Sequential Report

SEQ implementation - Where instructions are computed in sequential manner i.e. after the completion of one instruction only the other instruction is fetched. When one instruction is done with write_back stage then only the other instruction is fetched.

6 stages - Fetch , Decode , Execute , Memory , Write_back , pc_update

On positive edge of a clock the fetch fetches instruction -

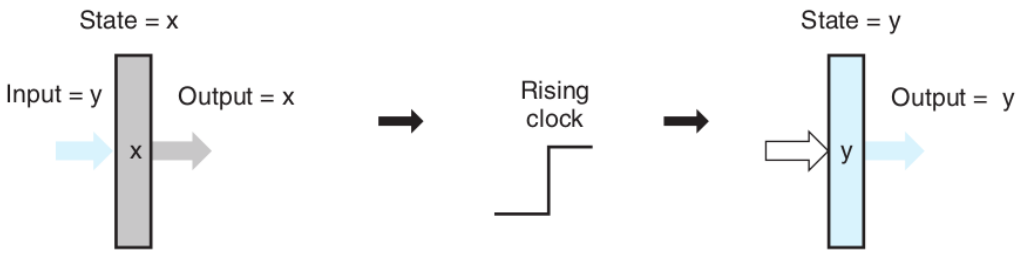
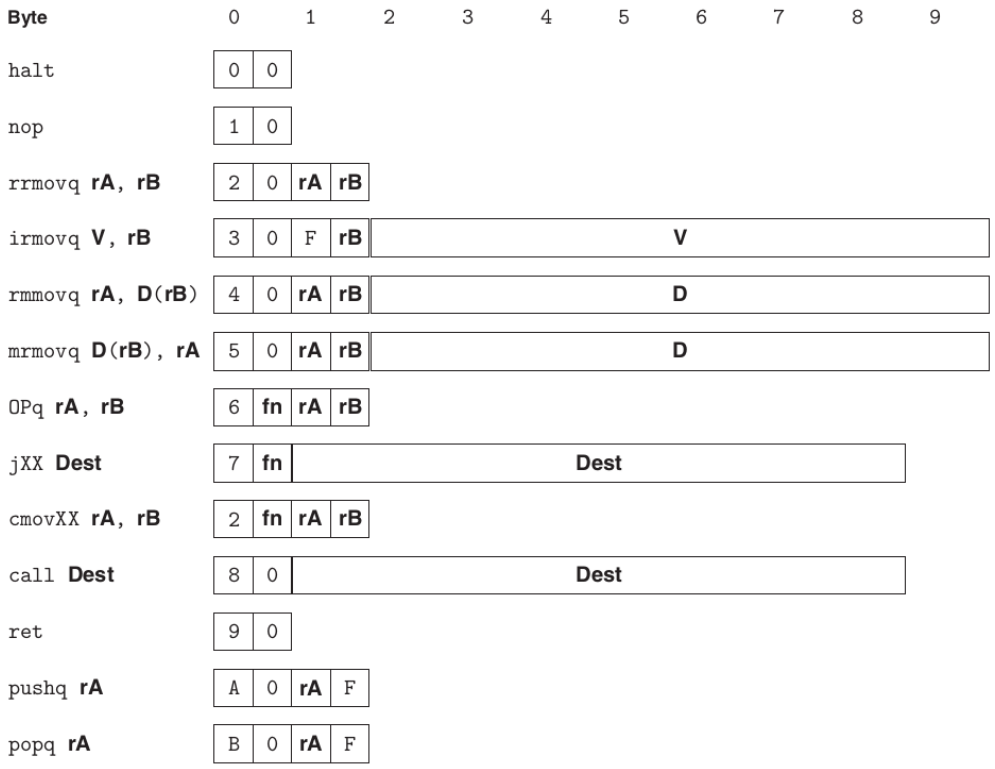


Figure 4.16 **Register operation.** The register outputs remain held at the current register state until the clock signal rises. When the clock rises, the values at the register inputs are captured to become the new register state.

Instruction set -

instruction memory ((memory / Bytes) allocated to each instruction) was formed by the following table -



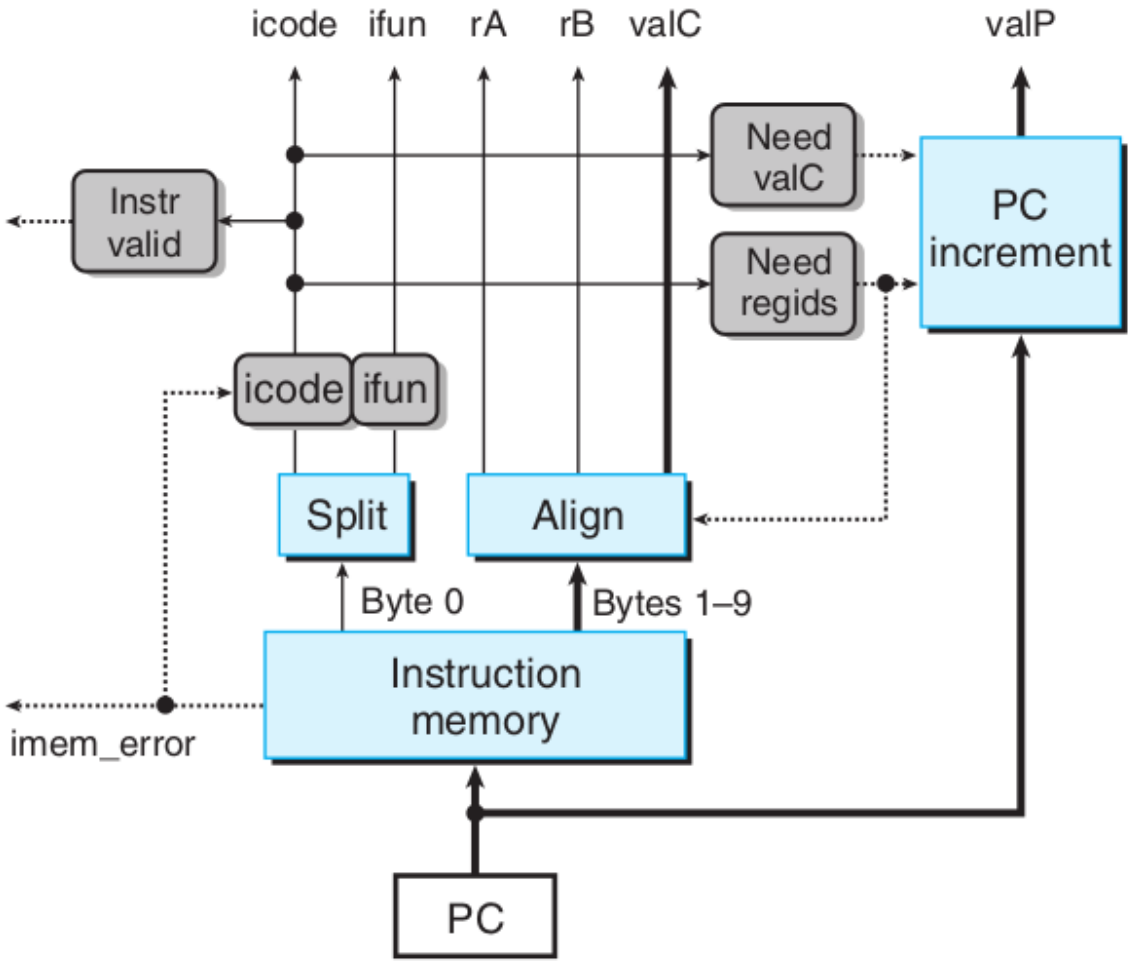
Register memory -

Number	Register name	Number	Register name
0	%rax	8	%r8
1	%rcx	9	%r9
2	%rdx	A	%r10
3	%rbx	B	%r11
4	%rsp	C	%r12
5	%rbp	D	%r13
6	%rsi	E	%r14
7	%rdi	F	No register

Figure 4.4 Y86-64 program register identifiers. Each of the 15 program registers has an associated identifier (ID) ranging from 0 to 0xE. ID 0xF in a register field of an instruction indicates the absence of a register operand.

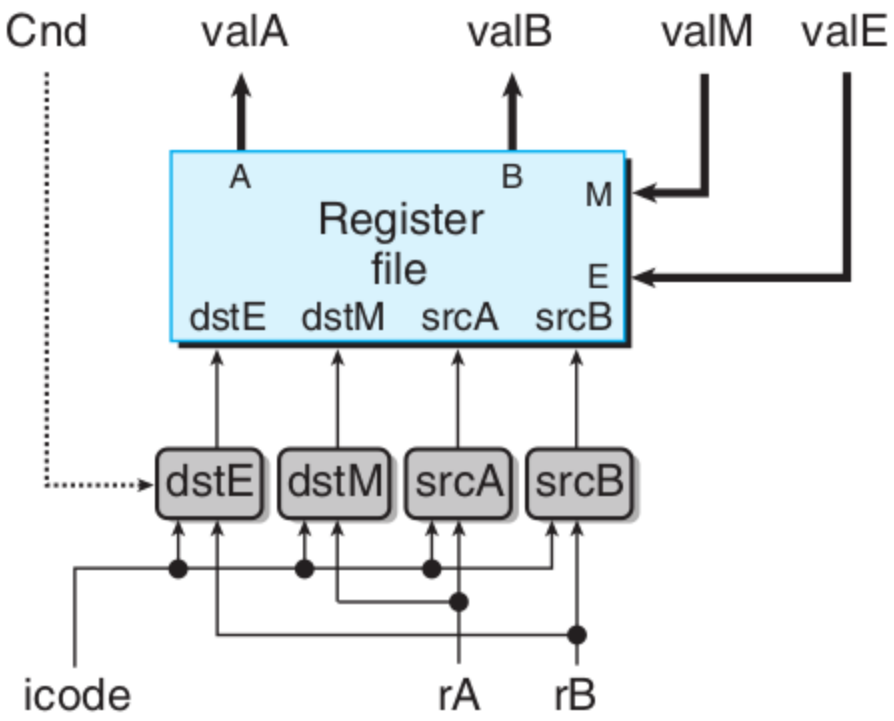
FETCH

- The fetch stage reads the bytes of an instruction from memory, using the program counter (PC) as the memory address.
- From the instruction it extracts the two 4-bit portions of the instruction specifier byte
- icode - the instruction code , ifun - the instruction function
- Now according to different operations it fetches registers rA and rB .
- Determines the value of next PC counter referred as valP.



DECODE

- it reads the registers designated by instruction fields rA and rB,referred as valA and valB , but for some instructions it reads register %rsp .

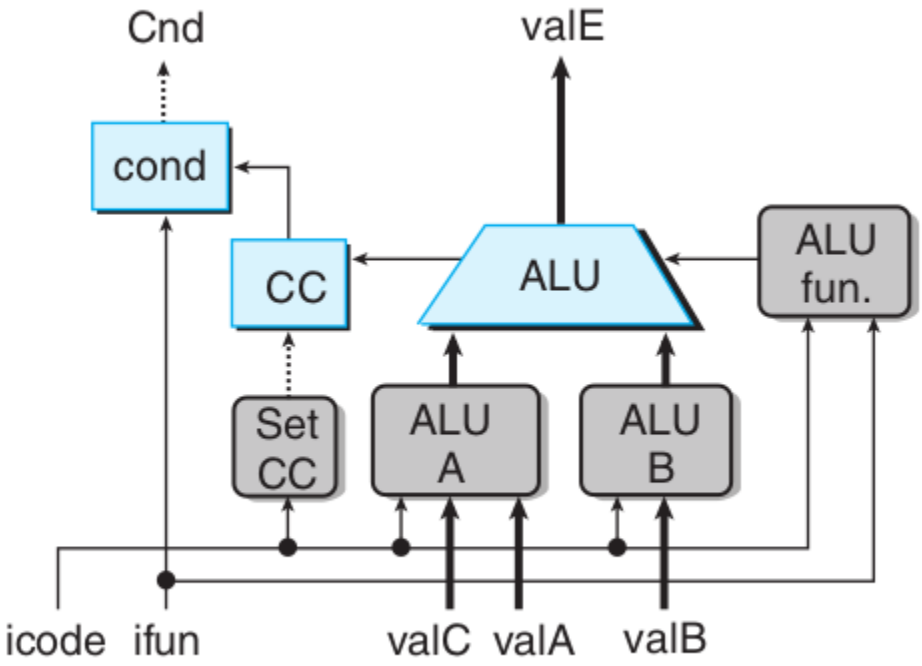


SEQ diagram of decode + write_back

The instruction fields are decoded to generate register identifiers for four addresses (two read and two write) used by the register file. The values read from the register file become the signals valA and valB. The two write-back values valE and valM serve as the data for the writes.

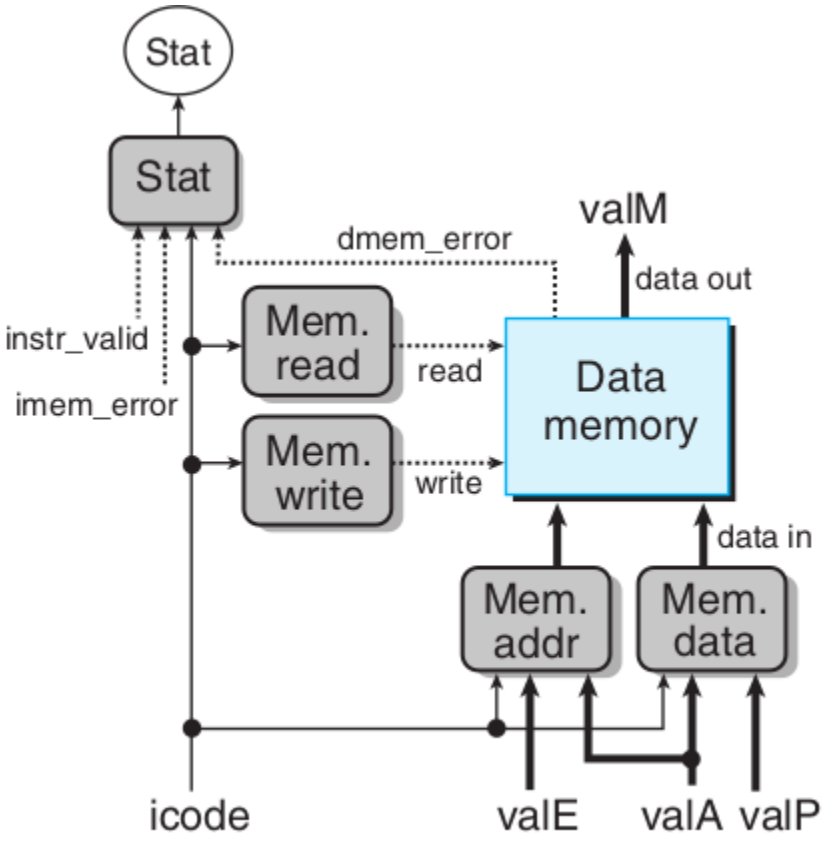
EXECUTE

- Arithmetic/logic unit (ALU) performs the operation specified by the instruction (according to the value of ifun) if required.
- Computes the effective address of a memory reference, or increments or decrements the stack pointer.
- The result is referred as valE.
- Sets the condition codes which check when to jump and when not to.



MEMORY

- May write data to memory, or it may read data from memory.
- Referred to as valM.

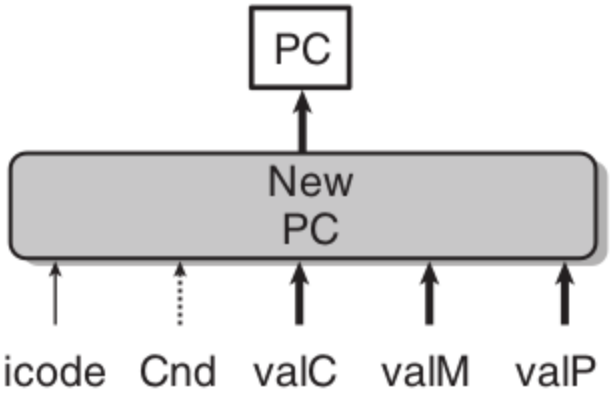


WRITE_BACK

- Writes results to the register file.

PC_UPDATE

- PC is set to the address of the next instruction.



SEQ PC update stage. The next value of the PC is selected from among the signals valC, valM,and valP, depending on the instruction code and the branch flag.

Working of all the stages for particular instruction

Stage	`rmmovq` rA, D(rB)	`mrmovq` D(rB), rA
Fetch	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1] valC $\leftarrow$ M_8[PC + 2] valP $\leftarrow$ PC + 10	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1] valC $\leftarrow$ M_8[PC + 2] valP $\leftarrow$ PC + 10
Decode	valA $\leftarrow$ R[rA] valB $\leftarrow$ R[rB]	valB $\leftarrow$ R[rB]
Execute	valE $\leftarrow$ valB + valC	valE $\leftarrow$ valB + valC
Memory	M_8[valE] $\leftarrow$ valA	valM $\leftarrow$ M_8[valE]
Write back		R[rA] $\leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

Stage	`OPq` rA, rB	`rrmovq` rA, rB	`irmovq` V, rB
Fetch	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1] valP $\leftarrow$ PC + 2	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1] valP $\leftarrow$ PC + 2	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1] valC $\leftarrow$ M_8[PC + 2] valP $\leftarrow$ PC + 10
Decode	valA $\leftarrow$ R[rA] valB $\leftarrow$ R[rB]	valA $\leftarrow$ R[rA]	
Execute	valE $\leftarrow$ valB OP valA Set CC	valE $\leftarrow$ 0 + valA	valE $\leftarrow$ 0 + valC
Memory			
Write back	R[rB] $\leftarrow$ valE	R[rB] $\leftarrow$ valE	R[rB] $\leftarrow$ valE
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP	PC $\leftarrow$ valP

Stage	`pushq` rA	`popq` rA
Fetch	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1]	icode:ifun $\leftarrow$ M_1[PC] rA:rB $\leftarrow$ M_1[PC + 1]
	valP $\leftarrow$ PC + 2	valP $\leftarrow$ PC + 2
Decode	valA $\leftarrow$ R[rA] valB $\leftarrow$ R[`%rsp`]	valA $\leftarrow$ R[`%rsp`] valB $\leftarrow$ R[`%rsp`]
Execute	valE $\leftarrow$ valB + (−8)	valE $\leftarrow$ valB + 8
Memory	M_8[valE] $\leftarrow$ valA	valM $\leftarrow$ M_8[valA]
Write back	R[`%rsp`] $\leftarrow$ valE	R[`%rsp`] $\leftarrow$ valE R[rA] $\leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

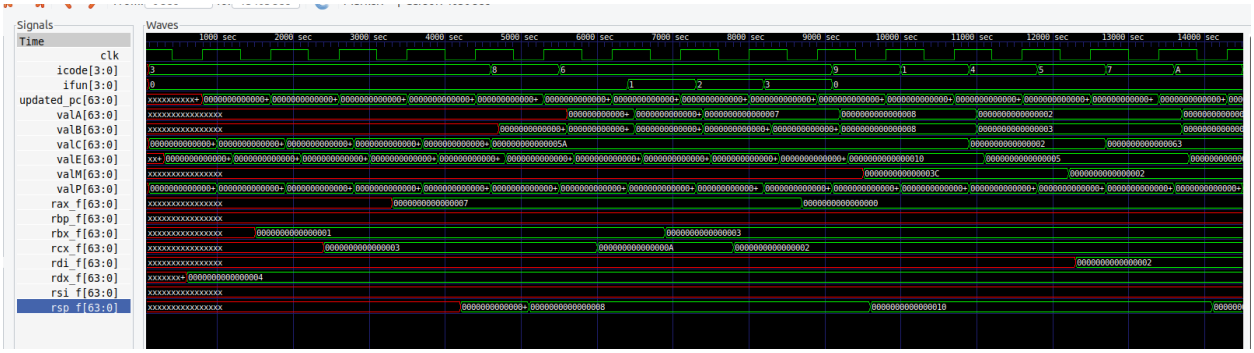
Stage	`jXX` Dest	`call` Dest	`ret`
Fetch	icode:ifun $\leftarrow$ M_1[PC]	icode:ifun $\leftarrow$ M_1[PC]	icode:ifun $\leftarrow$ M_1[PC]
	valC $\leftarrow$ M_8[PC + 1]	valC $\leftarrow$ M_8[PC + 1]	
	valP $\leftarrow$ PC + 9	valP $\leftarrow$ PC + 9	valP $\leftarrow$ PC + 1
Decode			valA $\leftarrow$ R[%rsp]
		valB $\leftarrow$ R[%rsp]	valB $\leftarrow$ R[%rsp]
Execute		valE $\leftarrow$ valB + (−8)	valE $\leftarrow$ valB + 8
	Cnd $\leftarrow$ Cond(CC, ifun)		
Memory		M_8[valE] $\leftarrow$ valP	valM $\leftarrow$ M_8[valA]
Write back		R[%rsp] $\leftarrow$ valE	R[%rsp] $\leftarrow$ valE
PC update	PC $\leftarrow$ Cnd ? valC : valP	PC $\leftarrow$ valC	PC $\leftarrow$ valM

Stage	`cmovXX` rA, rB
Fetch	icode:ifun $\leftarrow$ M_1[PC]
	rA:rB $\leftarrow$ M_1[PC + 1]
	valP $\leftarrow$ PC + 2
Decode	valA $\leftarrow$ R[rA]
Execute	valE $\leftarrow$ 0 + valA
Memory	
Write back	
	R[rB] $\leftarrow$ valE
PC update	PC $\leftarrow$ valP

Instructions to be written to run the processor -

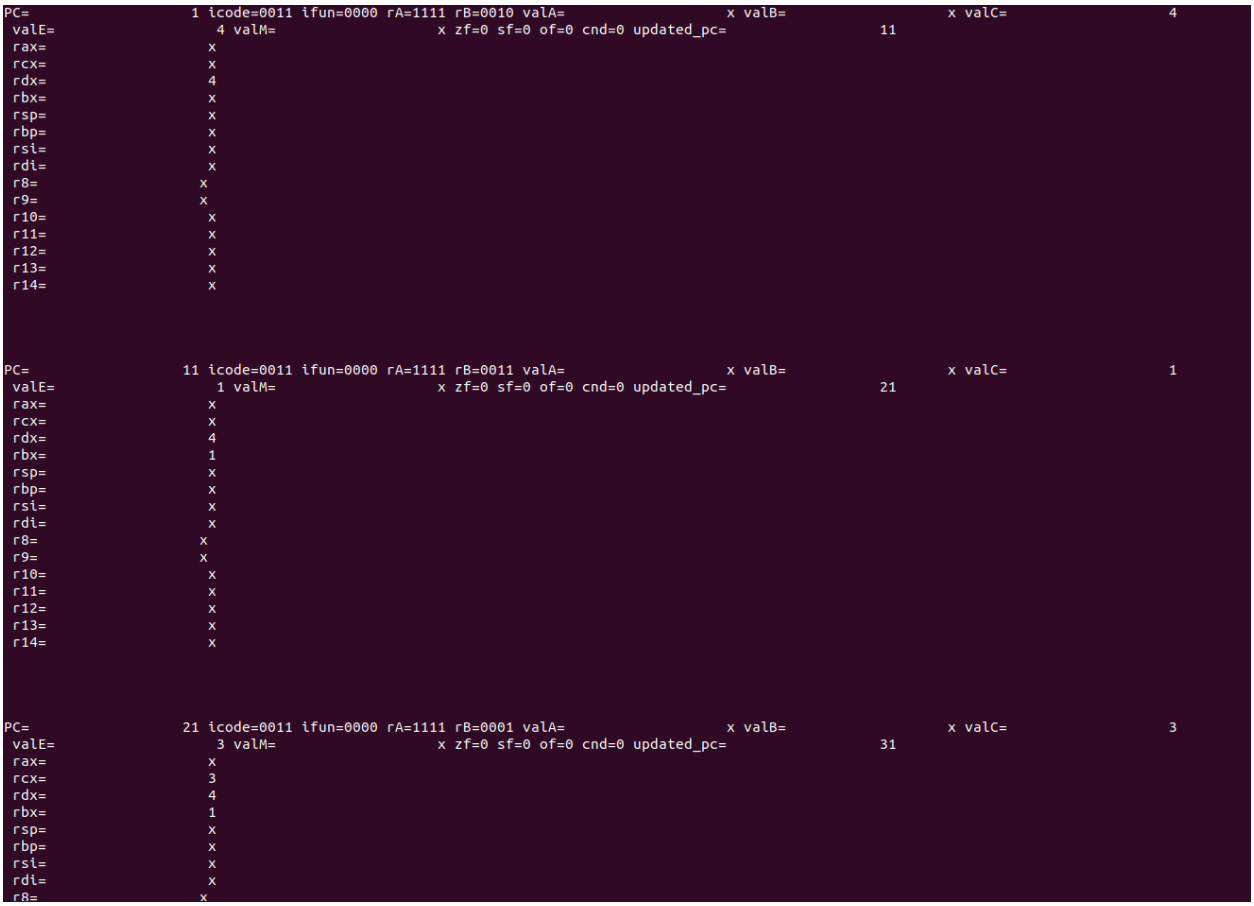
- **iverilog -o test_tb.vvp fetch.v decode.v execute.v memory.v write_back.v pc_update.v f_d_test.v**
- **vvp test_tb.vvp**
- **gtkwave**

gtkwave



we didn’t give any values to the registers by initial begin thus initially every register is in x state but as we give the commands to assign some values in registers then it gets changed from x to that assigned values.

Display results -



Instructions in instruction memory -

irmovq

irmovq

irmovq

irmovq

irmovq

call

nop

rmmovq

mrmovq

jmp

add

sub

and

xor

return

pushq

popq

halt