EE 309 Project Report
PIPELINED RISC-IITB

Jainesh Mehta - 210071001
Ishaan Manhar - 210070033
Harshraj Chaudhari- 210040060
Kushagra Ghelot - 21D070041

The project involved implementing a 6 stage- pipelined simple computer with the following stages :

1. **Instruction fetch** :
   This state is responsible for reading the contents of PC(R0) and then fetching the corresponding instruction from memory and passing it on to the IFID pipeline register.

2. **Instruction decode** :
   This state is responsible for collecting the instruction from the IFID pipeline register and then passing on the various control signals controlling the entire system to the IDRR pipeline register, this is also the state where the system recognises the LM/SM instructions and the decoder, passes on a LW/SW instruction to the IDRR register, there is a register counter whose value increases from 0 to 7 with each clock cycle and once the register hits 7, the control signal LMSM_hazard turns 0 and the stalling is stopped.

   There is also a “Cancel bit” responsible for flushing out the contents of the pipeline register

   We have also implemented an adder within this stage solely for determining the JUMP address.

3. **Register Read :**
   The state responsible for reading the contents of the registers with the corresponding addresses store in Register source1 (Rs1) and Register source2 (Rs2) , there are muxes (MUXA and MUXB)

provided in order to control the data that is passed on to the pipeline register, there is also an adder to add contents of Rf_D1 and imm*2, which is required for instructions where Rx + Imm*2 is required. Similar to the IDRR pipeline register, even the RREX pipeline register has the “cancel bit” required for flushing out the contents of the register.

4. **Execution state** :

   This is the state where ALU1A is implemented, ALU1A has a 2 bit control(ALUsel) and ADD,SUB, NAND is implemented based on the control bit sent by the instruction decoder, there are MUXes to control what is given as the inputs to the ALU, the carry and zero flags are implemented using a d-flipflop while MUXes are used again to control whether the carry and zero flags will be modified or not. There is also an adder present to implement PC+Imm*2 which might be required to WB.

5. **Memory access state** :

   This is the only state where any kind of memory data can be read or written (in the RAM), there are separate data lines for data read (Data out) and data write (Data in), data write is controlled by the mem_wr control signal sent by the instruction decoder, the memory address is implemented using a 2x1 MUX with input lines as ALU1C and ALU3C, there is a MUX also implemented to decide what is written back to the register file and this is passed on to the MEMWB register pipeline.

6. **Write Back state** :

   This is where we write back to the registers using Rf_D3 as the data to be written back and Rf_A3 as the address to which the data is to be written back, as always there is a control bit Rf_wr controlling whether registers are to be kept write enabled or not.

Instructions :-

⇒ NCZ / NCU / NCC / ADA / ADC / ADZ / AWC / ACA / ACC / ACZ / ACW / NDU / NDC / NDZ

RF_PC_R → ROM_Address, ALU_2-A
+2 → ALU_2-B
ALU_2C → PC_IF
ROM_DATA(11-9) → RS1
ROM_DATA(8-6) → RS2
ROM_DATA(5-3) → RD
RF_D1 → ALU_1A
RF_D2 → ALU_1B
If (cond^n) :-
ALU_1C → RF_D3

⇒ LLI RA Imm

RF_PC_R → ROM Add, ALU_2 A
ROM DATA(11-9) → RD
ROM DATA(8-0) $\xrightarrow{SE}$ RF_D3
+2 → ALU_2B
ALU_2C → RF_PC_WR

⇒ LW RA, RB, Imm

RF_PC_R → ROM Add, ALU_2A
ROM Data(11-9) → RFA3
ROM Data(8-6) → RFA1
RFD1 → ALU_1A
ROM Data(5-0) $\xrightarrow{SE}$ ALU_1B
ALU_1C → RAM Add
RAM Data → RFD3
+2 → ALU_2B
ALU_2C → RF_PC_WR

⇒ SW RA, RB, Imm

RF_PC_R → ROM Address, ALU_2A
+2 → ALU_2B
ALU_2C → RF_PC_WR
ROM Data (11–9) → RF_{A_1}
ROM Data (8–6) → RF_{A_2}
RF_{D_1} → ~~ALU_2A~~ RAM Datain
RF_{D_2} → ALU_3A
ROM Data (5–0) → ALU_3B
ALU_3C → RAM_Add

⇒ ADI RB, RA, Imm

RF_PC_R → ROM_Add, ALU_2A
+2 → ALU_2B
ALU_2C → RF_PC_WR
ROM Data (11–9) → RF_{A_3}
ROM Data (8–6) → RF_{A_3}
RF_{D_1} → ALU_1A
ROM Data (5–0) $\xrightarrow{SE}$ ALU_1B
ALU_1C → RF_{D_3}

⇒ LM :- System stalled for 8 cycles. In the instruction decoder, LW instruction is passed 8 times, each with an updated memory location and an updated register destination.

⇒ SM :- System stalled for 8 cycles. Instruction decoder sends SW instruction 8 times while updating memory location (using immediate) and register ~~destination~~ for source.

⇒ BEQ RA, RB Imm / BLT / BLE

RF_PC_R → ROM Add, ALU2A, ALU3A
Rom Data (11-9) → RF_A1
Rom Data (8-6) → RF_A2
RF_D1 → ALU1A
RF_D2 → ALU2B
Rom Data (5-0) SE → ALU3B
If (cond^n) :-
ALU2 _ C → RF_PC_WR
else :-
ALU3 - C → RF_PC_WR

Cond^n :-
BEQ : (Z == 1)
BLT : (Z == 00 && C == 1)
PLE : (C == 1)

⇒ JALR RA, Imm

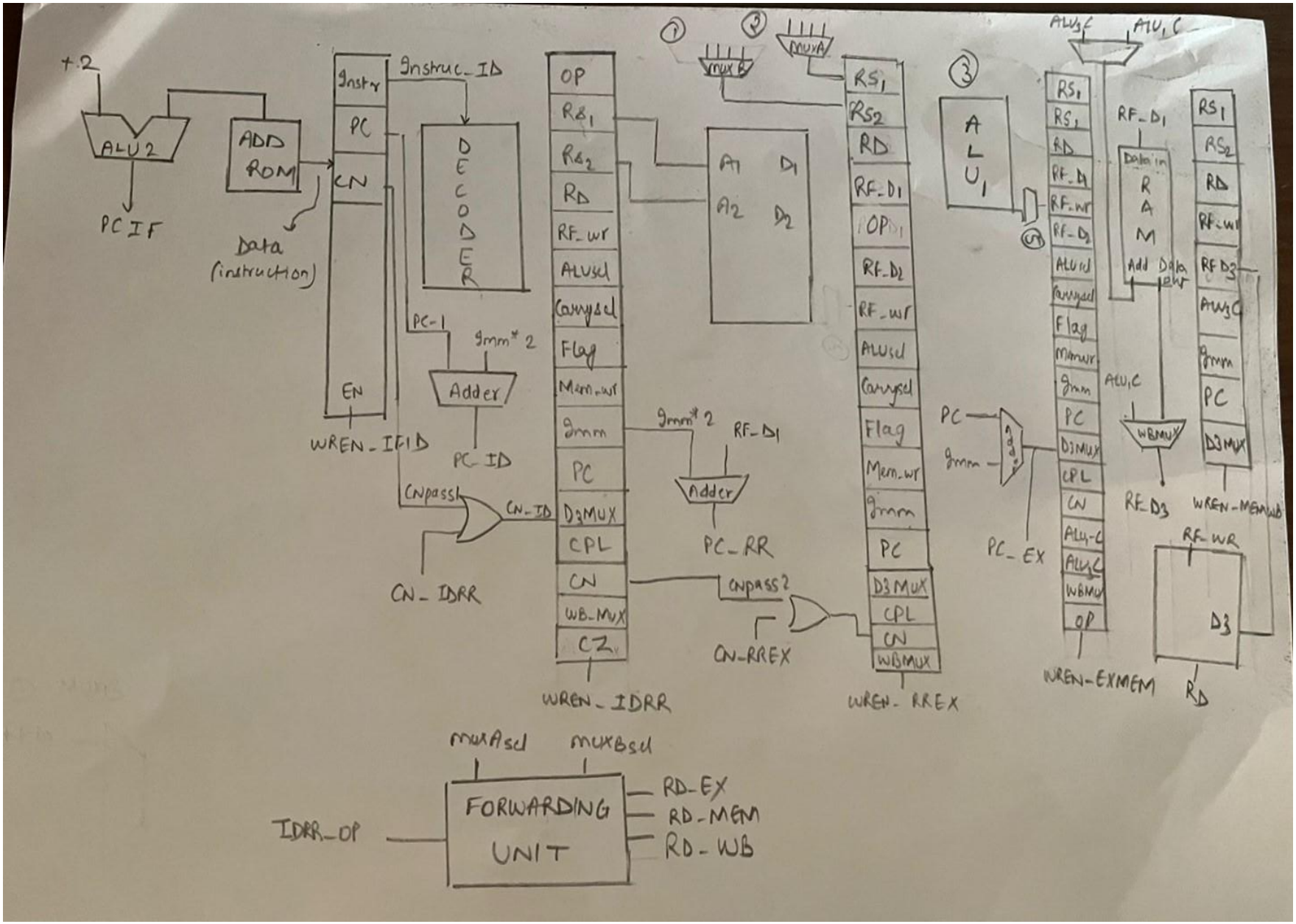
RF_PC_R → ROM Add, ALU2A, ALU3A
+2 → ALU2B
ROM Data (8→0) SE 2S → ALU3B
MEM, Data (11-9) → RF_A3
ALU2C → RFD3
ALU3C → RF_PC_WR

⇒ JLR RA, RB

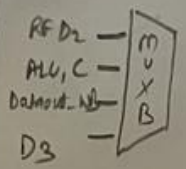
RF_PC_R → ROM Add, ALU2A, ALU3A
+2 → ALU2B
ROM_Data (11-9) → RF_A3
ALU2C → RFD3
ROM Data (8-6) → RFA2
RF_D2 → RF_PC_WR

⇒ JRI RA, Imm

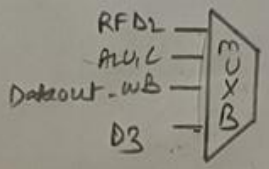
Rf_PC_R → ROM Add
Rom Data (11-9) → RFA1
RF_D1 → ALU3A
ROM Data SE 1S → ALU3B
ALU2C → RF_PC_WR


+2
ALU 2
PC IF
ADD ROM
Data (instruction)
Instr
PC
CN
EN
WREN_IFID
Instruc_ID
DECODER
PC-1
Imm*2
Adder
PC_ID
CNpass1
CN_ID
CN_IDRR
OP
RS1
RS2
RD
RF_wr
ALUsel
Carrysel
Flag
Mem_wr
Imm
PC
D3MUX
CPL
CN
WB_MUX
CZ
WREN_IDRR
A1
A2
D1
D2
Imm*2
RF_D1
Adder
PC_RR
CNpass2
CN_RREX
RF_D2
WREN_RREX
ALU1
ALU3C
ALU1C
RF_D1
Data in
RAM
Add
Data wr
WBMUX
RF_D3
WREN_MEMWB
RF_WR
D3
RD
WREN_EXMEM
PC_EX
muxA
muxB
muxAsel
muxBsel
FORWARDING UNIT
IDRR_OP
RD_EX
RD_MEM
RD_WB

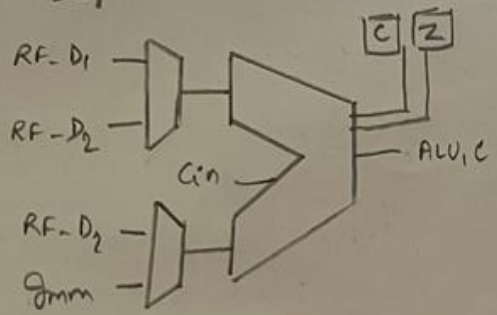
① MUX B



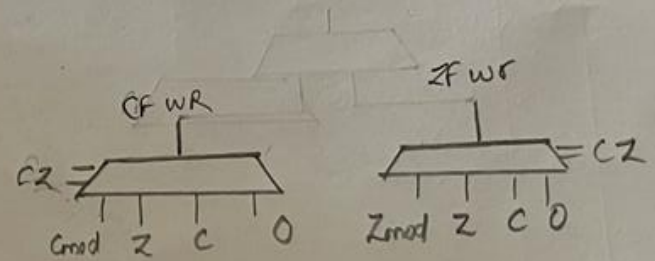
② MUX A



③ ALU



④ CF WR / ZF WR Logic



⑤ RF_WR – EX MUX