EE309 Course Project
Microprocessers

Aditya Kabare 21D070009
Anuj Gupta 21D070014
Tanishka Pradhan 210040159
Saurav Kumar 21D070063

May 3, 2023

1 Aim

Design a 6-stage pipelined processor, IITB-RISC-23, whose instruction set architecture is provided. IITB-RISC is a 16-bit very simple computer developed for the teaching that is based on the Little Computer Architecture. The IITB-RISC-23 is a 16-bit computer system with 8 registers. It should follow the standard 6 stage pipelines (Instruction fetch, instruction decode, register read, execute, memory access, and write back). The architecture should be optimized for performance, i.e., should include hazard mitigation techniques. Hence, it should have implemented forwarding mechanism.

2 CPU Design(data path)

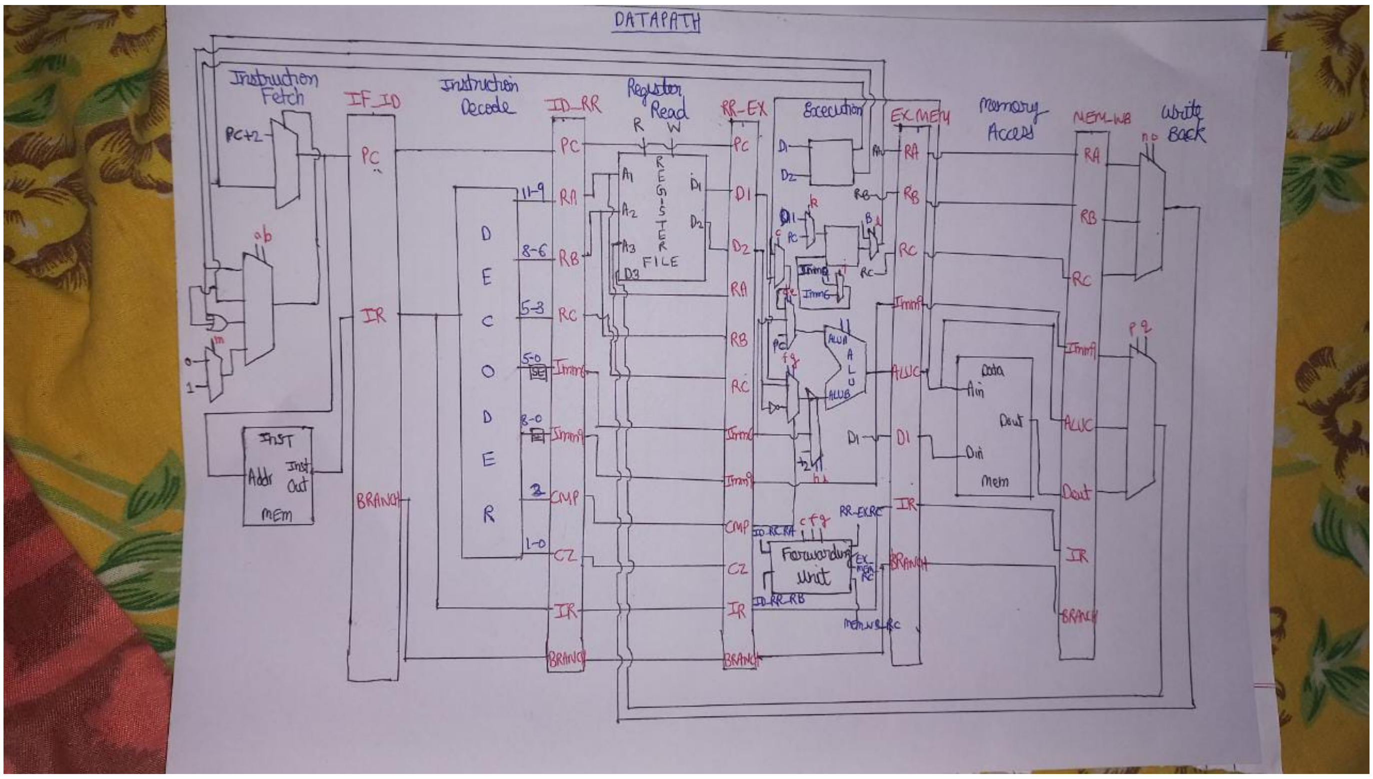


Figure 1: CPU Datapath

3 Pipeline Stages

- The CPU has one clock and each stage takes 1 clock cycle to complete.
- Read operation happens on rising edges and Write operation happens on falling Edges.

3.1 Instruction Fetch(IF)

- Here in this stage we will fetch instruction from our Instruction memory(16 bits) and store it in our Instruction Register(Present in our IF/ID interface).Our PC address is being fed into address line of our Instruction Memory.

- Our PC gets updated (either to PC+1 or gets branched to some PC address depending on branching condition) and gets stored in our PC register present in IF/ID interface.

3.2 Instruction Decode(ID)

- In this stage we pass the data stored in our IR register(our instruction) through our Instruction Decoder

- On Passing through the inst decoder, these bits are collected :
  (**15-12**) – as Opcode.
  (**11-9**) — as Ra (3-bit register)
  (**8-6**) — as Rb (3-bit register)
  (**5-3**) — as Rc (3-bit register)
  (**5-0**) — as Imm6 (16-bit Sign Extended Immediate-6 register)
  (**8-0**) — as Imm9 (16-bit Sign Extended Immediate-9 register)
  (**2**) —— as CMP (1-bit register)
  (**1-0**) — as CZ (2-bit register)

- We forward data stored in PC,IR and BRANCH register to next Interface(ID/RR)

3.3 Register Read(RR)

This stage contains a Register File(RF) which contains 8 16-bit Registers(R0-R7), with the following ports:

- Two address read lines A1,A2
- One address write line A3
- two Data out lines D1,D2 corresponding to data read in A1,A2 lines
- One data in line D3 to be written on address line A3
- Data stored in previous interface(ID/RR) registers along with D1,D2 is forwarded to next interface(EX)

3.4 Execution(EX)

- This stage contains one ALU for all ALU operations and one each of *equal to* and *less than* comparators for the Branch instructions.
- It contains several MUXes to decide input lines to the ALU depending on the instruction.
- The output line ALU-C of the main ALU is forwarded to next interface/stage along with data stored in some other registers present in previous interface/stage.

3.5 Memory Access(MEM)

In this stage we have a Data Memory with the following ports and functions:

- One Data-in line **(Din)**: Data stored in D1 register of our previous interface is passed through this line
- One Address line **(ADD)**: Data stored in ALU-C register of our previous interface is passed through this line
- The output line **(Dout)** is forwarded to next interface/stage along with data stored in a few registers present in previous interface/stage

Note that data is read from address (ADD) to Dout line on the rising edge while it data (Din) is written at the address on the falling edge.

3.6 Write Back(WB)

This stage contains two MUXes to control which data and address to be taken for writing in RF.

- MUX1 has 3 Input lines Ra, Rb, Rc connected to the A3 line of our RF.
- MUX2 has 3 Input lines Imm9, ALU-C, Dout(data memory) connected to D3 line of our RF.
- The select lines of the MUXes will be based on our instructions.

4 Hazard Mitigation

All instructions have some involvement in causing hazards in our data flow. A simple overview is given here.

- ALU, LLI and SW instructions:

  All these pose problems because of the **Write Back Delay**. All these instructions alter the register values and can cause problems with other succeeding instructions accessing those registers. The workaround is discussed in the next section.

- LW instruction:

  A compulsory one stage delay has to be given for it to fetch information from the **Memory Read** stage. It is considered in this program that the compiler automatically inserts a Null instruction after this instruction.

- Branch and Jump Instructions:

  Branch instructions have a conditional case which if turned true in the "EX Stage", causes a signal "BRANCH" to turn on for instructions in "IF", "ID" and "RR" stages. This prevents them from taking any further action in the pipeline by turning off any writes associated with these.

  Similarly for Jump instructions, same procedure happens as previously but unconditionally.

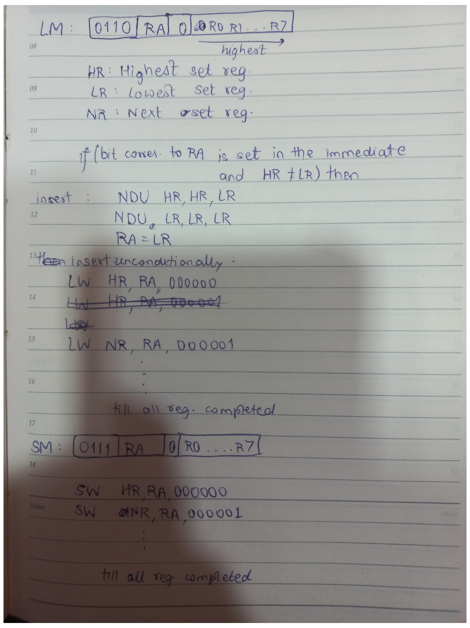
5 Data Forwarding

Apart from regular pipeline registers between the states, we have made some temporary registers after Write Back state, viz. **WB_OUT_*Name of Register***.
In general, we are comparing the address of destination register stored in the pipeline registers between Execution and Memory Access state, Memory Access and Write Back state and those temporary registers after the Write Back state with the address of souce registers in the pipeline register between Register Read and Execution state. Destination registers are different for different instructions, hence had to use multiple if-else statements to tackle all the possibilities. We used the values in the pipeline registers **EX_MEM_ALUC, MEM_WB_ALUC, WB_OUT_ALUC, EX_MEM_IMM9, MEM_WB_IMM9, MEM_WB_Dout, WB_OUT_IMM9 and WB_OUT _Dout** (different for different dependencies) instead of data of source registers when any dependency is encountered. (The corresponding Dout for EX_MEM is not required as immediate dependencies cannot be tackled in LOAD instructions.)

- **For all the arithmetic instructions except ADI** Destination register is R_c.
- **For ADI instruction** Destination register is R_b.
- **For LLI, LW, JAL and JLR instructions** Destination register is R_a.

6 Note on LM/SM

The LM and SM instructions are considered to be decoded into other simpler instructions already in the instruction set. The conversion logic is given in figure 2.



```
LM : | 0110 | RA | 0 | R0 R1 . . . R7 |
                       ---------------->
                            highest

     HR : Highest set reg.
     LR : Lowest set reg.
     NR : Next set reg.

     if (bit corres. to RA is set in the immediate
                        and HR ≠ LR) then
insert :   NDU  HR, HR, LR
           NDU  LR, LR, LR
           RA = LR
~~then~~ insert unconditionally :
     LW  HR, RA, 000000
     ~~LW  HR, RA, 000001~~
     LW  NR, RA, 000001
              :
              :
          till all reg. completed

SM : | 0111 | RA | 0 | R0 . . . . R7 |

     SW   HR, RA, 000000
     SW   NR, RA, 000001
              :
              :
        till all reg. completed
```

Figure 2: LM/SM compiler logic

References

[1] https://courses.cs.washington.edu/courses/cse378/10sp/lectures/

[2] https://nptel.ac.in/courses/106105165