**UNIVERSITY OF TEXAS AT ARLINGTON**

**(UTA)**

**PROJECT 1**

**EE 6313**

*32bit RISC*

*Microprocessor Design*

*With Pipeline*

*Depth* **iv**!

**ABSTRACT**

*The Project involves the design of a 32-bit Microprocessor with RISC architecture based on the pipeline depth being four. The four-stage pipeline reciprocates “****Instruction fetch****”, “****Read register / Address generation****”, “****Fetch operand / Execute****” and “****Write back****”. The microprocessor interfaces the pseudo memory and the registers by means of* ***Memory Interface Unit (MIU)*** *and* ***Register Interface Unit (RIU)****. The design also covers the avoidance of certain structural Hazards and Data hazards by using the* ***Stall****,* ***Flush*** *and* ***Data Forwarding*** *control logic making the design fully functional.*

*Certain considerations that have been included in the project on our convenience are as follows.*

*The format used for data read and write is “****Little Endian****”.*

*For the increment and decrement of the stack pointer we have used the standard method of how a stack works, that is*

*“****pre-decrement****” which is SP-4 in case of PUSH instruction.*

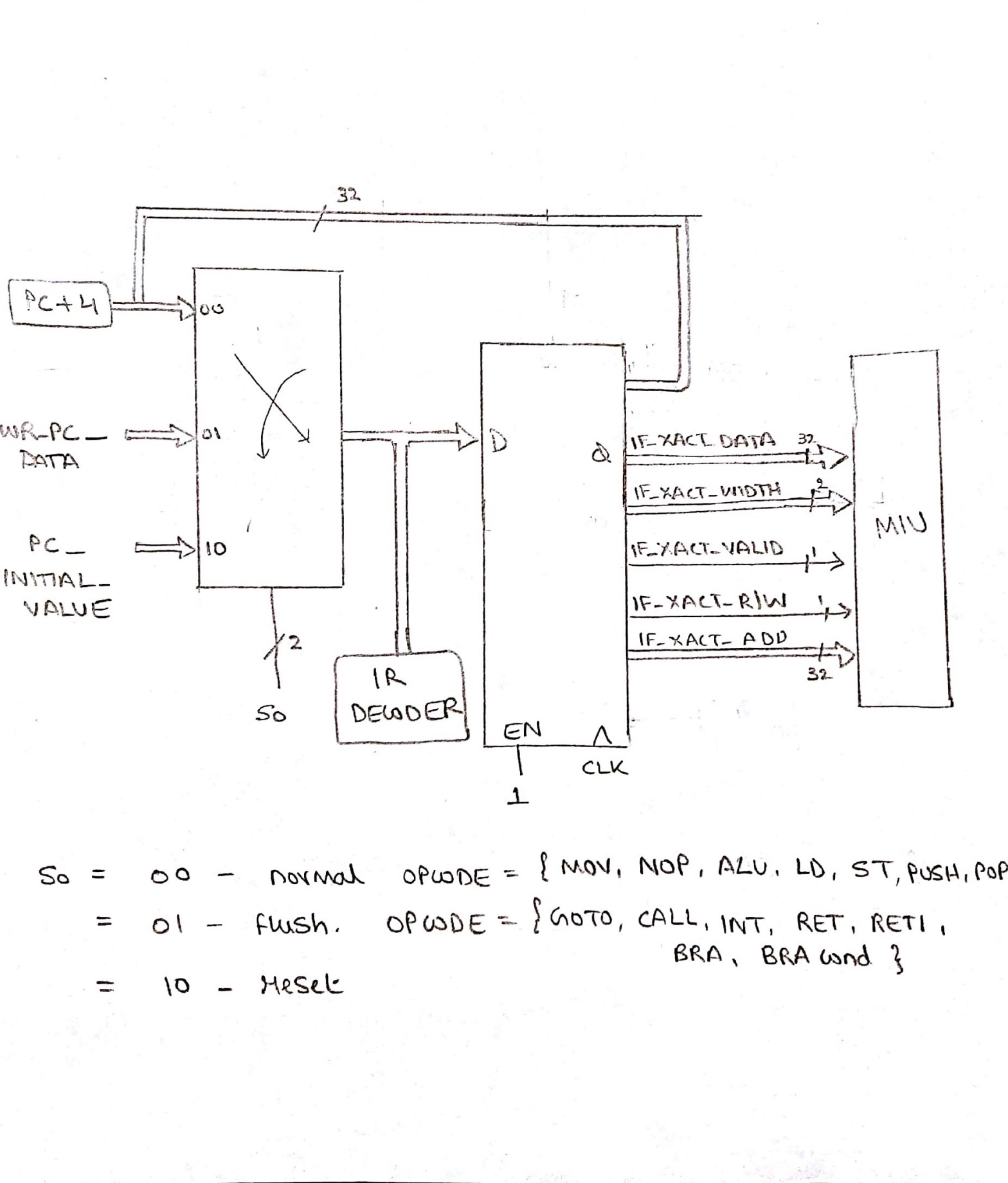
*and “****post-increment****” which is SP+4 in case of POP instruction.*

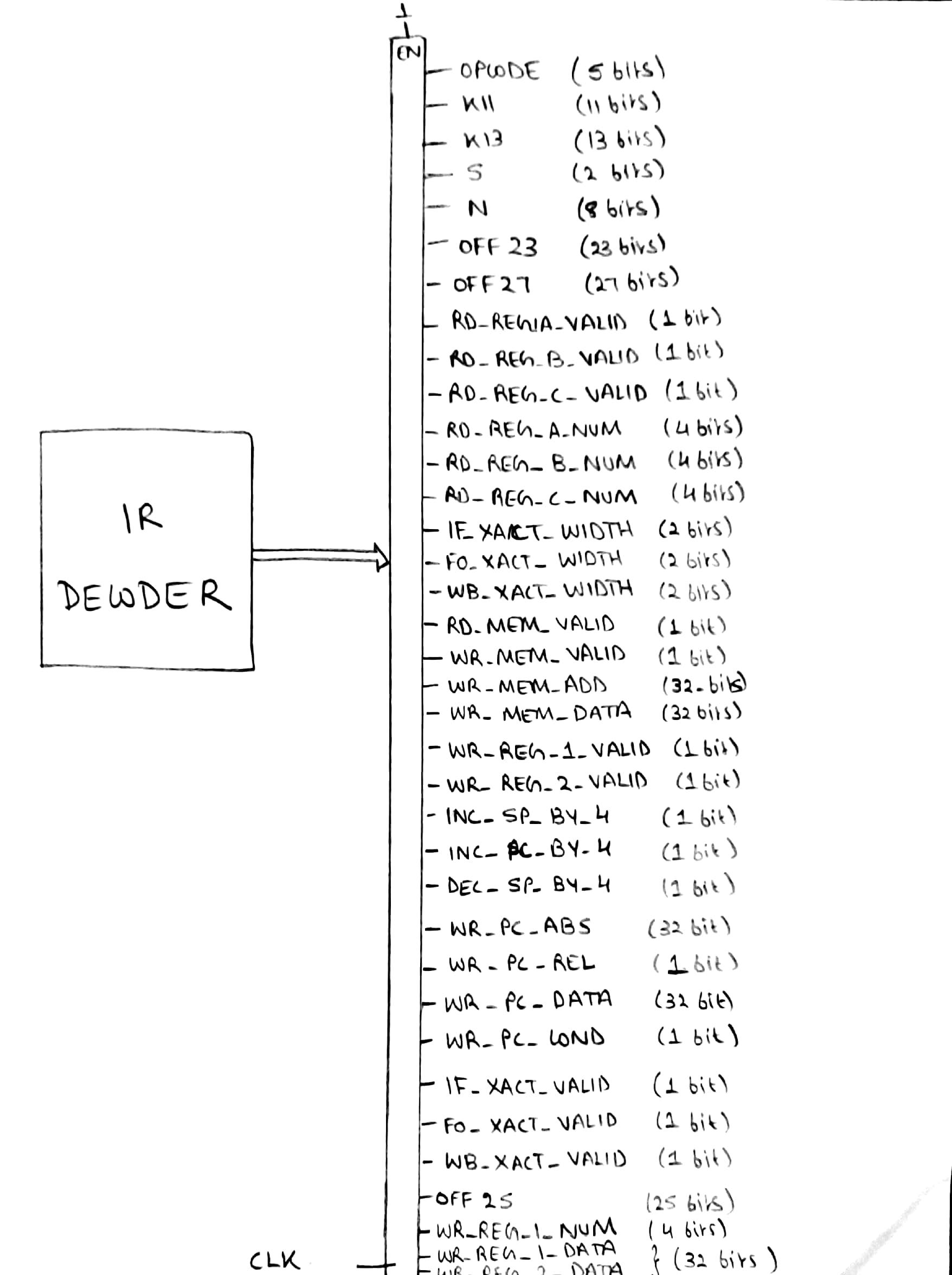
Table of Contents:

1. Instruction Fetch
   1. Theory of Operation
   2. Schematics
   3. Logical Equations
   4. Summary of Inputs / Outputs and Pass-through Data
2. Read Registers / Address Generation
   1. Theory of Operation
   2. Schematics and Logical Equations
   3. Summary of Inputs / Outputs and Pass-through Data
3. Fetch Operand / Execute
   1. Theory of Operation
   2. Schematics and Logical Equations
   3. Summary of Inputs / Outputs and Pass-through Data
4. Write Back
   1. Theory of Operation
   2. Schematics & Logical Equations
   3. Summary of Inputs / Outputs and Pass-through Data
5. Stall Control / Flush Logic
   1. Theory of Operation
   2. Schematics and Logical Equations
6. Data Forwarding
   1. Theory of Operation
   2. Schematics and Logical Equations
7. Register Logic
   1. Theory of Operation
   2. Schematics and Logical Equations
8. Memory Interface
   1. Theory of Operation
   2. Schematics and Logical Equations
9. **INSTRUCTION FETCH**

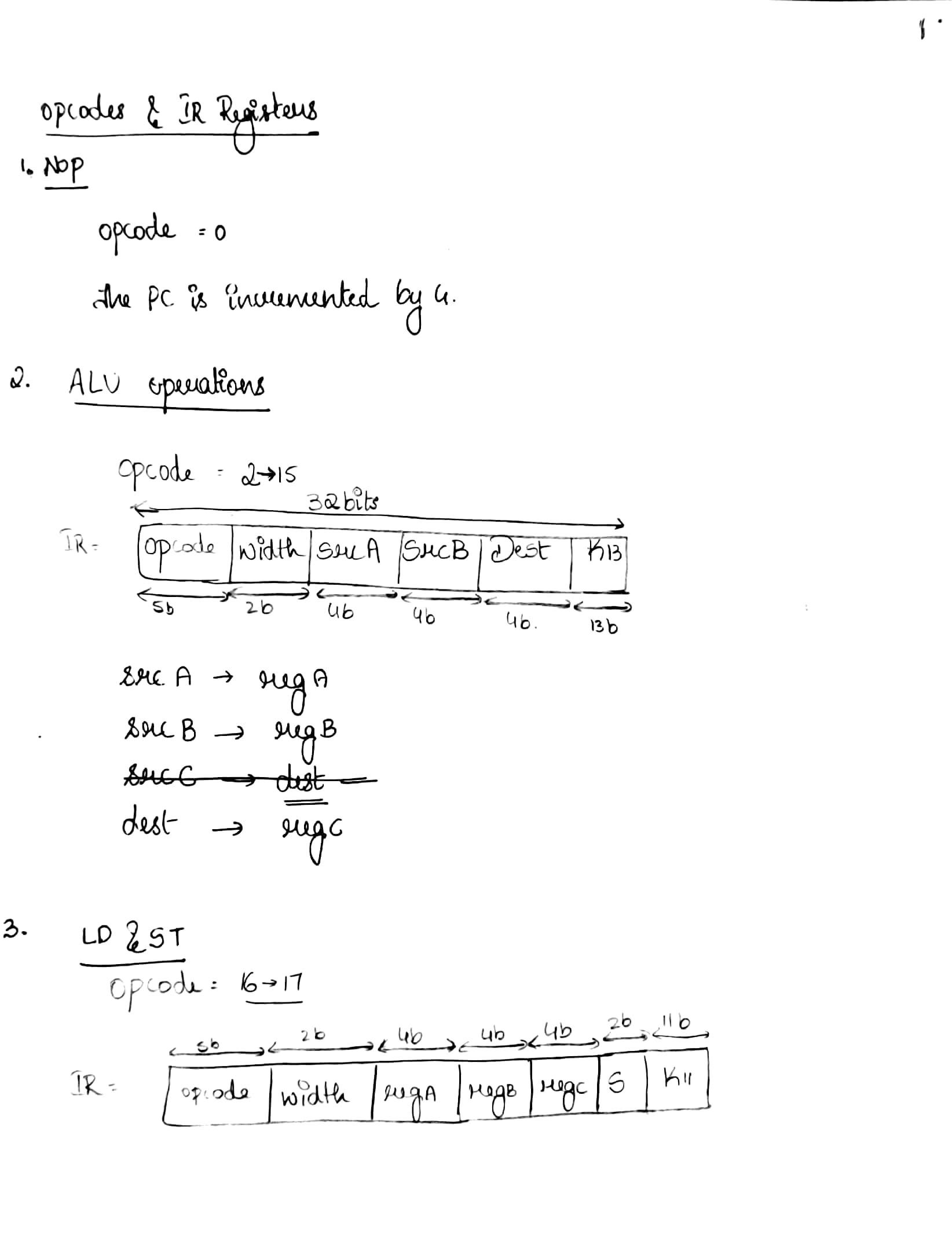
**1.1 Theory of operation:**

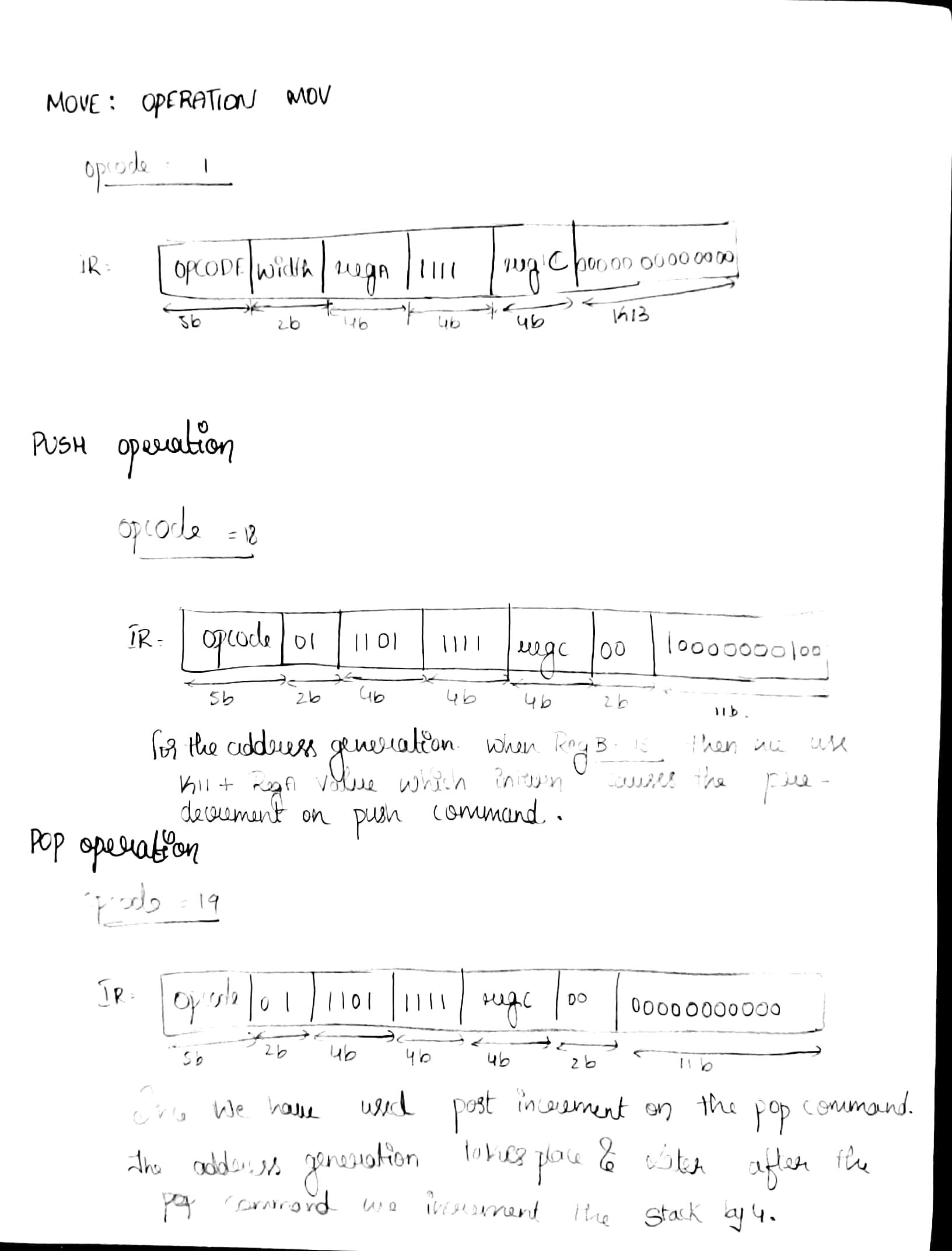
1. The first stage of pipeline is called the Instruction Fetch.
2. In this stage the instructions are fetched from the instruction memory or the IR register [31:0] which determines the instruction to be executed.
3. The instructions are decoded using the OPCODE which is different for each instruction.
4. The procedure involves PC and PC fetch which are takes PC\_INITIAL\_VALUE at reset and while executing an instruction it is incremented by 4 or taken from the writeback stage according to the next instruction.
5. The PC value at the end of the pipeline and at instruction fetch will have a difference of 12. If there is any discontinuous PC values, then the entire pipeline is flushed using the flush logic.
   1. **Schematics**

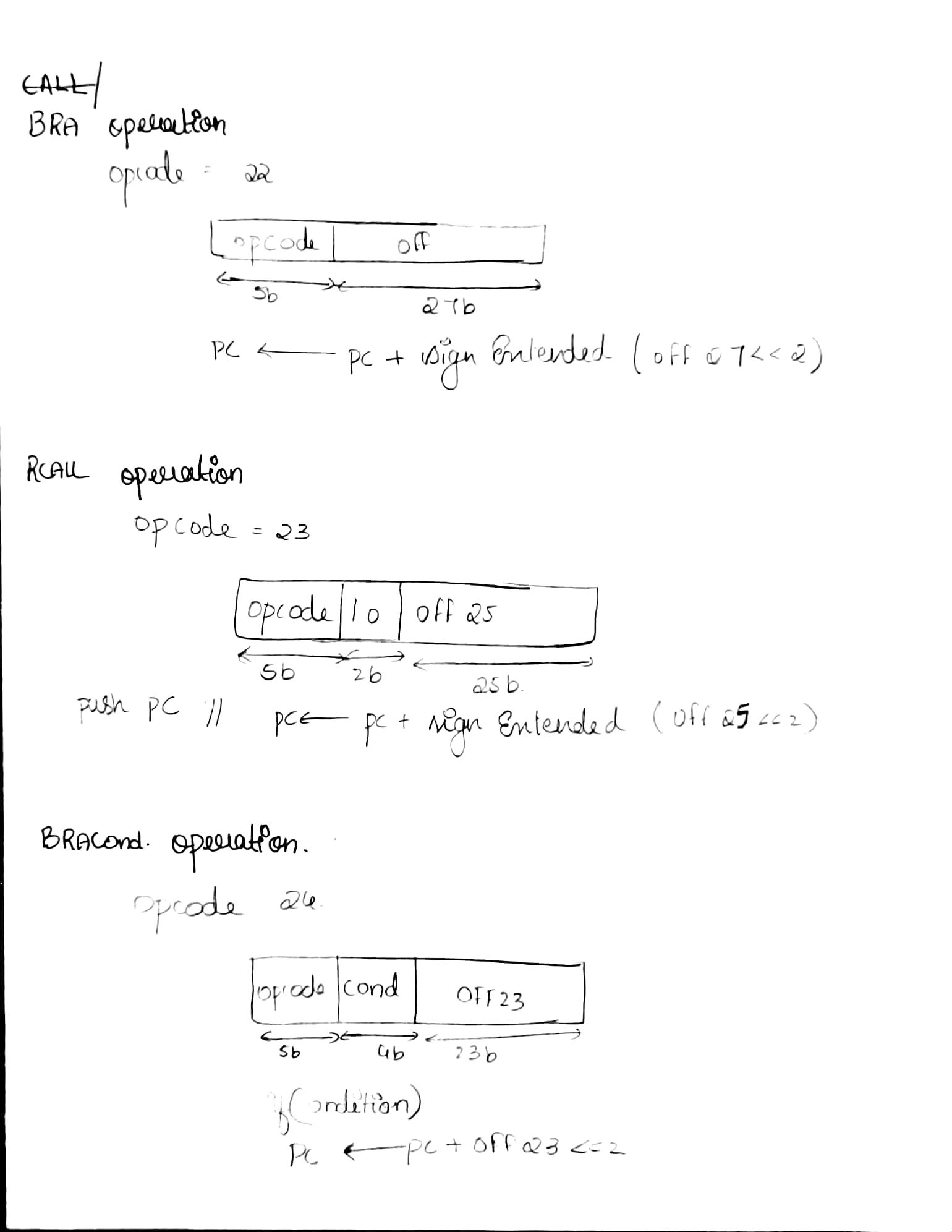


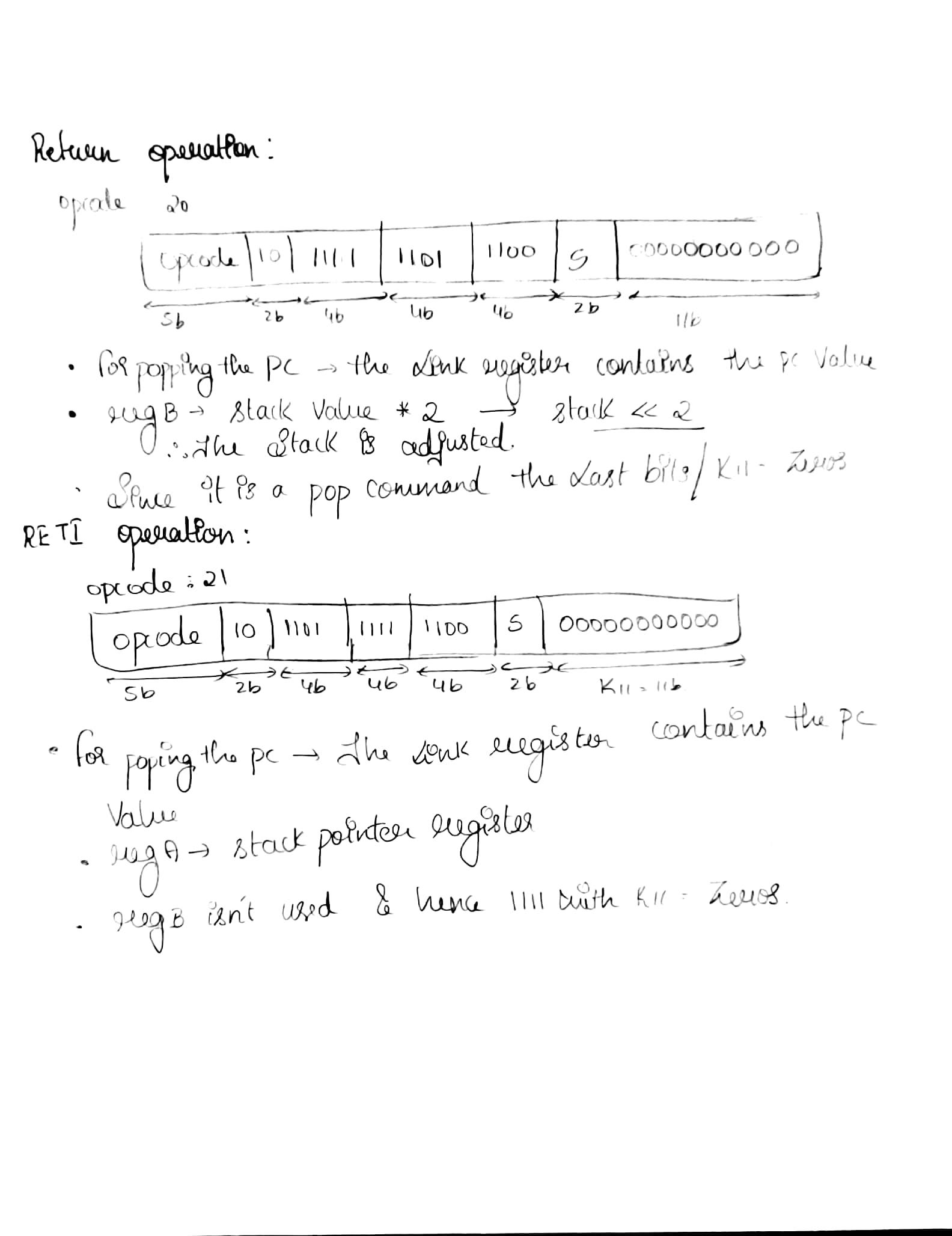


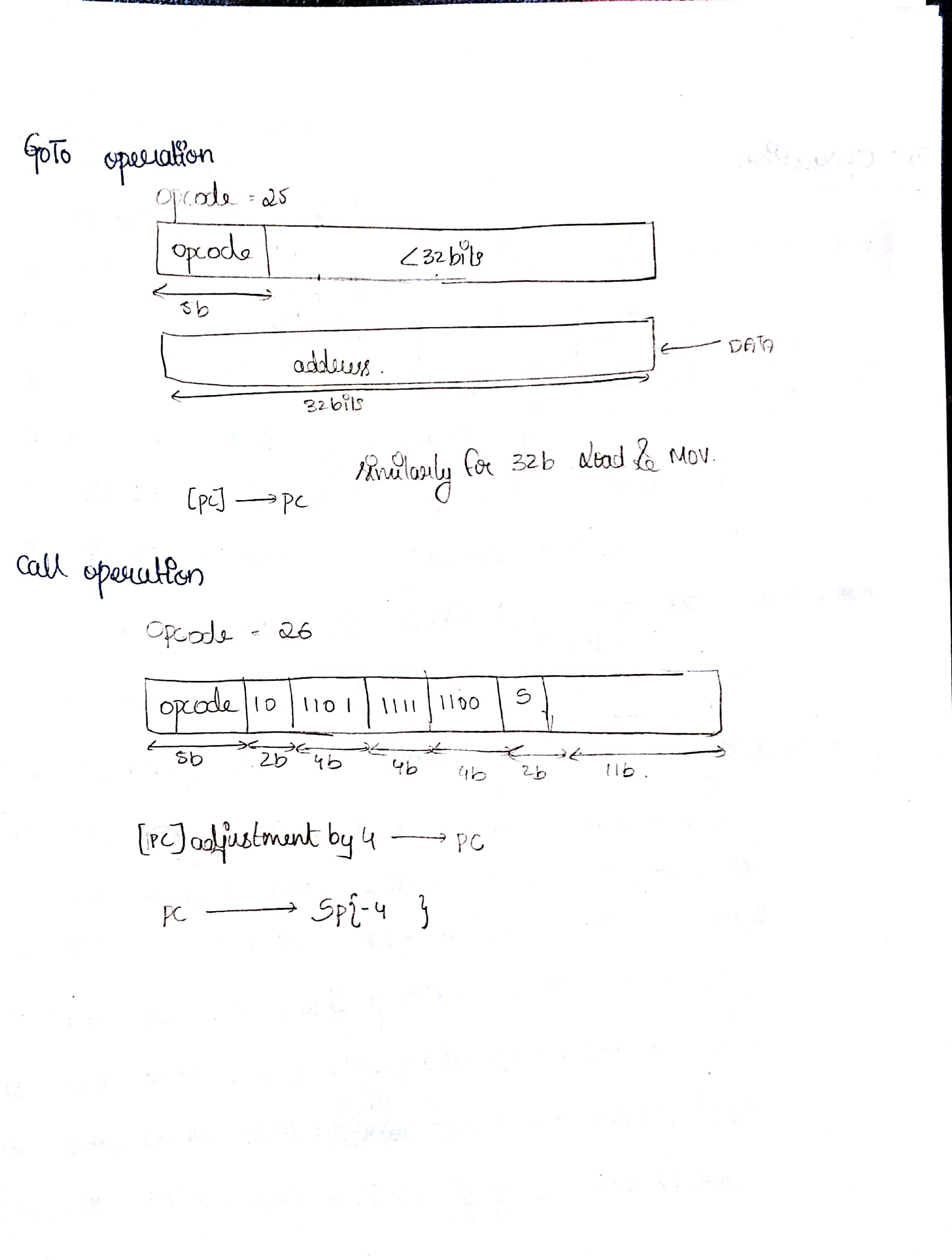
* 1. **Logical Equations**

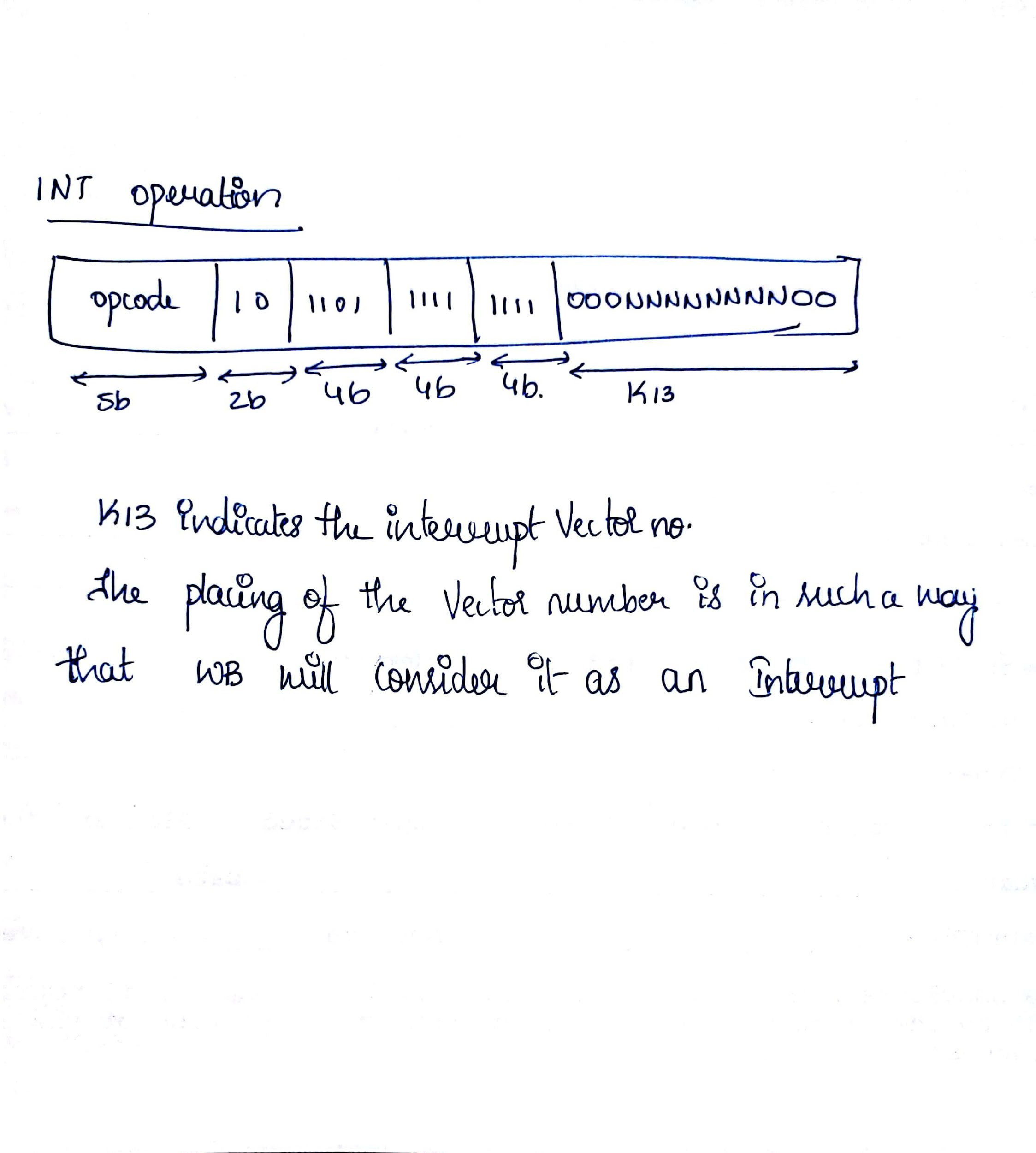


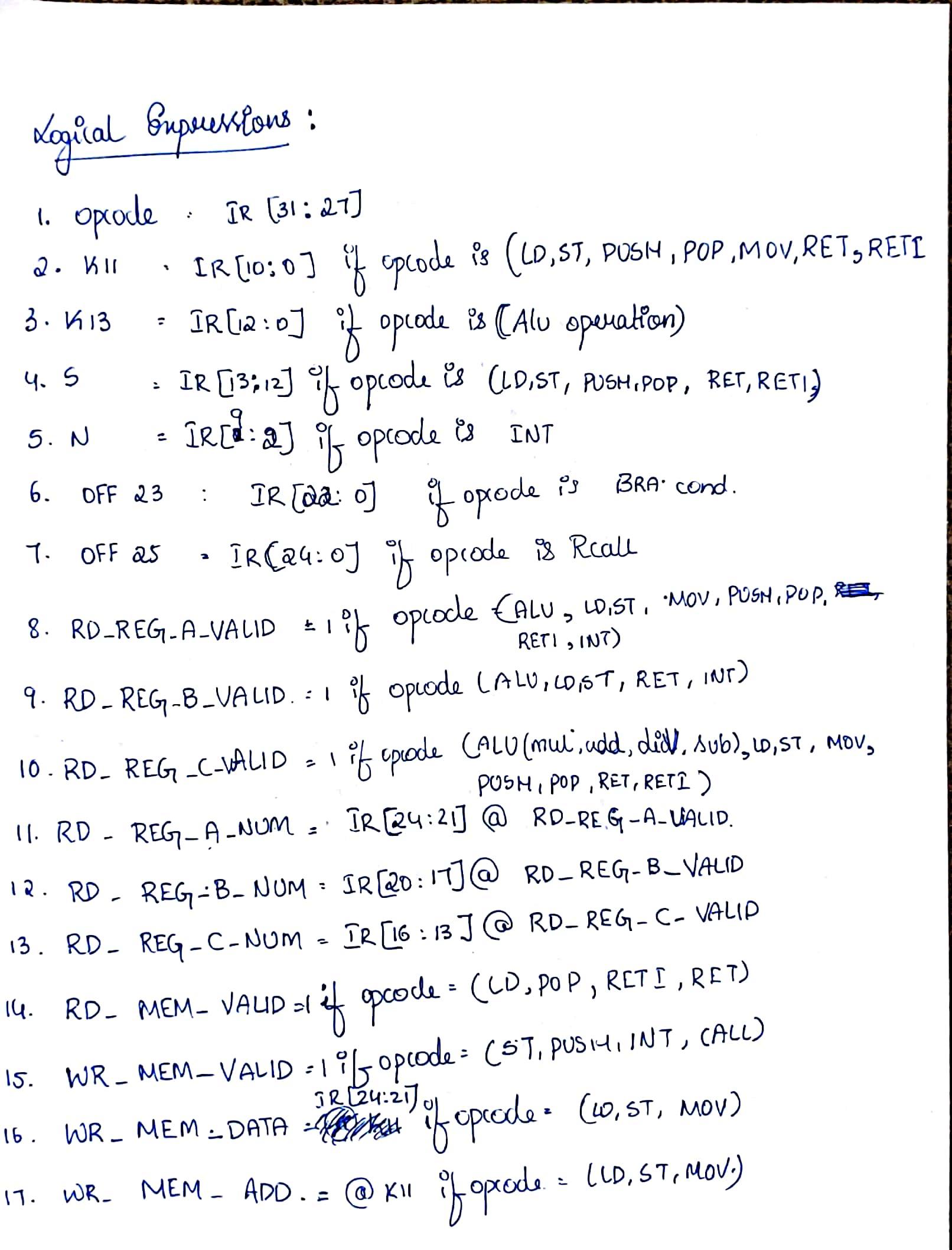


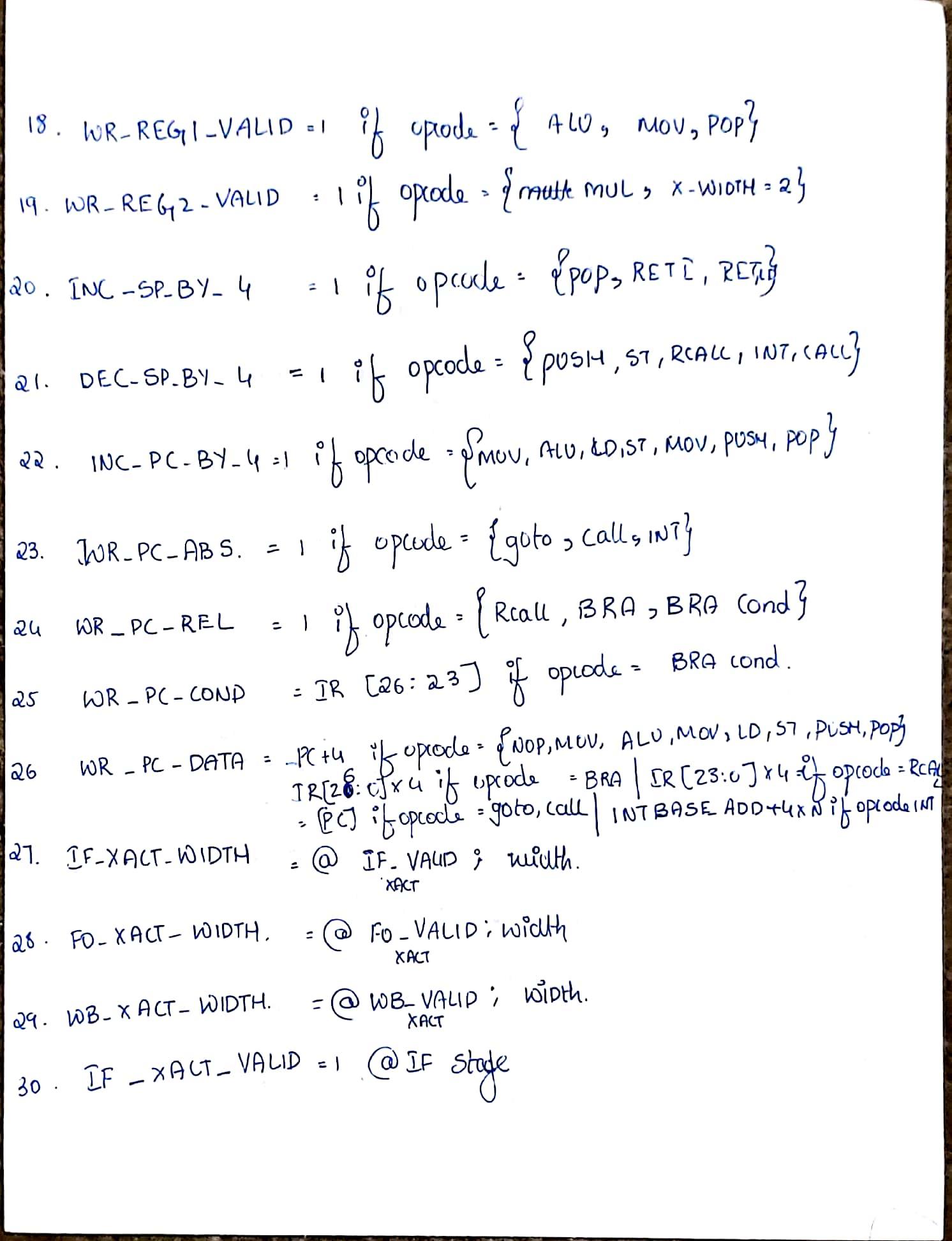


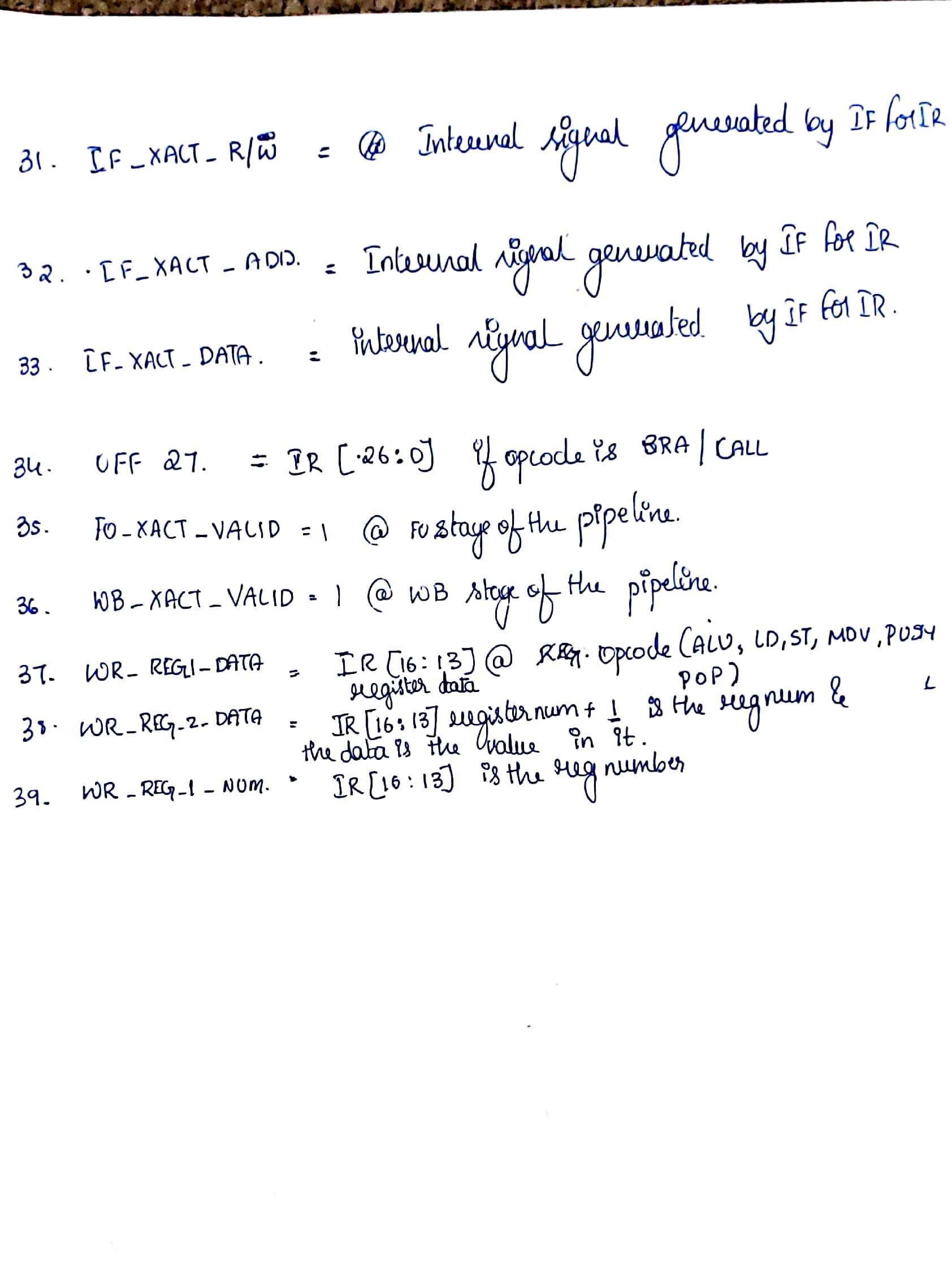










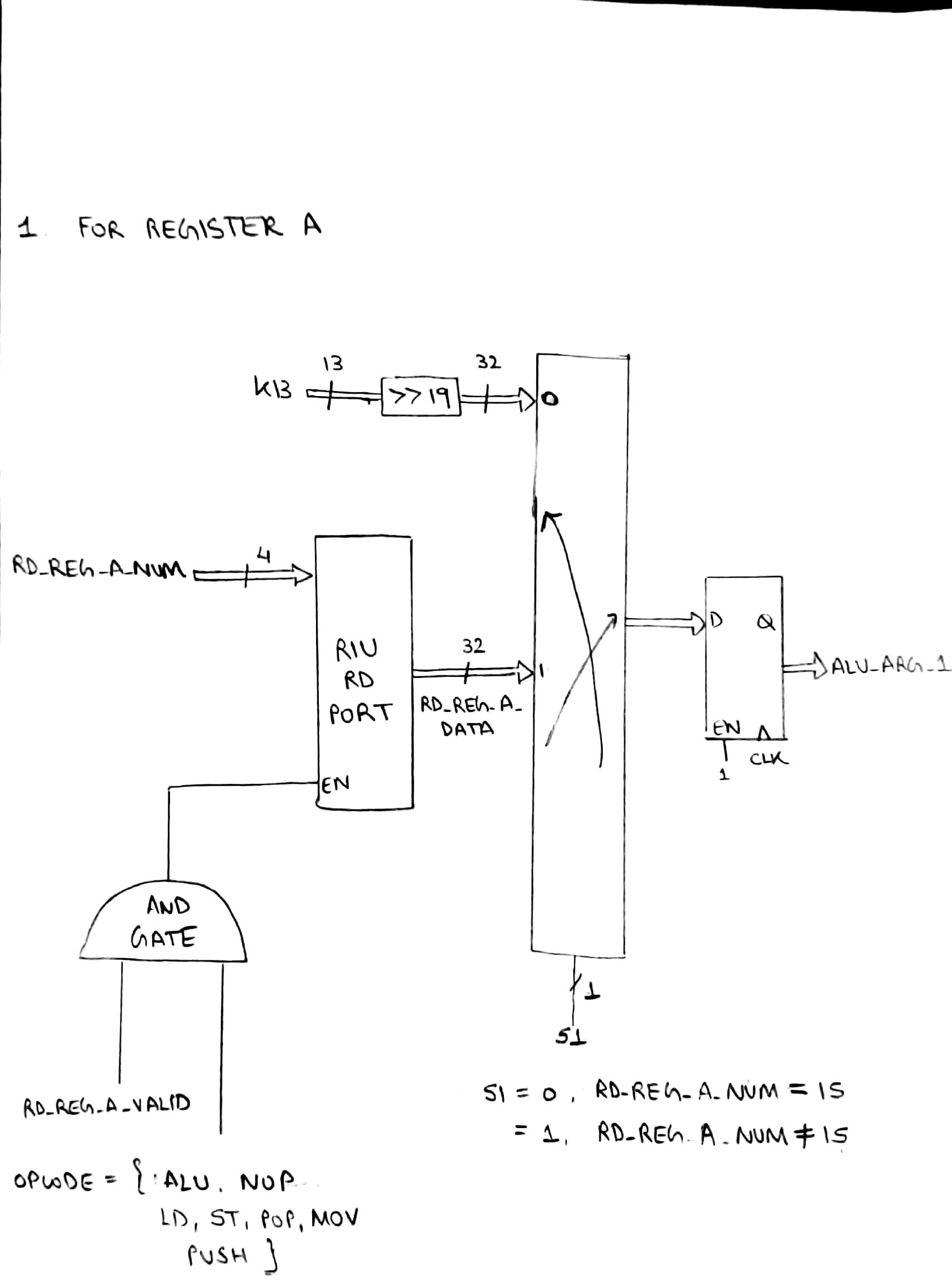


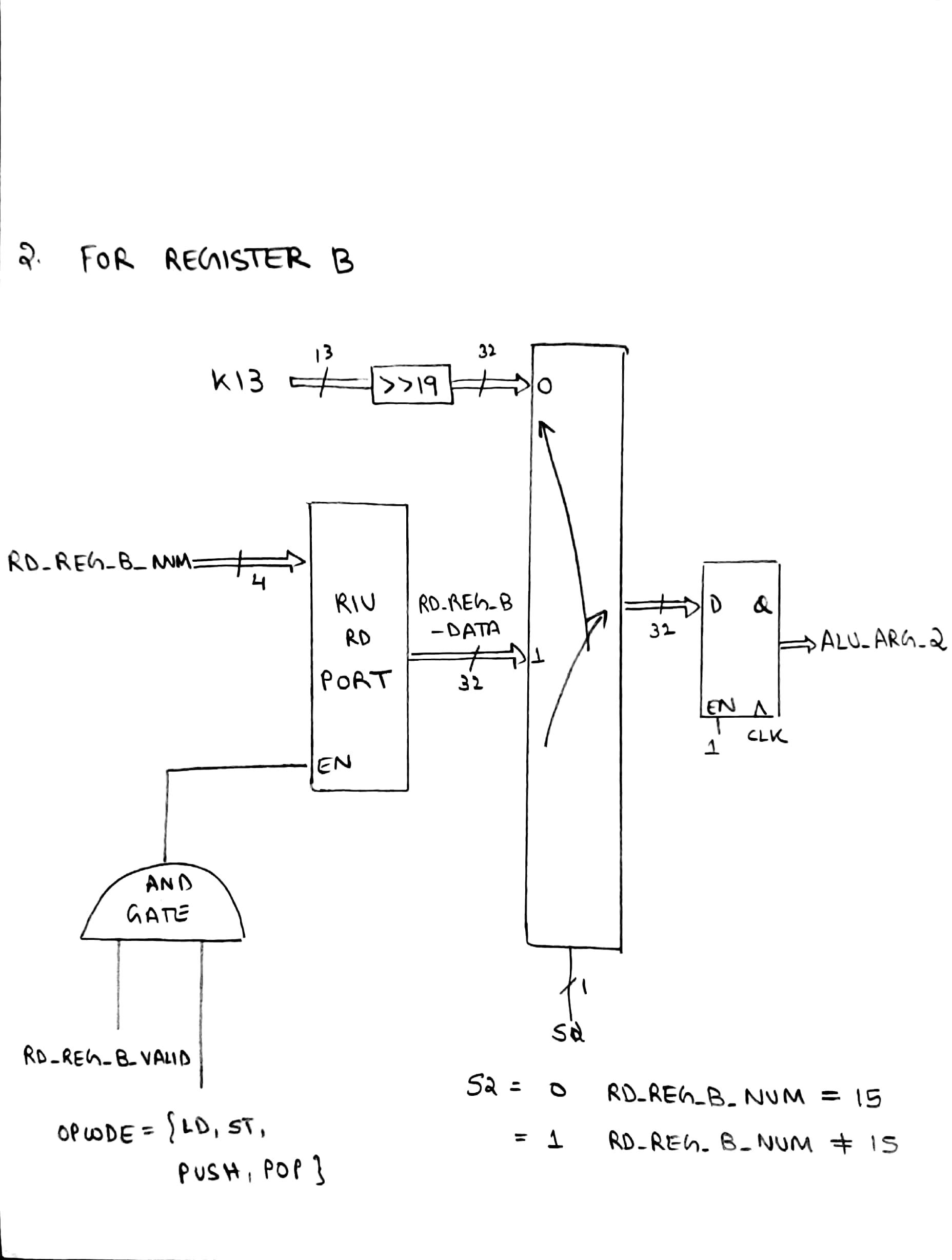
**1.4 Summary of Inputs / Outputs and Pass-through data:**

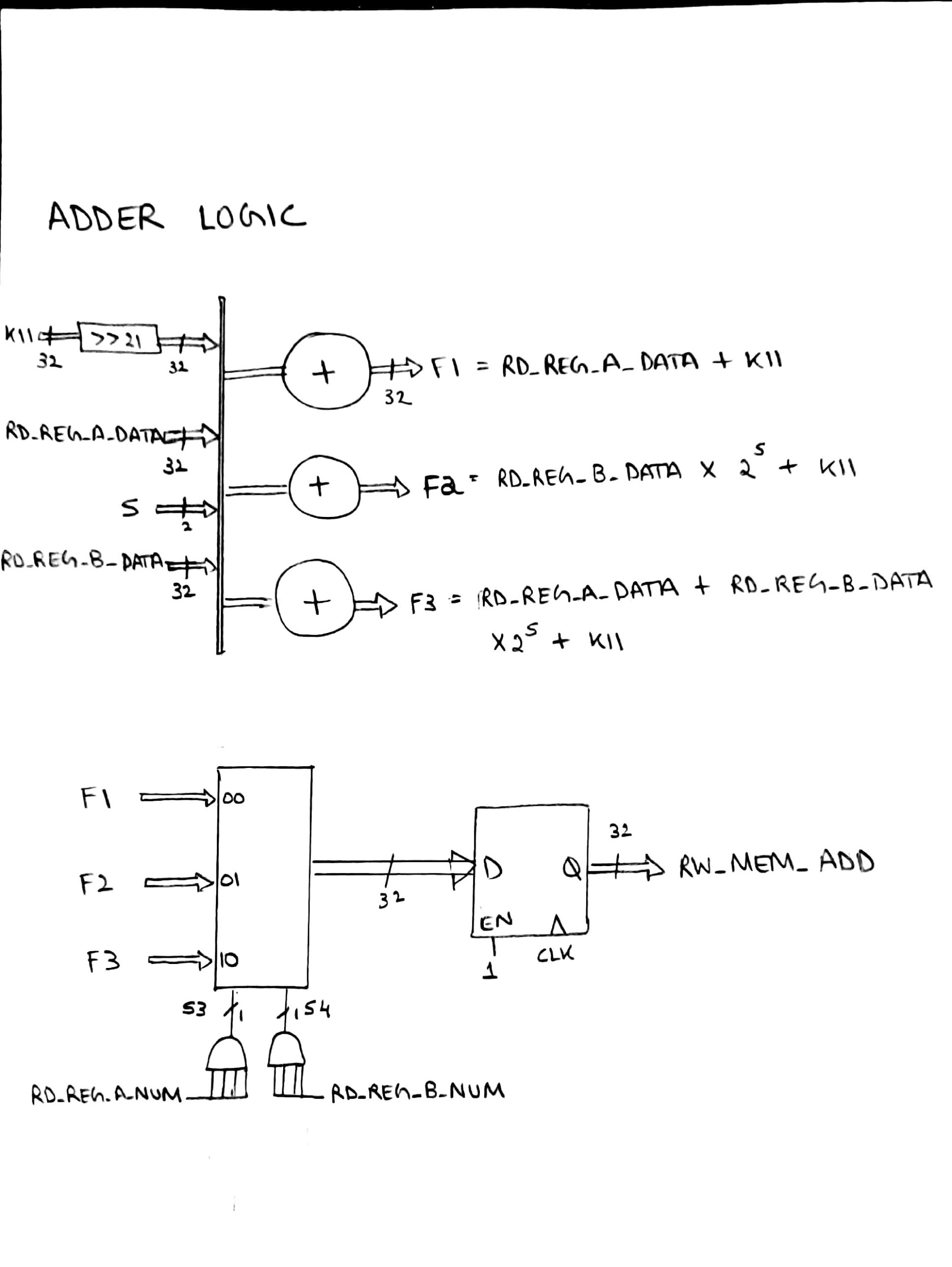
|  |  |  |  |
| --- | --- | --- | --- |
| Sl No. | Inputs | Outputs | Passthrough |
| 1. | WR\_PC\_VALUE | OPCODE | NO |
| 2. | PC\_INITIAL\_VALUE | K11 | PASS-THROUGH |
| 3. | PC [31:0] | K13 |  |
| 4. | S0 | S |  |
| 5. | IF\_XACT\_DONE | N |  |
| 6. |  | OFF23 |  |
| 7. |  | OFF27 |  |
| 8. |  | RD\_REG\_A\_VALID |  |
| 9. |  | RD\_REG\_B\_VALID |  |
| 10. |  | RD\_REG\_C\_VALID |  |
| 11. |  | RD\_REG\_A\_NUM |  |
| 12. |  | RD\_REG\_B\_NUM |  |
| 13. |  | RD\_REG\_C\_NUM |  |
| 14. |  | FO\_XACT\_WIDTH |  |
| 15. |  | WB\_XACT\_WIDTH |  |
| 16. |  | RD\_MEM\_VALID |  |
| 17. |  | WR\_MEM\_VALID |  |
| 18. |  | WR\_MEM\_DATA |  |
| 19. |  | WR\_MEM\_ADD |  |
| 20. |  | WR\_REG1\_VALID |  |
| 21. |  | WR\_REG2\_VALID |  |
| 22. |  | INC\_SP\_BY\_4 |  |
| 23. |  | DEC\_SP\_BY\_4 |  |
| 24. |  | INC\_PC\_BY\_4 |  |
| 25. |  | IF\_XACT\_DATA |  |
| 26. |  | IF\_XACT\_WIDTH |  |
| 27. |  | IF\_XACT\_VALID |  |
| 28. |  | IF\_XACT\_R/W |  |
| 29. |  | IF\_XACT\_ADD |  |
| 30. |  | WR\_PC\_ABSOLUTE |  |
| 31. |  | WR\_PC\_RELATIVE |  |
| 32. |  | WR\_PC\_COND |  |
| 33. |  | WR\_PC\_DATA |  |
| 34. |  | OFF 25 |  |
| 35. |  | FO\_XACT\_VALID |  |
| 36. |  | WB\_XACT\_VALID |  |
| 37. |  | WR\_REG\_1\_DATA |  |
| 38. |  | WR\_REG\_2\_DATA |  |
| 39. |  | WR\_REG\_1\_NUM |  |

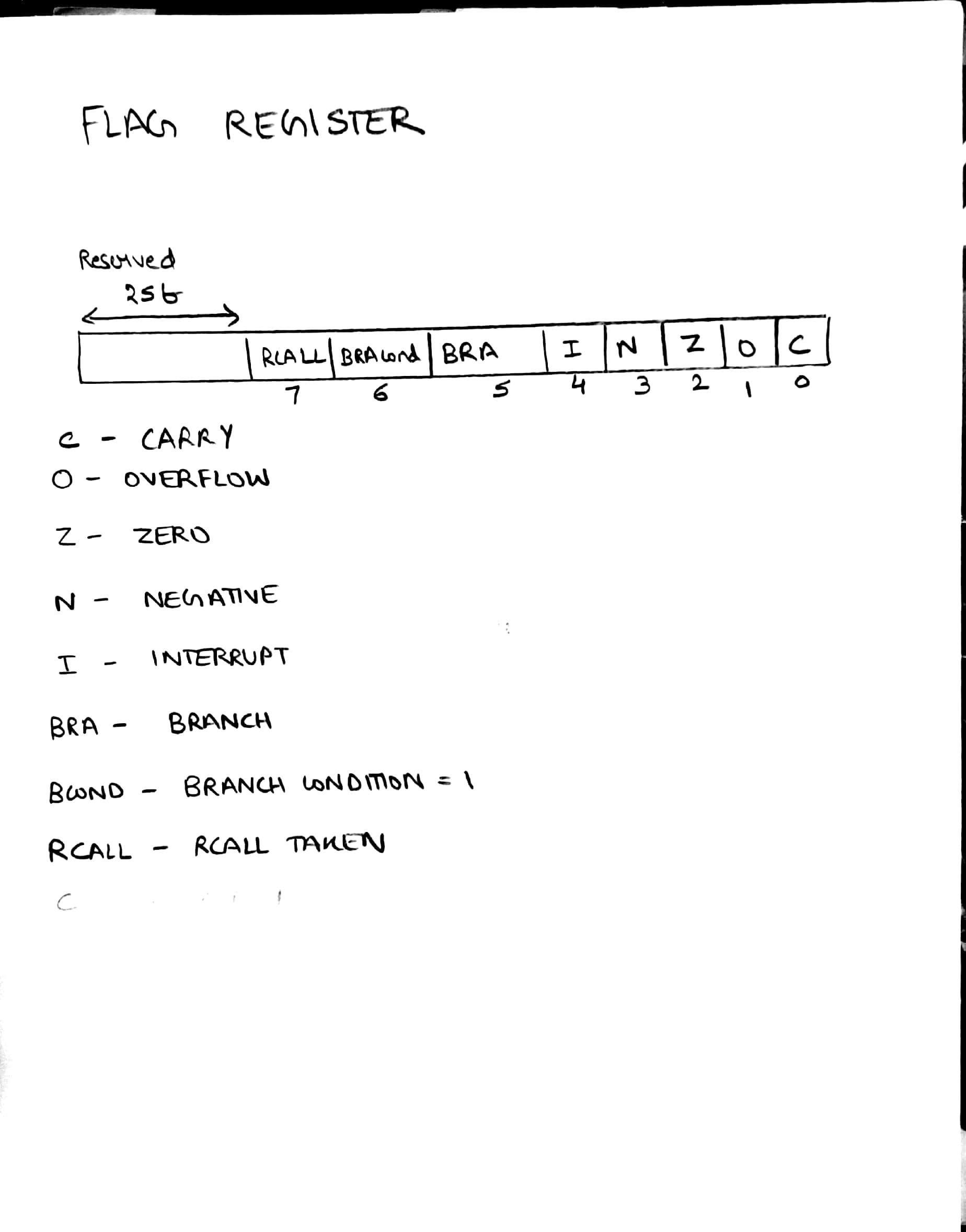
1. **READ REGISTER/ADDRESS GENERATION**
   1. **Theory of operation:**
2. The second stage of pipeline which involves the reading of registers and address generation.
3. The read register phase is mainly register values, validation signals and data which are outputs from the Instruction fetch are read as inputs to the Arithmetic Logical Unit (ALU).
4. The address generation is a pre-requisite to the Fetch Operand pipeline stage and this is done by computing the memory address.
5. The Opcode determines which registers to be used or which operation to be performed.

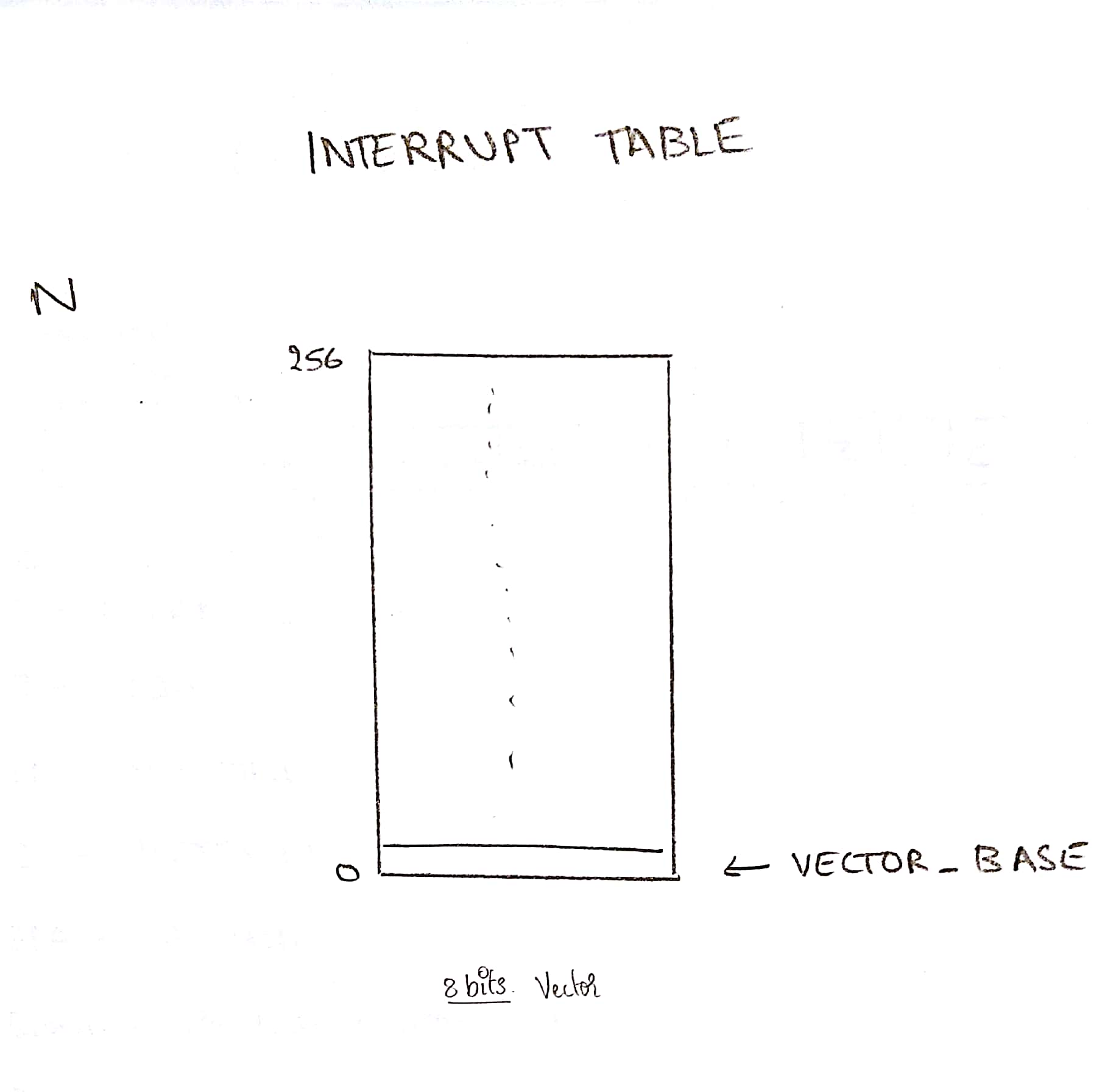
**2.2 Schematics and Logical Equations**











**2.3 Summary of Inputs / Outputs and Pass-through data:**

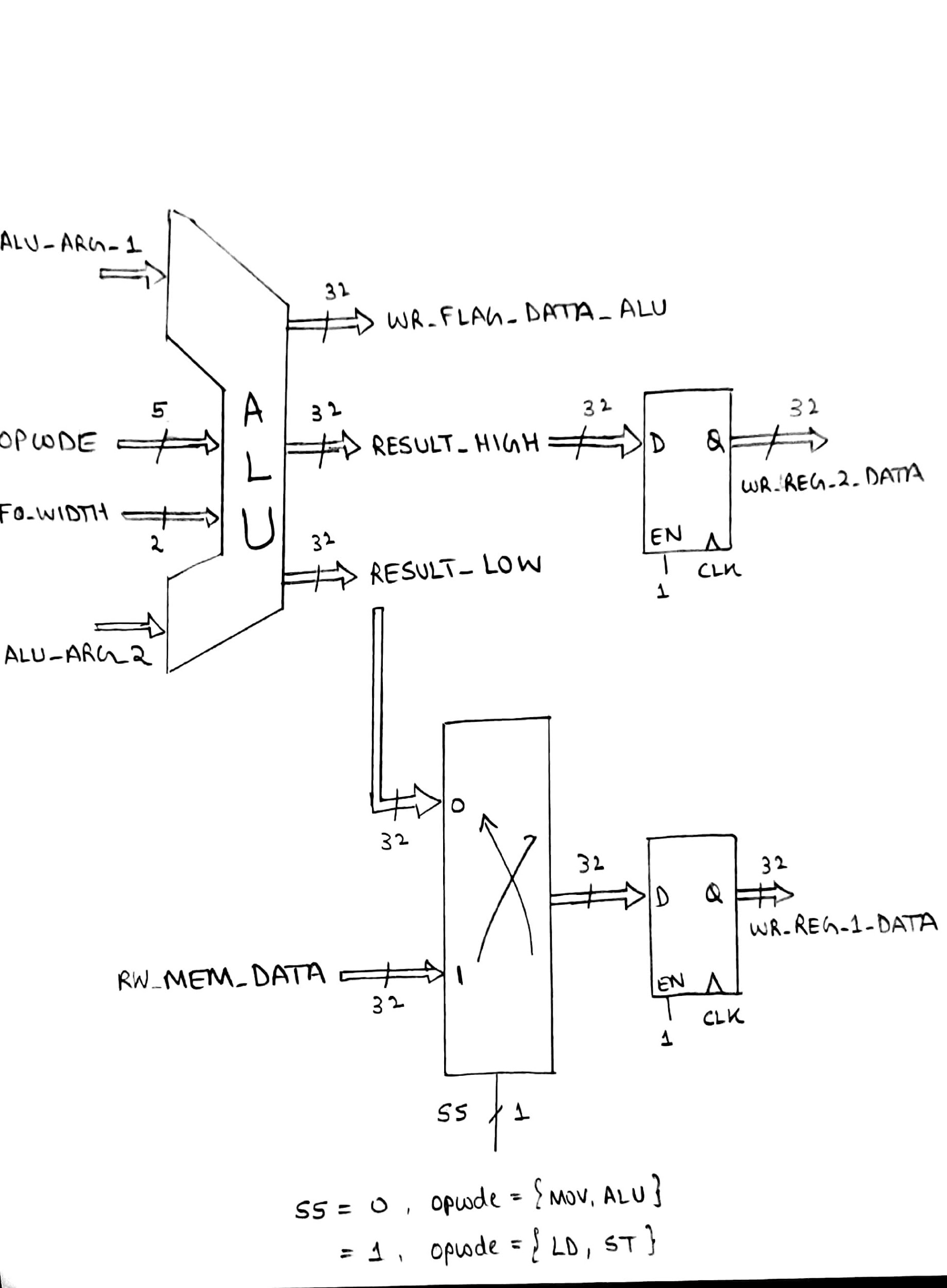
|  |  |  |  |
| --- | --- | --- | --- |
| Sl No. | Inputs | Outputs | Passthrough |
| 1. | K13 | RD\_REG\_A\_DATA | OPCODE |
| 2. | RD\_REG\_A\_NUM | ALU\_ARG\_1 | N |
| 3. | RD\_REG\_A\_VALID | RD\_REG\_B\_DATA | OFF23 |
| 4. | RD\_REG\_B\_NUM | ALU\_ARG\_2 | OFF27 |
| 5. | RD\_REG\_B\_VALID | RW\_MEM\_ADD | RD\_REG\_A\_VALID |
| 6. | K11 |  | RD\_REG\_B\_VALID |
| 7. | RD\_REG\_A\_DATA |  | RD\_REG\_C\_VALID |
| 8. | S |  | RD\_REG\_A\_NUM |
| 9. | RD\_REG\_B\_DATA |  | RD\_REG\_B\_NUM |
| 10. | OPCODE |  | RD\_REG\_C\_NUM |
| 11. |  |  | FO\_XACT\_WIDTH |
| 12. |  |  | WB\_XACT\_WIDTH |
| 13. |  |  | RD\_MEM\_VALID |
| 14. |  |  | WR\_MEM\_VALID |
| 15. |  |  | WR\_MEM\_DATA |
| 16. |  |  | WR\_MEM\_ADD |
| 17. |  |  | WR\_REG1\_VALID |
| 18. |  |  | WR\_REG2\_VALID |
| 19. |  |  | INC\_SP\_BY\_4 |
| 20. |  |  | DEC\_SP\_BY\_4 |
| 21. |  |  | INC\_PC\_BY\_4 |
| 22. |  |  | WR\_PC\_ABSOLUTE |
| 23. |  |  | WR\_PC\_RELATIVE |
| 24. |  |  | WR\_PC\_COND |
| 25. |  |  | WR\_PC\_DATA |
| 26. |  |  | FO\_XACT\_VALID |
| 27. |  |  | WB\_XACT\_VALID |
| 28. |  |  | WR\_REG\_1\_DATA |
| 29. |  |  | WR\_REG\_2\_DATA |
| 30. |  |  | WR\_REG\_1\_NUM |

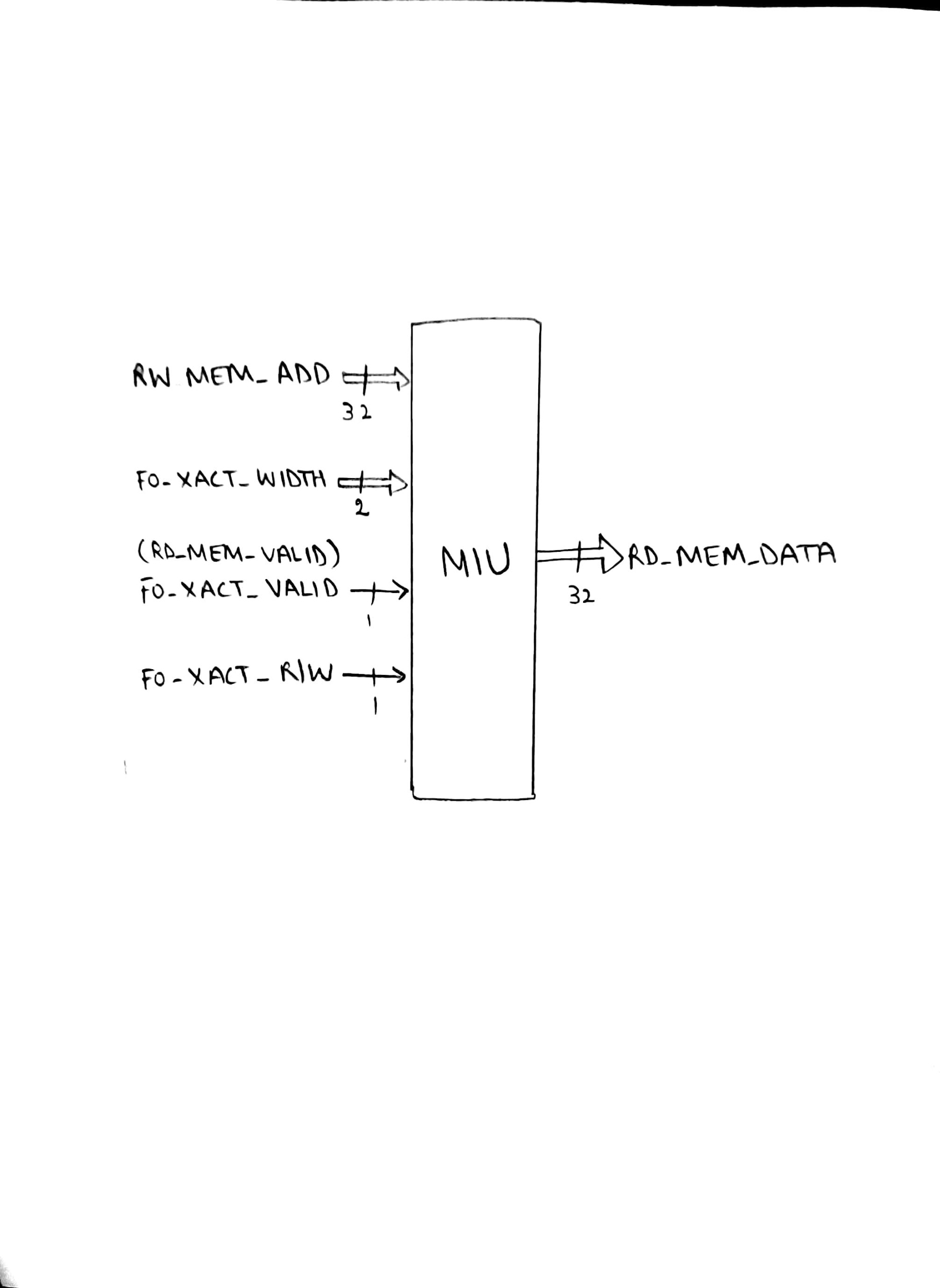
1. **FETCH OPERAND AND EXECUTE**

**3.1 Theory of Operation:**

* This is the 3rd stage of the pipeline. This stage executes the instructions as well as fetch operand from the memory which is also called read from memory.
* For certain opcodes like LD, POP, RETI, the date from memory is read.
* If the memory reading is slow because of the slow memory, then there might be a case for stalling which is a hazard.
* The stalling of a signal is shown after the WB stage.
* The mux decides the function and the ALU provides the data for further processing.

**3.2 Schematics and Logical Expressions**





**3.3** **Summary of Inputs / Outputs and Pass-through data:**

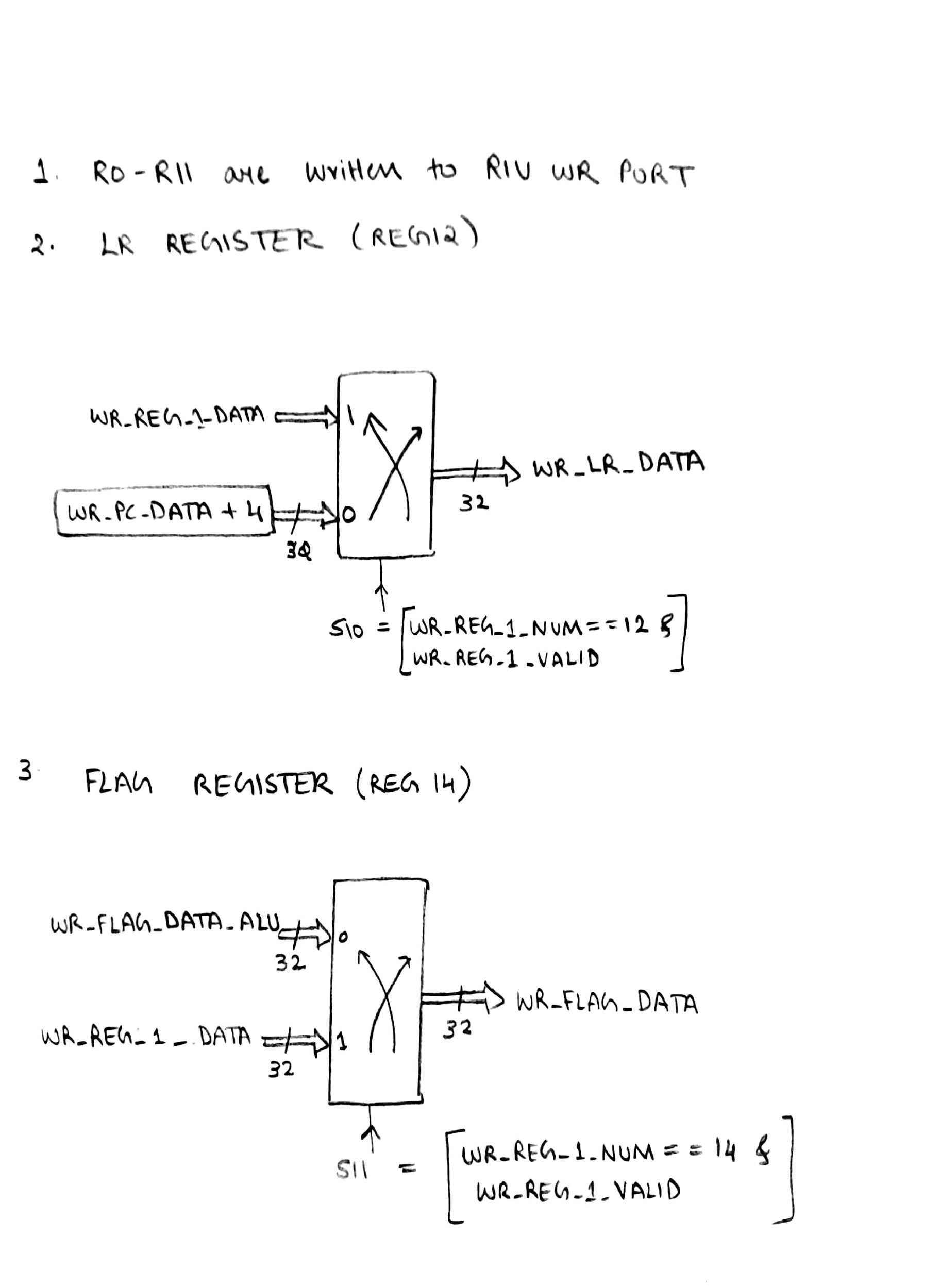
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sl No. | Inputs | Outputs | Passthrough |  |
| 1. | RD\_MEM\_VALID(FO\_XACT\_VALID) | WR\_REG1\_DATA | WR\_PC\_ABSOLUTE |  |
| 2. | RW\_MEM\_ADD | WR\_REG2\_DATA | WR\_PC\_RELATIVE |  |
| 3. | FO\_XACT\_WIDTH | WR\_FLAG\_DATA\_ALU | WR\_PC\_COND |  |
| 4. | OPCODE | RD\_MEM\_DATA | WR\_PC\_DATA |  |
| 5. | ALU\_ARG\_1 |  | DEC\_SP\_BY\_4 |  |
| 6. | ALU\_ARG\_2 |  | INC\_PC\_BY\_4 |  |
| 7. | FO\_XACT\_DONE |  | INC\_SP\_BY\_4 |  |
| 8. | RW\_MEM\_DATA(INTERNAL) |  | WR\_MEM\_ADD |  |
| 9. | SS |  | WR\_MEM\_VALID |  |
| 10. | FO\_XACT\_R/W |  | FO\_XACT\_WIDTH |  |
| 11. | FO\_XACT\_DONE |  | WR\_REG1\_NUM |  |
| 12. |  |  | WR\_REG1\_VALID |  |
| 13. |  |  | WR\_REG1\_DATA |  |
| 14. |  |  | WR\_REG2\_VALID |  |
| 15. |  |  | WR\_REG2\_DATA |  |
| 16. |  |  | WB\_XACT\_WIDTH |  |
| 17. |  |  | WR\_MEM\_DATA |  |
| 18. |  |  | WR\_REG\_1\_DATA |  |
| 19. |  |  | WR\_REG\_2\_DATA |  |
| 20. |  |  | WR\_REG\_1\_NUM |  |
| 21. |  |  | WB\_XACT\_WIDTH |  |
| 22. |  |  |  |  |

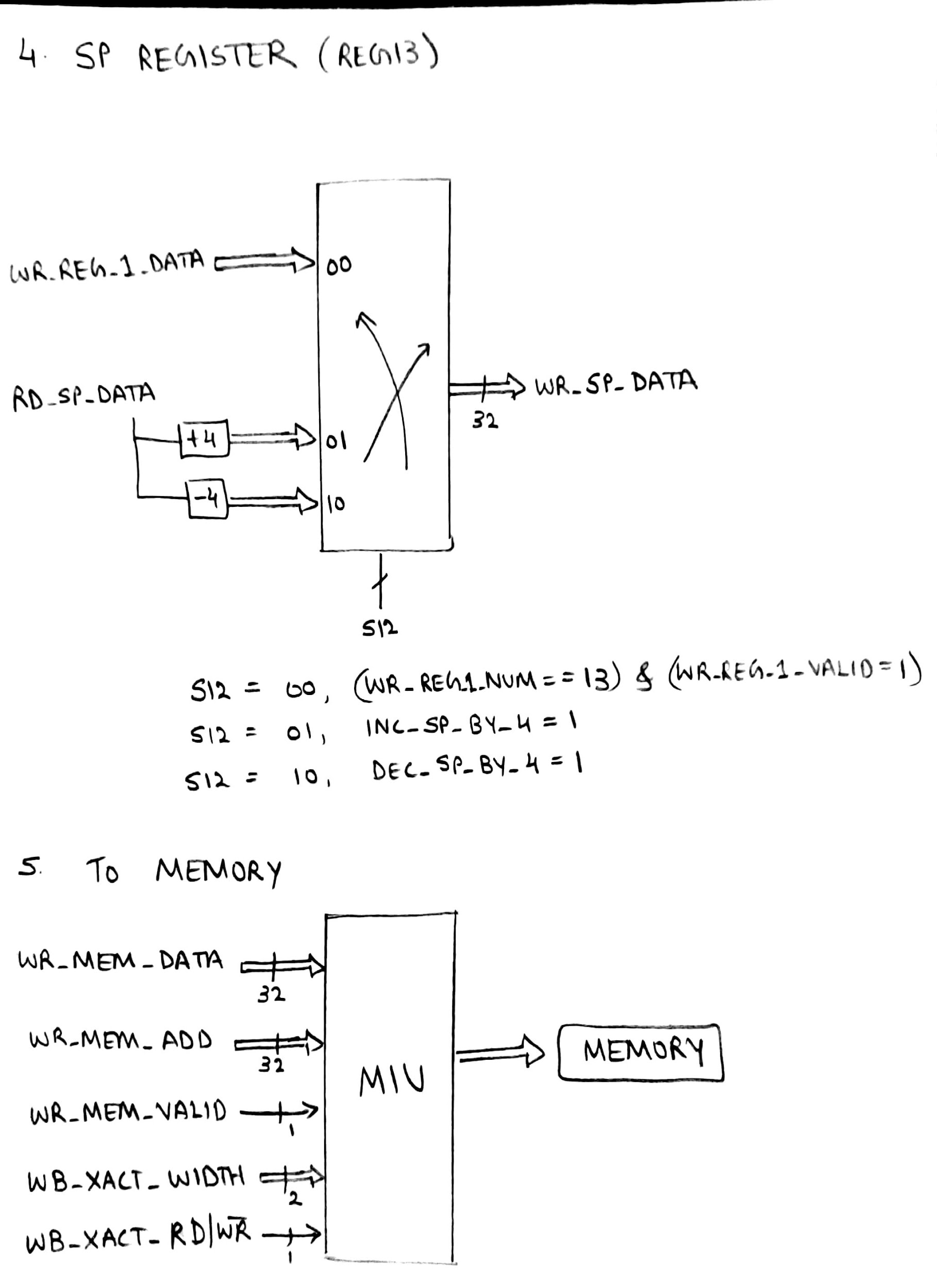
1. **WRITE BACK**

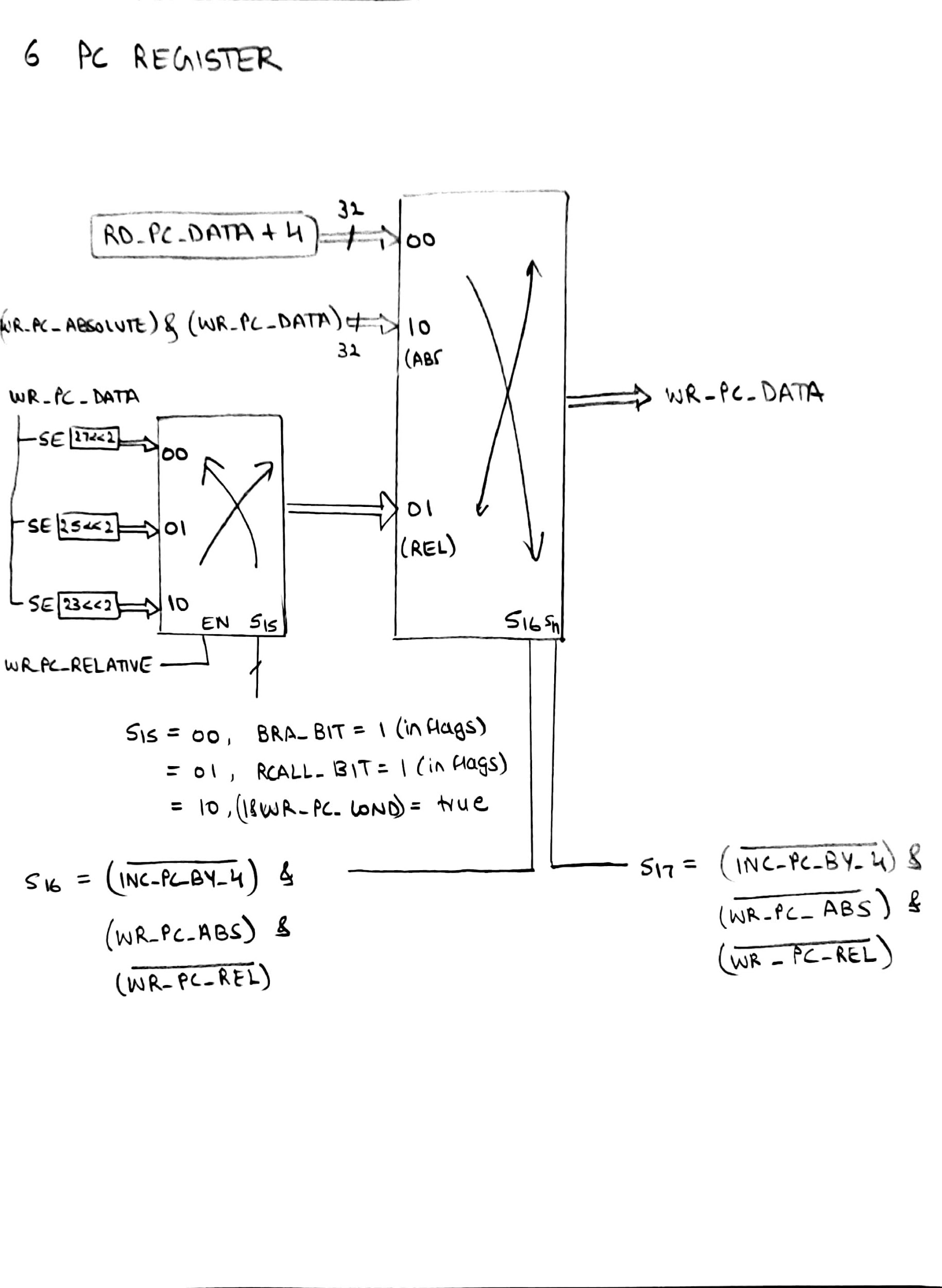
**4.1 Theory of Operation:**

1. The final stage of the pipeline where write to the registers and the memory occurs.
2. The registers PC, SP (RETI, PUSH, POP, INT), LR, Flags are written here.
3. The memory write is done by interfacing with the MIU.
4. The signals from FO/EX with the pass-through are inputs to this stage.
5. The interface to MIU and RIU are described later. The WB stage interfaces with the MIU and RIU for writing the data on to the registers and memory.

**4.2 Schematics and Logical Expressions**







**4.3 Summary of Inputs / Outputs and Pass-through data:**

This is the final stage of pipeline hence there is no passthrough of signals here.

NOTE: There are no pass through or outputs in this stage.

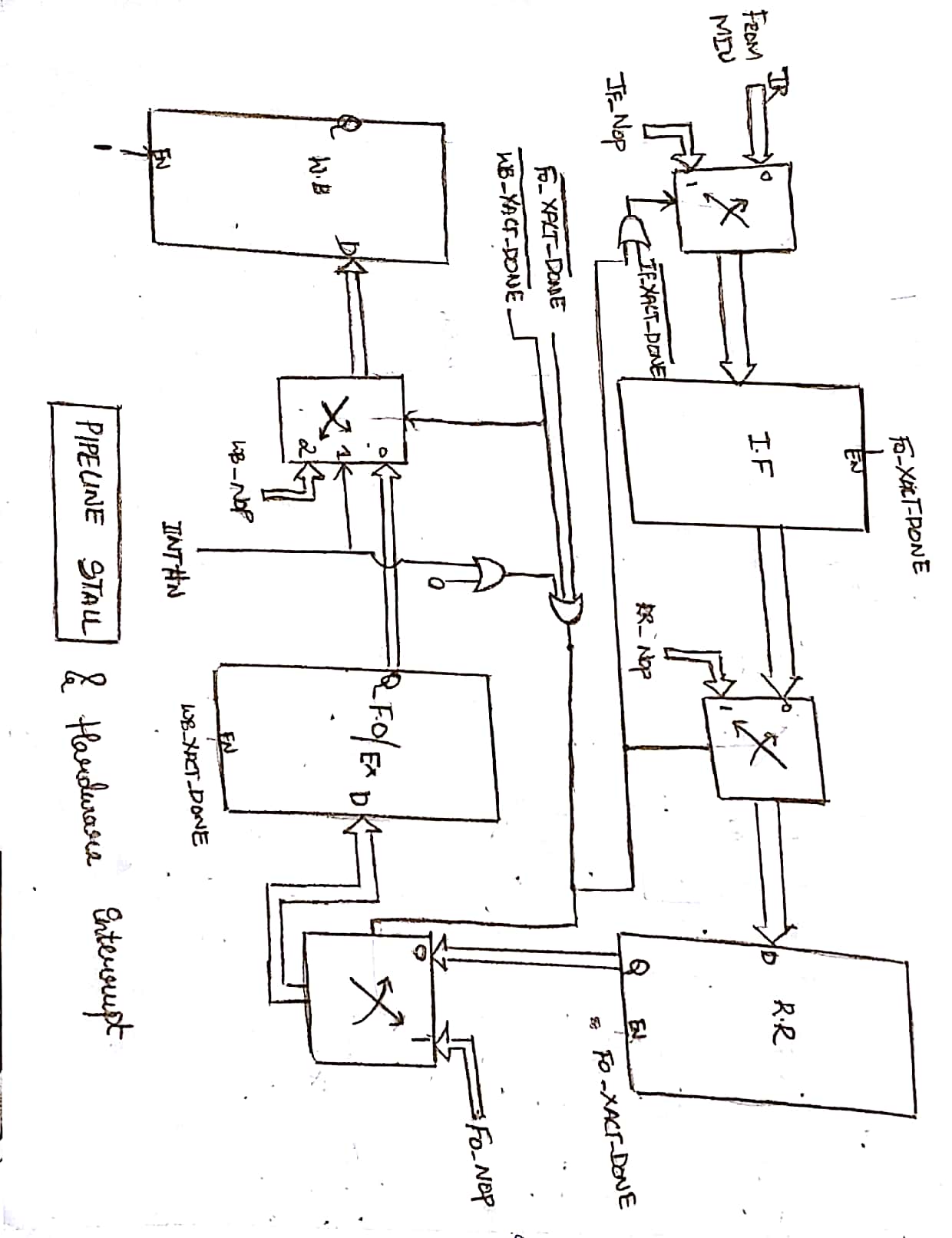
|  |  |
| --- | --- |
| Sl No. | Inputs |
| 1. | WR\_REG1\_VALID |
| 2. | WR\_REG1\_DATA |
| 3. | WR\_REG1\_NUM |
| 4. | WR\_REG2\_VALID |
| 5. | WR\_REG2\_DATA |
| 6. | WR\_MEM\_ADD |
| 7. | WR\_MEM\_DATA |
| 8. | WR\_MEM\_VALID |
| 9. | WB\_XACT**\_**WIDTH |
| 10. | WR\_PC\_ABSOLUTE |
| 11. | WR\_PC\_RELATIVE |
| 12. | WR\_PC\_COND |
| 13. | WR\_PC\_DATA |
| 14. | DEC\_SP\_BY\_4 |
| 15. | INC\_PC\_BY\_4 |
| 16. | INC\_SP\_BY\_4 |
| 17. | WR\_FLAG\_VALID |
| 18. | WR\_FLAG\_DATA\_ALU |
| 19. | WB\_XACT\_DONE |
| 20. | S10 |
| 21. | S11 |
| 22. | S12 |
| 23. | S16 |
| 24. | S17 |
| 25. |  |
| 26. |  |

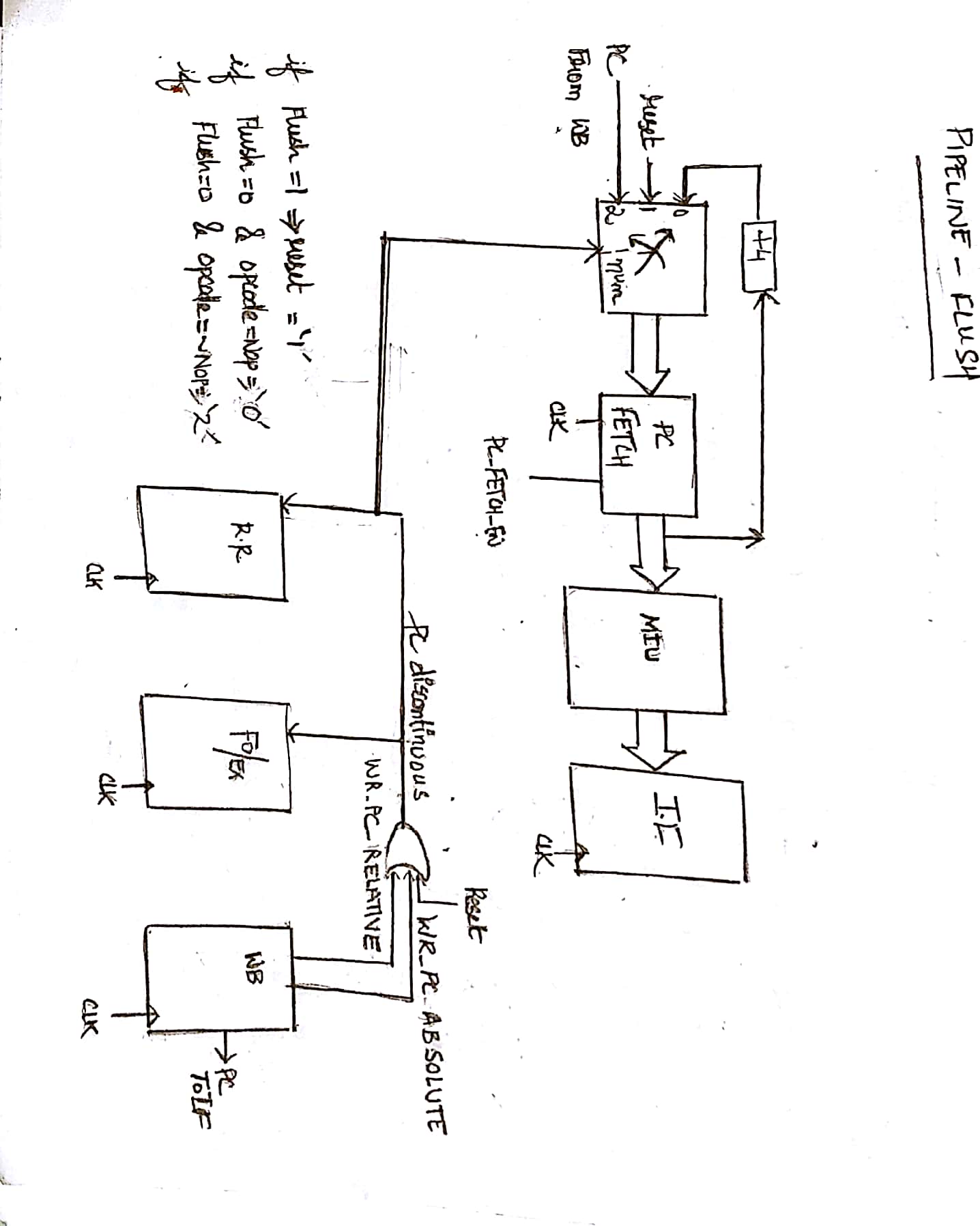
1. **STALL CONTROL/FLUSH LOGIC AND H/W INTERRUPT**

**5.1 Theory of Operation:**

1. A structural hazard in the design, which is detected and avoided by using the following design.
2. Each stage can be stalled, each has different conditions for stalling which is indicated below.
3. When a stage is stalled then the registers should not change its value hence, we use create a bubble in the stage.
4. The PC value is the same when stall is executed.
5. The flush logic is depended on the hazard where PC discontinuous values in the pipeline stages have difference, the PC value at IF and the PC value at WB have any difference other than 12. If the difference is not equal to 12 then the entire pipeline is flushed.
6. Flush is meant to remove all the data in the pipeline.
7. We have connected an external hardware that is connected as shown in the schematics.
8. Whenever the hardware interrupt is given the rest, all stages before WB are stalled and hence it is considered in this logic.
9. When interrupt is given here, then the interrupt vector is given to the WB stage along with the stack, PC values so that it can start the execution while stalling the other stages.

**5.2 Schematics and Logical Expressions**

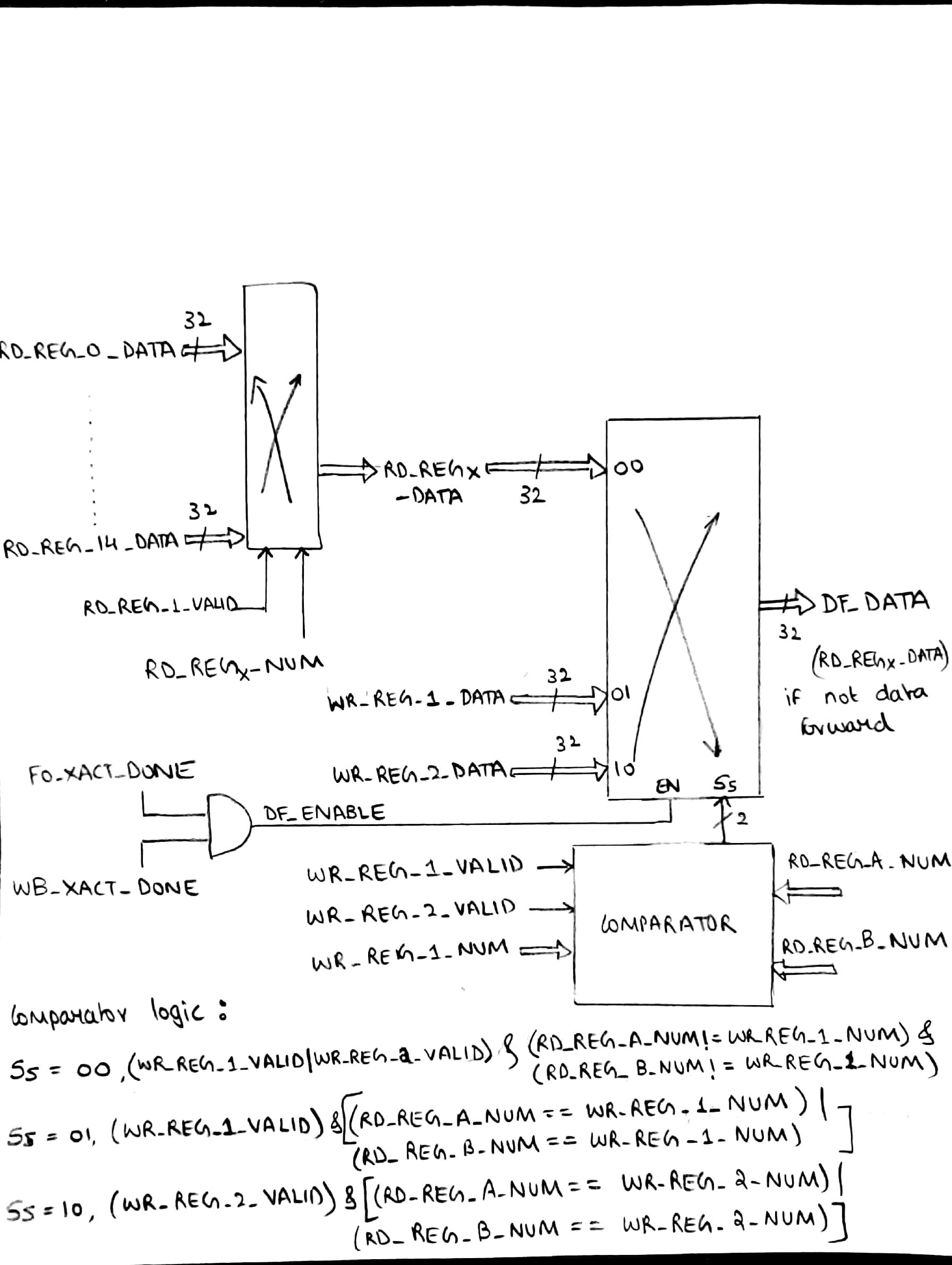




1. **DATA FORWARDING**

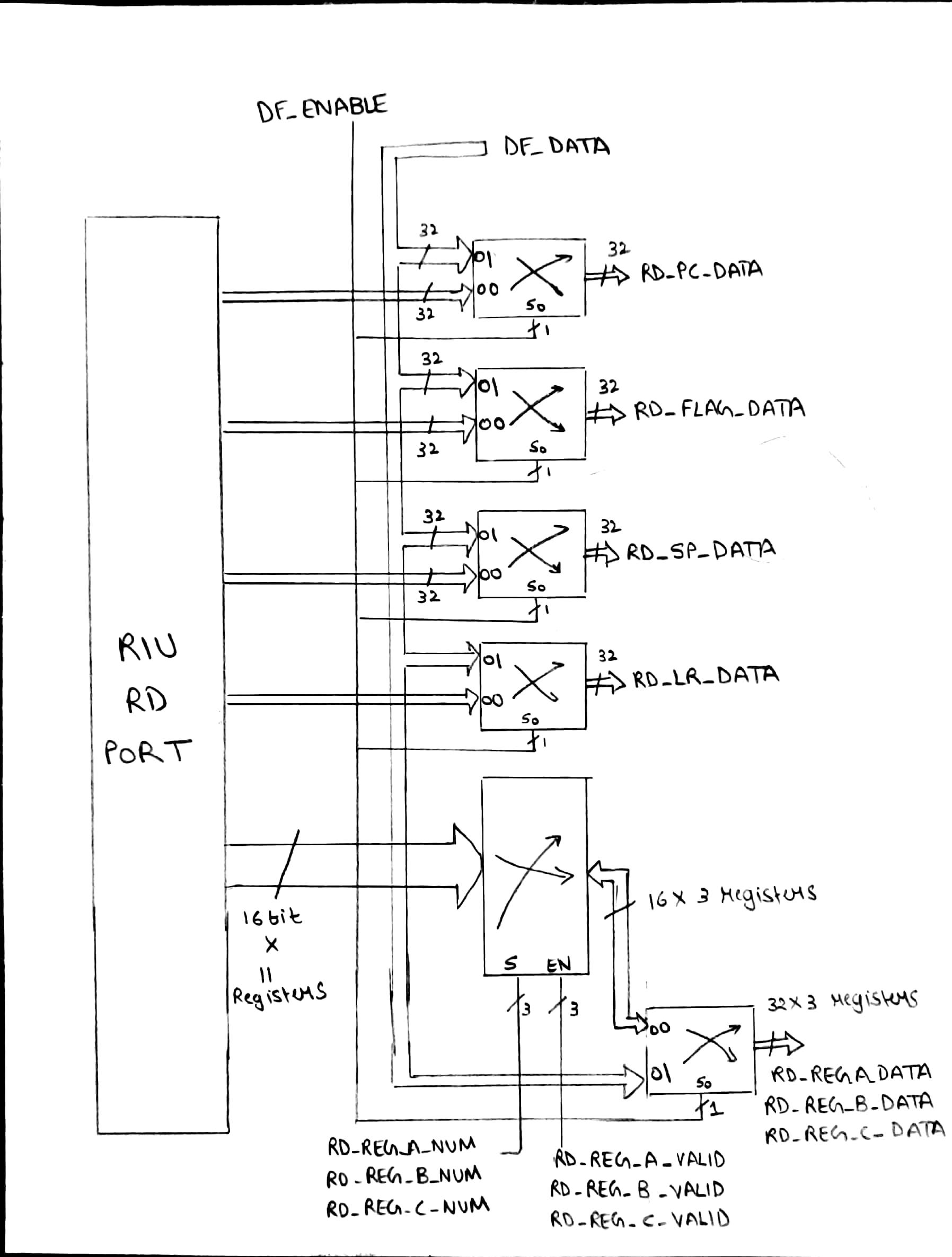
**6.1 Theory of Operation:**

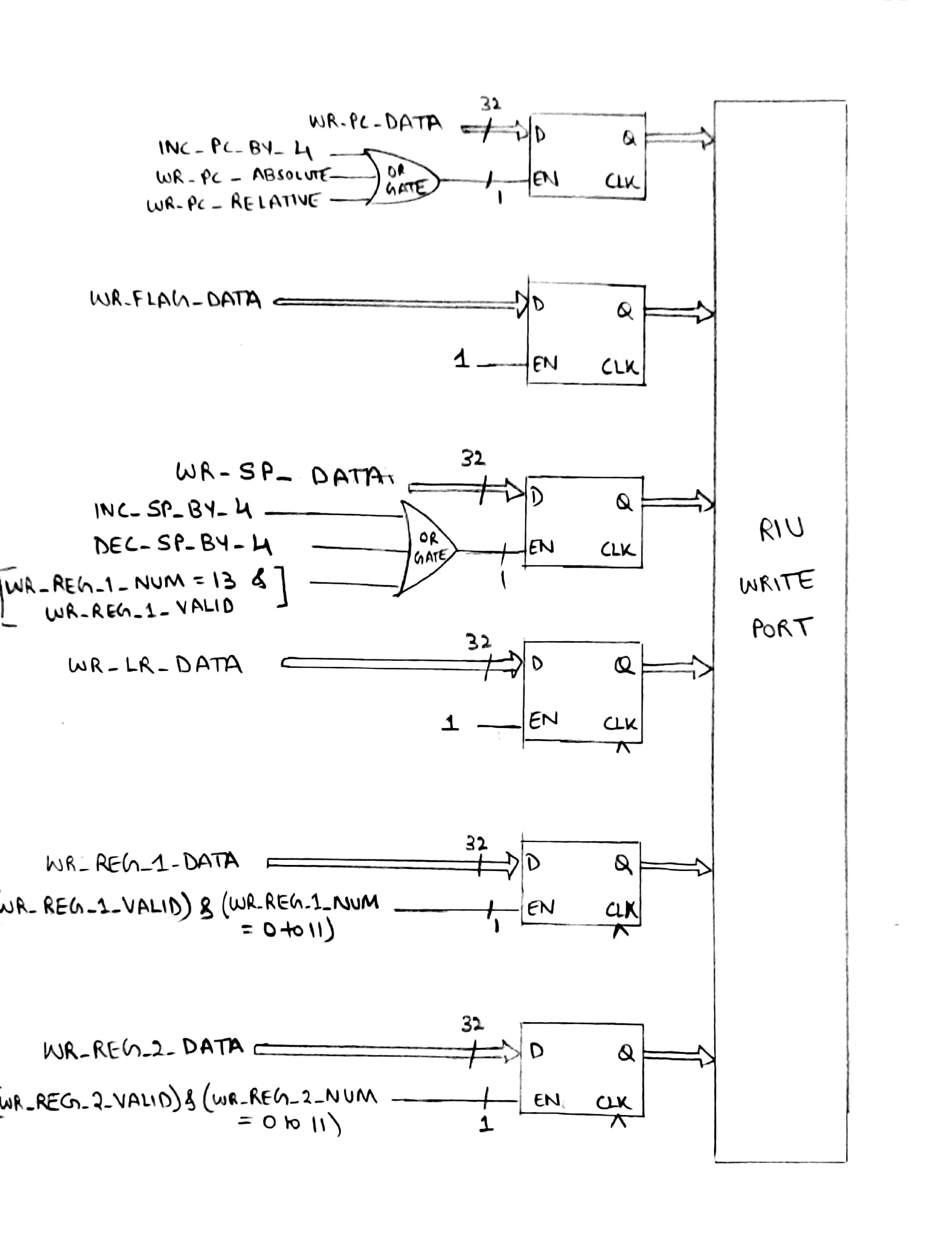
1. Data hazard with read after write can be avoided using this method.
2. Current Reg\_ num is compared with the next Reg\_ num, when it is valid, then, the data forwarding takes place. The WR\_REG\_1\_DATA is written to the ALU which requires the register value.
3. Similarly, for the 64bit data, the validation here requires from 2 registers and is sent to the ALU that requires the register values.



1. **REGISTER LOGIC**
   1. **Theory of Operation:**
2. We have a total of 16, 32-bit registers which are read and written at the RR and WB stages of the pipeline.
3. We have 4 hard coded registers they are PC, SP, LR, flags whose values are written and read by the following logic.

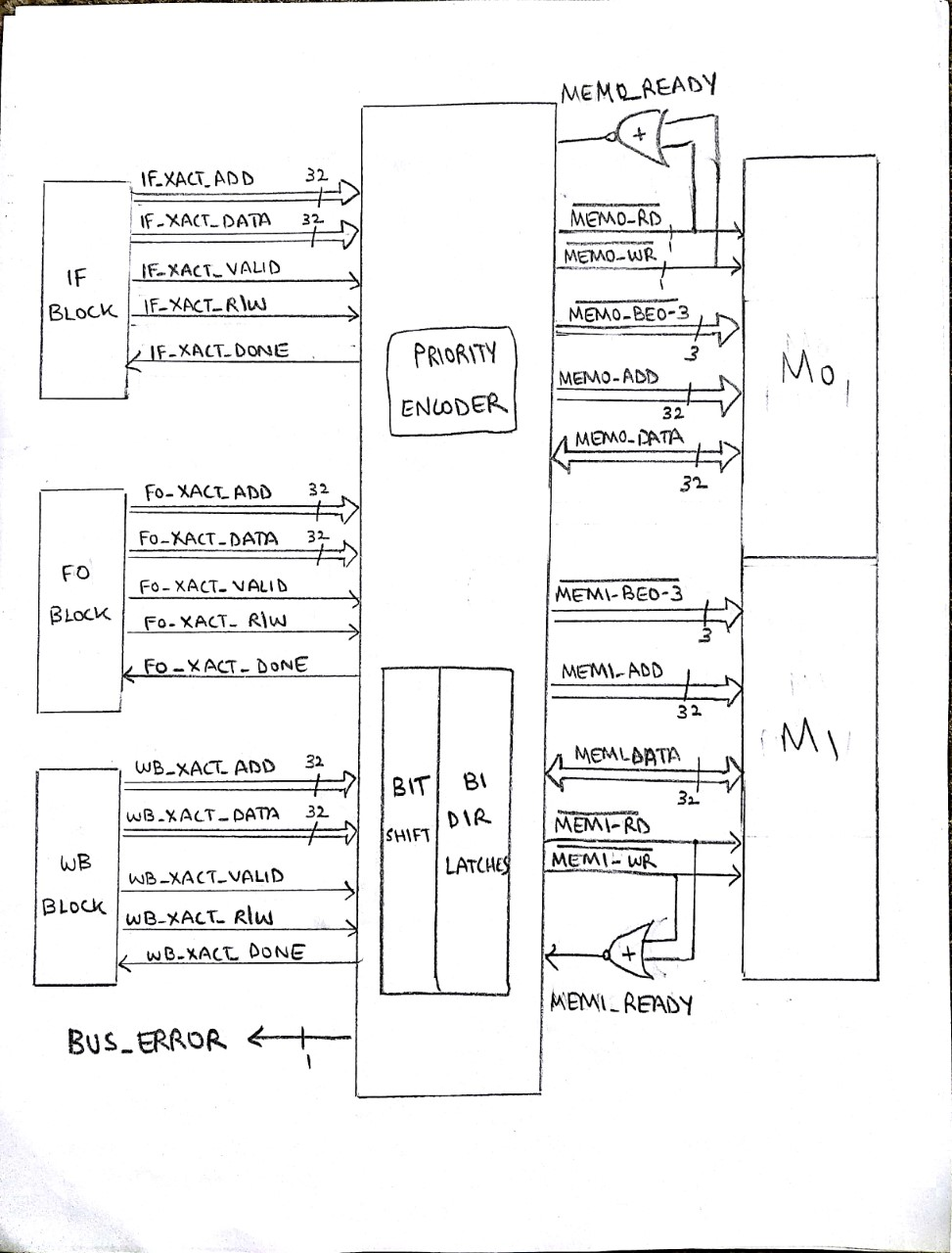
**8.2 Schematics and Logical Diagrams**



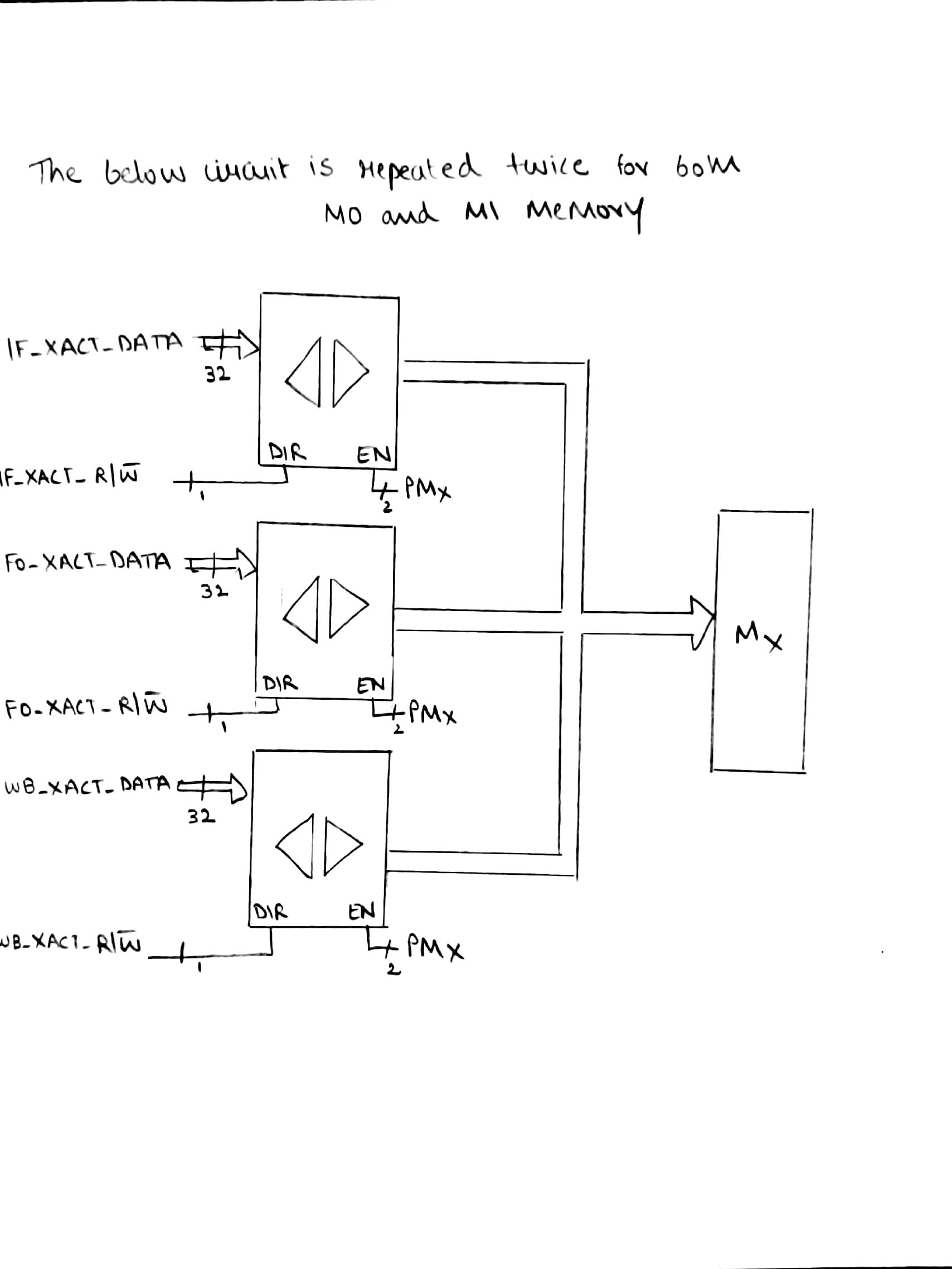


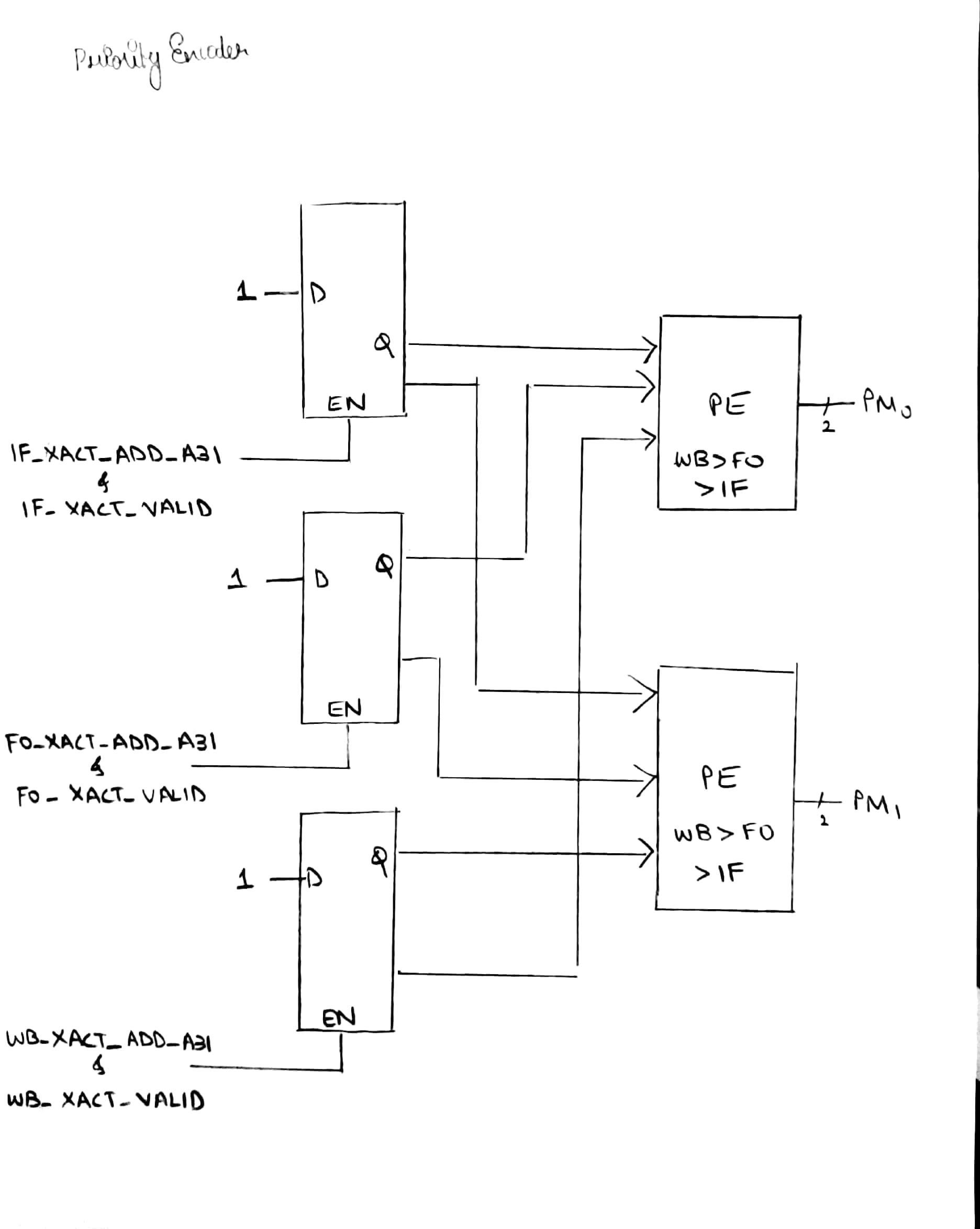
1. **MEMORY INTERFACE UNIT (MIU)**
   1. **Theory of Operation:**
2. The function of the memory interface unit depends on the pipeline stage accessing it.
3. The memory read is done at the FO stage whereas the write to memory is done at the WB stage.
4. The RD\_MEM\_ADD and RD\_MEM\_DATA read the data for FO.
5. The WR\_MEM\_ADD and WR\_MEM\_DATA write the data for WB.
6. The width is a standard command and it used by both read and write operation. It is specific for each instruction.

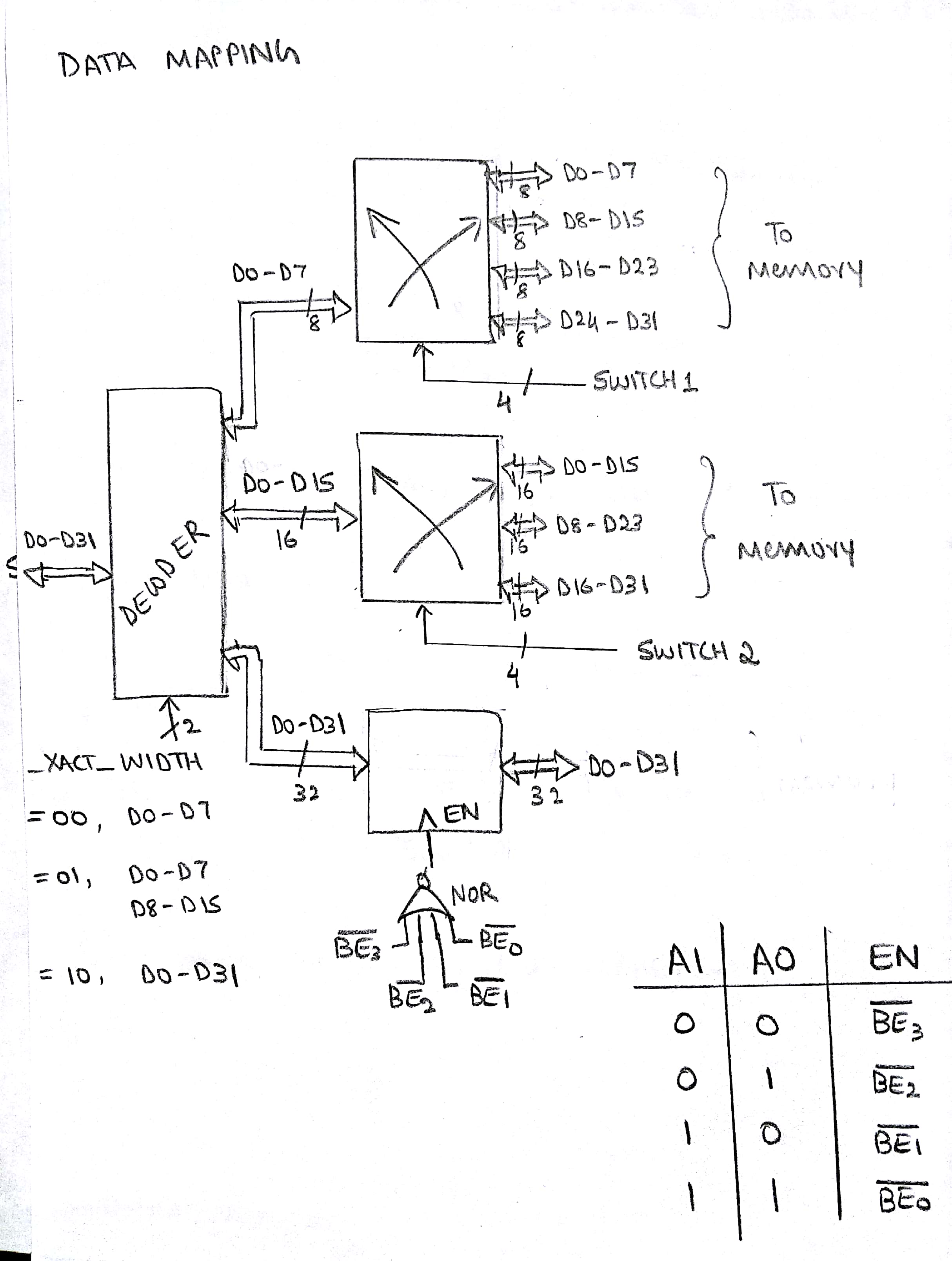
**8.2 Schematics and Logical Diagrams**

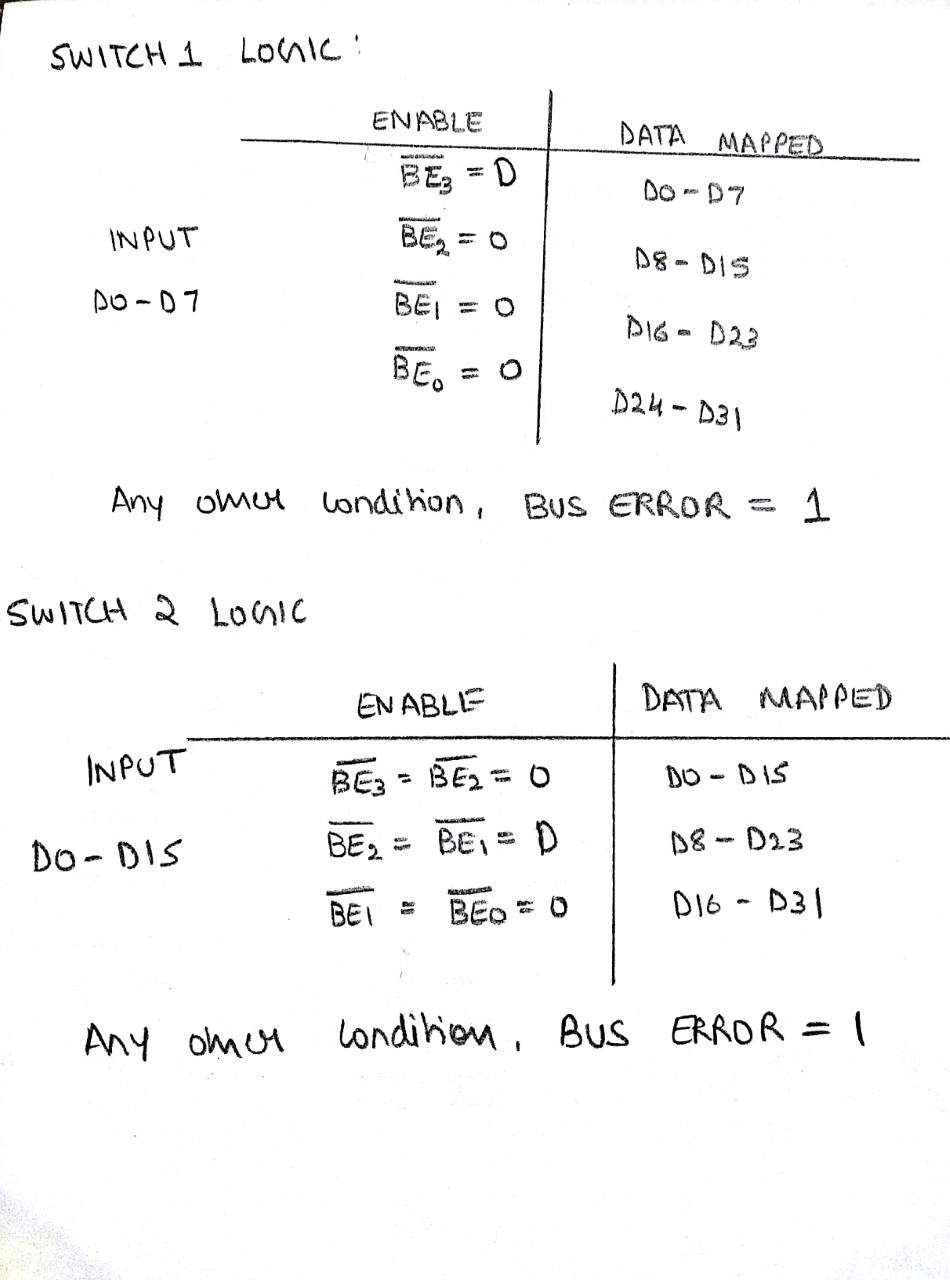


Bidirectional latches









------------------------------------------------------------------------------------------------------------------------------------