# IPA Project : Y-86 Instruction Set Architecture

Team Name - SIGMA
Team members - Pallav Subrahmanyam Koppisetti (2020102070) , KNV Karthikeya (2020102003)

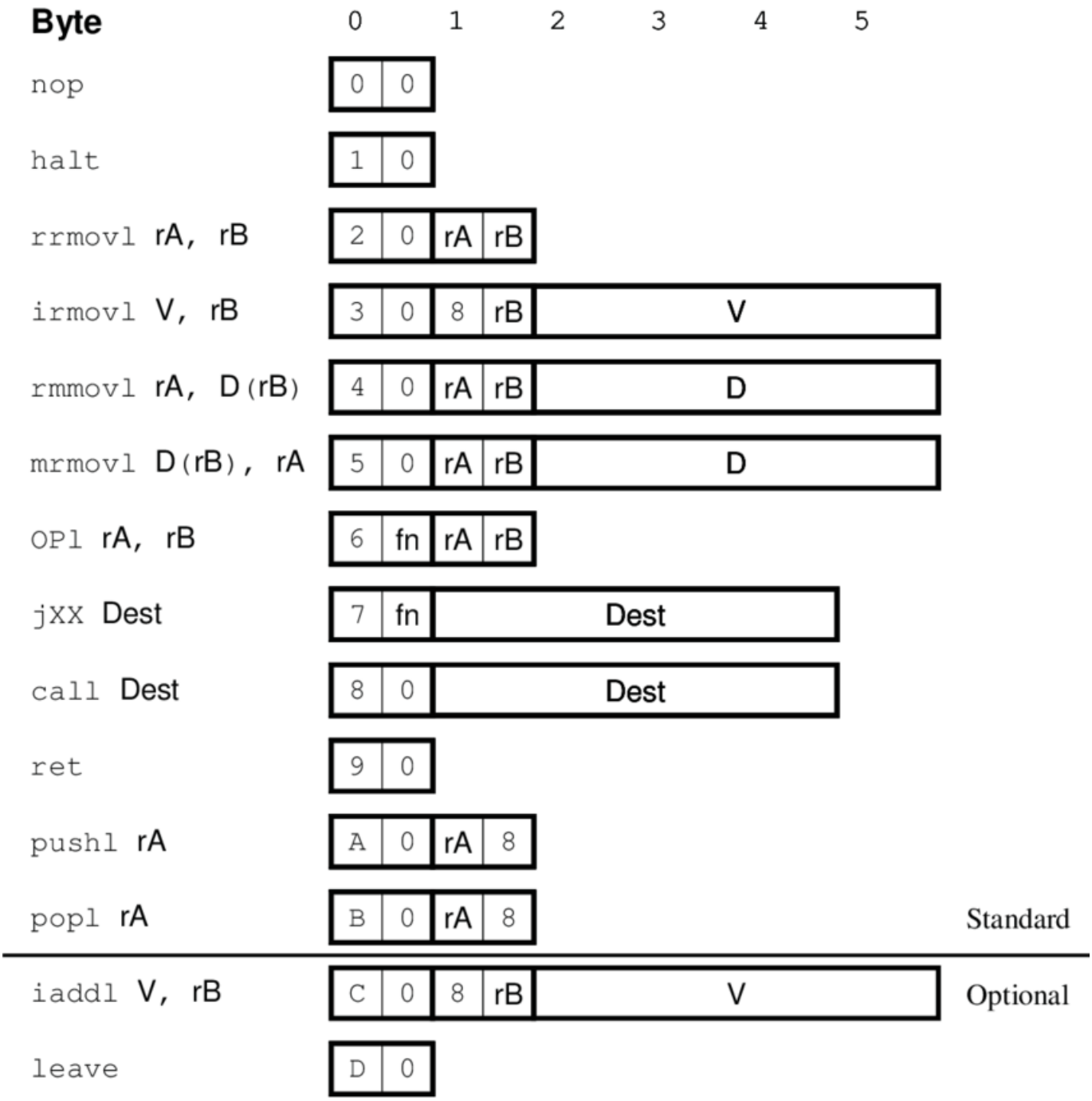
## Mid evaluation report :-

We have implemented a sequential y86 architecture processor using Verilog, which can be divided into 5 modules.

- Fetch
- Decode
- Execute
- Memory
- PC update/writeback

The commands supported by our processor are :-

- ihalt
- inop
- irrmovq
- iirmovq
- irmmovq
- imrmovq
- iopq
- ijxx
- icall
- iret
- ipushq
- ipop

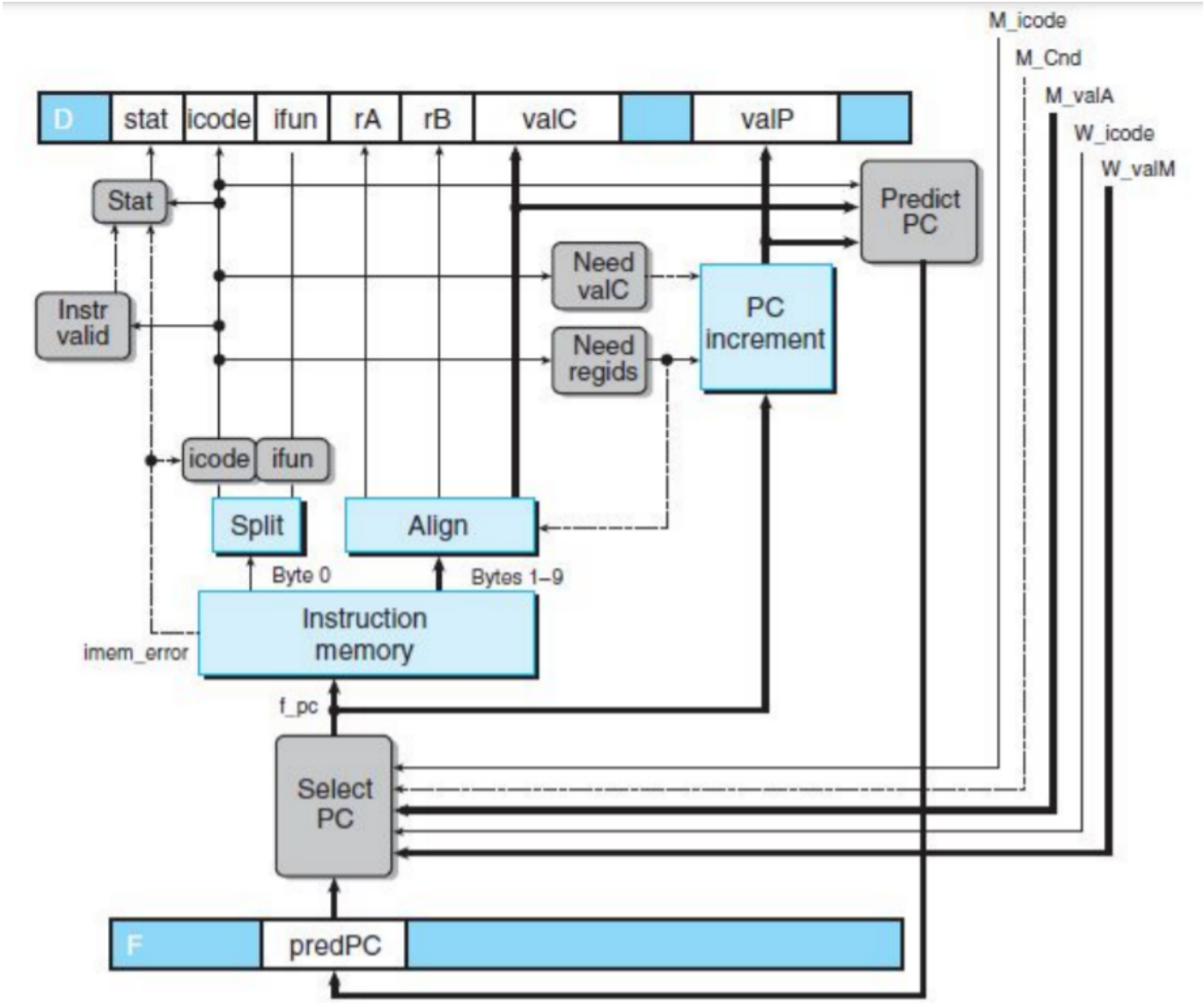



**The above image depicts the Y86-64 Instruction set**

## Fetch :-

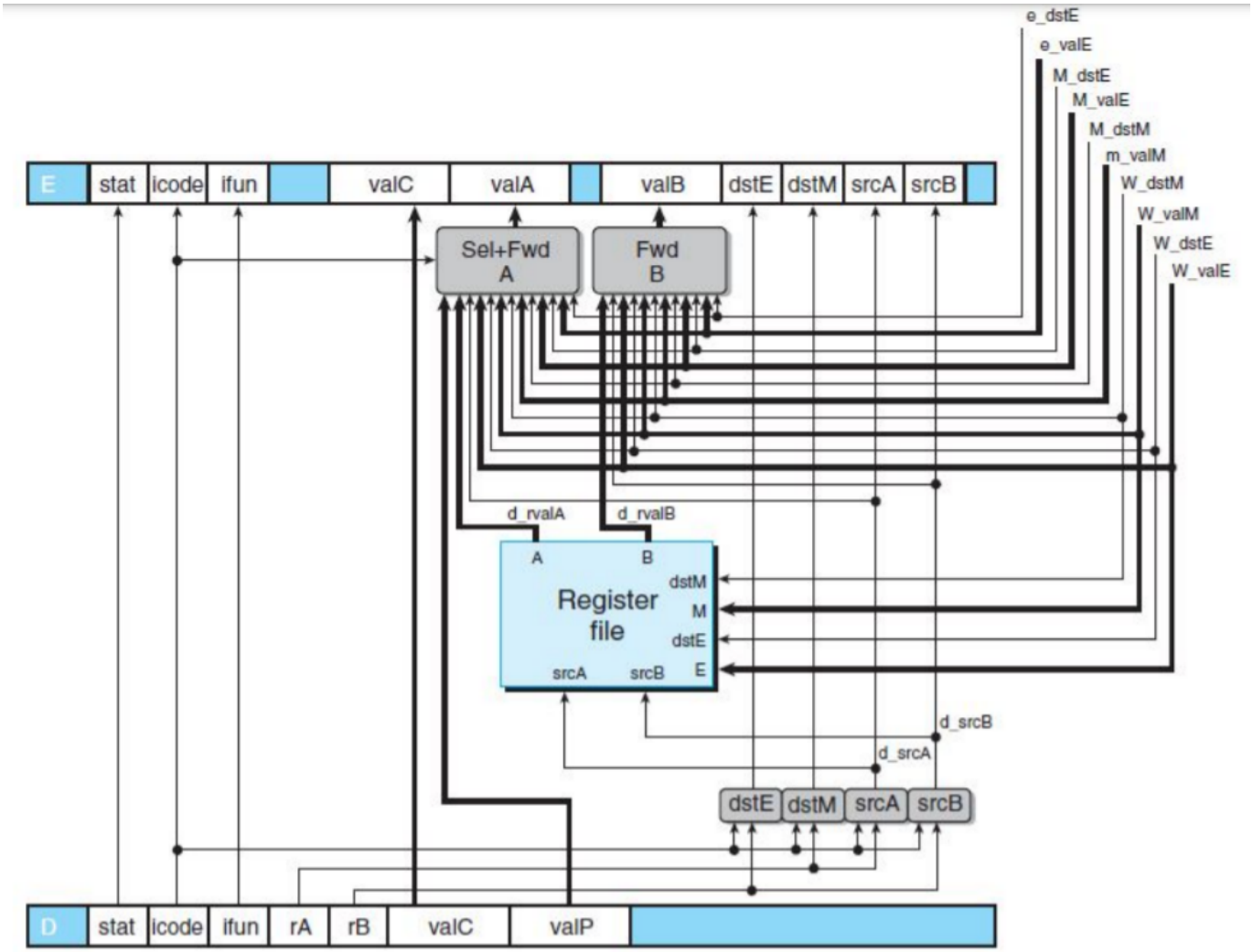
In the fetch stage, foremost step is to fetch the instruction from the memory address that is currently stored in the PC (program counter) and stored into the instruction register. Now, when this step ends, the PC points to the next instruction to be read in the next cycle and the cycle continues like this to fetch all the instructions. The first four bits are assigned to icode and the next 4 bits are assigned to ifun by the split module, if the pc points to a valid address in the instruction memory. The align module describes how the processor extracts the remaining fields from an instruction, depending on whether or not the instruction has a register specifier byte.

The last module used in Fetch module is known as pc_inc_ that is used to generate signal named valP, which is based upon the other values from other units.




## Decode :-

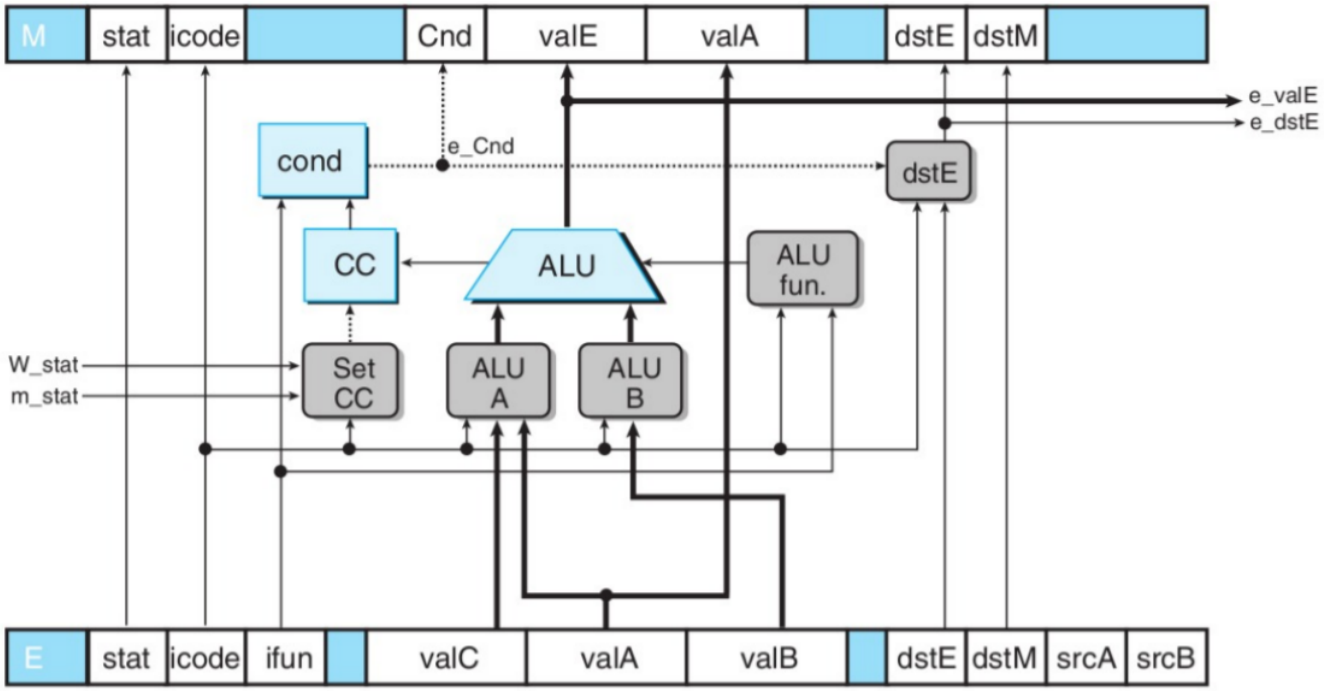
In the decode, we take icode,rA and rB as inputs and then depending on the value of icode we rA and assign it to valA, also read rB, and assign it to valB. The srcA and srcB blocks will determine whether we need to write valA, valB depending on icode. The block named register file in the diagram is used to get the values from the registers used in the instructions which is to be decoded. d. The ports A and B have address inputs as srcA and srcB. Ports E and M have address inputs dstE and dstM. The destination where valE is stored is held by the register ID dstE, similarly for valM its held by dstM. In the write back, valE and valM needs to write into the register file depending on the icode.




## Execute :-

The execute module is used to carry out the functional operations which are implemented by ALU. This stage computes the effective memory address, increments or decrements of the stack pointer. The module is used to send the decoded data as control signals to the functional units and then the required instructions are carried out before being transferred to the ALU, and finally the result being stored in the register. For conditional move instruction , the condition codes will be evaluated here, having inputs valC ,valA,valB,icode and ifun. Based on the inputs the module outputs valE and sets the cnd bit.

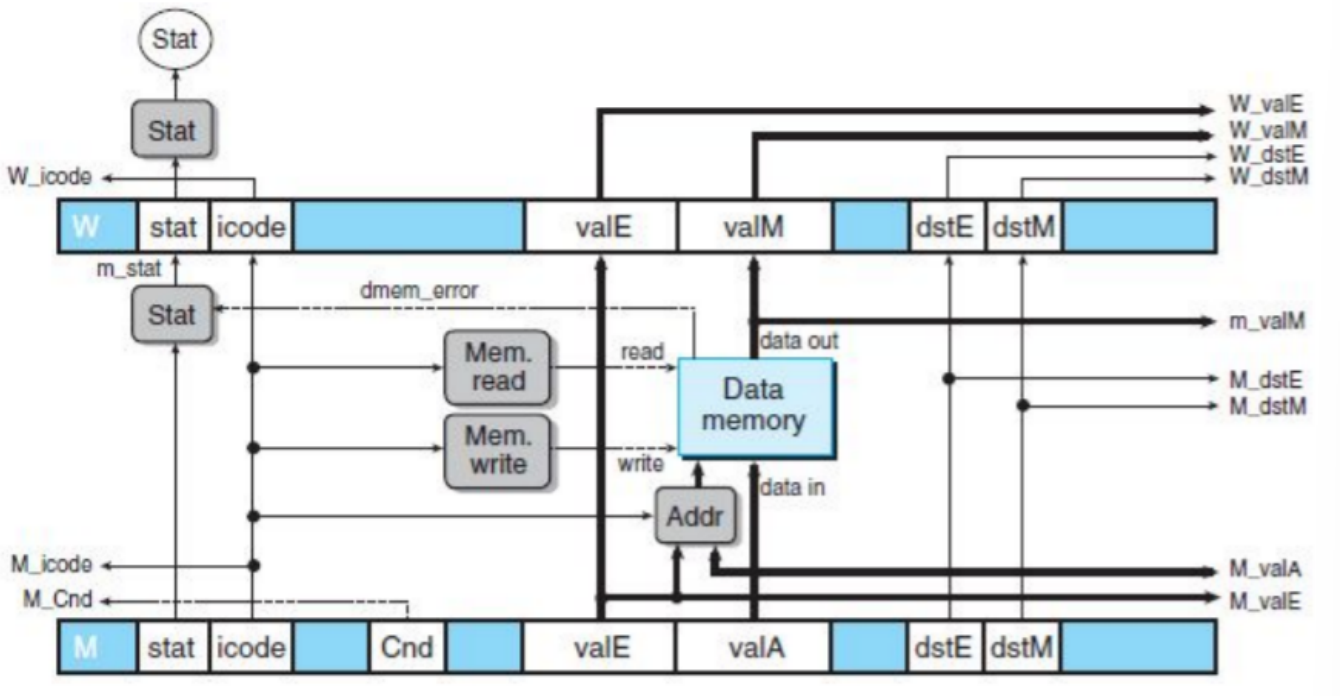
Our ALU generates the three signals everytime it operates on which the condition codes are based, i.e zero,sign,overflow.The execute stage connects to the memory stage, that determines where to store the computed values in memory.




## Memory :-

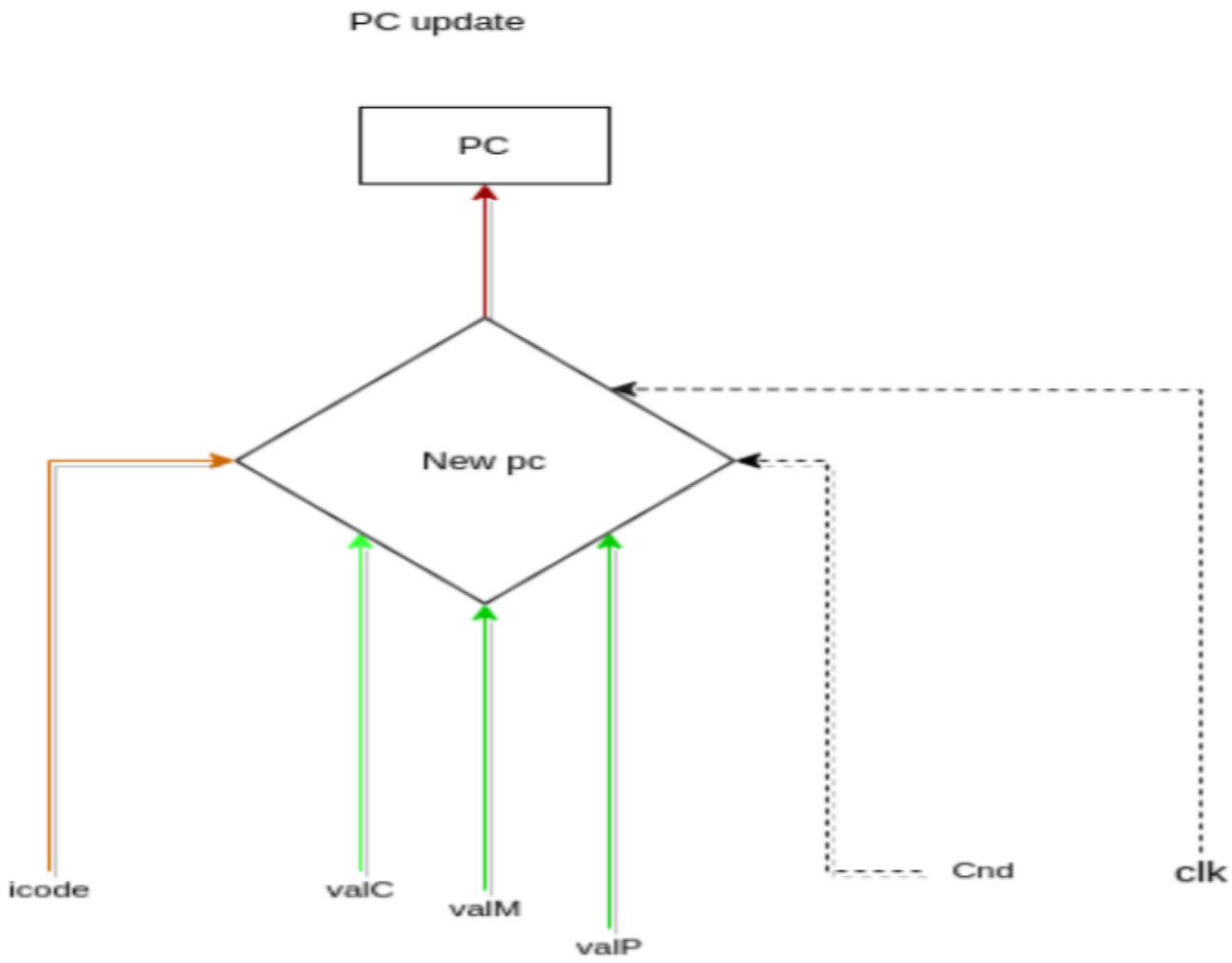
Memory stage can either read data from the memory or write data into the memory. The inputs for the memory stage are icode, valA, valE, valP and icode. The output is valM and also displays the current instruction status of the processor. When the read operation is performed, valM is generated. valE and valM are addresses for memory read and write.

**mem_addr** output the address of the memory that needs to be read or written. **The mem_data** outputs the data if anything needs to be written into the memory. mem_r and mem_w sets the read or write flags to 1 based on the instruction (icode).

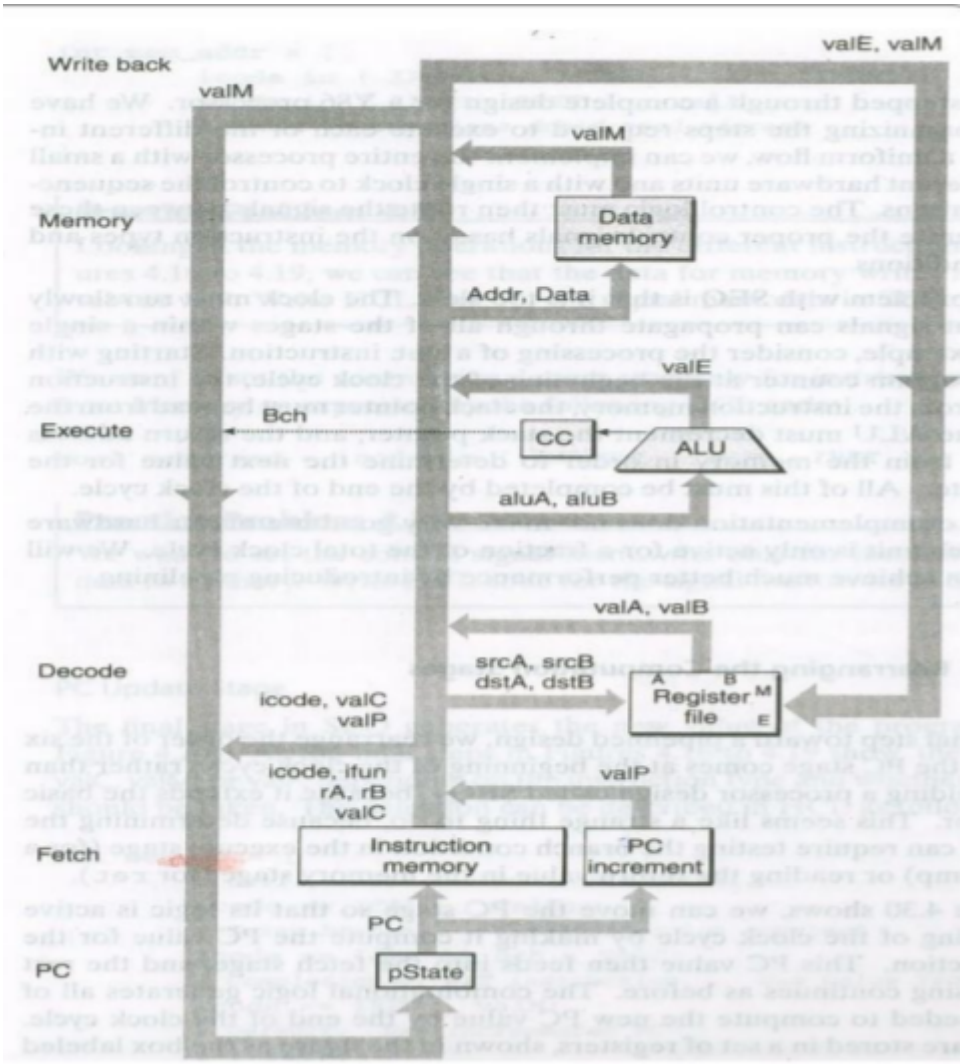



## PC update :-

PC update is a small module updating the PC value to the next instruction it needs to be pointed. The PC value can change to the next instruction or jump to a destination pc value if the condition is satisfied or based on call, ret instruction. All this is done based on the icode. The output of the pcupdate block is the next pc value. The pc update is only updated when all the previous modules are completed. We have used a clock and for every rising edge of the clk we update the pc value.




## Overall Implementation Flowchart :


Write back
valM
valE, valM
valM
Memory
Data memory
Addr, Data
valE
Bch
Execute
CC
ALU
aluA, aluB
valA, valB
Decode
srcA, srcB
dstA, dstB
A
B
M
Register file
E
icode, valC
valP
icode, ifun
rA, rB
valC
valP
Fetch
Instruction memory
PC increment
PC
PC
pState