAND instructions. The entire architecture has been optimized to give maximum performance (and thus high IPC).

Datapath

The following is a diagram of the entire datapath -



Some of the main components are the Register File, RAM, ROM, Pipeline Registers, and ALUs.

The format used for the pipeline registers is -

P1 =>	PC	IR	
P2 =>	PC	IR	

P3 =>	PC	IR	D1	D2
P4 =>	PC	IR	D1	ALU_OUT
P5 =>	IR	ALU_OUT/DMEM_OUT		

(Each block is 16 bits long)

Control Signals

Generated control signals for all MUXs, register file and memory write, PC write, ghosting signals, and hazard mitigations. The following Google sheet contains all the control signals and their logic -
https://docs.google.com/spreadsheets/d/1vHsMez2OwsR91mkRV6jaUmuj-n2em9IPGsOgrgyj6fM/edit?usp=sharing

Ghost Instruction => Used OPCODE = 1110 as a 'ghost instruction', i.e., it did not modify the memory, register file, or carry zero flags. It had no effect on the system anywhere and simply passed through the pipeline.

To disable/remove/flush out an instruction, we simply changed its opcode to 1110, and thus it became a ghost instruction.

Hazard Detection and Mitigation

I. Data Dependency

A. Destination Register = Source Register
   1. Identified a possible dependency by checking the source and destination register addresses in the instructions
   2. Used Data forwarding to send the updated data register data from ALU, Data Memory, Pipeline Registers P4, and P5

B. Special Case I - Load instruction
   1. Identified in IF/ID stage if the very next instruction depends on the Load instruction
   2. Stalled the pipeline for 1 cycle so that data can be loaded from memory

C. Special Case II - Source Address R0
   1. Identified if an instruction needs to read R0

2. Used the value of PC for the same instruction (stored in pipeline register) instead of reading register file (which will have updated the value of PC)

II. Branching

A. Simple branching - BEQ/BLT/BLE/JAL/JLR/JRI

1. Identified the destination address as well as whether branching would occur (for BEQ/BLT/BLE) in the EX stage. Updated the value of R0 accordingly.

B. Special Case III - Destination Address R0

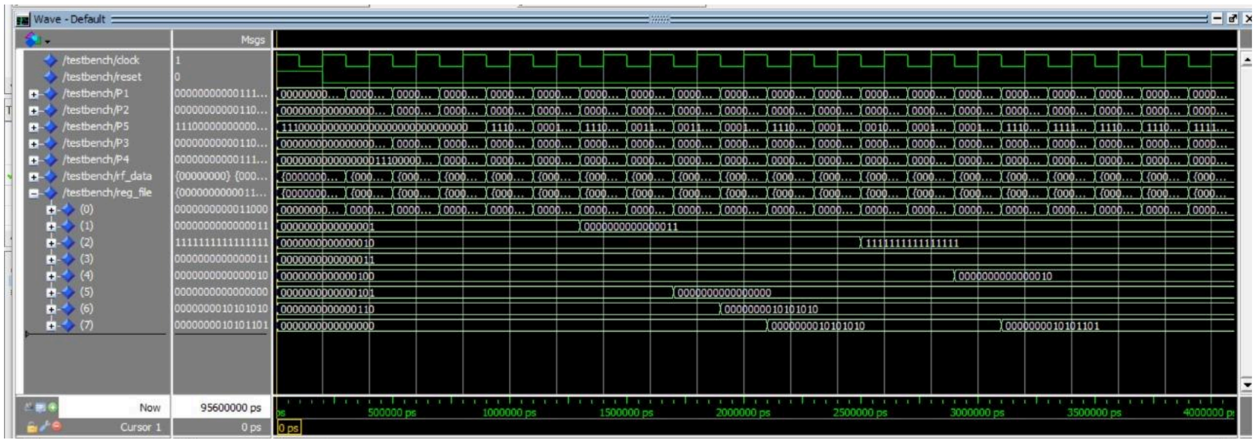
1. Identified whether the final destination address for an instruction is R0 and updated its value. Ghosted the intermediate instructions which are no longer valid.

Simulation Results

<u>Simulation #1</u>

The output shown is all pipeline registers and the 8 registers in the register file.

This code has multiple cases of data dependency and the output received is as expected.



```
Wave - Default
                          Msgs
/testbench/clock          1
/testbench/reset          0
/testbench/P1             0000000000000111...
/testbench/P2             0000000000000110...
/testbench/P5             11100000000000...
/testbench/P3             0000000000000110...
/testbench/P4             0000000000000111...
/testbench/rf_data        {00000000} {000...
/testbench/reg_file       {000000000000011...
  (0)                     0000000000011000
  (1)                     0000000000000011
  (2)                     1111111111111111
  (3)                     0000000000000011
  (4)                     0000000000000010
  (5)                     0000000000000000
  (6)                     0000000010101010
  (7)                     0000000010101101
Now                       95600000 ps
Cursor 1                  0 ps
```

ROM data for the above simulation (R0 initialized with 0, R1 with 1, etc.):-

0001010001001000 (ADA R2 R1 R1)

0010010001001010 (NDC R2 R1 R1)

0011101000000000 (LLI R5 000000000)

0011110010101010 (LLI R6 010101010)

0001101110111000 (ADA R5 R6 R7)

0001001010010001 (ADZ R1 R2 R2)

0001111110010100 (ACA R7 R6 R2)

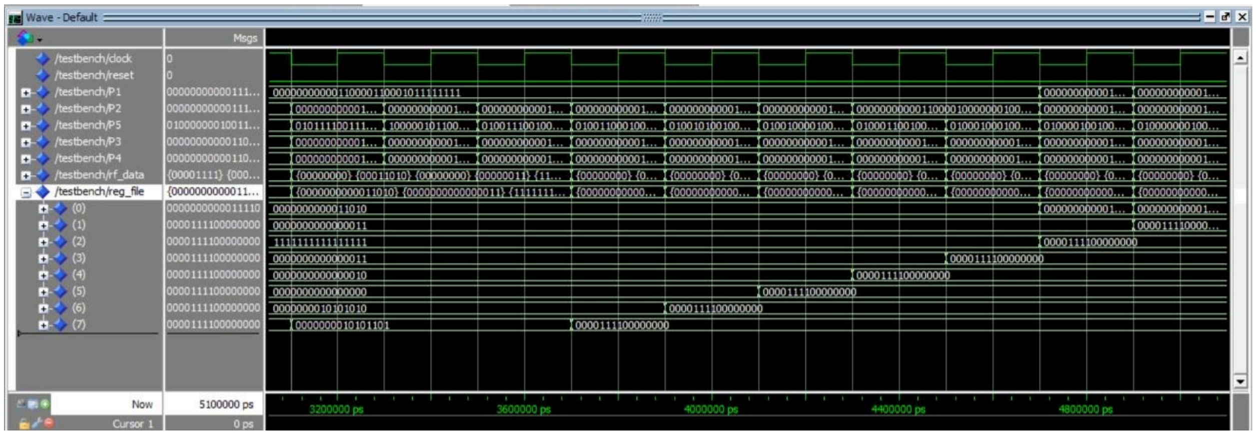
0010000010010100 (NCU R0 R2 R2)

0001010011100000 (ADA R2 R3 R4)

0001100110111011 (AWC R4 R6 R7)

1110111001111101 (Ghost Instruction)

Simulation #2

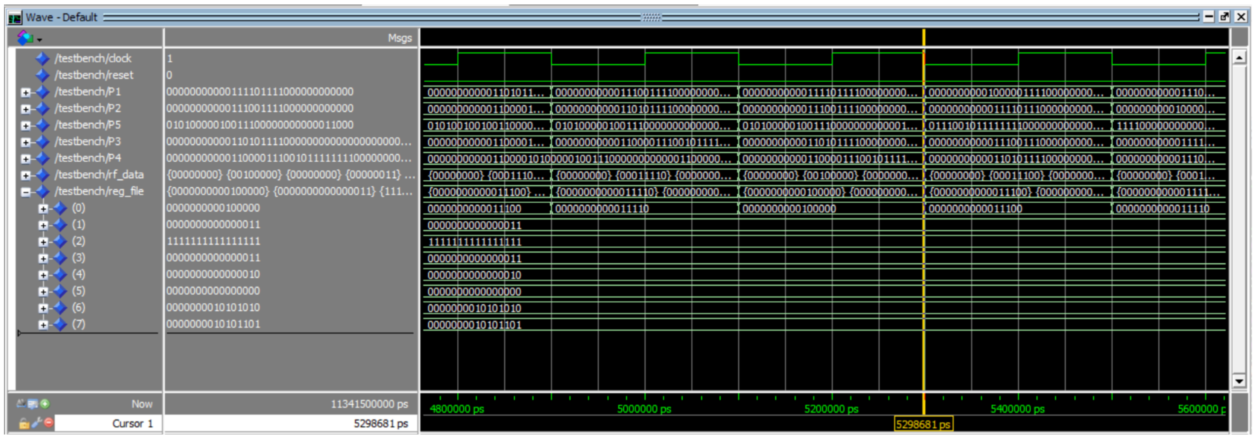


```
Wave - Default
                          Msgs
/testbench/clock          0
/testbench/reset          0
/testbench/P1             00000000000111...
/testbench/P2             00000000000111...
/testbench/P5             0100000000100 11...
/testbench/P3             00000000000110...
/testbench/P4             00000000000110...
/testbench/rf_data        {00001111} {000...
/testbench/reg_file       {00000000000011...
  (0)                     0000000000011110
  (1)                     0000111100000000
  (2)                     0000111100000000
  (3)                     0000111100000000
  (4)                     0000111100000000
  (5)                     0000111100000000
  (6)                     0000111100000000
  (7)                     0000111100000000
Now                       5100000 ps
Cursor 1                  0 ps
```

LM 001 011111111

RAM contains 0F00H at all locations, every cycle each register gets loaded with this value since all bits are high.

Simulation #3



```
Wave - Default
                          Msgs
/testbench/clock          1
/testbench/reset          0
/testbench/P1             00000000000111101111000000000000
/testbench/P2             00000000000111001111000000000000
/testbench/P5             01010000010011100000000000011000
/testbench/P3             000000000001101011110000000000000000000...
/testbench/P4             000000000001100001110010111111110000000...
/testbench/rf_data        {00000000} {00100000} {00000000} {00000011} ...
/testbench/reg_file       {0000000000100000} {0000000000000011} {111...
  (0)                     0000000000100000
  (1)                     0000000000000011
  (2)                     1111111111111111
  (3)                     0000000000000011
  (4)                     0000000000000010
  (5)                     0000000000000000
  (6)                     0000000010101010
  (7)                     0000000010101101
Now                       11341500000 ps
Cursor 1                  5298681 ps
```

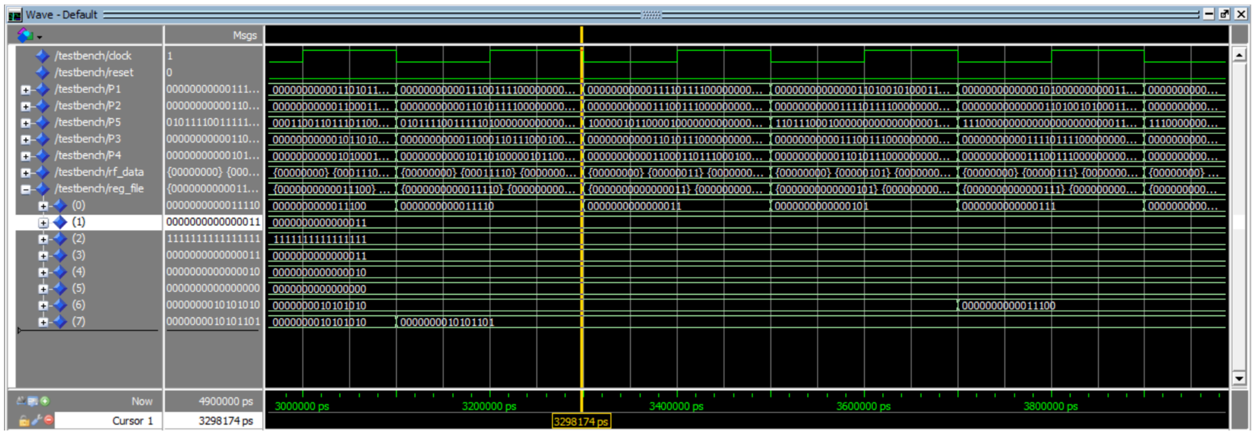
1000001011000010 (BEQ R1 R3 000010)

The value of R0 jumps to 11100 (decimal value 28) since R1 = R3 = 0000000000000011

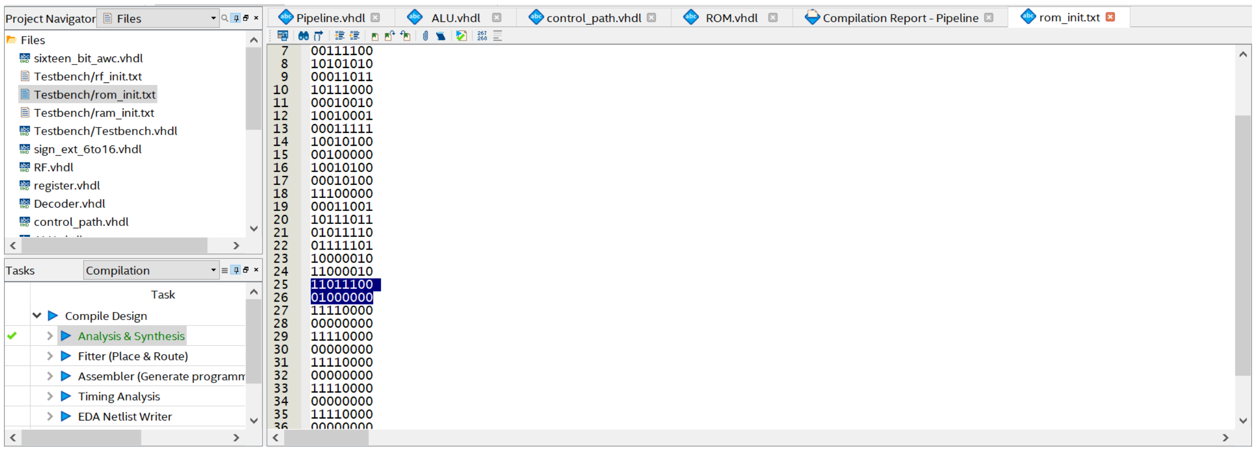
The value of PC of BEQ instruction is 11000 (decimal value 24) and immediate is 000010 (decimal value 2)

Thus, BEQ jumps to required location, PC + Imm*2

Simulation #4


```
Wave - Default
                      Msgs
/testbench/clock      1
/testbench/reset      0
/testbench/P1         00000000000111...    000000000001101011...  |000000000001110011110000000...  |000000000001111011110000000...  |000000000000001101001010001 1...  |000000000000001010000000000 11...  |0000000000...
/testbench/P2         00000000000110...    000000000001100011...  |000000000001101011110000000...  |000000000001110011100000000...  |000000000001111011110000000...    |000000000000001101001010001 1...  |0000000000...
/testbench/P5         01011110011111...    000110011011101100...  |010111100111110100000000000...  |100000101100001000000000000...  |110111000100000000000000001...    |111000000000000000000000001 1...   |1110000000...
/testbench/P3         00000000000110...    000000000001011010...  |000000000001100011011100010 0...  |000000000001101011100000000...  |000000000001110011100000000...   |000000000001111011110000000...    |0000000000...
/testbench/P4         00000000000101...    000000000001010001...  |000000000001011010000010110 0...  |000000000001100011011100010 0...  |000000000001101011100000000...   |000000000001110011100000000...    |0000000000...
/testbench/rf_data    {00000000} {000...   {00000000} {0001110...  |{00000000} {00011110} {0000000...  |{00000000} {00000011} {0000000...  |{00000000} {00000101} {0000000...  |{00000000} {00000111} {0000000...  |{00000000} ...
/testbench/reg_file   {000000000000 11...  {0000000000011100} ...  |{0000000000011110} {000000000...  |{0000000000000011} {000000000...  |{0000000000000101} {000000000...  |{0000000000000111} {000000000...  |{0000000000...
  (0)                 0000000000011110     0000000000011100        |0000000000011110                 |0000000000000011                 |0000000000000101                 |0000000000000111                  |0000000000...
  (1)                 0000000000000011     0000000000000011
  (2)                 1111111111111111     1111111111111111
  (3)                 0000000000000011     0000000000000011
  (4)                 0000000000000010     0000000000000010
  (5)                 0000000000000000     0000000000000000
  (6)                 0000000010101010     0000000010101010                                                                                                              |0000000000011100
  (7)                 0000000010101101     0000000010101010        |0000000010101101

Now        4900000 ps    3000000 ps    3200000 ps    3400000 ps    3600000 ps    3800000 ps
Cursor 1   3298174 ps    3298174 ps
```


```
Project Navigator   Files
Files
  sixteen_bit_awc.vhdl
  Testbench/rf_init.txt
  Testbench/rom_init.txt
  Testbench/ram_init.txt
  Testbench/Testbench.vhdl
  sign_ext_6to16.vhdl
  RF.vhdl
  register.vhdl
  Decoder.vhdl
  control_path.vhdl

Tasks   Compilation
  Task
  Compile Design
✓   Analysis & Synthesis
    Fitter (Place & Route)
    Assembler (Generate programm
    Timing Analysis
    EDA Netlist Writer
```

```
Pipeline.vhdl   ALU.vhdl   control_path.vhdl   ROM.vhdl   Compilation Report - Pipeline   rom_init.txt
7   00111100
8   10101010
9   00011011
10  10111000
11  00010010
12  10010001
13  00011111
14  10010100
15  00100000
16  10010100
17  00010100
18  11100000
19  00011001
20  10111011
21  01011110
22  01111101
23  10000010
24  11000010
25  11011100
26  01000000
27  11110000
28  00000000
29  11110000
30  00000000
31  11110000
32  00000000
33  11110000
34  00000000
35  11110000
36  00000000
```

1101110001000000 (JLR R6 R1)

R0 jumps to 0000000000000011 (in front of yellow cursor) which is the value stored in R1.

PC of the instruction is 26 in decimal and the value stored in R6 is 0000000000011100 (decimal equivalent 28). Thus, PC + 2 is stored in R6

Interesting Aspects

Use of a Ghost instruction: Using an unused opcode 1110 in order to disable instructions. To disable an instruction, changed its opcode to 1110 and such an instruction was defined to not cause any changes to the register files and memory.

Implementation of LM/SM: Used the Instruction Decode stage to send a set of Load/Store instructions according to which bits are high. The number of cycles taken to execute this instruction is equal to the number of high bits plus one cycle delay.

Implementation of ADD/NAND if carry/zero set: If the condition of carry/zero set is not satisfied, ghost the instruction so that it will not modify the register file.

Python script: Created a Python script to easily modify both ROM and RAM

Project Code on GitHub

The following link contains the entire project code in a GitHub repository - https://github.com/Ingenio17/IITB-RISC-23

Individual Contribution

Harshil Singla: Datapath design, Hazard mitigation, Decoder

Saumya Shah: Datapath design, Hazard mitigation, RF, ROM, Testbench

Himanshi Shende: Control signals, Python Code, RF, ROM, Testbench

Sachin Awasthi: Components (ALU's, sign extenders, RAM), Report flow chart and diagrams