

IITB- RISC-22 Processor

6-Stage Pipelined Processor



EE 739 - Processor Design

Project Report

| | |
|---|--|
| Submitted to: Prof. Virendra Singh Electrical Engineering | Submitted by: Sanjay Gupta(213070073) Pranav Dattatraya Pawar(213079020) Rahul(213079030) |
|---|--|

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:

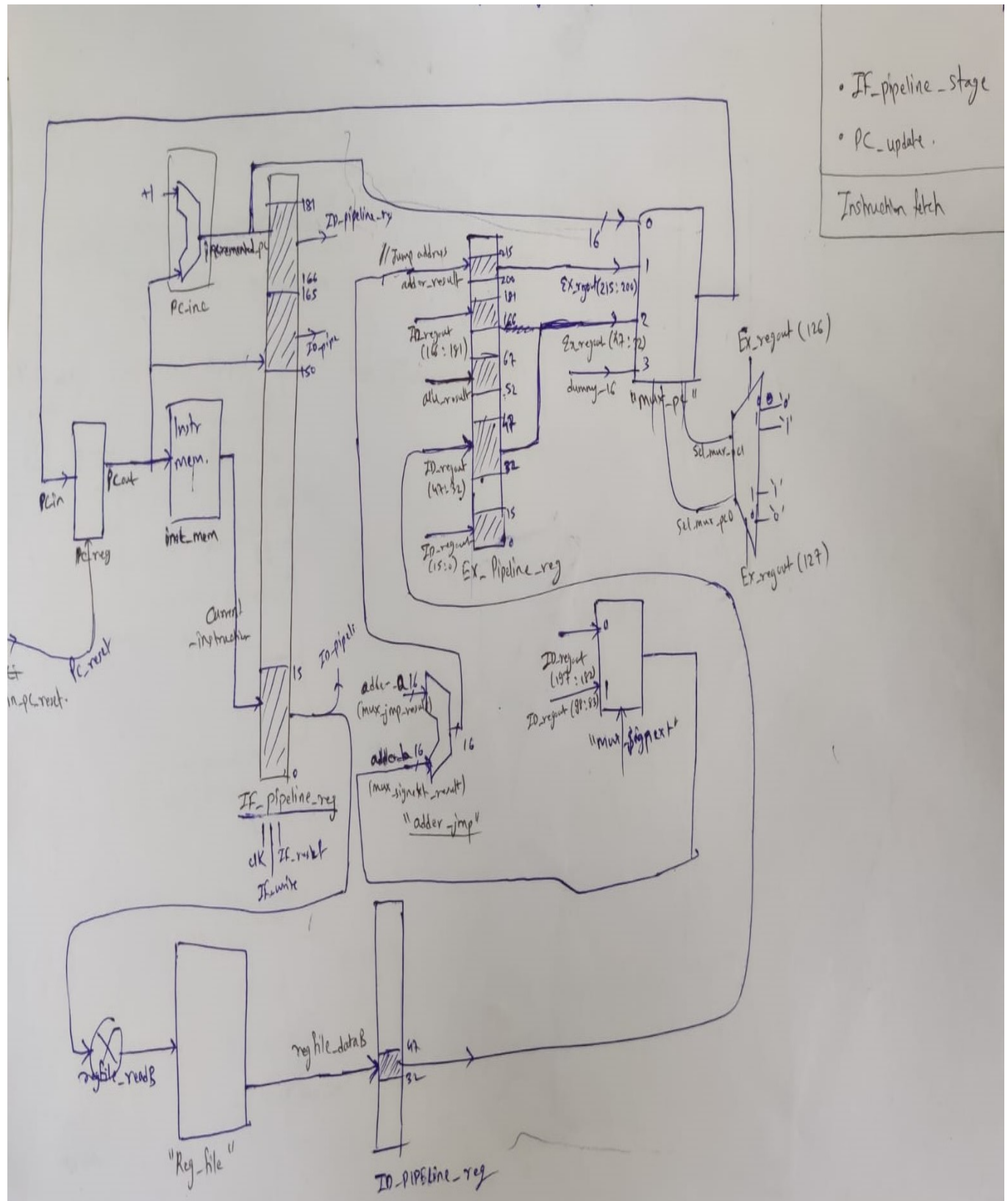
- Pipelined architecture improves the maximum operating frequency and throughput of processors by instruction level pipelining.
- In pipelining, divide datapath into nearly equal tasks, to be performed serially and requiring non-overlapping resources.
- Insert registers at task boundaries in the datapath. 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".

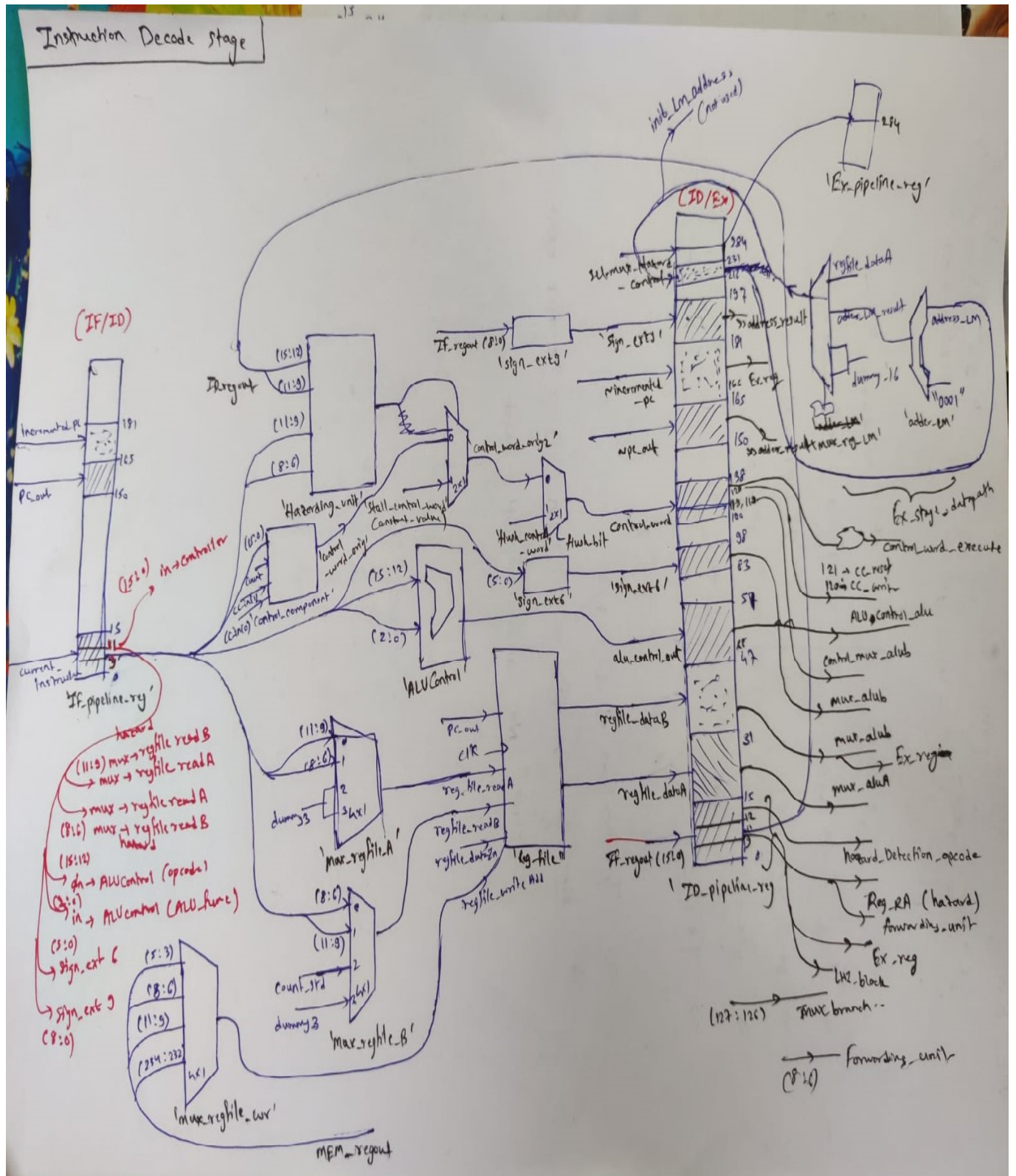
Instruction fetch stage:



The diagram illustrates a processor architecture with the following components and connections:

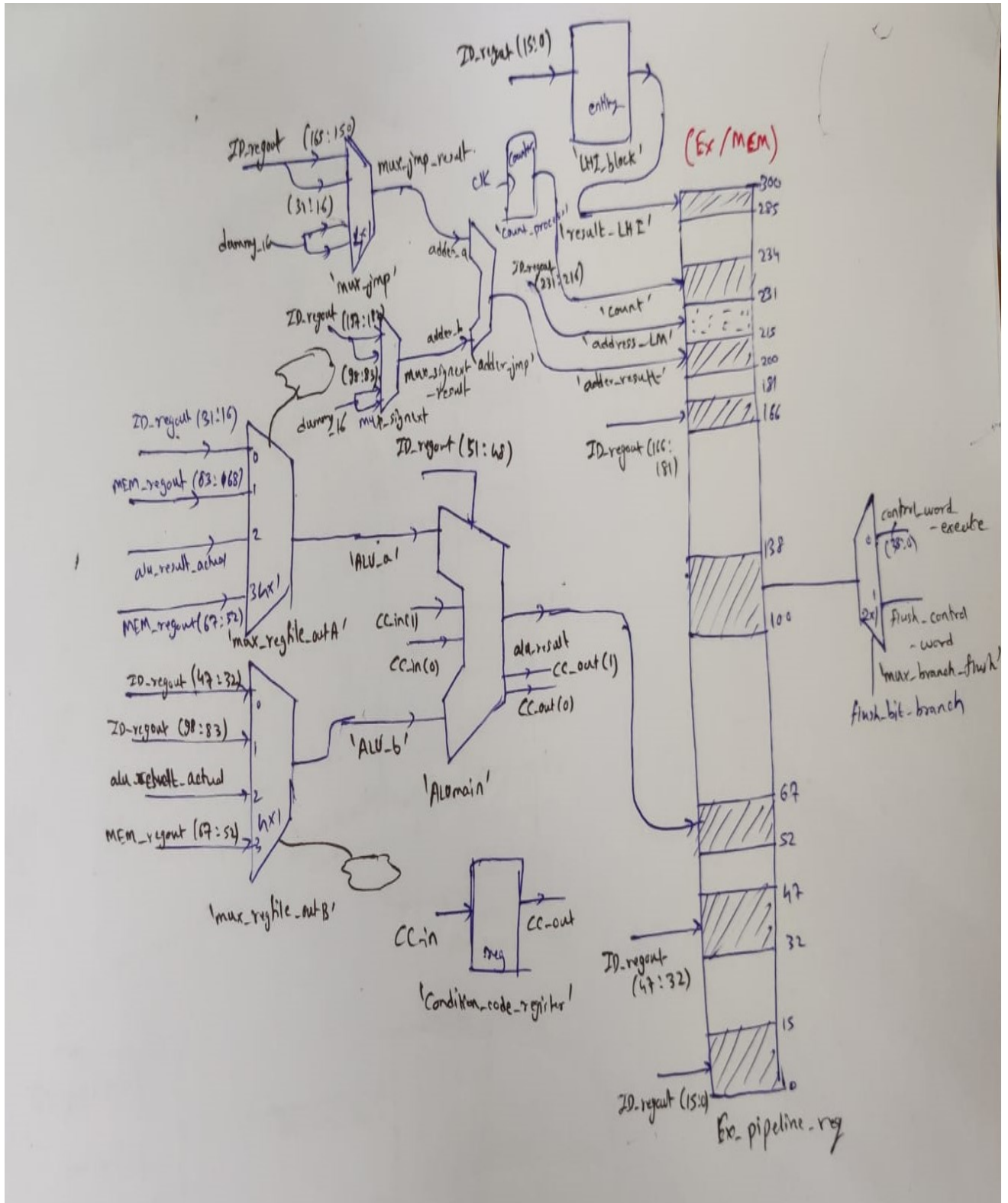
- IP/ED (Instruction Port/Execution Data):** The leftmost vertical block, connected to the main data bus.
- ID/RF (Instruction Decode/Register File):** A central vertical block containing 16 registers (0-15). It receives instructions from the IP/ED and provides data to the ALU and other units.
- RF/Ex (Register File/Execution):** The rightmost vertical block, which receives data from the ALU and other units and provides it back to the ID/RF.
- Control and Signaling:**
 - sign - kls:** A signal line connecting the ID/RF to the RF/Ex.
 - sign - pc:** A signal line connecting the ID/RF to the ALU.
 - sign - out 6:** A signal line connecting the ID/RF to the ALU.
 - sign - out:** A signal line connecting the ID/RF to the RF/Ex.
 - Control - out:** A signal line connecting the ID/RF to the ALU.
 - data control out:** A signal line connecting the ID/RF to the ALU.
 - data control in:** A signal line connecting the RF/Ex to the ALU.
 - data control out:** A signal line connecting the RF/Ex to the ALU.
 - data control in:** A signal line connecting the RF/Ex to the ALU.
- ALU (Arithmetic Logic Unit):** A central processing unit that receives data from the ID/RF and RF/Ex, and performs operations based on control signals. It outputs data to the RF/Ex.
- Registers:** A set of 16 registers (0-15) located between the ID/RF and RF/Ex. They store data and are connected to the ALU and other units.
- Other Components:**
 - PC (Program Counter):** A register that stores the current instruction address.
 - PC+1:** A register that stores the next instruction address.
 - PC+2:** A register that stores the next instruction address.
 - PC+3:** A register that stores the next instruction address.
 - PC+4:** A register that stores the next instruction address.
 - PC+5:** A register that stores the next instruction address.
 - PC+6:** A register that stores the next instruction address.
 - PC+7:** A register that stores the next instruction address.
 - PC+8:** A register that stores the next instruction address.
 - PC+9:** A register that stores the next instruction address.
 - PC+10:** A register that stores the next instruction address.
 - PC+11:** A register that stores the next instruction address.
 - PC+12:** A register that stores the next instruction address.
 - PC+13:** A register that stores the next instruction address.
 - PC+14:** A register that stores the next instruction address.
 - PC+15:** A register that stores the next instruction address.

Instruction Decode:

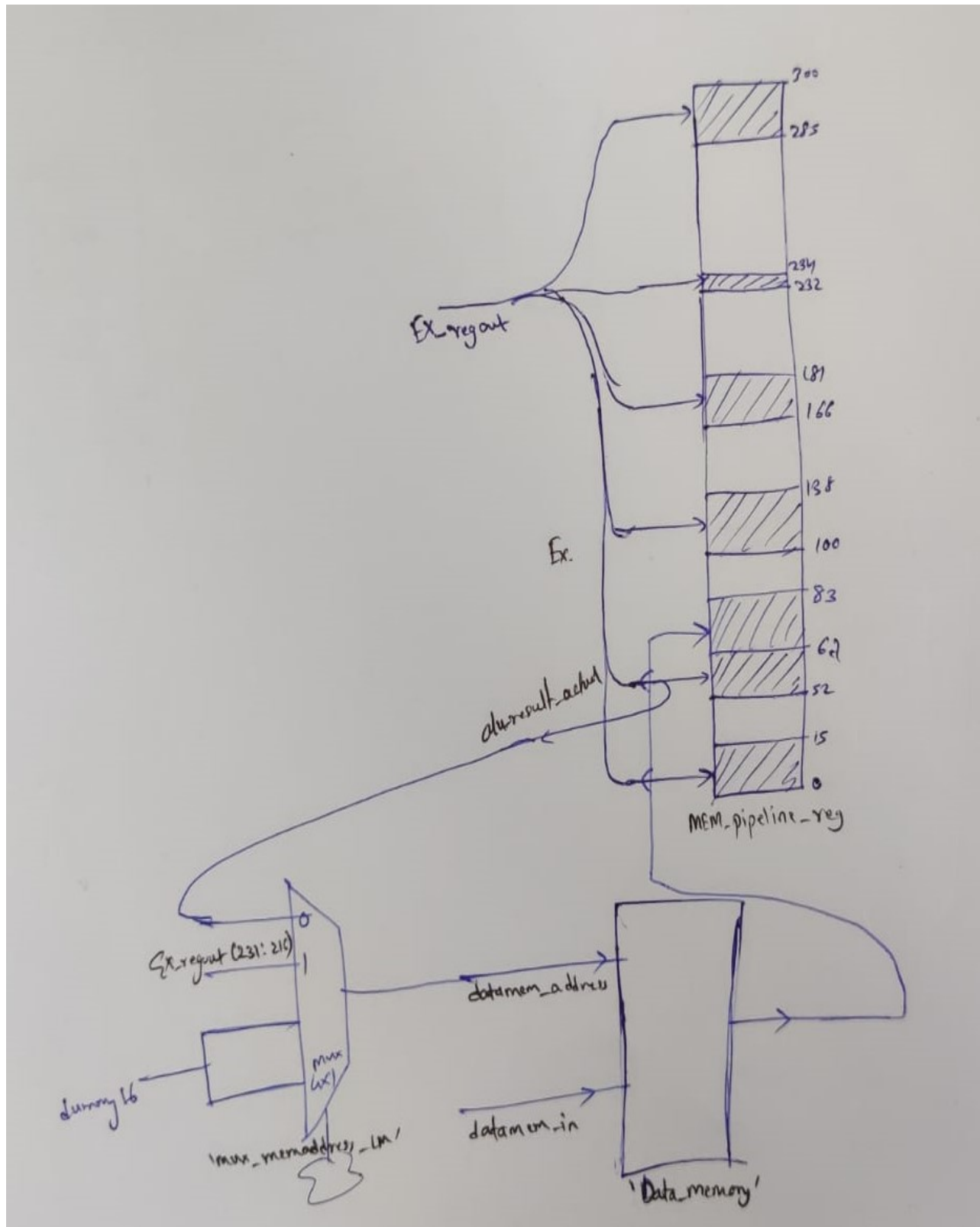


The diagram illustrates a 5-stage processor pipeline with the following components and connections:

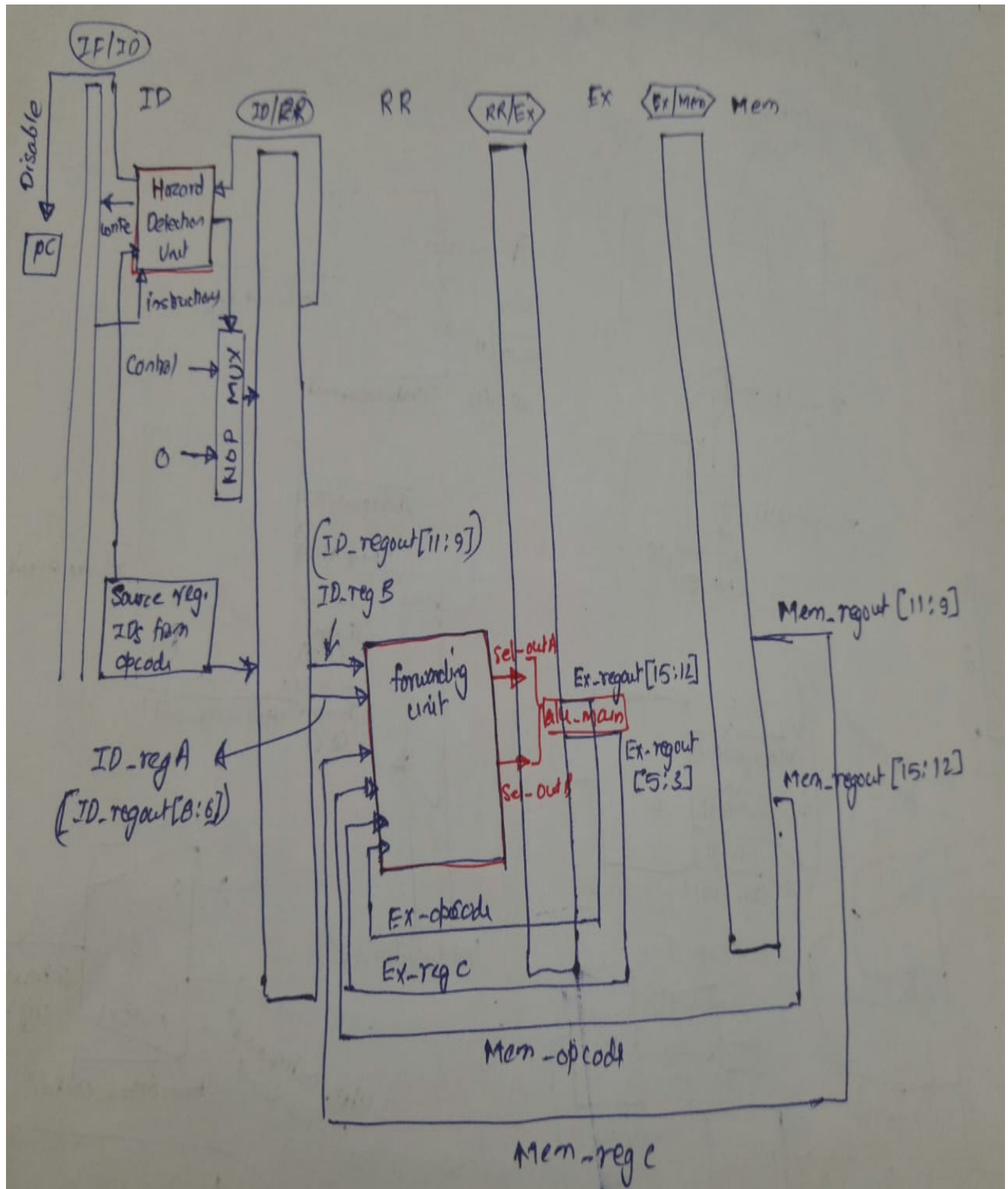
- Registers:**
 - EX/MEM:** A vertical stack of 32 registers (0 to 31). The top 16 registers (0-15) are shaded and labeled 'Ex/MEM'. The bottom 16 registers (16-31) are unshaded. The top register (0) is labeled 'control-word - execute'. The second register (1) is labeled 'flush-control - word'. The third register (2) is labeled 'max-branch-flush'. The bottom register (31) is labeled 'flush-bit-branch'.
 - ID-regout:** A vertical stack of 32 registers (0 to 31). The top 16 registers (0-15) are shaded. The bottom register (31) is labeled 'ID-regout (15:0)'.
 - MEM-regout:** A vertical stack of 32 registers (0 to 31). The top 16 registers (0-15) are shaded. The bottom register (31) is labeled 'MEM-regout (63:68)'.
 - alu-regout:** A vertical stack of 32 registers (0 to 31). The top 16 registers (0-15) are shaded. The bottom register (31) is labeled 'alu-regout (47:52)'.
 - alu-regout:** A vertical stack of 32 registers (0 to 31). The top 16 registers (0-15) are shaded. The bottom register (31) is labeled 'alu-regout (47:52)'.
 - alu-regout:** A vertical stack of 32 registers (0 to 31). The top 16 registers (0-15) are shaded. The bottom register (31) is labeled 'alu-regout (47:52)'.
- Multiplexers:**
 - Mux-jmp:** A 4-to-1 multiplexer with inputs from 'ID-regout (16:15)', 'ID-regout (31:16)', 'ID-regout (17:16)', and 'ID-regout (90:83)'. Its output is 'mux-jmp-result'.
 - Mux-signat:** A 4-to-1 multiplexer with inputs from 'ID-regout (31:16)', 'ID-regout (90:83)', 'ID-regout (51:40)', and 'ID-regout (16:15)'. Its output is 'mux-signat'.
 - Mux-regfile-outA:** A 4-to-1 multiplexer with inputs from 'MEM-regout (63:68)', 'alu-regout (47:52)', 'ID-regout (47:32)', and 'ID-regout (90:83)'. Its output is 'max-regfile-outA'.
 - Mux-regfile-outB:** A 4-to-1 multiplexer with inputs from 'MEM-regout (63:68)', 'alu-regout (47:52)', 'ID-regout (47:32)', and 'ID-regout (90:83)'. Its output is 'max-regfile-outB'.
- ALUs:**
 - ALU-a:** A 32-bit ALU with inputs from 'mux-jmp-result', 'mux-signat', and 'ID-regout (51:40)'. Its output is 'alu-result'.
 - ALU-b:** A 32-bit ALU with inputs from 'mux-jmp-result', 'mux-signat', and 'ID-regout (51:40)'. Its output is 'alu-result'.
- Control Logic:**
 - Condition-code-register:** A 32-bit register with inputs from 'alu-result' and 'alu-result-actual'. Its output is 'CC-out'.
 - Control-word:** A 32-bit register with inputs from 'alu-result' and 'alu-result-actual'. Its output is 'control-word - execute'.
 - Flush-control:** A 32-bit register with inputs from 'alu-result' and 'alu-result-actual'. Its output is 'flush-control - word'.
 - Max-branch-flush:** A 32-bit register with inputs from 'alu-result' and 'alu-result-actual'. Its output is 'max-branch-flush'.
 - Flush-bit-branch:** A 32-bit register with inputs from 'alu-result' and 'alu-result-actual'. Its output is 'flush-bit-branch'.



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:

