ISA Introduction:

- IITB-RISC is a 16-bit very simple computer developed for the teaching that is based on the Little Computer Architecture.
- IITB-RISC has 8 general purpose registers i.e. R0 to R7.
- This design have six stage pipeline architecture, namely
- Instruction Fetch (IF), Instruction Decode (ID), Register Read (RR), Execution (EX), Data memory (MEM) and Write Back (WB).
- Instruction set of IITB-RISC processor consists of three machine-code instruction formats namely Register (R), Immediate (I) and Jump (J).
- This architecture uses a condition code register which has two flags: Carry flag (C) and Zero flag (Z).
- This architecture is optimize for performance, it have hazard mitigation techniques i.e. hazard detection and forwarding technique.

Pipelined Processor:

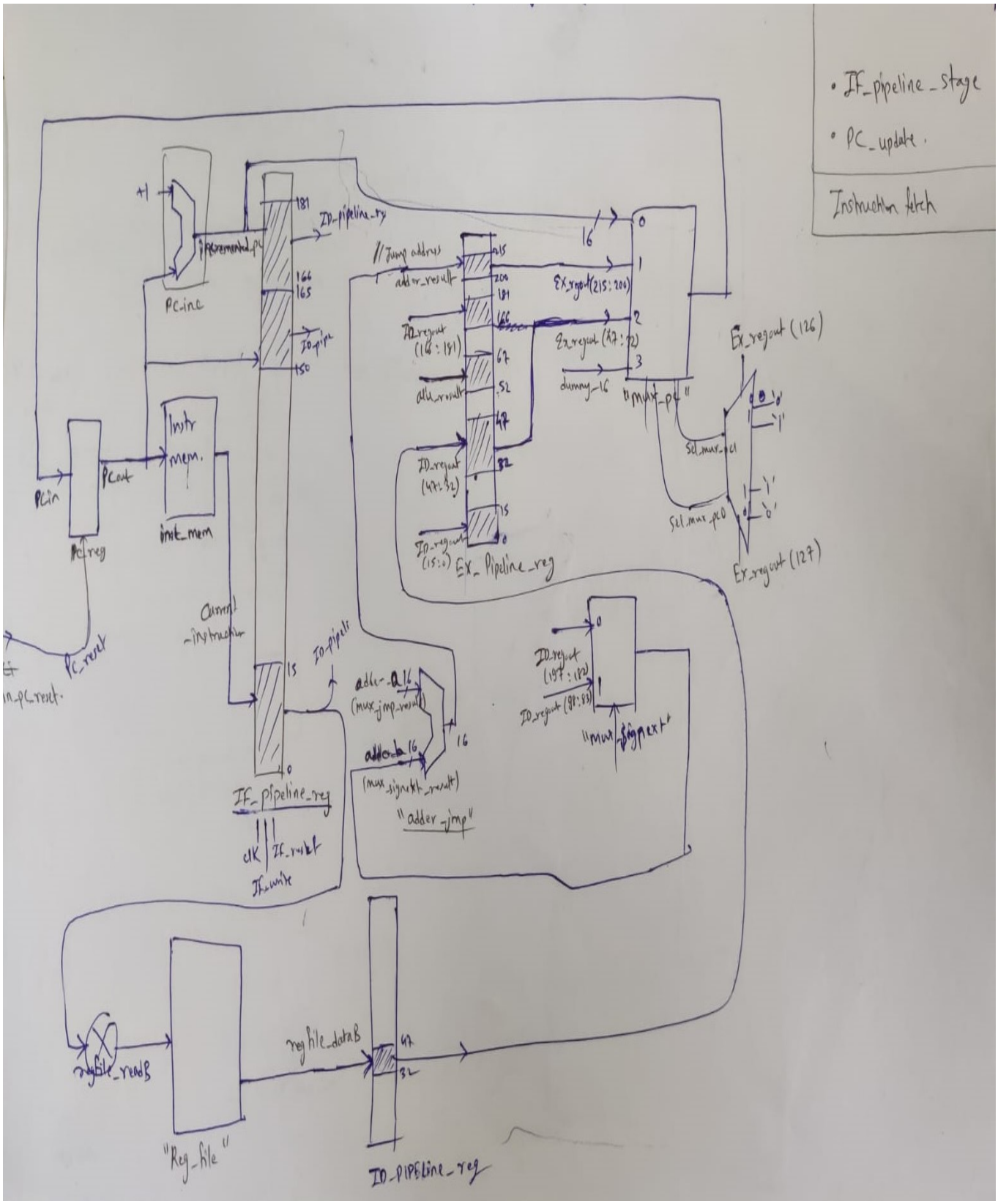
- Pipelined architecture improves the maximum operating frequency and throughput of processors by instruction level pipelining.
- In pipelining, divide datapath into nearly equal tasks, to be performed serially and requiring non-overlapping resources.
- Insert registers at task boundaries in the datapath. Registers pass the output data from one task as input data to the next task.
- Pipelining does not reduce the total time taken by an instruction, but it increases the number of instruction that can be executed together, and instruction throughput is increased.
- Even pipelining introduces latency in the output but the maximum frequency of operation is increased.

Pipeline Hazards:

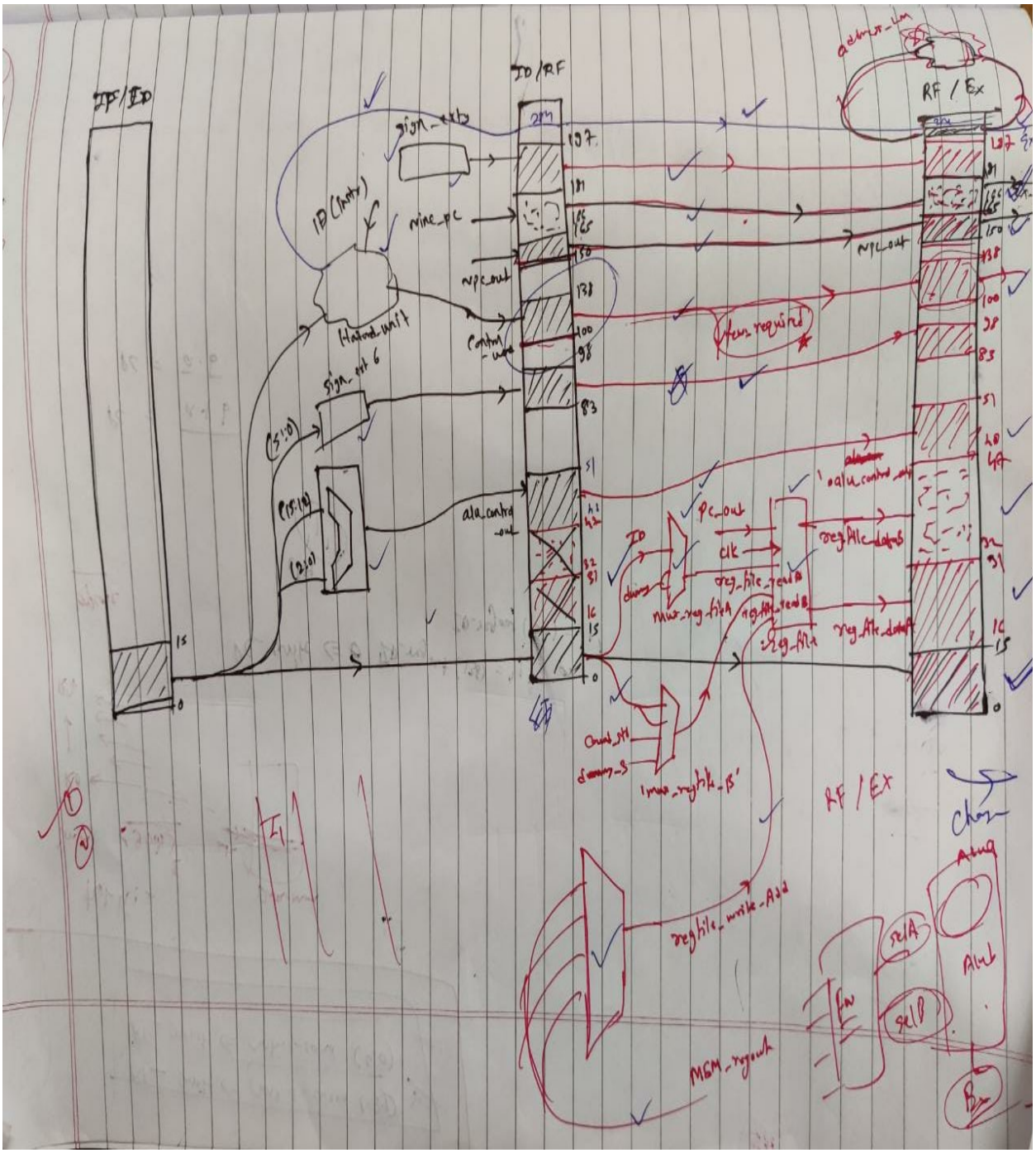
- Performance of any pipelined processor is degraded when an instruction depends on the result of previous instruction or any data which is not yet generated. In this case pipeline is stalled and processor send NOP instruction until the result for which the instruction was waiting is generated.
- In any pipelined architecture, three types of hazards occur they are control hazard, data hazard and structural hazard.
- To avoid these hazards, there is a need to forward data which is done by the forwarding unit. Hazards are resolved by Hazard detection and forwarding units.
- Compiler's understanding of how these units work can improve performance.

We have attached a README files in the folder, which will tells how to put data memory and instruction memory in the code and compile step by step using "do file".

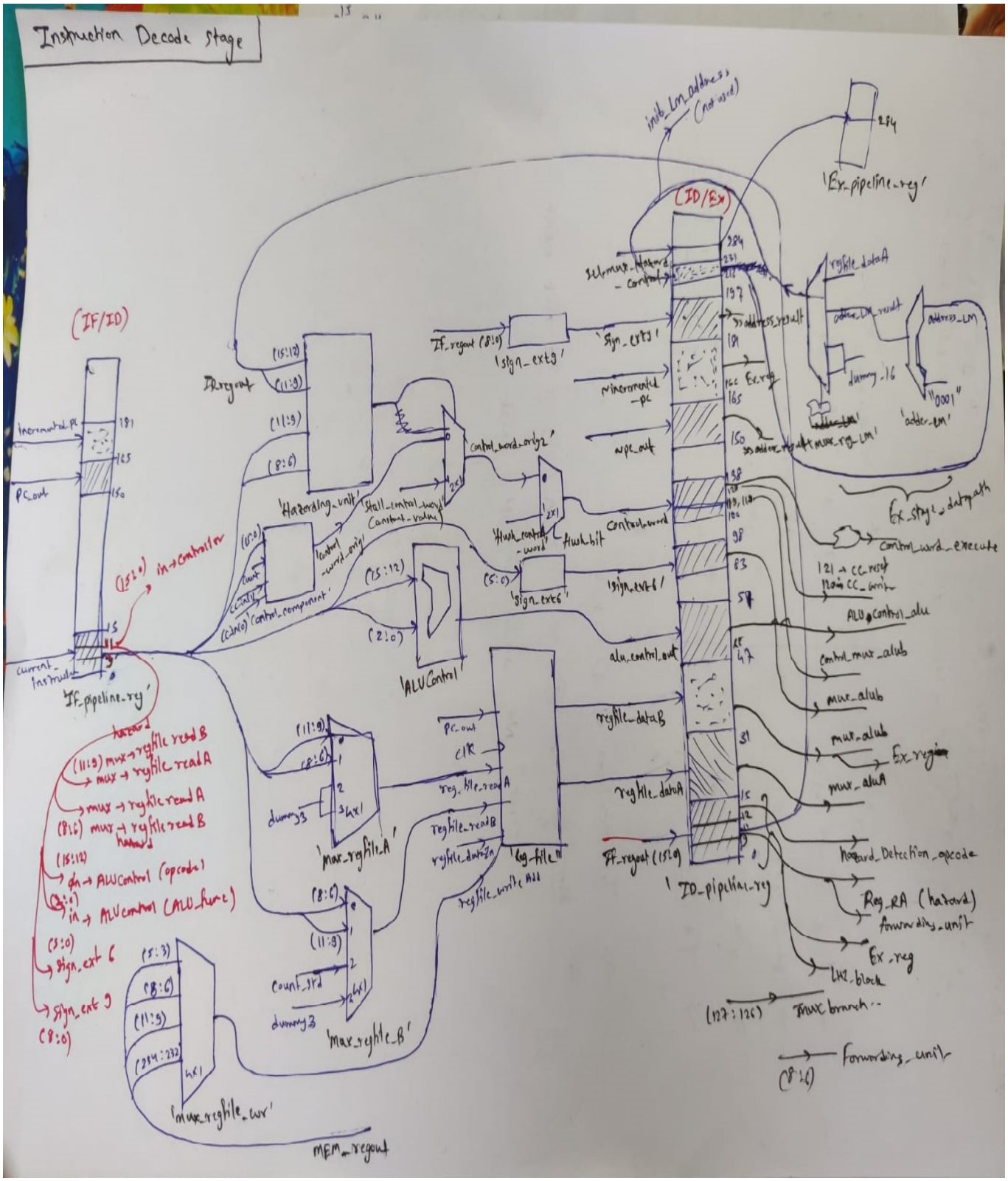
Instruction fetch stage:



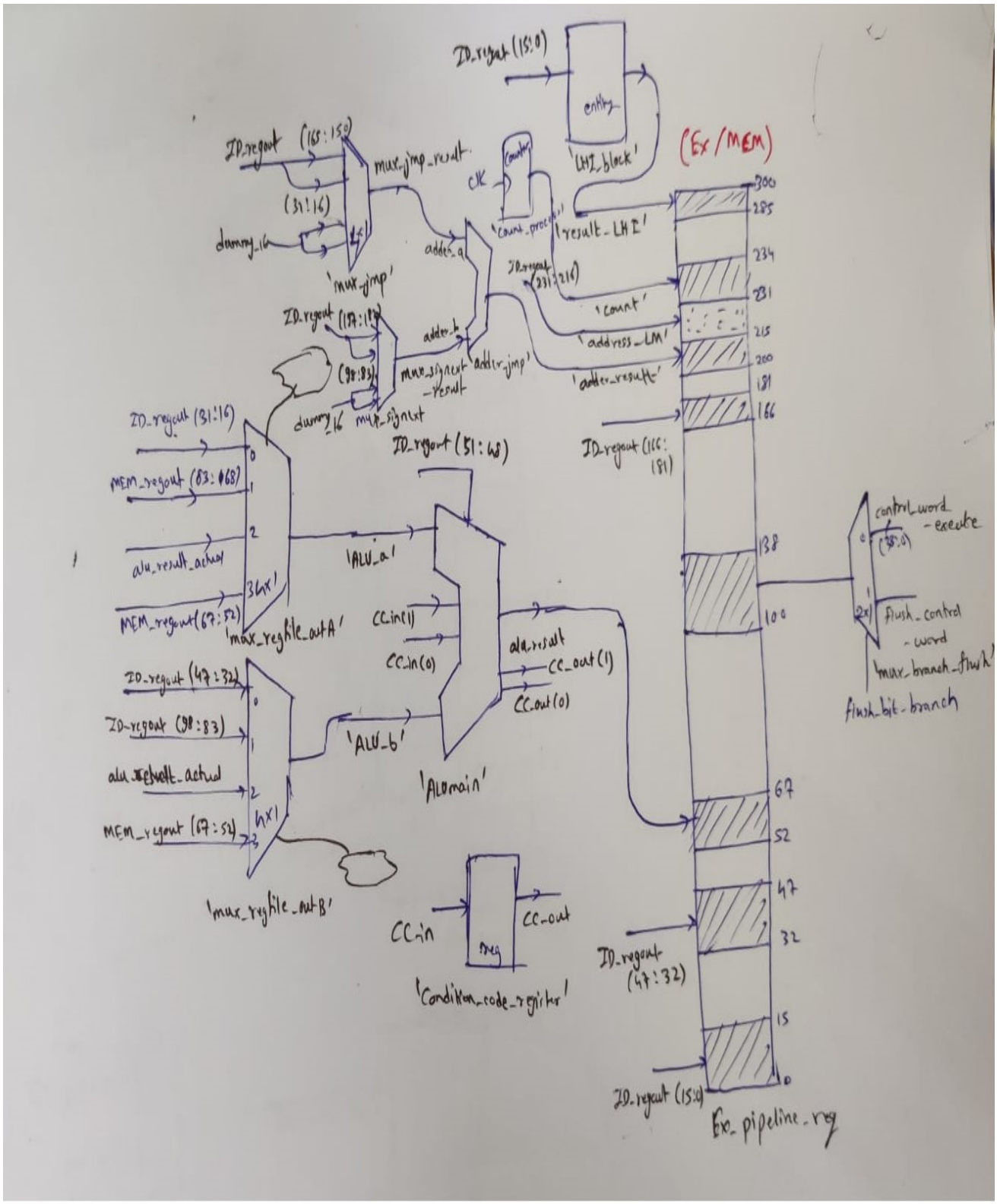
Register read stages:



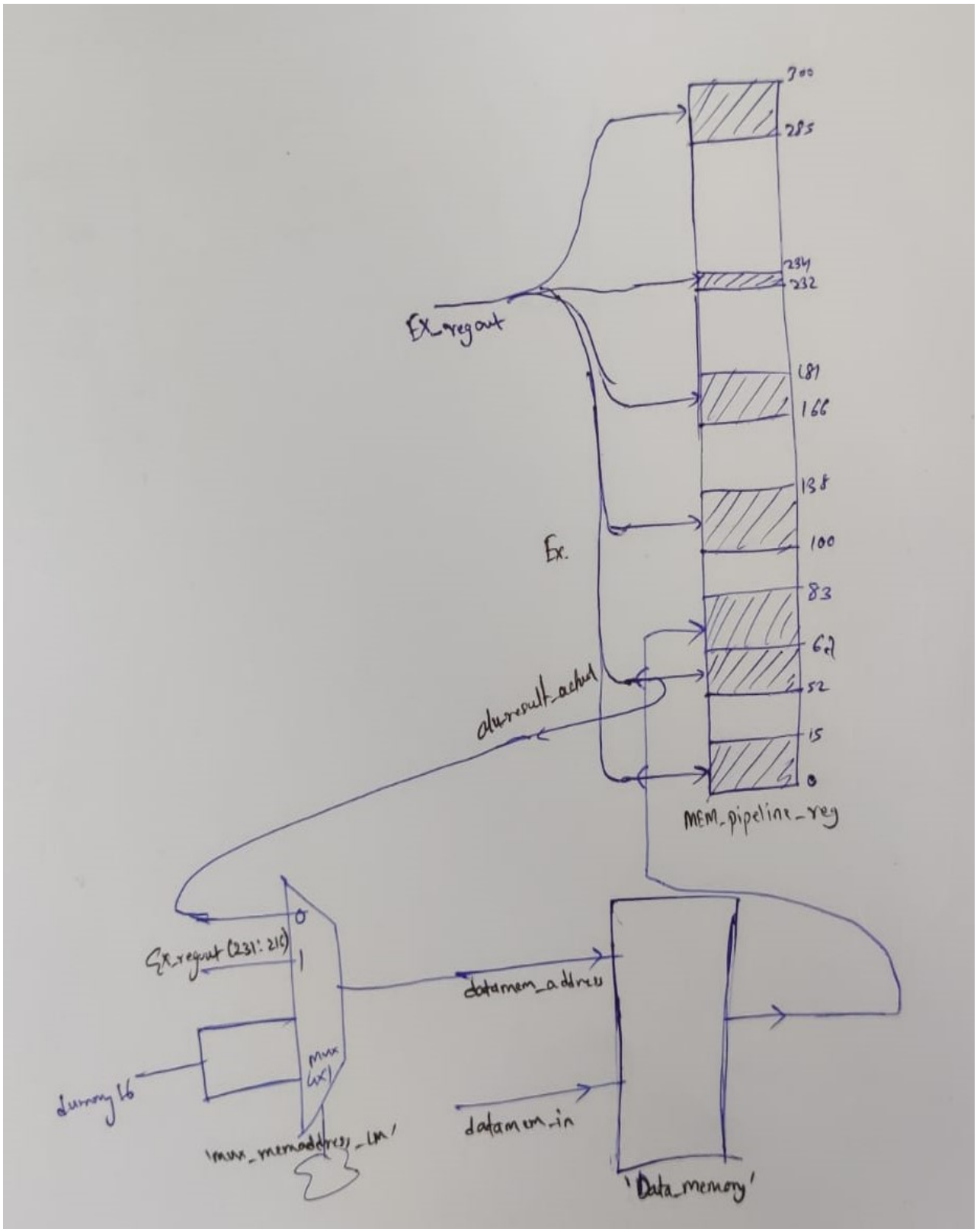
Instruction Decode:



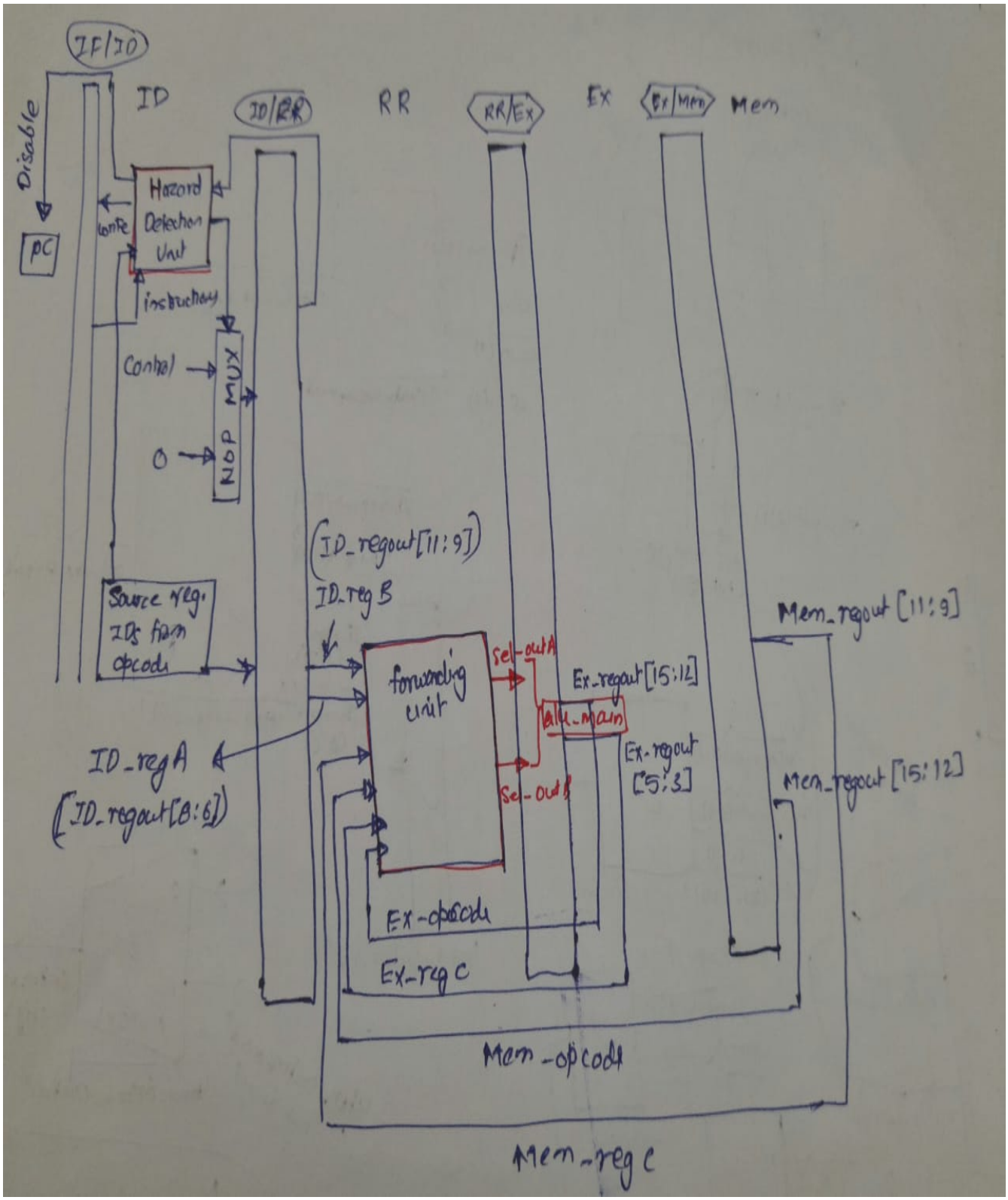
Execution stage:



Memory stage:



Hazard detection and forwarding unit:



Instruction sets:

Data Memory:	Instruction Memory:
X"0002" , X"0005" , X"0009" , X"0004" , X"0017" , X"000C" , X"0007" , X"0008" , X"0009" , X"000A" , X"000B"	"1100001011100100", "0001000010011000", "0001011001110000", "1000011101000010", "0001011010100000", "0010010001100000", "0000101000000111", "1100110010000010", "1001001000000010", "0001010011000000", "0100100110000000", "0100101110000001", "0001001110110000", "1101000010101010", "0000000000000000", "0000000000000000"

Results of instruction set: