Introduction to Processor Architecture
EC2.204
Course Project

Project Report submitted on the project of Designing
Sequential and Pipeline Implementation of
Y86-64 Processor for the course IPA in Verilog.

Submitted by
Vaishnavi 2022102068
Damini 2023122005

Date Of Submission
7th March 2024

CONTENTS

Pipelined Implementation

5)Problem Approach
- Pipeline Implementation
- Steps involved

6)Changes to Sequential Implementation
- Rearranging Steps - SEQ+
- Pipelined Registers - PIPE -
- Rearranging and Relabelling Signals - PIPE

7)Implementation
- PC Selection & Fetch
  Implementation of Fetch and PC
- Decode and write back
  Implementation of Decode and write back
- Execute
  Implementation of Execute
- Memory
  Implementation of Memory
- Pipeline Control
  Implementation of Pipeline Control

8)Pipeline Implementation
- Introduction
- Implementation through Verilog of Processor
- Testing of Pipeline Implementation - 9 Test Cases
- GTK Waves of Pipeline Implementation

9)Challenges Encountered

10)Conclusion

Objective of the Project

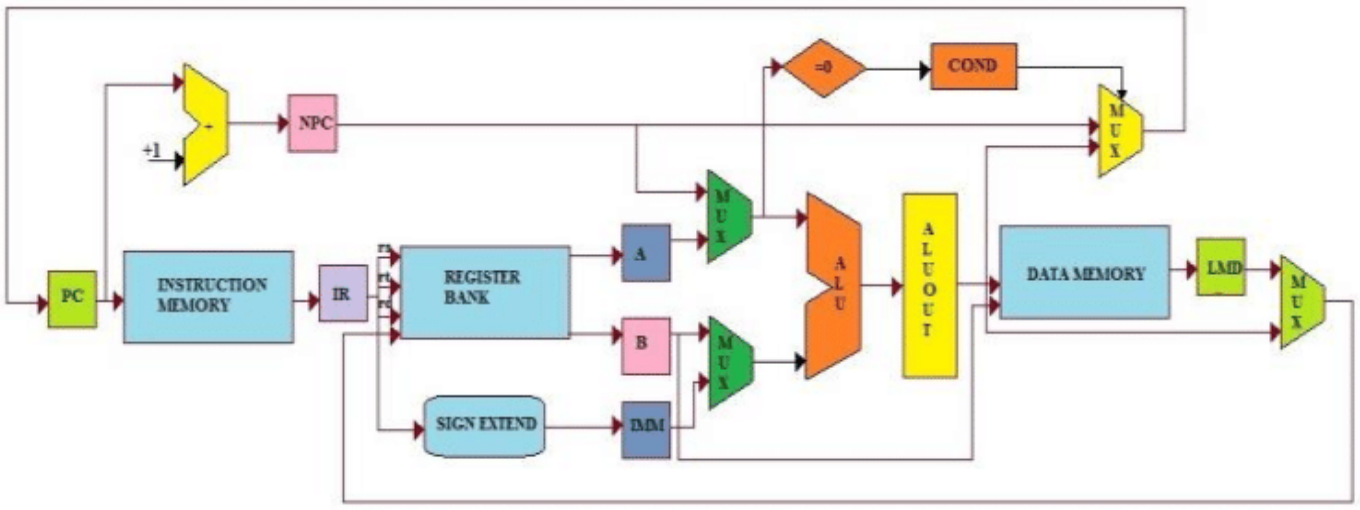
Aim:

To Design the Sequential and Pipeline Implementation of Y86 ISA that performs all stages such as Fetch , Decode , Execute , Memory , Write back along with PC update in Verilog.
To build ISA as a fully functioning Implementation for given instructions in Sequential and Pipelined Manner.

Tools/Softwares:

- Verilog

Block Diagram to be implemented:



The one given above is a basic idea which is to be implemented in a sequential and pipelined manner.

Sequential Implementation Problem Approach:

Sequential Implementation:

To Design the Sequential Implementation of Y86 ISA that performs all stages such as Fetch , Decode , Execute , Memory , Write back along with PC update.

On each clock cycle, SEQ performs all the steps required to process a complete instruction.

SEQUENCE OF STEPS BEING EXECUTED:

- Fetch
- Decode
- Execute
- Memory
- Write Back
- PC Update

H/w structure of SEQ

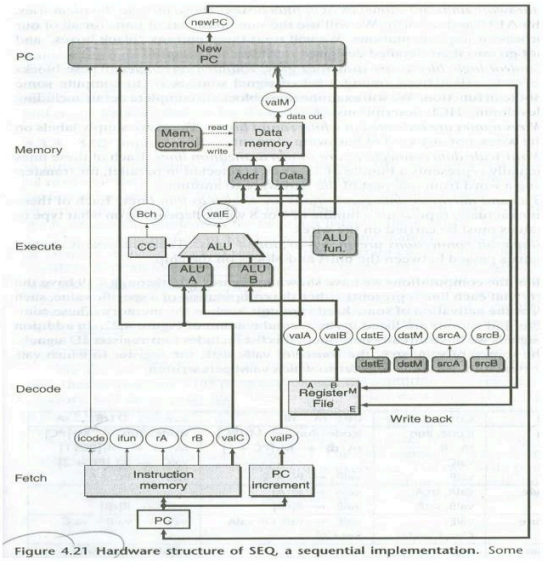


Figure 4.21 Hardware structure of SEQ, a sequential implementation. Some

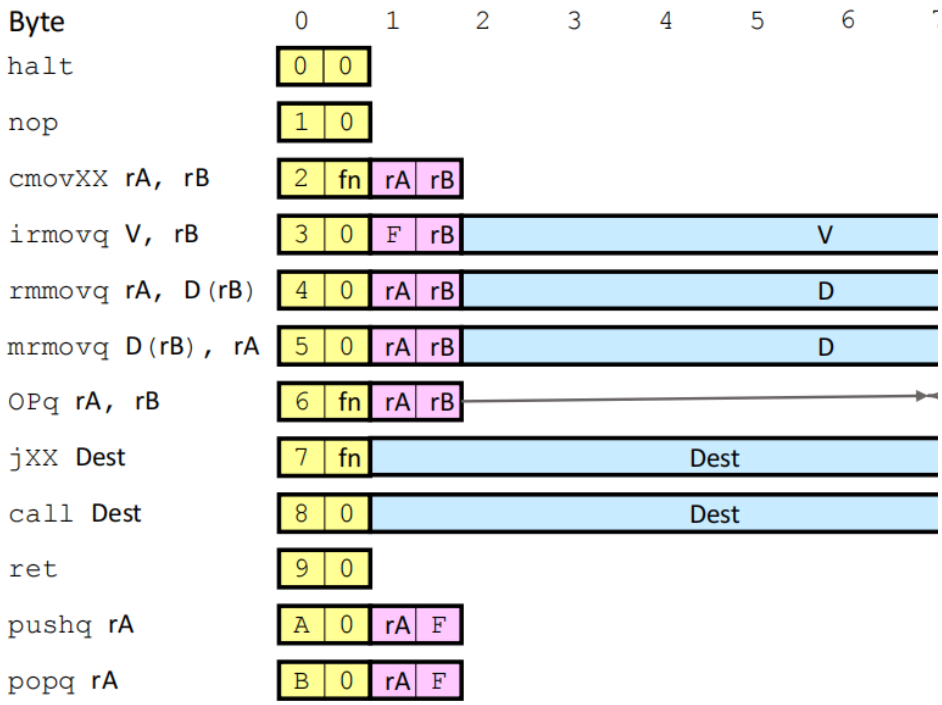
Implementation

In order to design the complete Implementation , the steps are to be designed individually and then combined to a single implementation.

The steps are :

1. Fetch - Read instruction from instruction memory.
2. Decode - Read program registers
3. Execute - Compute value or address
4. Memory - Read or write back data.
5. Write Back - Write program registers.
6. PC Update - Update the program counter.

The following instructions are to be implemented:



Instructions:

halt

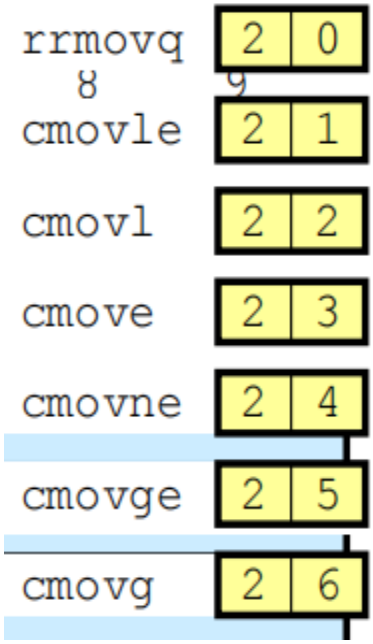
- halt stops instruction execution.
- It requires only a single byte

nop

- nop stands for no operation
- It requires a single byte

cmovxx

- cmovXX instruction represents seven different branch Instructions with different branch conditions. Branching is based on the setting of the condition codes by the arithmetic instructions.



- The seven instructions are rrmovq, cmovle, cmovl, cmove, cmovne, cmovge and cmovg

irmovq

- irmovq is immediate to register move instruction

rmmovq

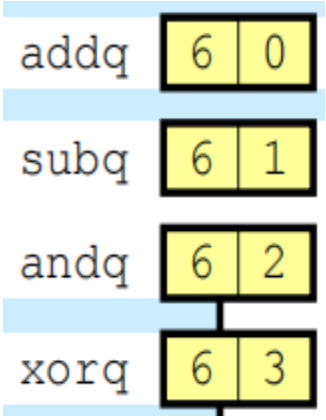
- rmmovq is register to memory move instruction.

mrmovq

- mrmovq is memory to register instruction

OPq

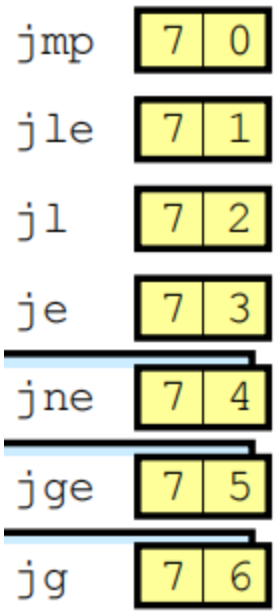
- OPq instruction performs four different arithmetic and logical operations.



Based on the function code , the operation gets selected. If 0 - addition , 1-Subtraction , 2- And , 3 -XOR is performed by the ALU

jXX

- jXX instructions represents seven different branch Instructions with different branch conditions.



Branches are taken according to the type of branch and the settings of the conditional codes.

call

● call instruction pushes the return address onto the stack and then jumps to the designation

ret

● ret instruction pops the return address from the stack and jumps to that location

pushq

● pushq instruction pushes 8 byte words onto the stack.
● pushing involves first decrementing the stack pointer by eight and then writing a word to the address given by the stack pointer.

popq

- popq instruction pops 8 byte words off the stack.
- popping involves reading the top word on the stack and then incrementing the stack pointer by eight.

Register Encoding:

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

The sequential implementation of the Y86-64 processor works according to the following table.

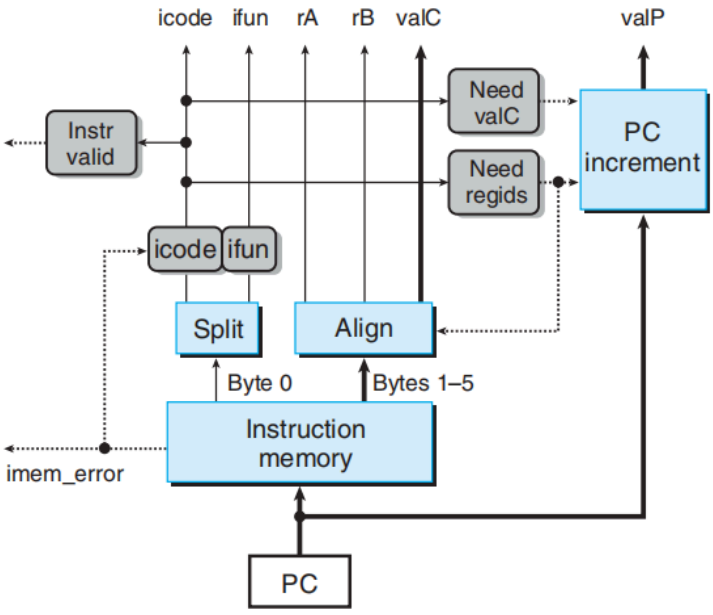
Stage	HALT	NOP	CMOV	IRMOVQ
Fch	icode:ifun ← M_1[PC] valP ← PC + 1	icode:ifun ← M_1[PC] valP ← PC + 1	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC + 10
Dec			valA ← R[rA]	
Exe	cpu.stat = HLT		valE ← valA Cnd ← Cond(CC,ifun)	valE ← valC
Mem				
WB			Cnd ? R[rB] ← valE	R[rB] ← valE
PC	PC ← 0	PC ← valP	PC ← valP	PC ← valP
Stage	**RMMOVQ**	**MRMOVQ**	**OPq**	**jXX**
Fch	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC + 10	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valC ← M_8[PC+2] valP ← PC + 10	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC + 9
Dec	valA ← R[rA] valB ← R[rB]	valB ← R[rB]	valA ← R[rA] valB ← R[rB]	
Exe	valE ← valB + valC	valE ← valB + valC	valE ← valB OP valA Set CC	Cnd ← Cond(CC,ifun)
Mem	M_8[valE] ← valA	valM ← M_8[valE]		
WB		R[rA] ← valM	R[rB] ← valE	
PC	PC ← valP	PC ← valP	PC ← valP	PC ← Cnd ? valC:valP

Stage	CALL	RET	PUSHQ	POPQ
Fch	icode:ifun ← M_1[PC] valC ← M_8[PC+1] valP ← PC + 9	icode:ifun ← M_1[PC] valP ← PC + 1	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2	icode:ifun ← M_1[PC] rA:rB ← M_1[PC+1] valP ← PC + 2
Dec	valB ← R[RSP]	valA ← R[RSP] valB ← R[RSP]	valA ← R[rA] valB ← R[RSP]	valA ← R[RSP] valB ← R[RSP]
Exe	valE ← valB - 8	valE ← valB + 8	valE ← valB - 8	valE ← valB + 8
Mem	M_8[valE] ← valP	valM ← M_8[valA]	M_8[valE] ← valA	valM ← M_8[valA]
WB	R[RSP] ← valE	R[RSP] ← valE	R[RSP] ← valE	R[RSP] ← valE R[rA] ← valM
PC	PC ← valC	PC ← valM	PC ← valP	PC ← valP

Steps:

Fetch:

● This stage reads the bytes of an instruction from memory using Program Counter (PC) as the memory address.



● Computed Values in this stage are -
icode – Instruction Code
ifun – Function Code

rA – Inst. Register A
rB – Inst. Register B
valC – Instruction Constant
valP – Increment Program Counter

Implementation:

halt icode = 0, ifun = 0, valP = PC+64'd1
nop icode = 1, ifun = 0, valP = PC+64'd2
cmovXX icode = 2,
ifun for-rrmovq =0,cmovle=1,cmovl = 2,cmove=3
,cmovne =4,cmovge = 5,cmovg = 6
valP = PC+64'd2
Irmovq icode = 3, ifun = 0, valP = PC+64'd10
rmmovq icode = 4, ifun = 0, valP = PC+64'd10
mrmovq icode = 5, ifun = 0, valP = PC+64'd10
OPq icode = 6, ifun for - addq =0,subq=1,andq = 2
xorq=3
valP = PC+64'd2.
jXX icode = 7, ifun for - jmp = 0, jle = 1, jl = 2, je = 3,
jne = 4,jge = 5, jg = 6
valP = PC+64'd9
call icode = 8, ifun = 0, valP = PC+64'd9
ret icode = 9, ifun = 0, valP = PC+64'd1
pushq icode = 10, ifun = 0, valP = PC+64'd2
popq icode = 11, ifun = 0, valP = PC+64'd2.

Verilog Implementation

Code :

File Name: fetch.v

```
module fetch(clk, PC, instr, icode, ifun, rA, rB, valC, valP, ins_mem_error, valid_instr,halt);
//inputs
input clk;                  // Clock input
input [63:0] PC;            // Program Counter
input [0:79] instr;         // 10 bytes are required instruction encodings

//outputs
output reg [3:0] icode;       //Instruction code
output reg [3:0] ifun;        //Function code
output reg [3:0] rA;          //Register A(src)
output reg [3:0] rB;          //Register B(dest)
output reg [63:0] valC;       // Immediate value
output reg [63:0] valP;       // Next program counter
output reg ins_mem_error = 0;         // Instruction memory error flag
output reg valid_instr = 1 ;      // Valid instruction flag
output reg halt = 0;              //halt the program

always @(*)
begin
   if(PC>20480)
    begin
        ins_mem_error=1;
    end
end

always @(*)
begin
    icode = instr[0:3];
    ifun = instr[4:7];

    //halt
    if(icode == 4'b0000)
    begin
        halt = 1;
        valP = PC + 64'd1;
    end
```

```
    //nop
    else if(icode == 4'b0001)
    begin
        valP = PC + 64'd1;
    end

    //cmovq
    else if(icode == 4'b0010)
    begin
        rA = instr[8:11];
        rB = instr[12:15];
```

```
49          valP = PC + 64'd2;
50      end
51
52      //irmovq
53      else if(icode == 4'b0011)
54      begin
55          rA = instr[8:11];
56          rB = instr[12:15];
57          valC = instr[16:79]; //8 bytes for constant value
58          valP = PC + 64'd10;
59      end
60
61      //rmmovq
62      else if(icode == 4'b0100)
63      begin
64          rA = instr[8:11];
65          rB = instr[12:15];
66          valC = instr[16:79]; //8 bytes for constant value
67          valP = PC + 64'd10;
68      end
69
70      //mrmovq
71      else if(icode == 4'b0101)
72      begin
73          rA = instr[8:11];
```

```
74          rB = instr[12:15];
75          valC = instr[16:79]; //8 bytes for constant value
76          valP = PC + 64'd10;
77      end
78
79      //opq
80      else if(icode == 4'b0110)
81      begin
82          rA = instr[8:11];
83          rB = instr[12:15];
84          valP = PC + 64'd2;
85      end
86
87      //jxx
88      else if(icode == 4'b0111)
89      begin
90          valC = instr[8:71];
91          valP = PC + 64'd9;
92      end
93
94      //call
95      else if(icode == 4'b1000)
96      begin
97          valC = instr[8:71];
98          valP = PC + 64'd9;
99      end
100
101     //ret
102     else if(icode == 4'b1001)
103     begin
104         valP = PC + 64'd1;
105     end
```

```
107     //pushq
108     else if(icode == 4'b1010)
```

```
109     begin
110         rA = instr[8:11];
111         rB = instr[12:15];
112         valP = PC + 64'd2;
113     end
114
115     //popq
116     else if(icode == 4'b1011)
117     begin
118         rA = instr[8:11];
119         rB = instr[12:15];
120         valP = PC + 64'd2;
121     end
122
123     else
124     begin
125         valid_instr = 1'b0;
126     end
127
128 end
129
130 endmodule
```

Testing:

File Name: fetch_test.v

```
1  `include "fetch.v"
2
3  module fetch_test;
4    reg clk;
5    reg [63:0] PC;
6
7    wire [3:0] icode;
8    wire [3:0] ifun;
9    wire [3:0] rA;
10   wire [3:0] rB;
11   wire [63:0] valC;
12   wire [63:0] valP;
13   wire ins_mem_error, valid_instr;
14   wire halt;
15   reg [7:0] instr_mem[0:20480];
16   reg [0:79] instr;
17
18 fetch fetch(
19   .clk(clk),
20   .PC(PC),
21   .icode(icode),
22   .ifun(ifun),
23   .rA(rA),
24   .rB(rB),
25   .valC(valC),
26   .valP(valP),
27   .ins_mem_error(ins_mem_error),
28   .valid_instr(valid_instr),
```

```
    .instr(instr),.halt(halt)
);

always@(PC) begin
  instr={
    instr_mem[PC],
    instr_mem[PC+1],
    instr_mem[PC+2],
    instr_mem[PC+3],
    instr_mem[PC+4],
    instr_mem[PC+5],
    instr_mem[PC+6],
    instr_mem[PC+7],
    instr_mem[PC+8],
    instr_mem[PC+9]
  };
end

always @(icode) begin
  if(halt == 1)
    $finish;
end

always #10 clk = ~clk;
always @(posedge clk) PC<=valP;

initial begin
$dumpfile("fetch_test.vcd");
$dumpvars(0,fetch_test);
$monitor("clk=%d PC=%d icode=%b ifun=%b rA=%b rB=%b,valC=%d,valP=%d\n",clk,PC,icode,ifun,rA,rB,valC,valP);
clk=1;
PC=64'd0;
//////////////////////////////Test Case 1//////////////////////////
//Reference for 1.txt

//opq , halt along with irmovq

//irmovq $0x100, %rbx
instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000001;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000000;
instr_mem[9]=8'b00000000;
```

```
  };
end

always @(icode) begin
  if(halt == 1)
    $finish;
end

always #10 clk = ~clk;
always @(posedge clk) PC<=valP;

initial begin
$dumpfile("fetch_test.vcd");
$dumpvars(0,fetch_test);
$monitor("clk=%d PC=%d icode=%b ifun=%b rA=%b rB=%b,valC=%d,valP=%d\n",clk,PC,icode,ifun,rA,rB,valC,valP);
clk=1;
PC=64'd0;
```

```
62  ////////////////////////////Test Case 1/////////////////////////
63  //Reference for 1.txt
64
65  //opq , halt along with irmovq
66
67  //irmovq $0x100, %rbx
68  instr_mem[0]=8'b00110000;
69  instr_mem[1]=8'b11110011;
70  instr_mem[2]=8'b00000000;
71  instr_mem[3]=8'b00000001;
72  instr_mem[4]=8'b00000000;
73  instr_mem[5]=8'b00000000;
74  instr_mem[6]=8'b00000000;
75  instr_mem[7]=8'b00000000;
76  instr_mem[8]=8'b00000000;
77  instr_mem[9]=8'b00000000;
78
79  //irmovq $0x200, %dx
80  instr_mem[10]=8'b00110000;
81  instr_mem[11]=8'b11110010;
82  instr_mem[12]=8'b00000000;
83  instr_mem[13]=8'b00000010;
84  instr_mem[14]=8'b00000000;
85  instr_mem[15]=8'b00000000;
86  instr_mem[16]=8'b00000000;
87  instr_mem[17]=8'b00000000;
88  instr_mem[18]=8'b00000000;
89  instr_mem[19]=8'b00000000;
90
91  //addq %rdx, %rbx
92  instr_mem[20]=8'b01100000;
```

```
92  instr_mem[20]=8'b01100000;
93  instr_mem[21]=8'b00100011;
94
95  //halt
96  instr_mem[22]=8'b00000000;
97
98  end
99  endmodule
```

Output:

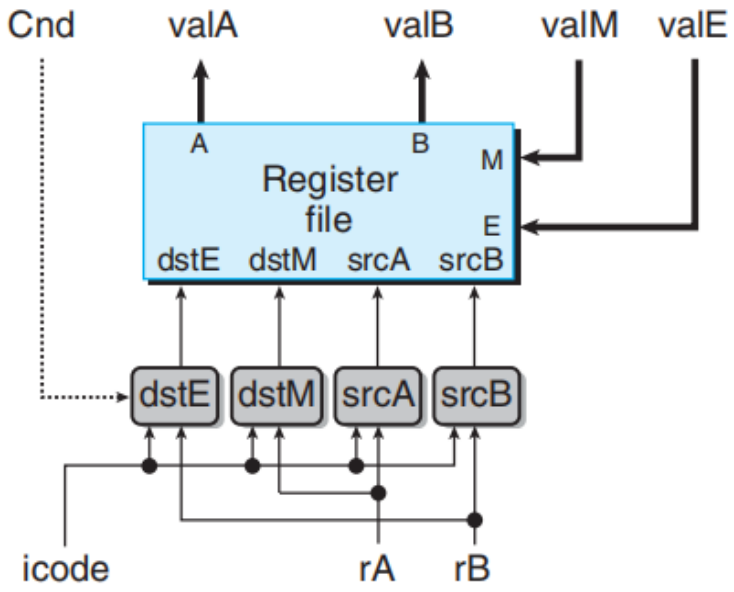
```
clk=1 PC=                 x icode=xxxx ifun=xxxx rA=1111 rB=0011,valC=     281474976710656,valP=                10

clk=0 PC=                 x icode=xxxx ifun=xxxx rA=1111 rB=0011,valC=     281474976710656,valP=                10

clk=1 PC=                10 icode=0011 ifun=0000 rA=1111 rB=0010,valC=     562949953421312,valP=                20

clk=0 PC=                10 icode=0011 ifun=0000 rA=1111 rB=0010,valC=     562949953421312,valP=                20

clk=1 PC=                20 icode=0110 ifun=0000 rA=0010 rB=0011,valC=     562949953421312,valP=                22

clk=0 PC=                20 icode=0110 ifun=0000 rA=0010 rB=0011,valC=     562949953421312,valP=                22

fetch_test.v:50: $finish called at 60 (1s)
clk=1 PC=                22 icode=0000 ifun=0000 rA=0010 rB=0011,valC=     562949953421312,valP=                23
```

Decode and Write back:

● Decode reads the registers designated by rA and rB and sometimes reads register %rsp.

● Write-Back write program registers.

● During the decode stage,both operands are read and supplied to the ALU in the execute stage



● The combined block of decode and write back takes **rA**, **rB**, **icode**, **Cnd**, **valM** and **valE** as inputs and gives **valA** and **valB** as outputs.

Implementation:

● For **decode** , **valA** and **valB** are computed according to the identified instructions.

OPq	Decode	valA ← R[rA]	Read operand A
rmmovq	Decode	valA ← R[rA]	Read operand A
mrmovq	Decode		
irmovq	Decode		
pushq	Decode	valA ← R[rA]	Read operand A
popq	Decode	valA ← R[%rsp]	Read stack pointer
cmovXX	Decode	valA ← R[rA]	Read operand A
jXX	Decode		
call	Decode		
ret	Decode	valA ← R[%rsp]	Read stack pointer

- For **writeback** , the values are written back into the register by identify the instructions and writeback into the respective registers.

OPq	Write Back	R[rB] ← valE	Write back result
rmmovq	Write Back		
mrmovq	Write Back		
irmovq	Write Back	R[rB] ← valE	Write back result
pushq	Write Back	R[%rsp] ← valE	Update stack pointer
popq	Write Back	R[%rsp] ← valE	Update stack pointer
cmovXX	Write Back	R[rB] ← valE	Write back result
jXX	Write Back		
call	Write Back	R[%rsp] ← valE	Update stack pointer
ret	Write Back	R[%rsp] ← valE	Update stack pointer

- Computed Values in this stage are
  valA - Register Value A
  valB - Register Value B

Verilog Implementation

Code :

File Name: decode_writeback.v

```
module decode_writeback(clk, icode, cnd, rA, rB, valA, valB, valE, valM,
                        reg_0,reg_1,reg_2, reg_3, reg_4, reg_5, reg_6, reg_7,
                        reg_8, reg_9, reg_10, reg_11, reg_12, reg_13, reg_14);
//inputs
input clk, cnd;
input [3:0] icode;
input [3:0] rA,rB;
input[63:0] valE,valM;

//outputs
output reg[63:0] valA, valB;
output reg[63:0] reg_0,reg_1,reg_2, reg_3, reg_4, reg_5, reg_6, reg_7,reg_8, reg_9, reg_10, reg_11, reg_12, reg_13, reg_14;

reg [63:0] reg_memory[0:14]; //15 program registers

initial
begin
    reg_memory[0] = 64'd0;
    reg_memory[1] = 64'd1;
    reg_memory[2] = 64'd2;
    reg_memory[3] = 64'd3;
    reg_memory[4] = 64'd4;
    reg_memory[5] = 64'd5;
    reg_memory[6] = 64'd6;
    reg_memory[7] = 64'd7;
    reg_memory[8] = 64'd8;
    reg_memory[9] = 64'd9;
    reg_memory[10] = 64'd10;
    reg_memory[11] = 64'd11;
    reg_memory[12] = 64'd12;
    reg_memory[13] = 64'd13;
    reg_memory[14] = 64'd14;
end

//////////////decode////////////////
always@(*)
begin
```

```
  //cmovq
  if(icode == 4'b0010)
  begin
    valA = reg_memory[rA];
    valB = 64'b0;
  end

  //irmovq
  if(icode == 4'b0011)
  begin
    valA = 64'b0;
    valB = 64'b0;
  end

  //rmmovq
  else if(icode == 4'b0100)
  begin
    valA = reg_memory[rA];
    valB = reg_memory[rB];
  end

  //mrmovq
  else if(icode == 4'b0101)
  begin
    valB = reg_memory[rB];
```

```
63    end
64
65    //opq
66    else if(icode == 4'b0110)
67    begin
68      valA = reg_memory[rA];
69      valB = reg_memory[rB];
70    end
71
72    //call
73    else if(icode == 4'b1000)
```

```
74    begin
75      valB = reg_memory[4];
76    end
77
78    //ret
79    else if(icode == 4'b1001)
80    begin
81      valA = reg_memory[4];
82      valB = reg_memory[4];
83    end
84
85    //pushq
86    else if(icode == 4'b1010)
87    begin
88      valA = reg_memory[rA];
89      valB = reg_memory[4];
90     end
91
92    //popq
93    else if(icode == 4'b1011)
94    begin
95      valA = reg_memory[4];
96      valB = reg_memory[4];
97    end
98  end
99
100
101 ///////////////write back/////////////////
102
103 always@(posedge clk) begin
104   //cmovxx
105   if(icode==4'b0010)
106   begin
107     if(cnd)
108       reg_memory[rB]=valE;
109   end
```

```
110
111   //irmovq
112   else if(icode==4'b0011)
113   begin
114     reg_memory[rB]=valE;
115   end
116
117   //mrmovq
118   else if(icode==4'b0101)
119   begin
120     reg_memory[rA] = valM;
121   end
122
123   //OPq
124   else if(icode==4'b0110)
125   begin
126     reg_memory[rB] = valE;
127   end
128
129   //call
130   else if(icode==4'b1000)
131   begin
132     reg_memory[4] = valE;
133   end
134
```

```
135     //ret
136     else if(icode==4'b1001)
137     begin
138       reg_memory[4] = valE;
139     end
140
141     //pushq
142     else if(icode==4'b1010)
143     begin
144       reg_memory[4] = valE;
145     end
```

```
146
147     //popq
148     else if(icode==4'b1011)
149     begin
150       reg_memory[4] = valE;
151       reg_memory[rA] = valM;
152     end
153
154     reg_0 <= reg_memory[0];
155     reg_1 <= reg_memory[1];
156     reg_2 <= reg_memory[2];
157     reg_3 <= reg_memory[3];
158     reg_4 <= reg_memory[4];
159     reg_5 <= reg_memory[5];
160     reg_6 <= reg_memory[6];
161     reg_7 <= reg_memory[7];
162     reg_8 <= reg_memory[8];
163     reg_9 <= reg_memory[9];
164     reg_10 <= reg_memory[10];
165     reg_11 <= reg_memory[11];
166     reg_12 <= reg_memory[12];
167     reg_13 <= reg_memory[13];
168     reg_14 <= reg_memory[14];
169   end
170   endmodule
```

Testing:

File Name: decode_wb_test.v

```
1   `include "fetch.v"
2   `include "decode_writeback.v"
3
4   module decode_wb_test;
5     reg clk;
6     reg [63:0] PC;
7     wire [3:0] icode;
8     wire [3:0] ifun;
9     wire [3:0] rA;
10    wire [3:0] rB;
11    wire [63:0] valA, valB, valC, valM,valE;
12    wire [63:0] valP;
13    wire ins_mem_error, valid_instr;
14    reg [7:0] instr_mem[0:20480];
15    reg [0:79] instr;
16    reg cnd;
17
18    wire [63:0] reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14;
19
20
21  fetch func1(
22    .clk(clk),
23    .PC(PC),
24    .instr(instr),
25    .icode(icode),
26    .ifun(ifun),
27    .rA(rA),
28    .rB(rB),
29    .valC(valC),
30    .valP(valP),
```

```
    .ins_mem_error(ins_mem_error),
    .valid_instr(valid_instr),.halt(halt)

  );

  decode_writeback func2(
    .clk(clk),.icode(icode),.rA(rA),.rB(rB),.valA(valA),.valB(valB),.valM(valM),.valE(valE),.cnd(cnd),
    .reg_0(reg_0),.reg_1(reg_1),.reg_2(reg_2),.reg_3(reg_3),.reg_4(reg_4),
    .reg_5(reg_5),.reg_6(reg_6),.reg_7(reg_7),.reg_8(reg_8),.reg_9(reg_9),
    .reg_10(reg_10),.reg_11(reg_11),.reg_12(reg_12),.reg_13(reg_13),.reg_14(reg_14)
  );

  // always@(PC) begin
  //   instr={
  //     instr_mem[PC],
  //     instr_mem[PC+1],
  //     instr_mem[PC+9],
  //     instr_mem[PC+8],
  //     instr_mem[PC+7],
  //     instr_mem[PC+6],
  //     instr_mem[PC+5],
  //     instr_mem[PC+4],
  //     instr_mem[PC+3],
  //     instr_mem[PC+2]
  //   };
  // end

  always@(PC) begin
    instr={
      instr_mem[PC],
      instr_mem[PC+1],
      instr_mem[PC+2],
      instr_mem[PC+3],
      instr_mem[PC+4],
      instr_mem[PC+5],
      instr_mem[PC+6],
      instr_mem[PC+7],
      instr_mem[PC+8],
      instr_mem[PC+9]
    };
  end

  always @(icode) begin
    if(halt==1)
      $finish;
  end

    always #10 clk = ~clk;
```

```
    always @(posedge clk) PC<=valP;

  initial begin
    $dumpfile("decode_wb_test.vcd");
    $dumpvars(0,decode_wb_test);
    $monitor("clk=%d icode=%b ifun=%b rA=%b rB=%b valA=%d valB=%d valE=%d valM=%d\n",clk,icode,ifun,rA,rB,valA,valB,valE,valM);
    clk=1;
    PC=64'd0;

  ////////////////////////////Test Case 1/////////////////////////
  //Reference for 1.txt

  //opq , halt along with irmovq
```

```
93   //irmovq $0x100, %rbx
94   instr_mem[0]=8'b00110000;
95   instr_mem[1]=8'b11110011;
96   instr_mem[2]=8'b00000000;
97   instr_mem[3]=8'b00000001;
98   instr_mem[4]=8'b00000000;
99   instr_mem[5]=8'b00000000;
100  instr_mem[6]=8'b00000000;
101  instr_mem[7]=8'b00000000;
102  instr_mem[8]=8'b00000000;
103  instr_mem[9]=8'b00000000;
104
105  //irmovq $0x200, %dx
106  instr_mem[10]=8'b00110000;
107  instr_mem[11]=8'b11110010;
108  instr_mem[12]=8'b00000000;
109  instr_mem[13]=8'b00000010;
110  instr_mem[14]=8'b00000000;
111  instr_mem[15]=8'b00000000;
112  instr_mem[16]=8'b00000000;
113  instr_mem[17]=8'b00000000;
114  instr_mem[18]=8'b00000000;
115  instr_mem[19]=8'b00000000;
116
117  //addq %rdx, %rbx
118  instr_mem[20]=8'b01100000;
119  instr_mem[21]=8'b00100011;
120
121  //halt
122  instr_mem[22]=8'b00000000;
123
124  end
125  endmodule
```

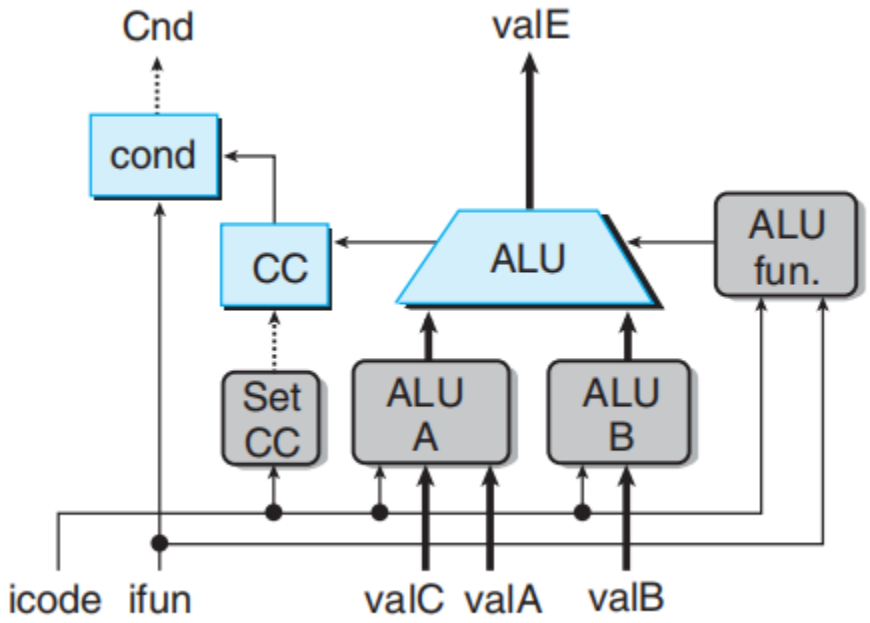
Output:

```
clk=1 icode=xxxx ifun=xxxx rA=1111 rB=0011 valA=                    0 valB=                    0 valE=                    z valM=                    z

clk=0 icode=xxxx ifun=xxxx rA=1111 rB=0011 valA=                    0 valB=                    0 valE=                    z valM=                    z

clk=1 icode=0011 ifun=0000 rA=1111 rB=0010 valA=                    0 valB=                    0 valE=                    z valM=                    z

clk=0 icode=0011 ifun=0000 rA=1111 rB=0010 valA=                    0 valB=                    0 valE=                    z valM=                    z

clk=1 icode=0110 ifun=0000 rA=0010 rB=0011 valA=                    z valB=                    3 valE=                    z valM=                    z

clk=0 icode=0110 ifun=0000 rA=0010 rB=0011 valA=                    z valB=                    3 valE=                    z valM=                    z

decode_wb_test.v:75: $finish called at 60 (1s)
clk=1 icode=0000 ifun=0000 rA=0010 rB=0011 valA=                    z valB=                    z valE=                    z valM=                    z
```

Execute:

●The execute stage includes the arithmetic/logic unit (ALU).

This unit performs the operation add, subtract, and, or Exclusive-Or on inputs aluA and aluB received.



●This stage performs the following either:

a) ALU performs the operation specified by ifun and computes effective address of memory

b) Increments (or) Decrements the stack pointer.

Condition Codes

●Carry Flag (CF) - This is used to detect overflow for unsigned operations .

This is determined by the most recent operation generated by carry out of the most significant bit.

●Zero Flag (ZF) - This flag comes into effect when the most recent operation yields zero.

●Sign Flag (SF) - This flag comes into effect when the most recent operation yields a negative value.
●Overflow Flag (OF) - This flag comes into effect if there is a two's complement overflow - either positive or negative

➔ All unary and binary arithmetic & logical operations (except leaq) set the single bit condition codes.
➔ In case of logical operations, the carry and overflow flags are set to zero.
➔ In case of shift operations, the carry flag is set the last shifted out, while the overflow flag is set to zero.
➔ INC and DEC instructions set the overflow flag and zero flag but leave the carry flag unchanged.

Implementation:

●The execute block takes **icode**, **ifun**, **valC**, **valA** and **valB** as inputs and gives the outputs as **Cnd** and **valE**. The execute selects the instructions according to the values of **icode** and **ifun**.
●The condition codes are also set for the instructions jXX and CMOV and **valE** is computed using ALU.

●Computed Values in this stage are -
valE - ALU Result
Cnd - To determine whether to take a branch or not

Verilog Implementation

Code :

File Name: execute.v

```
`include "alu.v"

module execute(cnd, icode, ifun, valA,valB, valC, valE, cond_c_in,zero_f,sign_f,overflow_f);
//inputs
input [3:0] icode, ifun;
input [63:0] valC, valA, valB;
input [2:0] cond_c_in;

//outputs
output reg [63:0] valE;
output reg cnd;
output reg zero_f,sign_f,overflow_f; //condition codes

wire overflow;
wire overflow1;

//input condition codes
wire z_f, s_f, o_f;
assign z_f = cond_c_in[0];
assign s_f = cond_c_in[1];
assign o_f = cond_c_in[2];

wire [63:0] valE_cmov, valE_CB, valE_opq, valE_inc, valE_dec;

always @* begin
    cnd=0;
    //for conditional instructions
    if (icode == 4'b0010 || icode == 4'b0111) begin
        if (ifun == 4'b0000) begin
            cnd = 1; // unconditional
        end

        // less than equal to
        else if (ifun == 4'b0001) begin
            cnd = (o_f ^ s_f) || z_f;
        end

        // less than
        else if (ifun == 4'b0010) begin
            cnd = o_f ^ s_f;
        end

        // equal to
        else if (ifun == 4'b0011) begin
            cnd = z_f;
        end
```

```
48          // not equal to
49          else if (ifun == 4'b0100) begin
50              cnd = ~z_f;
51          end
52
53          //greater than equal to
54          else if (ifun ==4'b0101) begin
55              cnd = ~(s_f ^ o_f);
56          end
57
58          //greater than
59          else if (ifun == 4'b0110) begin
60              cnd = ~(s_f ^ o_f) & ~z_f;
61          end
62      end
63  end
64
65
66  //ALU for executing instructions
67  alu alu1( 2'b0, valA, valB,valE_cmov, overflow1);
68  alu alu2(ifun[1:0], valA, valB,valE_opq, overflow);
69  alu alu3( 2'b0, valC, valB,valE_CB, overflow1);
70  alu alu4( 2'b0, 64'd8, valB,valE_inc, overflow1);
71  alu alu5( 2'b1, valB,64'd8,valE_dec, overflow1);
72
```

```
73  always @* begin
74      // cmovx
75      if (icode == 4'b0010) begin
76          valE = valE_cmov;
77      end
78
79      // irmovq, rmmovq, mrmovq
80      else if (icode == 4'b0011 || icode == 4'b0100 || icode == 4'b0101) begin
81          valE = valE_CB;
82      end
83
84      // opq
85      else if (icode == 4'b0110) begin
86          valE = valE_opq;
87          //setting condition codes
88          overflow_f <= overflow;
89          sign_f <= valE[63];
90          zero_f <= valE ? 0 : 1;
91      end
92
93      // call and pushq
94      else if (icode == 4'b1000 || icode == 4'b1010) begin
95          valE = valE_dec;
96      end
97
98      // return and popq
99      else if (icode == 4'b1001 || icode == 4'b1011) begin
100         valE = valE_inc;
101     end
```

```
        else begin
            valE = 0;
        end
    end

    endmodule
```

Testing:

File Name: execute_test.v

```
`include "fetch.v"
`include "decode_writeback.v"
`include "execute.v"
`include "pc_update.v"

module execute_test;
reg clk;
reg [63:0] PC;
wire [63:0] PC_next;
reg [0:79] instr;
reg [2:0] cond_c_in;
reg [7:0] instr_mem[0:20480];
wire cnd,halt;
wire valid_instr, ins_mem_error;
wire [3:0] icode,ifun,rA,rB;
wire [63:0] valA,valB,valC,valE,valM,valP;
wire [63:0] reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14;
wire zero_f,sign_f,overflow_f;

always@(PC) begin
  instr={
    instr_mem[PC],
    instr_mem[PC+1],
    instr_mem[PC+2],
    instr_mem[PC+3],
    instr_mem[PC+4],
    instr_mem[PC+5],
    instr_mem[PC+6],
    instr_mem[PC+7],
    instr_mem[PC+8],
    instr_mem[PC+9]
  };
end
always @(*) begin
    if(halt == 1)
      $finish;
end

always #10 clk = ~clk;

always @* PC = PC_next;

always @(posedge clk) begin
    if(icode==6)
    cond_c_in[0] = zero_f;
    cond_c_in[1] = sign_f;
    cond_c_in[2] = overflow_f;
  end
```

```
49
50  fetch func1(.clk(clk), .PC(PC), .instr(instr), .icode(icode), .ifun(ifun),
51                .rA(rA), .rB(rB), .valC(valC), .valP(valP),
52                .ins_mem_error(ins_mem_error), .valid_instr(valid_instr),.halt(halt));
53
54
55  decode_writeback func2(.clk(clk), .icode(icode),.cnd(cnd),.rA(rA), .rB(rB),.valA(valA), .valB(valB), .valE(valE), .valM(valM),
56                         .reg_0(reg_0),.reg_1(reg_1),.reg_2(reg_2), .reg_3(reg_3),.reg_4 (reg_4),.reg_5 (reg_5), .reg_6(reg_6), .reg_7(reg_7),
57                         .reg_8(reg_8), .reg_9(reg_9), .reg_10(reg_10),.reg_11(reg_11),.reg_12 (reg_12), .reg_13(reg_13), .reg_14(reg_14));
58
59
60  execute func3(.icode(icode),.ifun(ifun), .cnd(cnd), .valA(valA), .valB(valB),.valC(valC), .valE(valE),.cond_c_in(cond_c_in),.zero_f(zero_f),.sign_f(sign_f),.overflow_f(overflow_f));
61
62  pc_update func4(.clk(clk), .icode(icode),.cnd(cnd),.valC(valC),.valM(valM),.valP(valP),.PC(PC_next));
63
64
65  initial begin
66  $dumpfile("execute_test.vcd");
67  $dumpvars(0,execute_test);
68  //$monitor("clk=%d PC=%d icode=%b ifun=%b cnd=%d rA=%b rB=%b,valA=%d,valB=%d,valE=%d\n",clk,PC,icode,ifun,cnd,rA,rB,valA,valB,valE);
69    //$monitor("%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d\n", reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14);
70    $monitor("cnd=%d z_f=%d s_f=%d o_f=%d\n",cnd,zero_f,sign_f,overflow_f);
71    //$monitor("z_f=%d s_f=%d o_f=%d\n",cond_c_in[0],cond_c_in[1],cond_c_in[2]);
72  clk=1;
73  PC=64'd0;
74
75  ////////////////////////////////Test Case 1//////////////////////////
76  // Reference for 1.txt
77
78  //opq , halt along with irmovq
79
80  //irmovq $0x100, %rbx
81  instr_mem[0]=8'b00110000;
82  instr_mem[1]=8'b11110011;
83  instr_mem[2]=8'b00000000;
84  instr_mem[3]=8'b00000001;
85  instr_mem[4]=8'b00000000;
86  instr_mem[5]=8'b00000000;
87  instr_mem[6]=8'b00000000;
88  instr_mem[7]=8'b00000000;
89  instr_mem[8]=8'b00000000;
90  instr_mem[9]=8'b00000000;
```

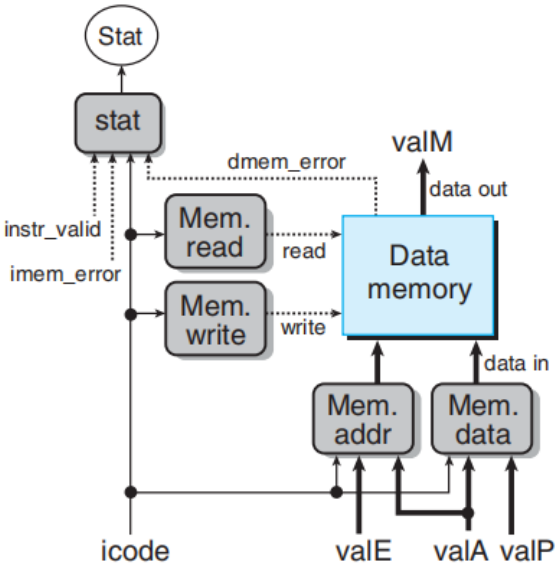
```
91
92    //irmovq $0x200, %dx
93    instr_mem[10]=8'b00110000;
94    instr_mem[11]=8'b11110010;
95    instr_mem[12]=8'b00000000;
96    instr_mem[13]=8'b00000010;
97    instr_mem[14]=8'b00000000;
98    instr_mem[15]=8'b00000000;
99    instr_mem[16]=8'b00000000;
100   instr_mem[17]=8'b00000000;
101   instr_mem[18]=8'b00000000;
102   instr_mem[19]=8'b00000000;
103
104   //addq %rdx, %rbx
105   instr_mem[20]=8'b01100000;
106   instr_mem[21]=8'b00100011;
107
108   //halt
109   instr_mem[22]=8'b00000000;
110
111   end
112   endmodule
```

Output:

```
clk=1 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=                  0,valB=                  0,valE=     281474976710656

clk=0 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=                  0,valB=                  0,valE=     281474976710656

clk=1 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=                  0,valB=                  0,valE=     562949953421312

clk=0 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=                  0,valB=                  0,valE=     562949953421312

clk=1 PC=                 20 icode=0110 ifun=0000 cnd=0 rA=0010 rB=0011,valA=     562949953421312,valB=     281474976710656,valE=     844424930131968

clk=0 PC=                 20 icode=0110 ifun=0000 cnd=0 rA=0010 rB=0011,valA=     562949953421312,valB=     281474976710656,valE=     844424930131968

execute_test.v:36: $finish called at 60 (1s)
clk=1 PC=                 22 icode=0000 ifun=0000 cnd=0 rA=0010 rB=0011,valA=     562949953421312,valB=     844424930131968,valE=                  0
```

Memory:

●Memory either reads data from memory or writes data to memory.



Implementation:

●The memory block has input values as **icode**, **valE**, **valA** and **valP** and the output values as **valM**

●The **dmem_error** is set to 1 when valE exceeds 1023 which is the size of the data memory.

●The value of **valM** is read from the registers and vice versa according to the value of **icode**.

●In case of rmovq, call and pushq we write to memory. Whereas, in the case of mrmovq, ret and popq we read from memory.

●Computed Values in this stage are -
valM - Value read from memory

Verilog Implementation

Code :

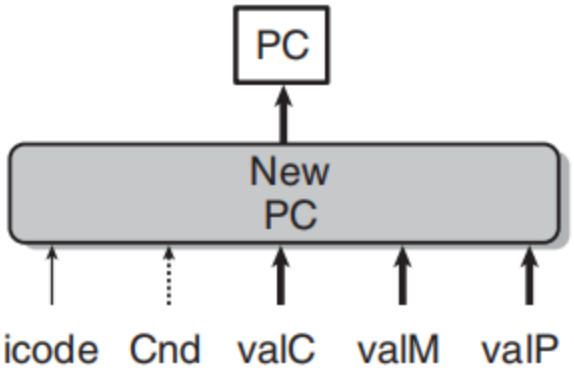
File Name: memory.v

```
1   module memory(clk,icode,valE,valP,valA,valM,dmem_error);
2
3   //inputs
4   input clk;
5   input [3:0] icode;
6   input [63:0] valE;
7   input [63:0] valP;
8   input [63:0] valA;
9
10  //outputs
11  output reg [63:0] valM;
12  output reg dmem_error=0;
13
14  reg [63:0] data_mem [0:1023];
15
16  always@(posedge clk)
17  begin
18  if(valE>18446744073709551616) //2^64
19  begin
20  dmem_error = 1;
21  end
22
23  // rmmovq
24  if(icode == 4'b0100)
25  begin
26  data_mem[valE] <= valA;
27  end
28
29  // mrmovq
30  else if(icode == 4'b0101)
31  begin
32  valM = data_mem[valE];
33  end
34
35  // call
36  else if(icode == 4'b1000)
37  begin
```

```
38    data_mem[valE] <= valP;
39    end
40
41    // ret
42    else if(icode == 4'b1001)
43    begin
44    valM = data_mem[valA];
45    end
46
47    // pushq
48    else if(icode == 4'b1010)
49    begin
50    data_mem[valE] <= valA;
51    end
52
53    // popq
54    else if(icode == 4'b1011)
55    begin
56    valM = data_mem[valA];
57    end
58
59    end
60    endmodule
```

PC Update:

●New value of the PC is updated as valC, valM or valP.



Implementation:

●The PC Update block has **icode**, **Cnd**, **valC**, **valM** and **valP** as inputs and the value of updated **PC** as output.

●The PC Update selects the instructions according to the values of **icode** and **Cnd** and updates the value of PC.

	OPq rA, rB	
PC update	PC ← valP	Update PC
	rmmovq rA, D(rB)	
PC update	PC ← valP	Update PC
	popq rA	
PC update	PC ← valP	Update PC
	jXX Dest	
PC update	PC ← Cnd ? valC : valP	Update PC
	call Dest	
PC update	PC ← valC	Set PC to destination
	ret	
PC update	PC ← valM	Set PC to return address

●Computed Values in this stage are -

PC_Update - Updated Program Counter

Verilog Implementation

Code :

File Name: pc_update.v

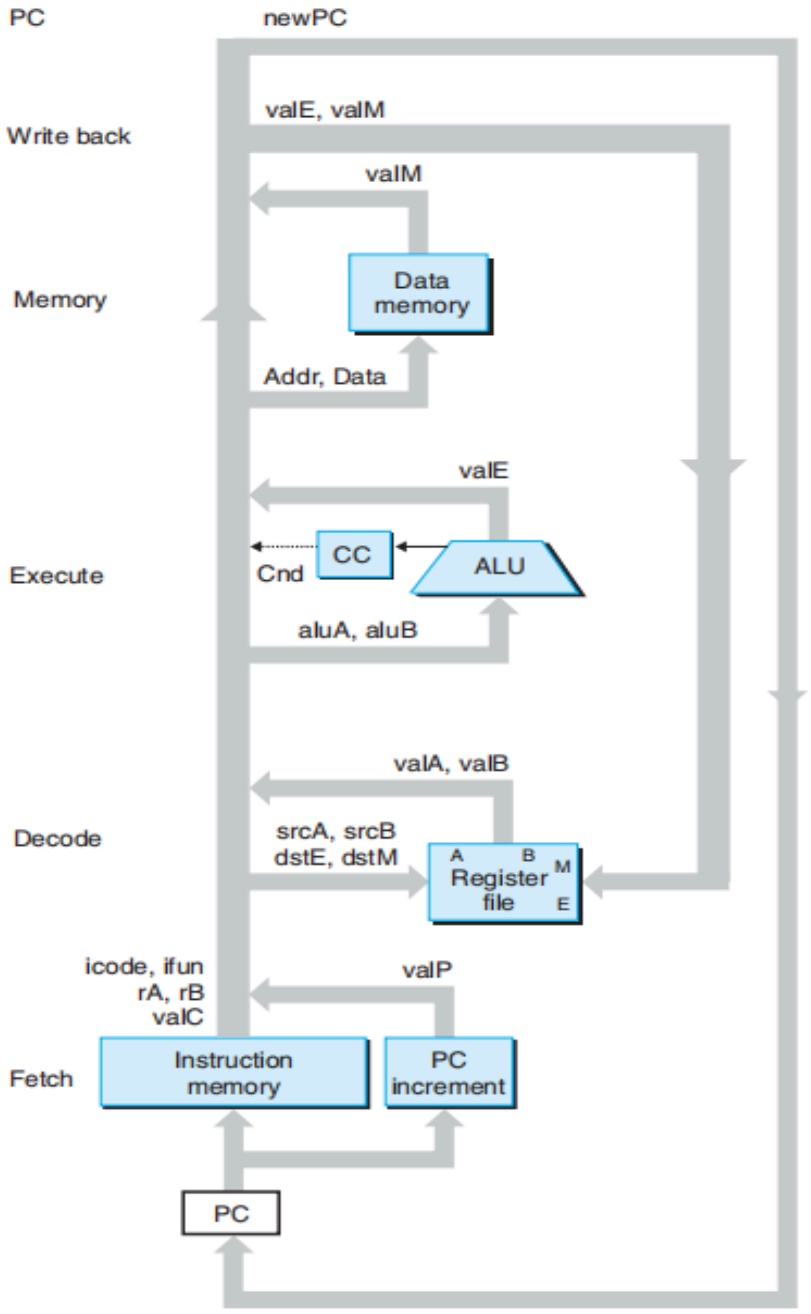
```
module pc_update(clk,icode,cnd,valC,valM,valP,PC);

//inputs
input clk;            // Clock input
input [3:0] icode;    //Instruction code
input cnd;           //Condition
input [63:0] valC;    // Immediate value
input [63:0] valM;    //Memory mod output
input [63:0] valP;    // Next program counter

//outputs
output reg [63:0] PC; // Program Counter

always@(posedge clk)
begin

if(icode == 4'b0000)
begin
PC <= 0;
end

if(icode == 4'b0001)
begin
PC <= valP;
end

if(icode == 4'b0010)
begin
PC <= valP;
end

if(icode == 4'b0011)
begin
PC <= valP;
end

if(icode == 4'b0100)
```

```
38    begin
39    PC <= valP;
40    end
41
42    if(icode == 4'b0101)
43    begin
44    PC <= valP;
45    end
46
47    if(icode == 4'b0110)
48    begin
49    PC <= valP;
50    end
51
52    if(icode == 4'b0111)
53    begin
54    PC <= cnd ? valC:valP;
55    end
56
57    if(icode == 4'b1000)
58    begin
59    PC <= valC;
60    end
61
62    if(icode == 4'b1001)
63    begin
64    PC <= valM;
65    end
66
67    if(icode == 4'b1010)
68    begin
69    PC <= valP;
70    end
71
72    if(icode == 4'b1011)
73    begin
74    PC <= valP;
```

```
75     end
76
77     end
78     endmodule
```

Sequential Implementation

The following has to be done when the above steps such as Fetch , Decode , Execute , Memory , Write back and PC update are combined and executed in a clock cycle.



Verilog Implementation

Testing :

File Name: processor.v

```
`include "fetch.v"
`include "decode_writeback.v"
`include "execute.v"
`include "memory.v"
`include "pc_update.v"

module processor;
reg clk;
reg [63:0] PC;
wire [63:0] PC_next;
reg [1:0] stat ; // AOK, HLT, ADR, INS
reg [0:79] instr;
reg [7:0] instr_mem[0:20480];
reg [2:0] cond_c_in;

wire [3:0] icode,ifun,rA,rB;
wire [63:0] valA,valB,valC,valE,valM,valP;

wire cnd;
wire dmem_error;
wire valid_instr, ins_mem_error,halt;

wire [63:0] reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14;

wire sign_f,zero_f,overflow_f;

//if constant is given from LSB to MSB
// always@(PC) begin

//   instr={
//     instr_mem[PC],
//     instr_mem[PC+1],
//     instr_mem[PC+9],
//     instr_mem[PC+8],
//     instr_mem[PC+7],
//     instr_mem[PC+6],
//     instr_mem[PC+5],
//     instr_mem[PC+4],
//     instr_mem[PC+3],
//     instr_mem[PC+2]
//   };
// end

//if constant given from MSB to LSB
always@(PC) begin
  instr={
    instr_mem[PC],
    instr_mem[PC+1],
    instr_mem[PC+2],
    instr_mem[PC+3],
    instr_mem[PC+4],
    instr_mem[PC+5],
    instr_mem[PC+6],
    instr_mem[PC+7],
    instr_mem[PC+8],
    instr_mem[PC+9]
  };
```

```
58    end
59
60    always #10 clk = ~clk;
61    always @(*) PC = PC_next;
62
63    always @(posedge clk) begin
64        if(icode==6)
65        cond_c_in[0] = zero_f;
66        cond_c_in[1] = sign_f;
67        cond_c_in[2] = overflow_f;
68      end
69    always@(valid_instr,ins_mem_error,dmem_error,icode)begin
70      if(valid_instr==0)
71        stat = 4;
72      else if(ins_mem_error==1 )
73        stat = 3;
74      else if(dmem_error==1 )
75        stat = 3;
76      else if(halt == 1)
77        stat = 2;
78      else
79        stat = 1;
80    end
81
82    always@(stat) begin
83        if(stat==2) begin
84          $display("Halting");
85          $finish;
86        end
87        else if(stat==3) begin
88          $display("Invalid address Error");
89          $finish;
90        end
91        else if(stat==4) begin
92          $display("Invalid Instruction Error");
93          $finish;
94        end
95    end
96
97
```

```
98  fetch func1(.clk(clk), .PC(PC), .instr(instr), .icode(icode), .ifun(ifun),
99                .rA(rA), .rB(rB), .valC(valC), .valP(valP),
100               .ins_mem_error(ins_mem_error), .valid_instr(valid_instr),.halt(halt));
101 decode_writeback func2(.clk(clk), .icode(icode),.cnd(cnd),.rA(rA), .rB(rB),.valA(valA), .valB(valB), .valE(valE), .valM(valM),
102                        .reg_0(reg_0),.reg_1(reg_1),.reg_2(reg_2), .reg_3(reg_3),.reg_4 (reg_4),.reg_5 (reg_5), .reg_6(reg_6), .reg_7(reg_7),
103                        .reg_8(reg_8), .reg_9(reg_9), .reg_10(reg_10),.reg_11(reg_11),.reg_12 (reg_12), .reg_13(reg_13), .reg_14(reg_14));
104 execute func3(.icode(icode),.ifun(ifun), .cnd(cnd), .valA(valA), .valB(valB),.valC(valC), .valE(valE),.cond_c_in(cond_c_in),.zero_f(zero_f),.sign_f(sign_f),.overflow_f(overflow_f));
105 memory func4(.clk(clk),.icode(icode),.valE(valE),.valP(valP),.valA(valA),.valM(valM),.dmem_error(dmem_error));
106 pc_update func5(.clk(clk), .icode(icode),.cnd(cnd),.valC(valC),.valM(valM),.valP(valP),.PC(PC_next));
107
108 initial begin
109     $dumpfile("processor.vcd");
110     $dumpvars(0,processor);
111     //$monitor("clk=%d PC=%d icode=%b ifun=%b rA=%b rB=%b,valC=%d,valP=%d\n",clk,PC,icode,ifun,rA,rB,valC,valP);
112     $monitor("clk=%d PC=%d icode=%b ifun=%b cnd=%d rA=%b rB=%b,valA=%d,valB=%d,valE=%d,\n",clk,PC,icode,ifun,cnd,rA,rB,valA,valB,valE);
113   // $monitor("clk=%d PC=%d icode=%b ifun=%b rA=%b rB=%b,valA=%d,valB=%d,vaE=%d,valM=%d,reg1=%d,reg2=%d,reg3=%d,reg5=%d,reg7=%d\n",clk,PC,icode,ifun,rA,rB,valA,valB,valE,valM,reg_1,reg_2,reg_3,reg_5,reg_7);
114     clk=1;
115     PC=64'd0;
116 ////////////////////////////////Test Case 1////////////////////////////////
```

Test Case 1 `opq , halt along with irmovq`:

```
////////////////////////////Test Case 1/////////////////////////////
// Reference for 1.txt
//opq , halt along with irmovq

//irmovq $0x100, %rbx
instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000001;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000000;
instr_mem[9]=8'b00000000;
//irmovq $0x200, %dx
instr_mem[10]=8'b00110000;
instr_mem[11]=8'b11110010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000010;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000000;
instr_mem[19]=8'b00000000;
//addq %rdx, %rbx
instr_mem[20]=8'b01100000;
instr_mem[21]=8'b00100011;
//halt
instr_mem[22]=8'b00000000;
```

Output:

```
clk=1 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=                  0,valB=                  0,valE=     281474976710656,

clk=0 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=                  0,valB=                  0,valE=     281474976710656,

clk=1 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=                  0,valB=                  0,valE=     562949953421312,

clk=0 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=                  0,valB=                  0,valE=     562949953421312,

clk=1 PC=                 20 icode=0110 ifun=0000 cnd=0 rA=0010 rB=0011,valA=     562949953421312,valB=     281474976710656,valE=     844424930131968,

clk=0 PC=                 20 icode=0110 ifun=0000 cnd=0 rA=0010 rB=0011,valA=     562949953421312,valB=     281474976710656,valE=     844424930131968,

Halting
processor.v:85: $finish called at 60 (1s)
clk=1 PC=                 22 icode=0000 ifun=0000 cnd=0 rA=0010 rB=0011,valA=     562949953421312,valB=     844424930131968,valE=                  0,
```

Test Case 2 `rrmovq,rmmovq,nop,halt with opq`

```
//////////////////////////////Test Case 2//////////////////////////

//rrmovq , rmmovq , nop , halt along with opq

//opq add
instr_mem[0] = 8'b01100000;
//%rax = 1 %rbx = 3
instr_mem[1] = 8'b00010011;
// rrmovq
instr_mem[2] = 8'b00100000;
instr_mem[3] = 8'b00010011;
// src = %rax dest = %rbx

//rmmovq
instr_mem[4] = 8'b01000000;
 //rax and (rbx)
instr_mem[5] = 8'b00010011;
//VALC
instr_mem[6] = 8'b00001111;
instr_mem[7] = 8'b00000000;
instr_mem[8] = 8'b00000000;
instr_mem[9] = 8'b00000000;
instr_mem[10] = 8'b00000000;
instr_mem[11] = 8'b00000000;
instr_mem[12] = 8'b00000000;
instr_mem[13] = 8'b00000000;

//no operation
instr_mem[14] = 8'b00010000;

//halt
instr_mem[15] = 8'b00000000;
```

Output:

```
clk=1 PC=                  0 icode=0110 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                3,valE=                4,

clk=0 PC=                  0 icode=0110 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                3,valE=                4,

clk=1 PC=                  2 icode=0010 ifun=0000 cnd=1 rA=0001 rB=0011,valA=                1,valB=                0,valE=                1,

clk=0 PC=                  2 icode=0010 ifun=0000 cnd=1 rA=0001 rB=0011,valA=                1,valB=                0,valE=                1,

clk=1 PC=                  4 icode=0100 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                1,valE= 1080863910568919041,

clk=0 PC=                  4 icode=0100 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                1,valE= 1080863910568919041,

clk=1 PC=                 14 icode=0001 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                1,valE=                0,

clk=0 PC=                 14 icode=0001 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                1,valE=                0,

Halting
processor.v:85: $finish called at 80 (1s)
clk=1 PC=                 15 icode=0000 ifun=0000 cnd=0 rA=0001 rB=0011,valA=                1,valB=                1,valE=                0,
```

Test Case 3 `jump halt with irmovq`

```
////////////////////////////////Test Case 3//////////////////////////

//jump, halt along with irmovq

//irmovq $0xc, %rbx
//3 0
  instr_mem[20]=8'b00110000;
  //F rB=3
  instr_mem[21]=8'b00000011;
  instr_mem[22]=8'b00000000;
  instr_mem[23]=8'b00000000;
  instr_mem[24]=8'b00000000;
  instr_mem[25]=8'b00000000;
  instr_mem[26]=8'b00000000;
  instr_mem[27]=8'b00000000;
  instr_mem[28]=8'b00000000;
  instr_mem[29]=8'b00001100;
// Val=12

  //jmp :find
   //7 jxx
  instr_mem[30]=8'b01110000;
  instr_mem[31]=8'b00000000;
  instr_mem[32]=8'b00000000;
  instr_mem[33]=8'b00000000;
  instr_mem[34]=8'b00000000;
  instr_mem[35]=8'b00000000;
  instr_mem[36]=8'b00000000;
  instr_mem[37]=8'b00000000;
  instr_mem[38]=8'b00100111;

  // find:
  // opq //6 add
  instr_mem[39]=8'b01100000;
  //rA=0 rB=3
  instr_mem[40]=8'b00000011;
  instr_mem[41]=8'b00000000;
```

Output:

```
clk=1 PC=                20 icode=0011 ifun=0000 cnd=0 rA=0000 rB=0011,valA=                0,valB=                0,valE=               12,

clk=0 PC=                20 icode=0011 ifun=0000 cnd=0 rA=0000 rB=0011,valA=                0,valB=                0,valE=               12,

clk=1 PC=                30 icode=0111 ifun=0000 cnd=1 rA=0000 rB=0011,valA=                0,valB=                0,valE=                0,

clk=0 PC=                30 icode=0111 ifun=0000 cnd=1 rA=0000 rB=0011,valA=                0,valB=                0,valE=                0,

clk=1 PC=                39 icode=0110 ifun=0000 cnd=0 rA=0000 rB=0011,valA=                0,valB=               12,valE=               12,

clk=0 PC=                39 icode=0110 ifun=0000 cnd=0 rA=0000 rB=0011,valA=                0,valB=               12,valE=               12,

Halting
processor.v:85: $finish called at 60 (1s)
clk=1 PC=                41 icode=0000 ifun=0000 cnd=0 rA=0000 rB=0011,valA=                0,valB=               12,valE=                0,
```

Test Case 4 `call ret,halt with irmovq`

```
////////////////////////////Test Case 4/////////////////////////////
//call , ret , halt along with irmovq
//irmovq $0x2, %rax
instr_mem[0]=8'b00110000;
//F rB=0
instr_mem[1]=8'b00000000;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000000;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000000;
instr_mem[9]=8'b00000010; //Val=2
//irmovq $0x10, %rdx
//3 0
instr_mem[10]=8'b00110000;
//F rB=2
instr_mem[11]=8'b00000010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000000;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000000;
instr_mem[19]=8'b01010001;
//Val=81

//1 0
instr_mem[20]=8'b00010000;
//call
//8 0
  instr_mem[21]=8'b10000000;
  //dest
  instr_mem[22]=8'b00000000;
  instr_mem[23]=8'b00000000;
  instr_mem[24]=8'b00000000;
  instr_mem[25]=8'b00000000;
  instr_mem[26]=8'b00000000;
  instr_mem[27]=8'b00000000;
  instr_mem[28]=8'b00000000;
  instr_mem[29]=8'b00000001;

//ret 9 0
  instr_mem[30]=8'b10010000;
//halt
  instr_mem[31]=8'b00000000;
```

Output:

```
clk=1 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=0000 rB=0000,valA=                  0,valB=                  0,valE=                  2,

clk=0 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=0000 rB=0000,valA=                  0,valB=                  0,valE=                  2,

clk=1 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=                  0,valE=                 81,

clk=0 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=                  0,valE=                 81,

clk=1 PC=                 20 icode=0001 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=                  0,valE=                  0,

clk=0 PC=                 20 icode=0001 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=                  0,valE=                  0,

clk=1 PC=                 21 icode=1000 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=                  4,valE=18446744073709551612,

clk=0 PC=                 21 icode=1000 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=                  4,valE=18446744073709551612,

Halting
processor.v:85: $finish called at 80 (1s)
clk=1 PC=                  1 icode=0000 ifun=0000 cnd=0 rA=0000 rB=0010,valA=                  0,valB=18446744073709551612,valE=                  0,
```

Test Case 5 `cmovxx,halt irmovq with opq`

```
//////////////////////////////Test Case 5//////////////////////////

//cmovxx , halt along with irmovq and opq

//addq %rdx, %rbx
instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000000;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000101;
instr_mem[9]=8'b00001000;
instr_mem[10]=8'b00110000;
instr_mem[11]=8'b11110010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000000;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000000;
instr_mem[19]=8'b00000001;

instr_mem[20]=8'b01100000;
instr_mem[21]=8'b00100011;

instr_mem[22] = 8'b00010000;

//cmov
// 2 fn
instr_mem[23]=8'b00100000;
// rA rB
instr_mem[24]=8'b00110100;

//halt
instr_mem[25] = 8'b00000000;
```

Output:

```
clk=1 PC=                0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=              0,valB=              0,valE=           1288,

clk=0 PC=                0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=              0,valB=              0,valE=           1288,

clk=1 PC=               10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=              0,valB=              0,valE=              1,

clk=0 PC=               10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=              0,valB=              0,valE=              1,

clk=1 PC=               20 icode=0110 ifun=0000 cnd=0 rA=0010 rB=0011,valA=              1,valB=           1288,valE=           1289,

clk=0 PC=               20 icode=0110 ifun=0000 cnd=0 rA=0010 rB=0011,valA=              1,valB=           1288,valE=           1289,

clk=1 PC=               22 icode=0001 ifun=0000 cnd=0 rA=0010 rB=0011,valA=              1,valB=           1289,valE=              0,

clk=0 PC=               22 icode=0001 ifun=0000 cnd=0 rA=0010 rB=0011,valA=              1,valB=           1289,valE=              0,

clk=1 PC=               23 icode=0010 ifun=0000 cnd=1 rA=0011 rB=0100,valA=           1289,valB=              0,valE=           1289,

clk=0 PC=               23 icode=0010 ifun=0000 cnd=1 rA=0011 rB=0100,valA=           1289,valB=              0,valE=           1289,

Halting
processor.v:85: $finish called at 100 (1s)
clk=1 PC=               25 icode=0000 ifun=0000 cnd=0 rA=0011 rB=0100,valA=           1289,valB=              0,valE=              0,
```

Test Case 6 `irmovq,halt with opq`

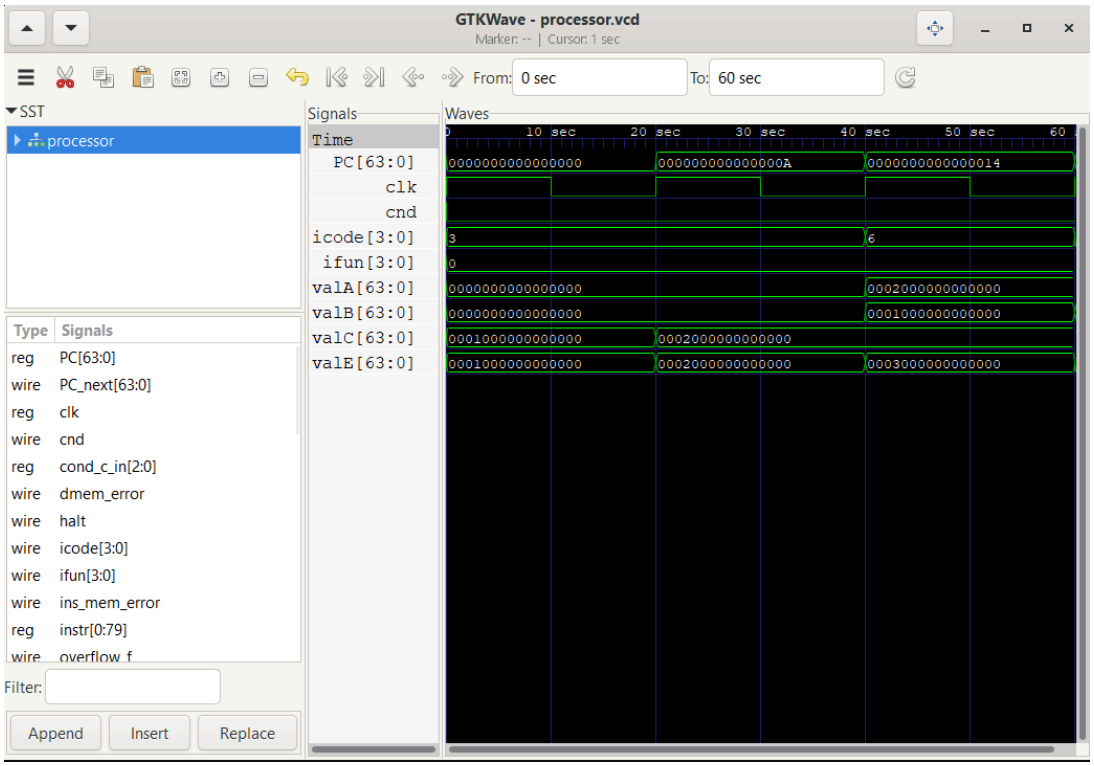
```
////////////////////////////////Test Case 6/////////////////////////////

//opq , halt along with irmovq

instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000000;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000000;
instr_mem[9]=8'b00000100;
instr_mem[10]=8'b00110000;
instr_mem[11]=8'b11110010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000000;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000110;
instr_mem[19]=8'b00000000;
instr_mem[20]=8'b01100001;
instr_mem[21]=8'b00100011;

instr_mem[22]=8'b00000000;
end
endmodule
```

Output:

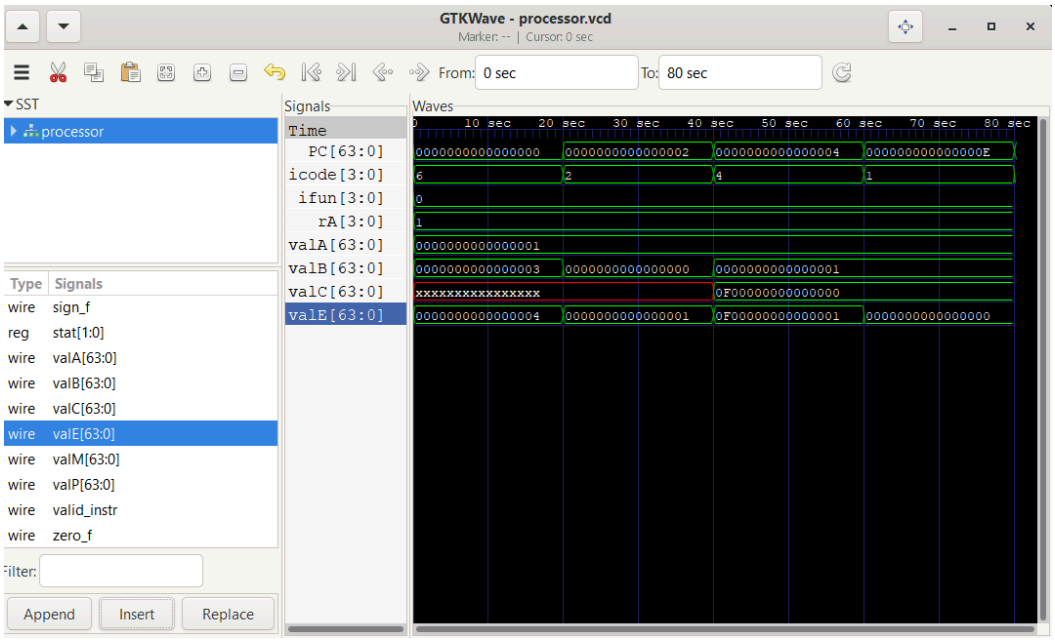
```
VCD info: dumpfile processor.vcd opened for output.
clk=1 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=                 0,valB=                 0,valE=                 4,

clk=0 PC=                  0 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0011,valA=                 0,valB=                 0,valE=                 4,

clk=1 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=                 0,valB=                 0,valE=              1536,

clk=0 PC=                 10 icode=0011 ifun=0000 cnd=0 rA=1111 rB=0010,valA=                 0,valB=                 0,valE=              1536,

clk=1 PC=                 20 icode=0110 ifun=0001 cnd=0 rA=0010 rB=0011,valA=              1536,valB=                 4,valE=              1532,

clk=0 PC=                 20 icode=0110 ifun=0001 cnd=0 rA=0010 rB=0011,valA=              1536,valB=                 4,valE=              1532,

Halting
processor.v:85: $finish called at 60 (1s)
clk=1 PC=                 22 icode=0000 ifun=0000 cnd=0 rA=0010 rB=0011,valA=              1536,valB=              1532,valE=                 0,
```

GTK Wave for Sequential Implementation:

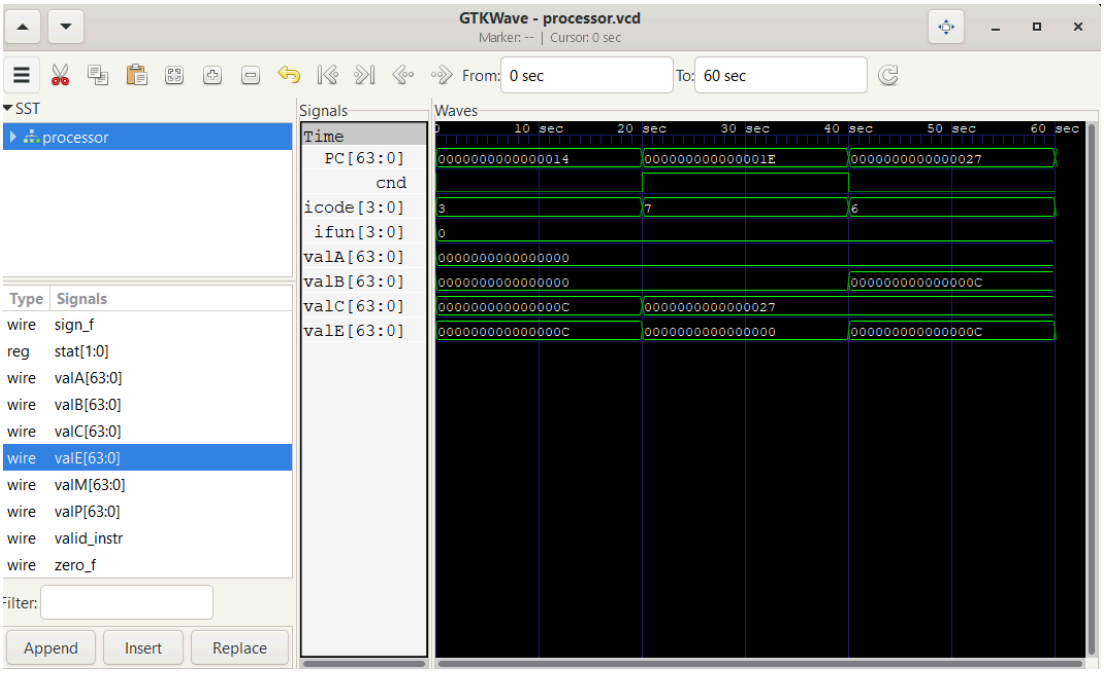
Test Case 1 `opq , halt along with irmovq:`



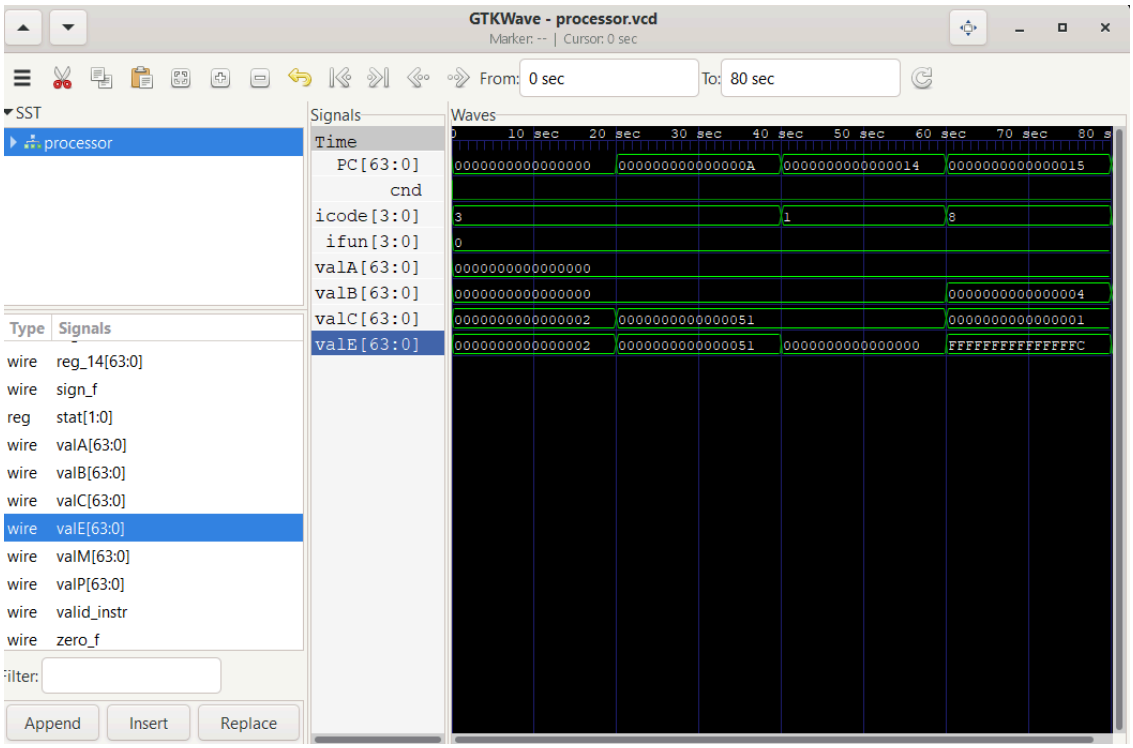
Test Case 2 `rrmovq,rmmovq,nop,halt with opq:`



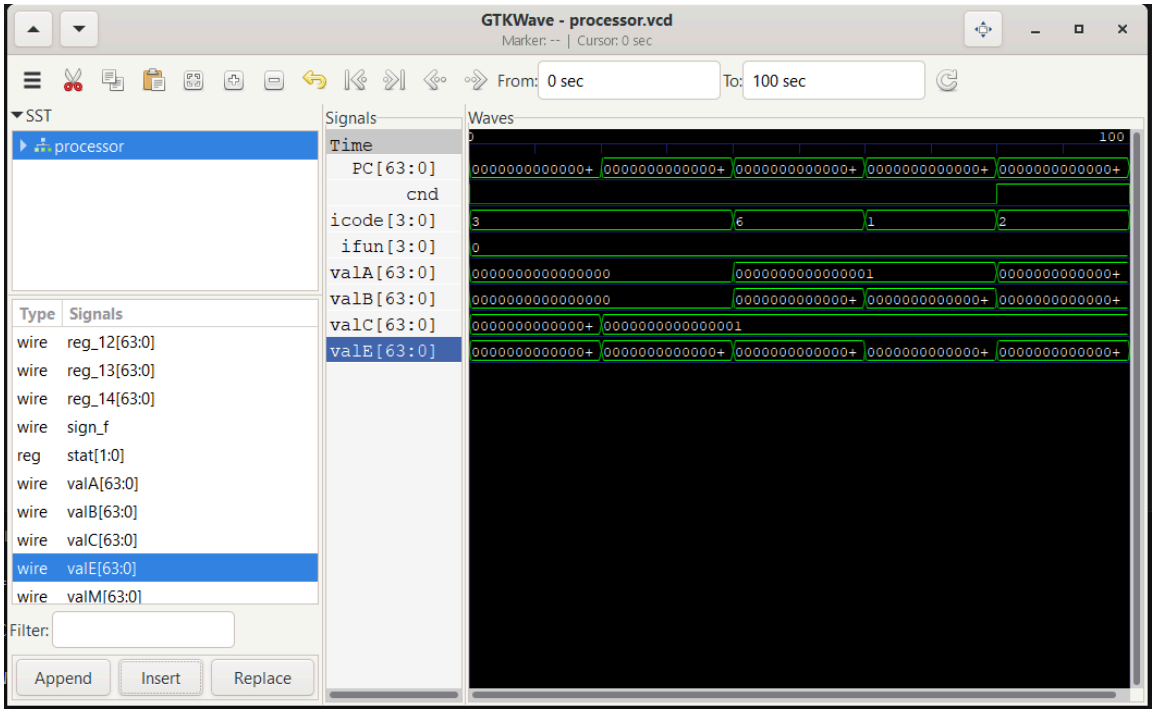
Test Case 3 `jump halt with irmovq`



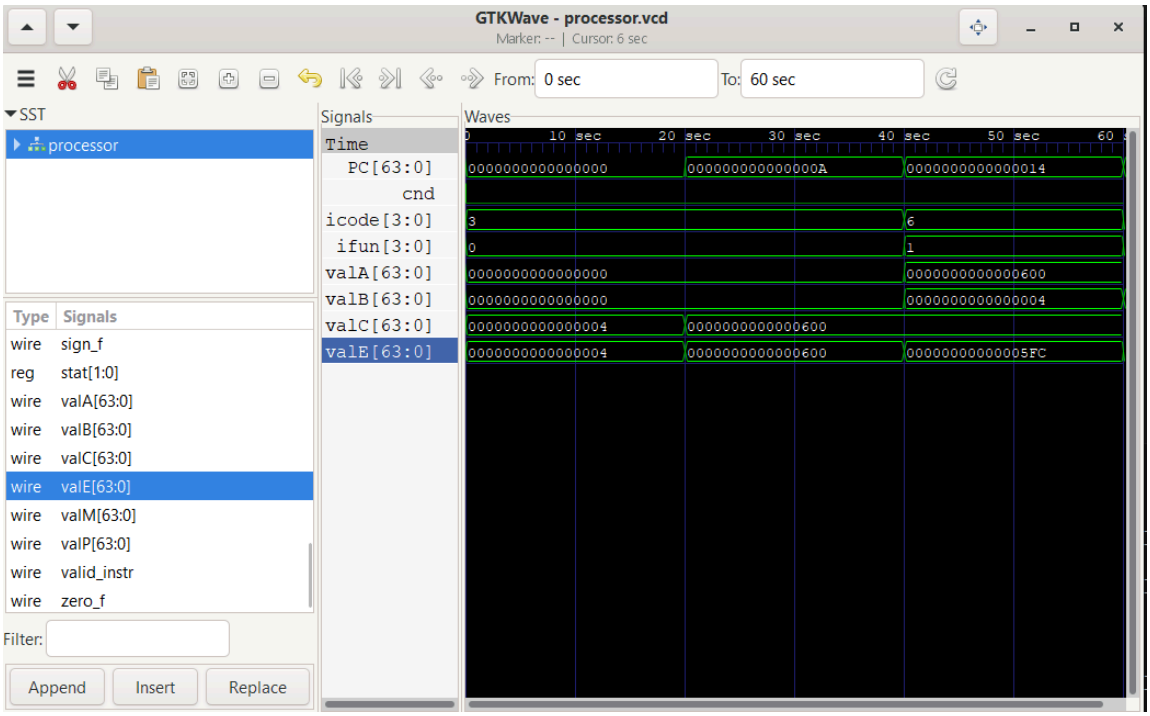
Test Case 4 `call ret,halt with irmovq`



Test Case 5 `cmovxx,halt irmovq with opq`



Test Case 6 `irmovq,halt with opq`

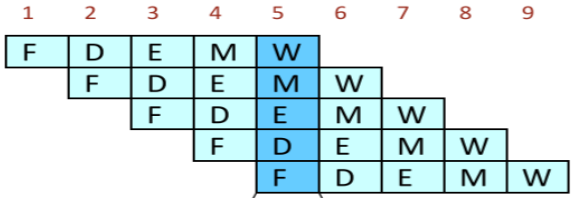


Pipeline Implementation
Problem Approach:

To Design the Pipeline Implementation of Y86 ISA that performs all stages such as Fetch , Decode , Execute , Memory , Write back along with PC update parallely some changes have to be made to sequential Implementation.
A key feature of pipelining is that it increases the throughput of the system,but it may also slightly increase the latency.
The pipelined implementation of the Y86-64 works the same way as that of the sequential implementation with the modules being the same but with inclusion of the pipelined registers.
Slight changes have to be done in the fetch and decode blocks for data forwarding and PC prediction for improving the performance and the addition of the pipeline control logic for eliminating pipeline hazards.

STEPS BEING DONE PARALLELY:

- Fetch
- Decode
- Execute
- Memory
- Write Back



Changes to Sequential Implementation:

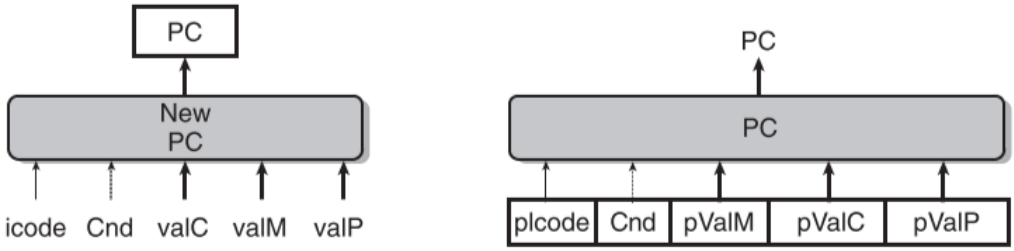
The first step to pipelining is the rearrangement of computation stages.

Rearranging Steps:

Rearrange the order of the five stages in SEQ so that the PC update stage comes at the beginning of the clock cycle, rather than at the end.

This transformation will work better with the sequencing of activities within the pipeline stages and is referred to as “SEQ+”.

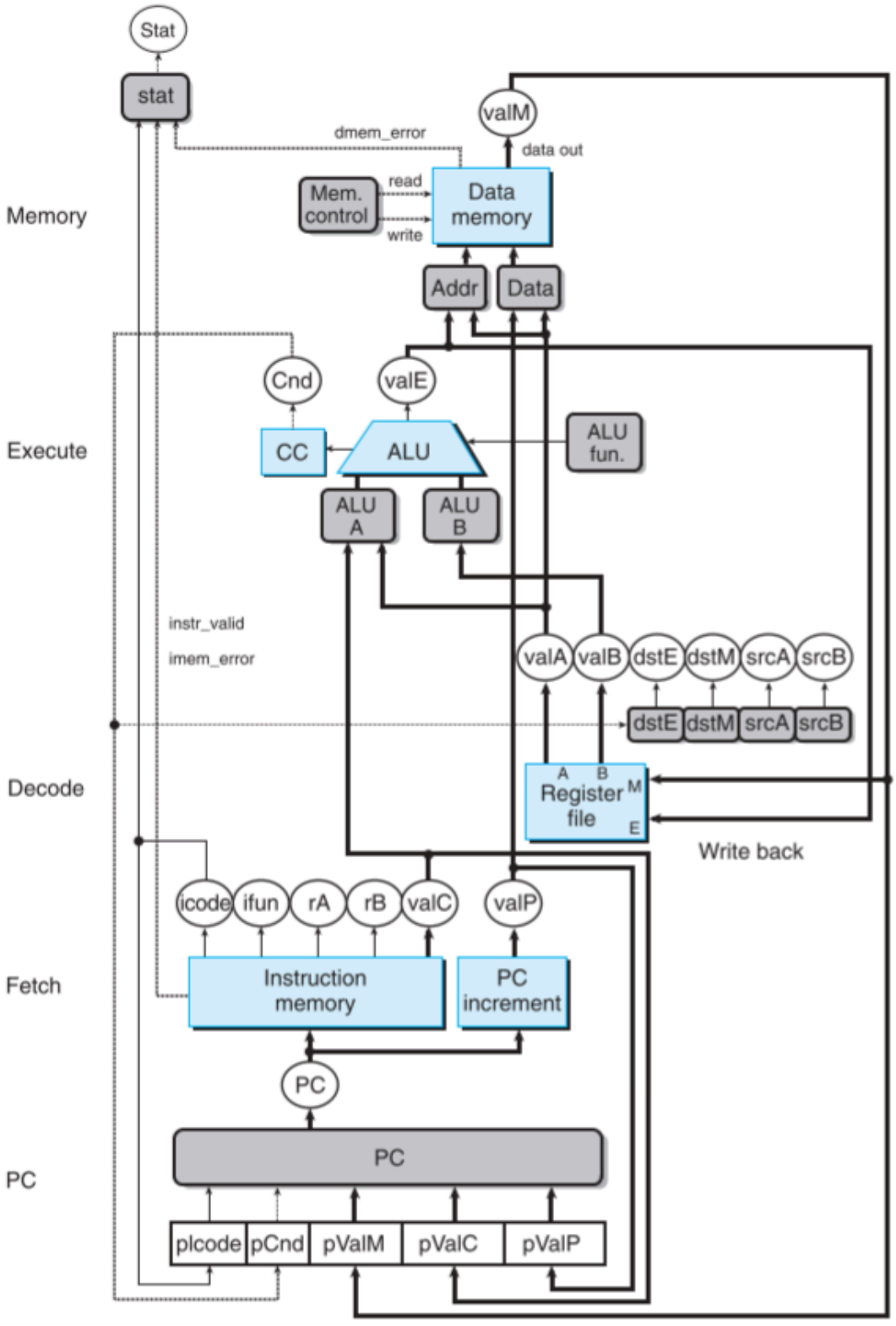
With SEQ+ we create state registers to hold the signals computed during an instruction .As a new clock cycle begins, the values propagate through the exact same logic to compute the PC through the labeled registers “pIcode,” “pCnd,” indicating that on any given cycle, they hold the control signals generated during the previous cycle.



(a) SEQ new PC computation (b) SEQ+ PC selection

The most important feature of SEQ+ is that there is no hardware register storing the program counter.

Instead, the PC is computed dynamically based on some state information stored from the previous instruction.



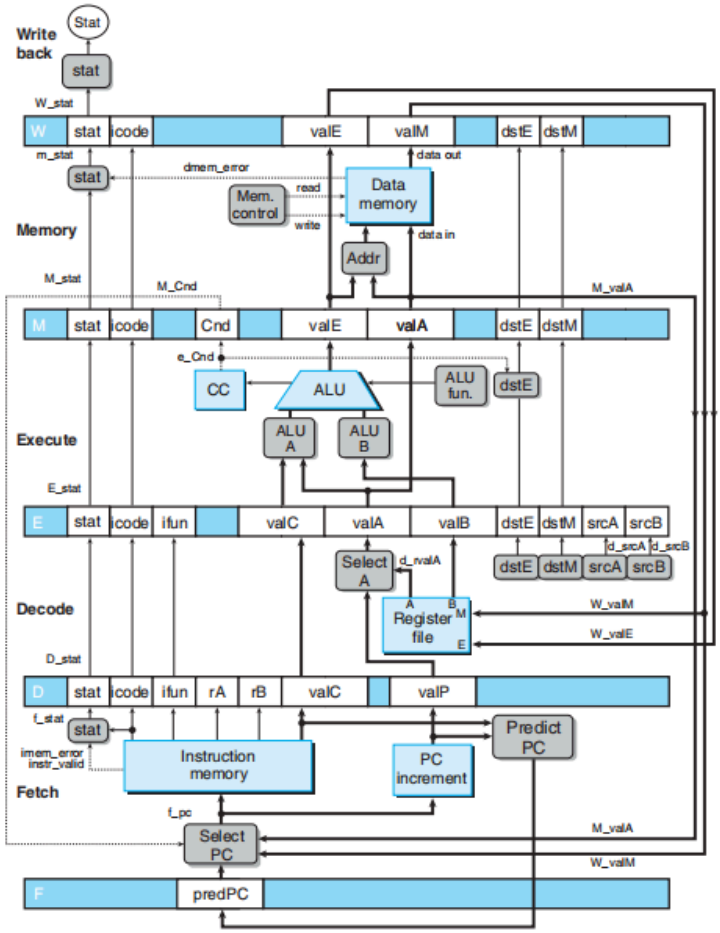
SEQ+ with PC being at the beginning

The next step to achieve pipelining is the insertion of Pipeline registers.

Pipelined Registers:

Insertion of pipeline registers between the stages of SEQ+ and rearrange signals somewhat, yielding the PIPE– processor, which is the sub stage before getting the desired pipelined processor.

Indicated by the multiple fields, each pipeline register holds multiple bytes and words.The hardware used by PIPE- is almost similar to SEQ+ but with the inclusion of pipeline registers.



PIPE- with Pipeline Registers

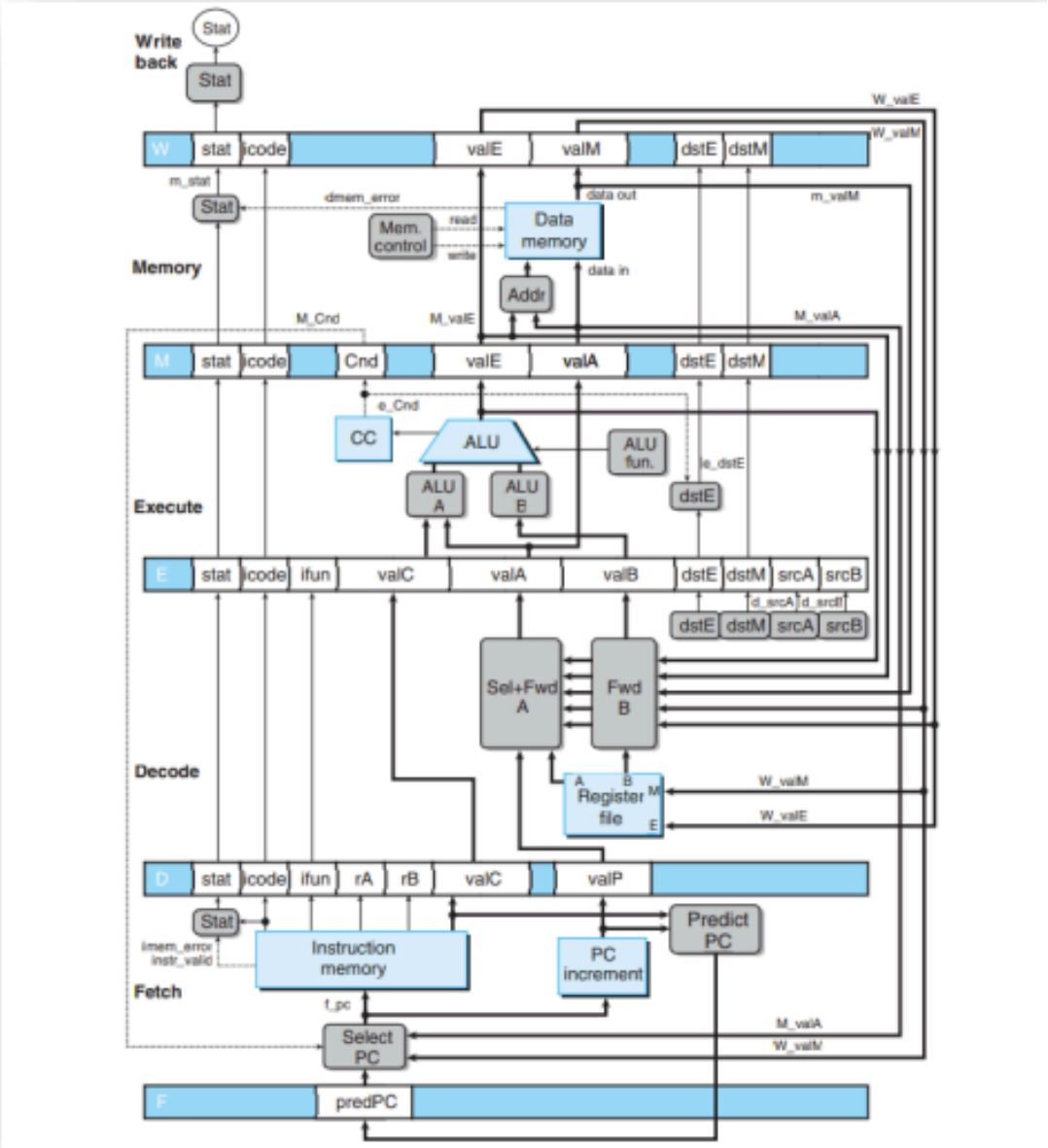
The pipeline registers are labeled as follows:
F holds a predicted value of the program counter, as will be discussed shortly.
D sits between the fetch and decode stages. It holds information about the most recently fetched instruction for processing by the decode stage.
E sits between the decode and execute stages. It holds information about the most recently decoded instruction and the values read from the register file for processing by the execute stage.
M sits between the execute and memory stages. It holds the results of the most recently executed instruction for processing by the memory stage.
It also holds information about branch conditions and branch targets for processing conditional jumps.
W sits between the memory stage and the feedback paths that supply the computed results to the register file for writing and the return address to the PC selection logic when completing a ret instruction

Rearranging and Relabelling Signals:
There are four “stat” that hold the status codes for four different instructions.
A naming scheme where a signal stored in a pipeline register can be uniquely identified by prefixing its name with that of the pipe register written in uppercase.
ex: D_stat, E_stat, M_stat, and W_stat.

The signals that have just been computed within a stage are labeled by prefixing the signal name with the first character of the stage name, written in lowercase.
ex: f_stat and m_stat.

With all the above changes Pipeline Implementation is done.

Pipeline Implementation:



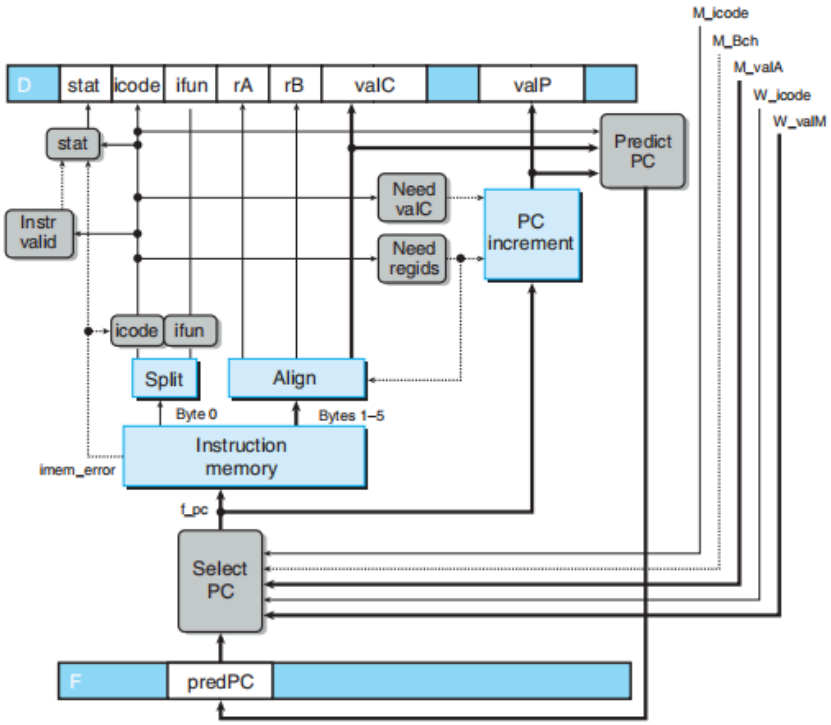
PC Selection & Fetch:

This stage must select a current value for PC as well as predict the next PC value. The hardware units for reading the instruction from memory and for extracting the different instruction fields are the same as SEQ

PC Prediction:

- Instructions that Don't Transfer Control
  - Predict next PC to be valP - Always reliable
- Call and Unconditional Jumps
  - Predict next PC to be valC (dst)- Always reliable
- Conditional Jumps
  - Predict next PC to be valC (destination)

  Only correct if branch is taken (around 60%)
- Return Instruction
  - Don't try to predict



Implementation:

●The **f_pc** is initialized to the initial value of PC from the start.This takes the input as the PC and compute the values of **stat, icode, ifun, rA,rB, valC** and **valP**

●Here the fetch part functions the same as that of sequential Implementation.

●The addition of the Predict PC block adds the functionality of the predicting PC and after this f_pc is named as **f_predPC** which gets updated on every clock cycle.

●In the fetch stage, memory error is tested which occurs due to an out-of-range instruction address.

●The Values computed are **D_icode, D_ifun, D_rA, D_rB, D_valC, D_stat D_valP**

Verilog Implementation

Code :

File Name: fetch.v

```
module fetch(clk, f_predPC, F_predPC, D_stat, D_icode, D_ifun, D_rA, D_rB, D_valC, D_valP,
             M_icode, M_cnd, M_valA, W_icode, W_valM, F_stall, D_stall, D_bubble);

    //inputs
    input clk,M_cnd;
    input [3:0] M_icode,W_icode;
    input [63:0] M_valA,W_valM,F_predPC;
    input F_stall,D_stall,D_bubble;

    //outputs
    output reg [3:0] D_icode, D_ifun, D_rA, D_rB;
    output reg [63:0] D_valC,D_valP,f_predPC;
    output reg [0:3] D_stat=4'b1000;

    reg [3:0] icode, ifun,rA, rB;
    reg [63:0] valC,valP;
    reg [0:3] stat;                  // AOK, HLT, ADR, INS
    reg [0:79] instr;
    reg [7:0] instr_mem[0:2048];
    reg [63:0] PC;
    reg instr_mem_error=0, instr_valid=1;
```

```
22
23      initial
24          PC = F_predPC;    //predicted PC_value
25
26      always@*
27      begin
28          // Jump not taken
29          if(M_icode==4'b0111 & !M_cnd)
30              PC = M_valA;
31          // Return
32          else if(W_icode==4'b1001)
33              PC = W_valM;
34          else
35              PC = F_predPC;
36      end
37
38      always@*
39      begin
40
41          instr_valid=1;
42          if(PC>2048)
43          begin
44              instr_mem_error=1;
45          end
46          instr = {instr_mem[PC],instr_mem[PC+1],instr_mem[PC+2],
47                   instr_mem[PC+3],instr_mem[PC+4],instr_mem[PC+5],
48                   instr_mem[PC+6],instr_mem[PC+7],instr_mem[PC+8],instr_mem[PC+9]};
49          {icode, ifun} = instr[0:7];
50
```

```
51          // halt
52          if (icode == 4'b0000)
53          begin
54              valP = PC;
55              f_predPC = valP;
56          end
57
58          // nop
59          else if (icode == 4'b0001)
60          begin
61              valP = PC + 1;
62              f_predPC = valP;
63          end
64
65          // cmovq
66          else if (icode == 4'b0010)
67          begin
68              {rA, rB} = instr[8:15];
69              valP = PC + 2;
70              f_predPC = valP;
71          end
72
73          // irmovq
74          else if (icode == 4'b0011)
75          begin
76              {rA, rB, valC} = instr[8:79];
77              valP = PC + 10;
78              f_predPC = valP;
79          end
80
81          // rmmovq
82          else if (icode == 4'b0100)
83          begin
84              {rA, rB, valC} = instr[8:79];
```

```
            valP = PC + 10;
            f_predPC = valP;
        end

        // mrmovq
        else if (icode == 4'b0101)
        begin
            {rA, rB, valC} = instr[8:79];
            valP = PC + 10;
            f_predPC = valP;
        end

        // OPq
        else if (icode == 4'b0110)
        begin
            {rA, rB} = instr[8:15];
            valP = PC + 2;
            f_predPC = valP;
        end

        // jxx
        else if (icode == 4'b0111)
        begin
            valC = instr[8:71];
            valP = PC + 9;
            f_predPC = valC;
        end

        // call
        else if (icode == 4'b1000)
        begin
            valC = instr[8:71];
            valP = PC + 9;
            f_predPC = valC;
        end

        // ret
        else if (icode == 4'b1001)
        begin
            valP = PC + 1;
        end

        // pushq
        else if (icode == 4'b1010)
        begin
            {rA, rB} = instr[8:15];
            valP = PC + 2;
```

```
            f_predPC = valP;
        end

        // popq
        else if (icode == 4'b1011)
        begin
            {rA, rB} = instr[8:15];
            valP = PC + 2;
            f_predPC = valP;
        end
        else
```

```
143         begin
144             instr_valid = 1'b0;
145         end
146 
147         if(instr_valid==0)
148         stat = 4'b0001;
149         else if(instr_mem_error==1)
150         begin
151         stat = 4'b0010;
152         end
153         else if(icode==4'b0000)
154         stat = 4'b0100;
155         else
156         stat = 4'b1000;
157     end
158 
159      // Assigning to decode registers
160     always@(posedge clk)
161     begin
162         if(F_stall)
163         begin
164             PC = F_predPC;
165         end
166 
167         if(D_stall)
168         begin
169         end
170         else if(D_bubble)
171         begin
172             D_icode <= 4'b0001;
173             D_ifun <= 4'b0000;
174             D_rA <= 4'b0000;
175             D_rB <= 4'b0000;
176             D_valC <= 64'b0;
177             D_valP <= 64'b0;
178             D_stat <= 4'b1000;
```

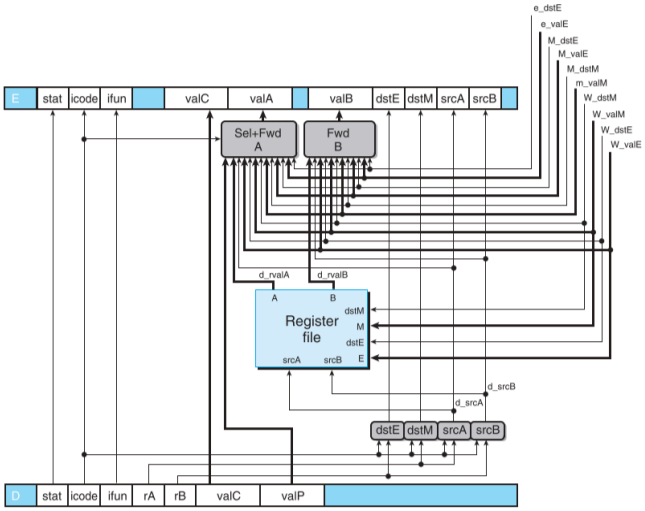
```
179         end
180         else
181         begin
182             D_icode <= icode;
183             D_ifun <= ifun;
184             D_rA <= rA;
185             D_rB <= rB;
186             D_valC <= valC;
187             D_valP <= valP;
188             D_stat <= stat;
189         end
190     end
191 
192     initial
193     begin
194 
195       // Reference for 1.txt
196       //opq , halt along with irmovq
197       //$readmemb("test1_pipe.txt", instr_mem);
198 
199      //rrmovq , rmmovq , nop , halt along with opq
200      //$readmemb("test2_pipe.txt", instr_mem);
201 
202      //jump, halt along with irmovq
203      //$readmemb("test3_pipe.txt", instr_mem);
204 
```

```
205     //opq , halt along with irmovq
206     //$readmemb("test4_pipe.txt", instr_mem);
207
208     //cmovxx , halt along with irmovq and opq
209     //$readmemb("test5_pipe.txt", instr_mem);
210
211     //opq , halt along with irmovq
212     //$readmemb("test6_pipe.txt", instr_mem);
213
214     //push, pop, halt along with irmovq
215     //$readmemb("test7_pipe.txt", instr_mem);
216
217     //irmovq, mrmovq, opq along with halt
218     //$readmemb("test8_pipe.txt", instr_mem);
219
220     //call, return, rrmovq along with conditional and unconditional jumps
221     $readmemb("test9_pipe.txt", instr_mem);
222
223 end
224 endmodule
```

Decode and Write Back:

The blocks labeled as “dstE”, “dstM”, “srcA”, and “srcB” are very similar to the implementation of SEQ.

The register IDs supplied to the write ports come from the write-back stage (signals W_dstE and W_dstM), rather than from the decode stage.



Forwarding Logic:

Data word	Register ID	Source description
e_valE	e_dstE	ALU output
m_valM	M_dstM	Memory output
M_valE	M_dstE	Pending write to port E in memory stage
W_valM	W_dstM	Pending write to port M in write-back stage
W_valE	W_dstE	Pending write to port E in write-back stage

The complexity of this stage is with the forwarding logic.

The block labeled “Sel+Fwd A” serves two roles:

- It merges the valP signal into the valA signal for later stages in order to reduce the amount of state in the pipeline register.

• It also implements the forwarding logic for source operand valA.
Same process repeats for Fwd B.

Implementation:

●This takes the inputs from the previous PC selection and Fetch stage in order to process it further.
●The inputs required for data forwarding are also taken which include **e_dstE**, **e_valE**, **M_dstE**, **M_valE**, **M_dstM**, **m_valM**, **W_dstM**, **W_valM**, **W_dstE** and **W_valE**.
●**Sel+Fwd A** and **Fwd B** blocks help in data forwarding and directly give the values for valA or valB from the execute, memory or writeback stages as part of data forwarding.
●The outputs obtained from this block include **E_stat**, **E_ifun**, **E_icode**, **E_valA**, **E_valB**, **E_valC**, **dstE**, **dstM**, **srcA** and **srcB**
●Inputs **D_icode**, **D_ifun**, **D_rA**, **D_rB**, **D_valC**, **D_stat**, **D_valP** are taken similar to that of the sequential
●**E_stat**, **E_ifun**, **E_icode**, **E_valA**, **E_valB** and **E_valC** and all the 14 registers get updated from the writeback part.
●The registers get updated as per clock and output for **E_stat**, **E_ifun**,**E_icode**, **E_valA**, **E_valB** and **E_valC** is available as an output of the execute register.

Verilog Implementation

Code :

File Name: decode_wb.v

decode_wb.v

```
module decode_wb(clk, D_stat, D_icode, D_ifun, D_rA, D_rB, D_valC,D_valP,
                 d_srcA, d_srcB, E_bubble, E_stat, E_icode, E_ifun, E_valC,
                 E_valA, E_valB, E_dstE, E_dstM, E_srcA, E_srcB, e_dstE, e_valE,
                 M_dstE, M_dstM, M_valE, m_valM, W_dstE, W_dstM, W_valE,W_valM,W_icode,
                 reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,
                 reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14 );

//decode stage + reg
input clk;
input [0:3] D_stat;
input [3:0] D_icode,D_ifun,D_rA,D_rB;
input [63:0] D_valC,D_valP;

output reg [3:0] d_srcA,d_srcB;

//execute stage + reg
input E_bubble;
input [3:0] e_dstE;
input [63:0] e_valE;

output reg [0:3] E_stat;
output reg [3:0] E_icode,E_ifun;
output reg [63:0] E_valC,E_valA,E_valB;
output reg [3:0] E_dstE,E_dstM,E_srcA,E_srcB;

//memory stage + reg
input [3:0] M_dstE, M_dstM;
input [63:0] M_valE, m_valM;

//writeback stage + reg
input [3:0] W_icode, W_dstE, W_dstM;
input [63:0] W_valE, W_valM;

output reg[63:0] reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,
reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14;

reg [3:0] d_dstE,d_dstM;
reg [63:0] d_rvalA,d_rvalB,d_valA,d_valB,reg_mem[0:14];


initial begin
    reg_mem[0] = 0;
    reg_mem[1] = 1;
    reg_mem[2] = 2;
    reg_mem[3] = 3;
    reg_mem[4] = 4;
    reg_mem[5] = 5;
    reg_mem[6] = 6;
    reg_mem[7] = 7;
    reg_mem[8] = 8;
    reg_mem[9] = 9;
    reg_mem[10] = 10;
    reg_mem[11] = 11;
    reg_mem[12] = 12;
    reg_mem[13] = 13;
    reg_mem[14] = 14;
end
```

```
58   always@(*)
59   begin
60     d_srcA = 4'hF;
61     d_srcB = 4'hF;
62     d_dstE = 4'hF;
63     d_dstM = 4'hF;
64
65   //cmovq
66   if (D_icode == 4'b0010) begin
67     d_srcA = D_rA;
68     d_dstE = D_rB;
69     d_rvalA=reg_mem[D_rA];
70     d_rvalB=64'b0;
71   end
72
73   //irmovq
74   else if (D_icode == 4'b0011) begin
75     d_dstE = D_rB;
76     d_rvalB=64'b0;
77   end
78
79   //rmmovq
80   else if (D_icode == 4'b0100) begin
81     d_srcA = D_rA;
82     d_srcB = D_rB;
83     d_rvalA=reg_mem[D_rA];
84     d_rvalB=reg_mem[D_rB];
85   end
86
87   //mrmovq
88   else if (D_icode == 4'b0101) begin
89     d_srcB = D_rB;
90     d_dstM = D_rA;
91     d_rvalB=reg_mem[D_rB];
92   end
93
94   //OPq
95   else if (D_icode == 4'b0110) begin
96     d_srcA = D_rA;
97     d_srcB = D_rB;
98     d_dstE = D_rB;
99     d_rvalA=reg_mem[D_rA];
100    d_rvalB=reg_mem[D_rB];
101  end
102
103  //call
104  else if (D_icode == 4'b1000) begin
105    d_srcB = 4;
106    d_dstE = 4;
107    d_rvalB=reg_mem[4];
```

```
108  end
109
110  //ret
111  else if (D_icode == 4'b1001) begin
112    d_srcA = 4;
113    d_srcB = 4;
114    d_dstE = 4;
```

```
    d_rvalA=reg_mem[4];
    d_rvalB=reg_mem[4];
  end

  //pushq
  else if (D_icode == 4'b1010) begin
    d_srcA = D_rA;
    d_srcB = 4;
    d_dstE = 4;
    d_rvalA=reg_mem[D_rA];
    d_rvalB=reg_mem[4];
  end

  //popq
  else if (D_icode == 4'b1011) begin
    d_srcA = 4;
    d_srcB = 4;
    d_dstE = 4;
    d_dstM = D_rA;
    d_rvalA=reg_mem[4];
    d_rvalB=reg_mem[4];
  end

   // Forwarding A
   //jxx or call
  if(D_icode==4'b0111 | D_icode == 4'b1000)
      d_valA = D_valP;
  else if(d_srcA==e_dstE & e_dstE!=4'hF)
      d_valA = e_valE;
  else if(d_srcA==M_dstM & M_dstM!=4'hF)
      d_valA = m_valM;
  else if(d_srcA==W_dstM & W_dstM!=4'hF)
      d_valA = W_valM;
  else if(d_srcA==M_dstE & M_dstE!=4'hF)
      d_valA = M_valE;
  else if(d_srcA==W_dstE & W_dstE!=4'hF)
      d_valA = W_valE;
  else
      d_valA = d_rvalA;

  // Forwarding B
  //  from execute
  if(d_srcB==e_dstE & e_dstE!=4'hF)
      d_valB = e_valE;

  // from memory
  else if(d_srcB==M_dstM & M_dstM!=4'hF)
      d_valB = m_valM;

```

```
  // memory value from write back stage
  else if(d_srcB==W_dstM & W_dstM!=4'hF)
      d_valB = W_valM;

  // execute value from memory stage
  else if(d_srcB==M_dstE & M_dstE!=4'hF)
      d_valB = M_valE;

  // execute value from write back stage
  else if(d_srcB==W_dstE & W_dstE!=4'hF)
```

```
174     d_valB = W_valE;
175 else
176     d_valB = d_rvalB;
177 end
178
179 always@(posedge clk)
180 begin
181     if(E_bubble)
182     begin
183       E_stat <= 4'b1000;
184       E_icode <= 4'b0001;
185       E_ifun <= 4'b0000;
186       E_valC <= 4'b0000;
187       E_valA <= 4'b0000;
188       E_valB <= 4'b0000;
189       E_dstE <= 4'hF;
190       E_dstM <= 4'hF;
191       E_srcA <= 4'hF;
192       E_srcB <= 4'hF;
193     end
194
195     // Execute register update
196     else
197     begin
198       E_stat <= D_stat;
199       E_icode <= D_icode;
200       E_ifun <= D_ifun;
201       E_valC <= D_valC;
202       E_valA <= d_valA;
203       E_valB <= d_valB;
204       E_srcA <= d_srcA;
205       E_srcB <= d_srcB;
206       E_dstE <= d_dstE;
207       E_dstM <= d_dstM;
208     end
209 end
210
211 // writeback
212 always@(posedge clk) begin
```

```
214 //cmovxx
215 if(W_icode==4'b0010)
216 begin
217     reg_mem[W_dstE]=W_valE;
218 end
219
220 //irmovq
221 else if(W_icode==4'b0011)
222 begin
223     reg_mem[W_dstE]=W_valE;
224 end
225
226 //mrmovq
227 else if(W_icode==4'b0101)
228 begin
229     reg_mem[W_dstM] = W_valM;
230 end
231
```

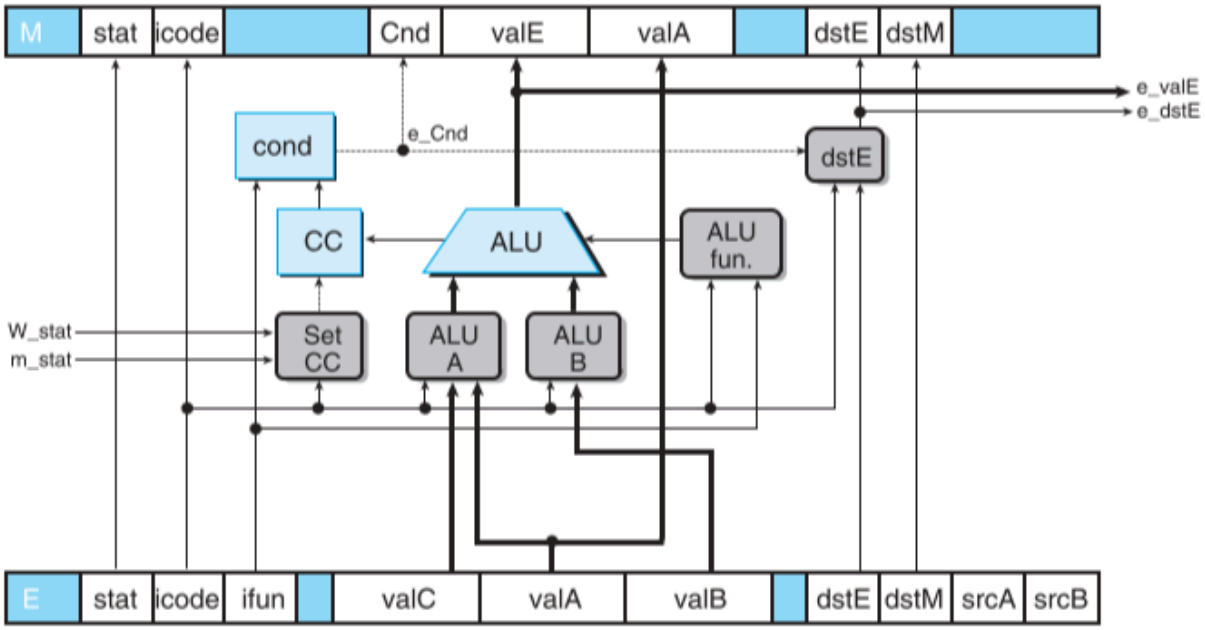
```
231
232     //OPq
233     else if(W_icode==4'b0110)
234     begin
235         reg_mem[W_dstE] = W_valE;
236     end
237
238     //call
239     else if(W_icode==4'b1000)
240     begin
241         reg_mem[W_dstE] = W_valE;
242     end
243
244     //ret
245     else if(W_icode==4'b1001)
246     begin
247         reg_mem[W_dstE] = W_valE;
248     end
249
250     //pushq
251     else if(W_icode==4'b1010)
252     begin
253         reg_mem[W_dstE] = W_valE;
254     end
255
256     //popq
257     else if(W_icode==4'b1011)
258     begin
259         reg_mem[W_dstE] = W_valE;
260         reg_mem[W_dstM] = W_valM;
261     end
```

```
262
263     reg_0 <= reg_mem[0];
264     reg_1 <= reg_mem[1];
265     reg_2 <= reg_mem[2];
266     reg_3 <= reg_mem[3];
267     reg_4 <= reg_mem[4];
268     reg_5 <= reg_mem[5];
269     reg_6 <= reg_mem[6];
270     reg_7 <= reg_mem[7];
271     reg_8 <= reg_mem[8];
272     reg_9 <= reg_mem[9];
273     reg_10 <= reg_mem[10];
274     reg_11 <= reg_mem[11];
275     reg_12 <= reg_mem[12];
276     reg_13 <= reg_mem[13];
277     reg_14 <= reg_mem[14];
278     end
279     endmodule
```

Execute:

One difference between Sequential and Pipeline is that the logic labeled “Set CC,” which determines whether or not to update the condition codes, has signals m_stat and W_stat as inputs.

These signals are used to detect cases where an instruction causing an exception is passing through later pipeline stages, and therefore any updating of the condition codes should be suppressed.



Implementation:

●This stage takes inputs as the outputs from the execute pipelined register which include **E_stat**, **E_ifun**, **E_icode**, **E_valA**, **E_valB**, **E_valC**, **E_dstE**, **E_dstM**, **W_stat** and **m_stat.**

●The value of **e_dstE** is computed based on the **e_Cnd** which will make it either E_dstE or an empty register.
●**W_stat** and **m_stat** take care to not change the conditional codes when an instruction such as halt occurs.
●The outputs obtained from this block include **M_stat**, **M_icode**, **Cnd**, **M_valE**, **M_valA**, **M_dstE**, **M_dstM**, **e_valE** and **e_dstE**.

Verilog Implementation

Code :

File Name: execute.v

```
`include "alu.v"
module execute(clk,E_stat,E_icode,E_ifun,E_valC,E_valA,E_valB,E_dstE,E_dstM,
               M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,M_bubble,
               W_stat,e_valE,e_dstE,e_cnd,m_stat,SetCC);

//inputs
input clk, M_bubble, SetCC;
input [0:3] E_stat,m_stat, W_stat;
input [3:0] E_icode,E_ifun,E_dstE,E_dstM;
input [63:0] E_valC,E_valA,E_valB;

//outputs
output reg M_cnd;
output reg [0:3] M_stat;
output reg [3:0] M_icode, M_dstE,M_dstM,e_dstE;
output reg [63:0] M_valE,M_valA,e_valE;
output reg e_cnd=1;

reg [2:0] CC_out = 3'b000;      // o_f,s_f,z_f condition codes

wire overflow;
wire overflow1;

wire [63:0] valE_cmov, valE_CB,valE_opq, valE_inc, valE_dec;


//assigning condition codes
wire z_f, s_f, o_f;
assign z_f = CC_out[0];
assign s_f = CC_out[1];
assign o_f = CC_out[2];

always @*
begin
    if (E_icode == 4'b0010 || E_icode == 4'b0111)
    begin
        // unconditional
```

```
        if (E_ifun == 4'b0000)
        begin
            e_cnd = 1;
        end

        // less than equal to
        else if (E_ifun == 4'b0001)
        begin
            e_cnd = (o_f ^ s_f) | z_f;
        end

        // less than
        else if (E_ifun == 4'b0010)
```

```
        begin
            e_cnd = o_f ^ s_f;
        end

        // equal
        else if (E_ifun == 4'b0011)
        begin
            e_cnd = z_f;
        end

        // not equal
        else if (E_ifun == 4'b0100)
        begin
            e_cnd = ~z_f;
        end

        // greater than equal to
        else if (E_ifun == 4'b0101)
        begin
            e_cnd = ~(s_f ^ o_f);
        end

        // greater than
        else if (E_ifun == 4'b0110)
        begin
            e_cnd = ~(s_f ^ o_f) & ~z_f;
        end
        e_dstE = e_cnd ? E_dstE : 4'hF;
    end
    else
    begin
        e_dstE = E_dstE;
    end
end

//ALU for executing instructions
alu func1(2'b0,E_valA,E_valB,valE_cmov,overflow1);
alu func2(E_ifun[1:0],E_valA,E_valB,valE_opq,overflow);
alu func3(2'b0,E_valC,E_valB,valE_CB,overflow1);
alu func4(2'b0,64'd1,E_valB,valE_inc,overflow1);
alu func5(2'b1,64'd1,E_valB,valE_dec,overflow1);


always @*
begin
```

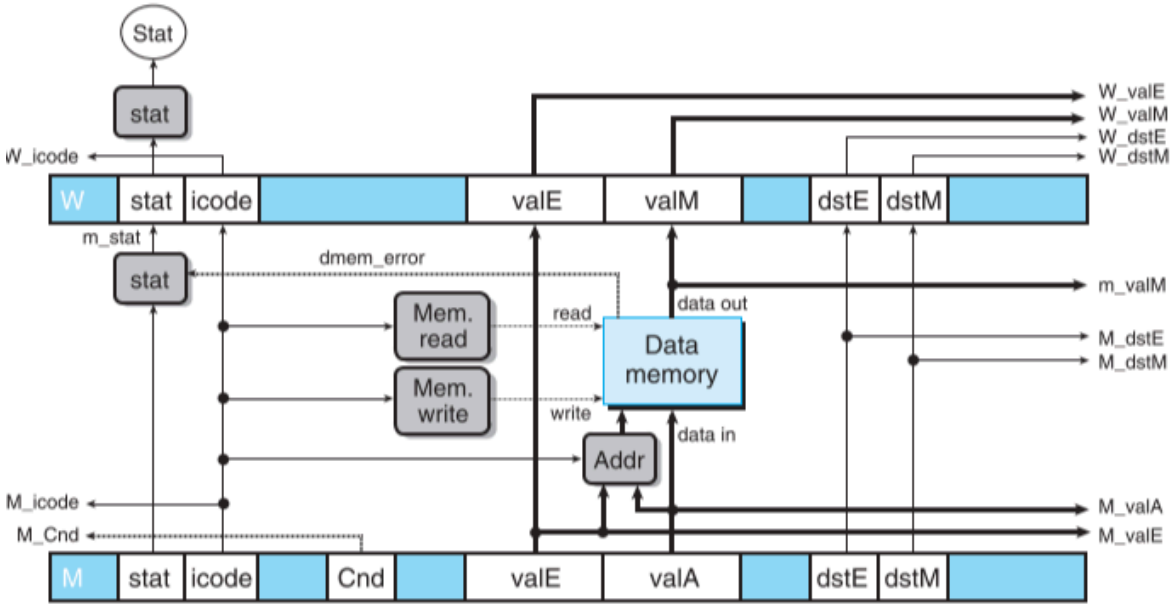
```
96      // cmovx
97      if (E_icode == 4'b0010)
98      begin
99          e_valE = valE_cmov;
100     end
101
102     // irmovq
103     else if (E_icode == 4'b0011)
104     begin
105         e_valE = valE_CB;
106     end
107
108     //rmmovq
109     else if (E_icode == 4'b0100)
110     begin
111         e_valE = valE_CB;
112     end
113
114     // mrmovq
115     else if (E_icode == 4'b0101)
116     begin
117         e_valE = valE_CB;
118     end
119
120     // opq
121     else if (E_icode == 4'b0110)
122     begin
123         e_valE = valE_opq;
124         if (SetCC)
125         begin
126             CC_out[2] = o_f;
127             CC_out[1] = e_valE[63];
128             CC_out[0] = e_valE ? 0 : 1;
129         end
130     end
131
132     // call
133     else if (E_icode == 4'b1000)
134     begin
135         e_valE = valE_dec;
136     end
137
138     // ret
139     else if (E_icode == 4'b1001)
140     begin
141             e_valE = valE_inc;
142     end
```

```
144     // pushq
145     else if (E_icode == 4'b1010)
146     begin
147         e_valE = valE_dec;
148     end
```

```
149     // popq
150     else if (E_icode == 4'b1011)
151     begin
152         e_valE = valE_inc;
153     end
154     else
155     begin
156         e_valE = 0;
157     end
158 end
159 
160 
161 always@(posedge clk)
162 begin
163     if(M_bubble)
164     begin
165         M_stat <= 4'b1000;
166         M_icode <= 4'b001;
167         M_cnd <= 1;
168         M_valE <= 0;
169         M_valA <= 0;
170         M_dstE <= 4'hF;
171         M_dstM <= 4'hF;
172     end
173     else
174     begin
175         M_stat <= E_stat;
176         M_icode <= E_icode;
177         M_cnd <= e_cnd;
178         M_valE <= e_valE;
179         M_valA <= E_valA;
180         M_dstE <= e_dstE;
181         M_dstM <= E_dstM;
182     end
183 end
184 
185 
186 endmodule
```

Memory:

One Major Difference which can be observed from the Pipelined Hardware is absence of block labeled “Data”. This block serves to select between data sources valP (for call instructions) and valA, but this selection is now performed by the block labeled “Sel+Fwd A” in the decode stage.



Implementation:

- This takes the inputs as the outputs from the memory pipelined register which include **M_stat**, **M_icode**, **M_Cnd**, **M_valE, M_valA, M_dstE and M_dstM**
- This block functions the same way as that of the sequential memory block

which takes **M_icode**, **M_Cnd**, **M_valE, M_valA, M_dstE and M_dstM** as

inputs and gives **W_icode, W_valE, W_valM, W_dstE, W_dstM and**
m_valM as outputs.

●This also updates the **m_stat** to **M_stat** if dmem_error has not occurred.

●The outputs obtained from this block include **W_stat, W_icode, W_valE,**
W_valM, W_dstE, W_dstM and m_valM.

Verilog Implementation

Code :

File Name: memory.v

memory.v

```
1   module memory(clk, M_stat, M_icode, M_cnd, M_valE, M_valA, M_dstE, M_dstM,
2                 m_valM,m_stat,W_stat,W_icode,W_valE,W_valM,W_dstE,W_dstM,W_stall);
3
4   //inputs
5   input clk, M_cnd, W_stall;
6   input [0:3] M_stat;
7   input [3:0] M_icode, M_dstE, M_dstM;
8   input [63:0] M_valE, M_valA;
9
10  //outputs
11  output reg [0:3] W_stat, m_stat;
12  output reg [3:0] W_icode, W_dstE, W_dstM;
13  output reg [63:0] m_valM, W_valE, W_valM;
14
15  reg mem_read;
16  reg [63:0] data_memory[0:1023];
17  reg data_mem_error = 0;
18
19  always @*
20  begin
21      if(data_mem_error)
22          m_stat = 4'b0010;
23      else
24          m_stat = M_stat;
25  end
26
27  always @*
28  begin
29      // rmmovq, call, pop
30      if(M_icode == 4'b0100 | M_icode == 4'b1000 | M_icode == 4'b1011)
```

```
31      begin
32          mem_read = 1;
33      end
34      else
35      begin
36          mem_read = 0;
37      end
38  end
39
40  always@(posedge clk)
41  begin
42      if(M_valE > 1023 & mem_read)
43      begin
44          data_mem_error = 1;
45      end
46      else
47      begin
```

```
32          mem_read = 1;
33      end
34      else
35      begin
36          mem_read = 0;
37      end
38  end
39
40  always@(posedge clk)
41  begin
42      if(M_valE > 1023 & mem_read)
43      begin
44          data_mem_error = 1;
45      end
46      else
47      begin
48
49          // rmmovq
50          if(M_icode == 4'b0100)
51          begin
52              data_memory[M_valE] <= M_valA;
53          end
54
55          // mrmovq
56          if(M_icode == 4'b0101)
57          begin
58              m_valM = data_memory[M_valE];
59          end
60
61          // call
62          if(M_icode == 4'b1000)
63          begin
64              data_memory[M_valE] <= M_valA;
65          end
66
67          // return
68          if(M_icode == 4'b1001)
```

```
69          begin
70              m_valM = data_memory[M_valA];
71          end
72
73          // pushq
74          if(M_icode == 4'b1010)
75          begin
76              data_memory[M_valE] <= M_valA;
77          end
78
79          // popq
80          if(M_icode == 4'b1011)
81          begin
82              m_valM = data_memory[M_valA];
83          end
84
85      end
86
87  end
88
89  always@(posedge clk)
90  begin
91      if(W_stall)
92      begin
93
94      end
95      else
96      begin
97          W_stat <= m_stat;
98          W_icode <= M_icode;
99          W_valE <= M_valE;
100         W_valM <= m_valM;
101         W_dstE <= M_dstE;
102         W_dstM <= M_dstM;
103     end
104 end
105 endmodule
```

Pipeline Control:

There is a huge need to see the overall workflow of the pipeline which involves observing the parallelism of different stages and countering the Hazards which occur. Handling Hazards is therefore very important and constitutes the last stages of the Pipeline Implementation. This logic must handle the following four control cases:

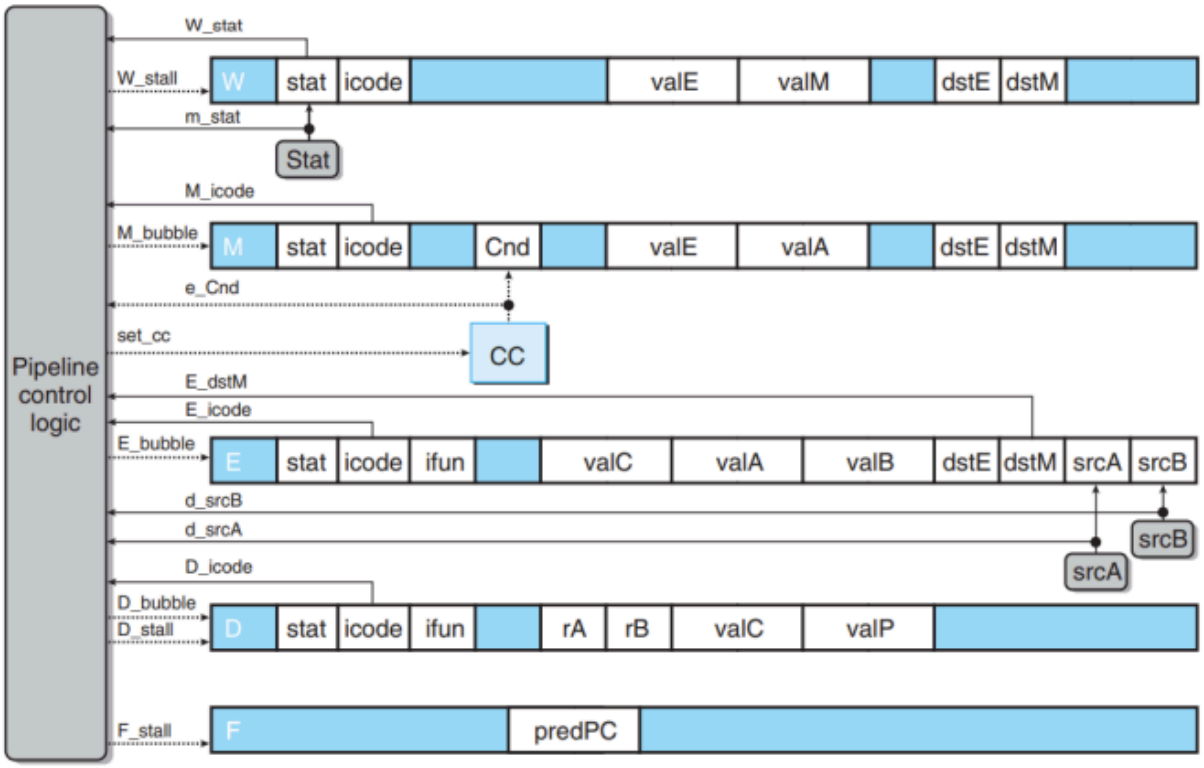
Processing ret: The pipeline must stall until the ret instruction reaches the write-back stage.

Load/use hazards: The pipeline must stall for one cycle between an instruction that reads a value from memory and an instruction that uses this value.

Mispredicted branches: By the time the branch logic detects that a jump should not have been taken, several instructions at the branch target will have started down the pipeline. These instructions must be removed from the pipeline.

	Pipeline register				
Condition	F	D	E	M	W
Processing `ret`	stall	bubble	normal	normal	normal
Load/use hazard	stall	stall	bubble	normal	normal
Mispredicted branch	normal	bubble	bubble	normal	normal

Therefore the final pipeline implementation structure involving all the 5 stages that is Fetch , Decode , Memory , Execute , Write Back along with handling Data Hazards give the below one.



Implementation:

- This takes the inputs as **m_stat**, **W_stat**, **D_icode**, **E_icode**, **M_icode**, **d_srcA**,**d_srcB**, **d_dstM** and **e_cnd**
- The outputs obtained are as **F_stall**, **D_stall**, **D_bubble**, **E_bubble**, **M_bubble**, **W_stall** and **set_cc**.

Verilog Implementation

Code :

File Name: pipecontrolling.v

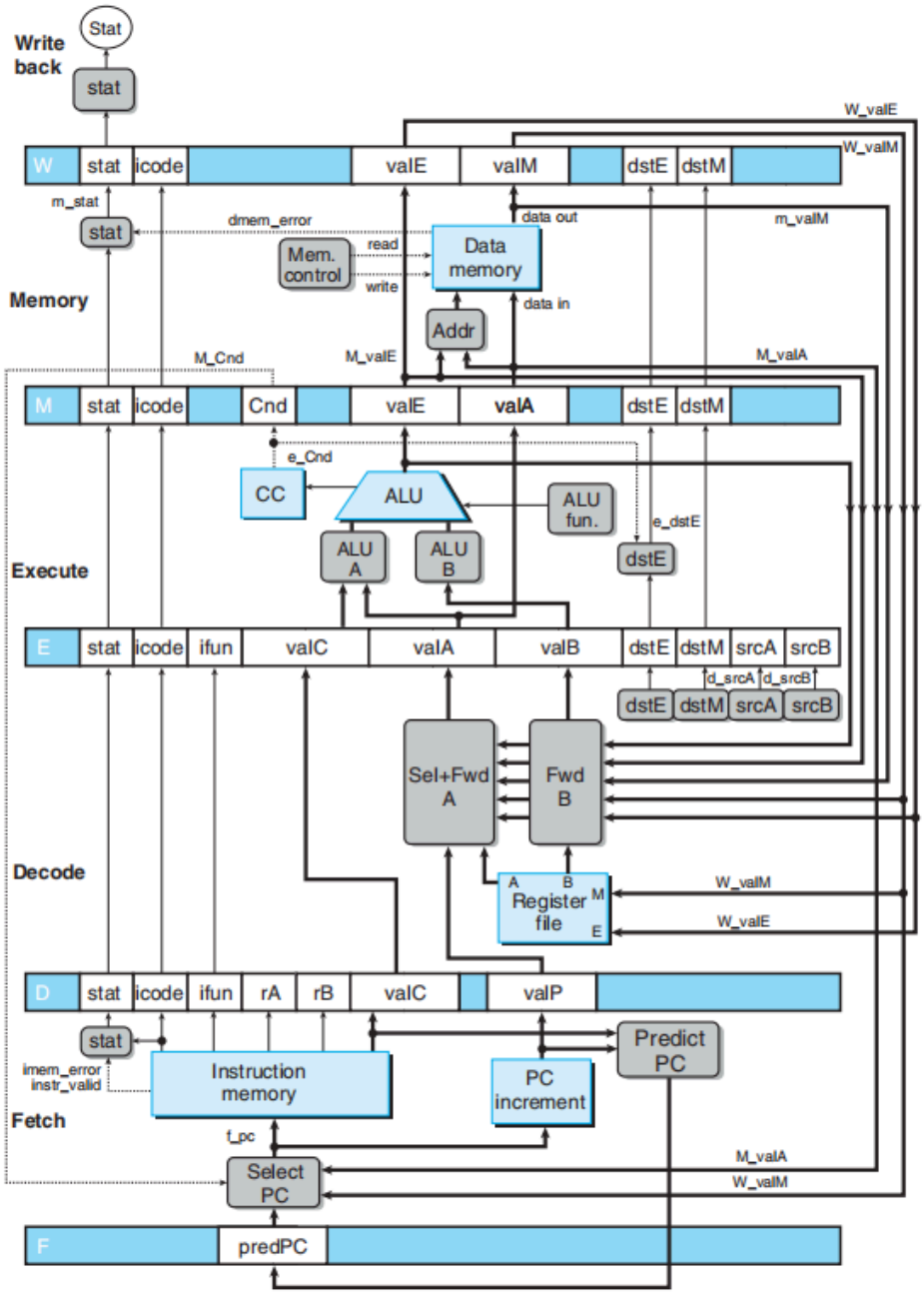
```
module pipe_controlling(D_icode, d_srcA, d_srcB, E_icode, E_dstM, e_cnd,M_icode, m_stat, W_stat,
                        F_stall,D_stall,D_bubble,E_bubble,M_bubble,W_stall, SetCC);


//inputs
input [3:0] D_icode,E_icode,M_icode,d_srcA,d_srcB,E_dstM;
input [0:3] m_stat, W_stat;
input e_cnd;
```

```
 //outputs
 output reg F_stall, D_stall, D_bubble, E_bubble, M_bubble, W_stall;
 output reg SetCC;

 always@*
 begin
    F_stall = 0;D_stall = 0;D_bubble = 0;
    E_bubble = 0;M_bubble = 0;W_stall = 0;
    SetCC = 1;
    if(E_icode==4'b0111 & !e_cnd)
    begin
        D_bubble = 1;
        E_bubble = 1;
    end
    else if((E_icode == 4'b0101 | E_icode == 4'b1011) & (E_dstM==d_srcA | E_dstM==d_srcB))
    begin
        F_stall = 1;
        D_stall = 1;
        E_bubble = 1;
    end
    else if(E_icode == 4'b1001 | M_icode == 4'b1001 | D_icode == 4'b1001)
    begin
        F_stall = 1;
        D_bubble = 1;
    end
    else if(E_icode == 4'b0000 | m_stat!=4'b1000 | W_stat!=4'b1000)
    begin
        SetCC = 0;
    end

    end

 endmodule
```

Pipeline Implementation

Combining all the above steps , the final Pipeline Implementation of the processor is finished .

The steps such as Fetch , Decode , Execute , Memory , Write back are combined and executed parallelly for various instructions.



Verilog Implementation

Testing :

File Name: processor.v

```
`include "fetch.v"
`include "decode_wb.v"
`include "execute.v"
`include "memory.v"
`include "pipe_controlling.v"

module processor;
reg clk;
reg[63:0] F_predPC;
reg[7:0] instr_mem[0:2048];
reg [0:3] stat = 4'b1000;    // AOK, HLT, ADR, INS

wire F_stall, D_stall, D_bubble, E_bubble, M_bubble, W_stall, SetCC;
wire M_cnd, e_cnd;
wire [0:3] D_stat,E_stat,M_stat,W_stat,m_stat;
wire [3:0] D_icode, D_ifun, D_rA, D_rB, d_srcA, d_srcB,
           E_icode, E_ifun, E_dstE, E_dstM, E_srcA, E_srcB, e_dstE,
           M_icode, M_dstE, M_dstM,
           W_icode, W_dstE, W_dstM ;

wire [63:0] reg_0,reg_1,reg_2,reg_3,reg_4,reg_5,reg_6,reg_7,
            reg_8,reg_9,reg_10,reg_11,reg_12,reg_13,reg_14;

wire [63:0] f_predPC, D_valC, D_valP,
            E_valC, E_valA, E_valB, e_valE,
            M_valE, M_valA, m_valM,
            W_valE, W_valM;

 // stat is calculated at every stage of the processor
always@(W_stat)
begin
    stat = W_stat;
end

  // always halt logic
always@(stat)
begin
    if(stat==4'b0001) begin
      $display("Instruction Error - INS");
      $finish;
    end

    else if(stat==4'b0010) begin
      $display("Address Error - ADR");
      $finish;
    end

    else if(stat==4'b0100) begin
      $display("Halting");
      $finish;
```

processor.v

```
37  begin
51      end
52  end
53
54  always #10 clk = ~clk;
55  always @(posedge clk) F_predPC <= f_predPC;
56
57  fetch fetch(.clk(clk), .f_predPC(f_predPC), .F_predPC(F_predPC),.D_stat(D_stat), .D_icode(D_icode), .D_ifun(D_ifun), .D_rA(D_rA), .D_rB(D_rB), .D_valC(D_valC),.D_valP(D_valP),
58          .M_icode(M_icode), .M_cnd(M_cnd), .M_valA(M_valA), .W_icode(W_icode), .W_valM(W_valM),
59          .F_stall(F_stall),.D_stall(D_stall),.D_bubble(D_bubble));
60
61
62  decode_wb decode_wb(.clk(clk), .D_stat(D_stat),.D_icode(D_icode), .D_ifun(D_ifun), .D_rA(D_rA), .D_rB(D_rB), .D_valC(D_valC),.D_valP(D_valP),
63              .d_srcA(d_srcA),.d_srcB(d_srcB), .E_bubble(E_bubble), .E_stat(E_stat), .E_icode(E_icode), .E_ifun(E_ifun),.E_valC(E_valC),
64              .E_valA(E_valA), .E_valB(E_valB), .E_dstE(E_dstE), .E_dstM(E_dstM), .E_srcA(E_srcA), .E_srcB(E_srcB), .e_dstE(e_dstE), .e_valE(e_valE),
65              .M_dstE(M_dstE), .M_dstM(M_dstM), .M_valE(M_valE), .m_valM(m_valM), .W_dstE(W_dstE), .W_dstM(W_dstM), .W_valE(W_valE),.W_valM(W_valM),.W_icode(W_icode),
66              .reg_0(reg_0),.reg_1(reg_1),.reg_2(reg_2),.reg_3(reg_3),.reg_4(reg_4),.reg_5(reg_5),.reg_6(reg_6),.reg_7(reg_7),
67              .reg_8(reg_8),.reg_9(reg_9),.reg_10(reg_10),.reg_11(reg_11),.reg_12(reg_12),.reg_13(reg_13),.reg_14(reg_14));
68
69  execute execute(.clk(clk),.E_stat(E_stat),.E_icode(E_icode),.E_ifun(E_ifun),.E_valC(E_valC),.E_valA(E_valA),.E_valB(E_valB),.E_dstE(E_dstE),.E_dstM(E_dstM),
70          .M_stat(M_stat),.M_icode(M_icode),.M_cnd(M_cnd),.M_valE(M_valE),.M_valA(M_valA),.M_dstE(M_dstE),.M_dstM(M_dstM),.M_bubble(M_bubble),
71          .W_stat(W_stat),.e_valE(e_valE),.e_dstE(e_dstE),.e_cnd(e_cnd),.m_stat(m_stat),.SetCC(SetCC));
72
73  memory memory(.clk(clk), .M_stat(M_stat), .M_icode(M_icode), .M_cnd(M_cnd), .M_valE(M_valE), .M_valA(M_valA), .M_dstE(M_dstE), .M_dstM(M_dstM),
74          .m_valM(m_valM),.m_stat(m_stat),.W_stat(W_stat),.W_icode(W_icode),.W_valE(W_valE),.W_valM(W_valM),.W_dstE(W_dstE),.W_dstM(W_dstM),.W_stall(W_stall));
75
76  pipe_controlling pipe_controlling(.D_icode(D_icode), .d_srcA(d_srcA), .d_srcB(d_srcB), .E_icode(E_icode), .E_dstM(E_dstM), .e_cnd(e_cnd),.M_icode(M_icode), .m_stat(m_stat), .W_stat(W_stat),
77              .F_stall(F_stall),.D_stall(D_stall),.D_bubble(D_bubble),.E_bubble(E_bubble),.M_bubble(M_bubble),.W_stall(W_stall), .SetCC(SetCC));
78
79
80
81  initial begin
82
83      $dumpfile("processor.vcd");
84      $dumpvars(0,processor);
85      F_predPC=64'd0;
86      clk=0;
87
88      $monitor("clk=%d f_predPC=%d F_predPC=%d D_icode=%d,E_icode=%d, M_icode=%d, ifun=%d,rax=%d,rdx=%d,rbx=%d,rcx=%d\n",clk,f_predPC,F_predPC, D_icode,E_icode,M_icode,D_ifun,reg_0,reg_2,reg_3,reg_1);
89      //$monitor("clk=%d f_predPC=%d F_predPC=%d rax=%d rdx=%d rbx=%d rsp=%d rsi=%d\n",clk,f_predPC,F_predPC,reg_0,reg_2,reg_3,reg_4,reg_6);
90  end
91  endmodule
```

Test Case 1 `opq , halt along with irmovq`:

```
2   ////////////////////////////////// test1_pipe ////////////////////////////////
3   // Reference for 1.txt
4
5   //opq , halt along with irmovq
6
7   //irmovq $0x100, %rbx
8   instr_mem[0]=8'b00110000;
9   instr_mem[1]=8'b11110011;
10  instr_mem[2]=8'b00000000;
11  instr_mem[3]=8'b00000001;
12  instr_mem[4]=8'b00000000;
13  instr_mem[5]=8'b00000000;
14  instr_mem[6]=8'b00000000;
15  instr_mem[7]=8'b00000000;
16  instr_mem[8]=8'b00000000;
17  instr_mem[9]=8'b00000000;
18
```

```
19    //irmovq $0x200, %dx
20    instr_mem[10]=8'b00110000;
21    instr_mem[11]=8'b11110010;
22    instr_mem[12]=8'b00000000;
23    instr_mem[13]=8'b00000010;
24    instr_mem[14]=8'b00000000;
25    instr_mem[15]=8'b00000000;
26    instr_mem[16]=8'b00000000;
27    instr_mem[17]=8'b00000000;
28    instr_mem[18]=8'b00000000;
29    instr_mem[19]=8'b00000000;
30
31    //addq %rdx, %rbx
32    instr_mem[20]=8'b01100000;
33    instr_mem[21]=8'b00100011;
34
35    //halt
36    instr_mem[22]=8'b00000000;
```

Output:

```
clk=0 f_predPC=          10 F_predPC=           0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=          x,rdx=          x,rbx=          x,rcx=          x
clk=1 f_predPC=          20 F_predPC=          10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          20 F_predPC=          10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          22 F_predPC=          20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 6, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 6, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=  281474976710656,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=  281474976710656,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=  562949953421312,rbx=  281474976710656,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=  562949953421312,rbx=  281474976710656,rcx=          1
Halting
processor.v:50: $finish called at 130 (1s)
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=  562949953421312,rbx=  844424930131968,rcx=          1
```

Test Case 2 `rrmovq,rmmovq,nop,halt with opq`

```
////////////////////////////// test2_pipe ////////////////////////////
//rrmovq , rmmovq , nop , halt along with opq

// //opq add
instr_mem[0] = 8'b01100000;

//%rax = 1 %rbx = 3
instr_mem[1] = 8'b00010011;
// rrmovq
instr_mem[2] = 8'b00100000;
instr_mem[3] = 8'b00010011;
// src = %rax dest = %rbx

//rmmovq
instr_mem[4] = 8'b01000000;
//rax and (rbx)
instr_mem[5] = 8'b00010011;
//VALC
instr_mem[6] = 8'b00001111;
instr_mem[7] = 8'b00000000;
instr_mem[8] = 8'b00000000;
instr_mem[9] = 8'b00000000;
instr_mem[10] = 8'b00000000;
instr_mem[11] = 8'b00000000;
instr_mem[12] = 8'b00000000;
instr_mem[13] = 8'b00000000;

//no operation
instr_mem[14] = 8'b00010000;

// //halt
// instr_mem[15] = 8'b00000000;
```

Output:

```
clk=0 f_predPC=              2 F_predPC=              0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=          x,rdx=          x,rbx=          x,rcx=          x
clk=1 f_predPC=              4 F_predPC=              2 D_icode= 6,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=              4 F_predPC=              2 D_icode= 6,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=             14 F_predPC=              4 D_icode= 2,E_icode= 6, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=             14 F_predPC=              4 D_icode= 2,E_icode= 6, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=             15 F_predPC=             14 D_icode= 4,E_icode= 2, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=             15 F_predPC=             14 D_icode= 4,E_icode= 2, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=             15 F_predPC=             15 D_icode= 1,E_icode= 4, M_icode= 2, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=             15 F_predPC=             15 D_icode= 1,E_icode= 4, M_icode= 2, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=             15 F_predPC=             15 D_icode= 0,E_icode= 1, M_icode= 4, ifun= 0,rax=          0,rdx=          2,rbx=          4,rcx=          1
clk=0 f_predPC=             15 F_predPC=             15 D_icode= 0,E_icode= 1, M_icode= 4, ifun= 0,rax=          0,rdx=          2,rbx=          4,rcx=          1
clk=1 f_predPC=             15 F_predPC=             15 D_icode= 0,E_icode= 0, M_icode= 1, ifun= 0,rax=          0,rdx=          2,rbx=          1,rcx=          1
clk=0 f_predPC=             15 F_predPC=             15 D_icode= 0,E_icode= 0, M_icode= 1, ifun= 0,rax=          0,rdx=          2,rbx=          1,rcx=          1
Address Error - ADR
processor.v:45: $finish called at 130 (1s)
clk=1 f_predPC=             15 F_predPC=             15 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          2,rbx=          1,rcx=          1
```

Test Case 3 `jump halt with irmovq`

```
////////////////////////////// test3_pipe ////////////////////////////

//jump, halt along with irmovq

//irmovq $0xc, %rbx
//3 0
  instr_mem[0]=8'b00110000;
  //F rB=3
  instr_mem[1]=8'b00000011;
  instr_mem[2]=8'b00000000;
  instr_mem[3]=8'b00000000;
  instr_mem[4]=8'b00000000;
  instr_mem[5]=8'b00000000;
  instr_mem[6]=8'b00000000;
  instr_mem[7]=8'b00000000;
  instr_mem[8]=8'b00000000;
  instr_mem[9]=8'b00001100;
// Val=12

  //jmp :find
   //7 jxx
  instr_mem[10]=8'b01110000;
  instr_mem[11]=8'b00000000;
  instr_mem[12]=8'b00000000;
  instr_mem[13]=8'b00000000;
  instr_mem[14]=8'b00000000;
  instr_mem[15]=8'b00000000;
  instr_mem[16]=8'b00000000;
  instr_mem[17]=8'b00000000;
  instr_mem[18]=8'b00010011;

  // find:
  // opq //6 add
  instr_mem[19]=8'b01100000;
  //rA=0 rB=3
  instr_mem[20]=8'b00000011;
  instr_mem[21]=8'b00000000;
```

Output:

```
clk=0 f_predPC=          10 F_predPC=           0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=          x,rdx=          x,rbx=          x,rcx=          x
clk=1 f_predPC=          19 F_predPC=          10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          19 F_predPC=          10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          21 F_predPC=          19 D_icode= 7,E_icode= 3, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          21 F_predPC=          19 D_icode= 7,E_icode= 3, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          21 F_predPC=          21 D_icode= 6,E_icode= 7, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          21 F_predPC=          21 D_icode= 6,E_icode= 7, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 6, M_icode= 7, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 6, M_icode= 7, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=         12,rcx=          1
clk=0 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=         12,rcx=          1
clk=1 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          2,rbx=         12,rcx=          1
clk=0 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          2,rbx=         12,rcx=          1

Halting
processor.v:50: $finish called at 130 (1s)
clk=1 f_predPC=          21 F_predPC=          21 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          2,rbx=         12,rcx=          1
```

Test Case 4 `opq , halt along with irmovq`:

```
////////////////////////////// test4_pipe ////////////////////////////

//opq , halt along with irmovq

//irmovq $0b11, %rbx
instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000000;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000000;
instr_mem[9]=8'b000000011;

//irmovq $0b10, %dx
instr_mem[10]=8'b00110000;
instr_mem[11]=8'b11110010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000000;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000000;
instr_mem[19]=8'b00000010;

//addq %rdx, %rbx
instr_mem[20]=8'b01100001;
instr_mem[21]=8'b00100011;

//halt
instr_mem[22]=8'b00000000;
```

Output:

```
clk=0 f_predPC=              10 F_predPC=               0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=                x,rdx=              x,rbx=              x,rcx=              x
clk=1 f_predPC=              20 F_predPC=              10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              20 F_predPC=              10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              22 F_predPC=              20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              22 F_predPC=              20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              22 F_predPC=              22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 1,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              22 F_predPC=              22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 1,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 6, M_icode= 3, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 6, M_icode= 3, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=                0,rdx=              2,rbx=              3,rcx=              1

Halting
processor.v:50: $finish called at 130 (1s)
clk=1 f_predPC=              22 F_predPC=              22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=                0,rdx=              2,rbx=              1,rcx=              1
```

Test Case 5 cmovxx,halt irmovq with opq

```
////////////////////////////// test5_pipe ////////////////////////////

//cmovxx , halt along with irmovq and opq

//addq %rdx, %rbx
instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000000;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000101;
instr_mem[9]=8'b00001000;
instr_mem[10]=8'b00110000;
instr_mem[11]=8'b11110010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000000;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000000;
instr_mem[19]=8'b00000001;

instr_mem[20]=8'b01100000;
instr_mem[21]=8'b00100011;

instr_mem[22] = 8'b00010000;

//cmov
// 2 fn
instr_mem[23]=8'b00100000;
// rA rB
instr_mem[24]=8'b00110100;

//halt
instr_mem[25] = 8'b00000000;
```

Output:

```
clk=0 f_predPC=                10 F_predPC=                 0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=                x,rdx=                x,rbx=                x,rcx=                x
clk=1 f_predPC=                20 F_predPC=                10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                20 F_predPC=                10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                22 F_predPC=                20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                22 F_predPC=                20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                23 F_predPC=                22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                23 F_predPC=                22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                25 F_predPC=                23 D_icode= 1,E_icode= 6, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                25 F_predPC=                23 D_icode= 1,E_icode= 6, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                25 F_predPC=                25 D_icode= 2,E_icode= 1, M_icode= 6, ifun= 0,rax=                0,rdx=                2,rbx=             1288,rcx=                1
clk=0 f_predPC=                25 F_predPC=                25 D_icode= 2,E_icode= 1, M_icode= 6, ifun= 0,rax=                0,rdx=                2,rbx=             1288,rcx=                1
clk=1 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 2, M_icode= 1, ifun= 0,rax=                0,rdx=                1,rbx=             1288,rcx=                1
clk=0 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 2, M_icode= 1, ifun= 0,rax=                0,rdx=                1,rbx=             1288,rcx=                1
clk=1 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 0, M_icode= 2, ifun= 0,rax=                0,rdx=                1,rbx=             1289,rcx=                1
clk=0 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 0, M_icode= 2, ifun= 0,rax=                0,rdx=                1,rbx=             1289,rcx=                1
clk=1 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=                0,rdx=                1,rbx=             1289,rcx=                1
clk=0 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=                0,rdx=                1,rbx=             1289,rcx=                1

Halting
processor.v:50: $finish called at 170 (1s)
clk=1 f_predPC=                25 F_predPC=                25 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=                0,rdx=                1,rbx=             1289,rcx=                1
```

Test Case 6 `irmovq,halt with opq`

```
////////////////////////////// test6_pipe ////////////////////////////

 //opq , halt along with irmovq

instr_mem[0]=8'b00110000;
instr_mem[1]=8'b11110011;
instr_mem[2]=8'b00000000;
instr_mem[3]=8'b00000000;
instr_mem[4]=8'b00000000;
instr_mem[5]=8'b00000000;
instr_mem[6]=8'b00000000;
instr_mem[7]=8'b00000000;
instr_mem[8]=8'b00000000;
instr_mem[9]=8'b00000100;
instr_mem[10]=8'b00110000;
instr_mem[11]=8'b11110010;
instr_mem[12]=8'b00000000;
instr_mem[13]=8'b00000000;
instr_mem[14]=8'b00000000;
instr_mem[15]=8'b00000000;
instr_mem[16]=8'b00000000;
instr_mem[17]=8'b00000000;
instr_mem[18]=8'b00000110;
instr_mem[19]=8'b00000000;
instr_mem[20]=8'b01100010;
instr_mem[21]=8'b00100011;
instr_mem[22]=8'b00000000;
```

Output:

```
clk=0 f_predPC=          10 F_predPC=           0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=          x,rdx=          x,rbx=          x,rcx=          x
clk=1 f_predPC=          20 F_predPC=          10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          20 F_predPC=          10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          22 F_predPC=          20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 2,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 6,E_icode= 3, M_icode= 3, ifun= 2,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 6, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 6, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=          4,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=          0,rdx=          2,rbx=          4,rcx=          1
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=       1536,rbx=          4,rcx=          1
clk=0 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=       1536,rbx=          4,rcx=          1
Halting
processor.v:50: $finish called at 130 (1s)
clk=1 f_predPC=          22 F_predPC=          22 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=       1536,rbx=          0,rcx=          1
```

Test Case 7 `pushq,pop,halt with irmovq`

```
////////////////////////////// test7_pipe ////////////////////////////////////////
//push, pop, halt along with irmovq

// irmovq $0x10 %rax
    instr_mem[0]=8'h30; //3 0
    instr_mem[1]=8'b11110000; //F rA=0
    instr_mem[2]=8'h00;
    instr_mem[3]=8'h00;
    instr_mem[4]=8'h00;
    instr_mem[5]=8'h00;
    instr_mem[6]=8'h00;
    instr_mem[7]=8'h00;
    instr_mem[8]=8'h00;
    instr_mem[9]=8'h10; //V=16
// irmovq $0xc %rbx
    instr_mem[10]=8'h30; //3 0
    instr_mem[11]=8'hF3; //F rB=3
    instr_mem[12]=8'h00;
    instr_mem[13]=8'h00;
    instr_mem[14]=8'h00;
    instr_mem[15]=8'h00;
    instr_mem[16]=8'h00;
    instr_mem[17]=8'h00;
    instr_mem[18]=8'h00;
    instr_mem[19]=8'h0c; //V=12

// irmovq $0xc %rsp
    instr_mem[20]=8'h30; //3 0
    instr_mem[21]=8'hF4; //F rsp=4
    instr_mem[22]=8'h00;
    instr_mem[23]=8'h00;
    instr_mem[24]=8'h00;
    instr_mem[25]=8'h00;
    instr_mem[26]=8'h00;
    instr_mem[27]=8'h00;
    instr_mem[28]=8'h00;
    instr_mem[29]=8'h0c; //V=12
// Example: pushq %rdi (rA=6, rB=0, V=0)
    instr_mem[30] = 8'b10100000; // call push
    instr_mem[31] = 8'b0010011; // rA=6, rB=0

// Example: popq %rdx (rA=0, rB=0, V=0)
    instr_mem[32] = 8'b10110000; // call pop
    instr_mem[33] = 8'b00100000; // rA=0, rB=0

    instr_mem[34] = 8'b00000000; // halt
```

Output:

```
clk=0 f_predPC=                10 F_predPC=                 0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=                x,rdx=                x,rbx=                x,rcx=                x
clk=1 f_predPC=                20 F_predPC=                10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                20 F_predPC=                10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                30 F_predPC=                20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                30 F_predPC=                20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                32 F_predPC=                30 D_icode= 3,E_icode= 3, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                32 F_predPC=                30 D_icode= 3,E_icode= 3, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                34 F_predPC=                32 D_icode=10,E_icode= 3, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                34 F_predPC=                32 D_icode=10,E_icode= 3, M_icode= 3, ifun= 0,rax=                0,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                34 F_predPC=                34 D_icode=11,E_icode=10, M_icode= 3, ifun= 0,rax=               16,rdx=                2,rbx=                3,rcx=                1
clk=0 f_predPC=                34 F_predPC=                34 D_icode=11,E_icode=10, M_icode= 3, ifun= 0,rax=               16,rdx=                2,rbx=                3,rcx=                1
clk=1 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode=11, M_icode=10, ifun= 0,rax=               16,rdx=                2,rbx=               12,rcx=                1
clk=0 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode=11, M_icode=10, ifun= 0,rax=               16,rdx=                2,rbx=               12,rcx=                1
clk=1 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode= 0, M_icode=11, ifun= 0,rax=               16,rdx=                2,rbx=               12,rcx=                1
clk=0 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode= 0, M_icode=11, ifun= 0,rax=               16,rdx=                2,rbx=               12,rcx=                1
clk=1 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=               16,rdx=                2,rbx=               12,rcx=                1
clk=0 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=               16,rdx=                2,rbx=               12,rcx=                1

Halting
processor.v:50: $finish called at 170 (1s)
clk=1 f_predPC=                34 F_predPC=                34 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=               16,rdx=                1,rbx=               12,rcx=                1
```

Test Case 8 `mrmovq,opq,halt with irmovq`

```
////////////////////////////// test8_pipe ////////////////////////////////////////////
/irmovq, mrmovq, opq along with halt
// irmovq $0xc %rbx
   instr_mem[0]=8'h30; //3 0
   instr_mem[1]=8'hF3; //F rB=3
   instr_mem[2]=8'h00;
   instr_mem[3]=8'h00;
   instr_mem[4]=8'h00;
   instr_mem[5]=8'h00;
   instr_mem[6]=8'h00;
   instr_mem[7]=8'h00;
   instr_mem[8]=8'h00;
   instr_mem[9]=8'h0c; //V=12
/irmovq $0x10 %rax
   instr_mem[10]=8'h30; //3 0
   instr_mem[11]=8'b11110010; //F rA=0
   instr_mem[12]=8'h00;
   instr_mem[13]=8'h00;
   instr_mem[14]=8'h00;
   instr_mem[15]=8'h00;
   instr_mem[16]=8'h00;
   instr_mem[17]=8'h00;
   instr_mem[18]=8'h00;
   instr_mem[19]=8'h10; //V=16
/mrmovq
   instr_mem[20]=8'b01010000; //5 0
   instr_mem[21]=8'b00000011  ; //rA rB
   instr_mem[22]=8'b00000000; //D
   instr_mem[23]=8'b00000000; //D
   instr_mem[24]=8'b00000000; //D
   instr_mem[25]=8'b00000000; //D
   instr_mem[26]=8'b00000000; //D
   instr_mem[27]=8'b00000000; //D
   instr_mem[28]=8'b00000000; //D
   instr_mem[29]=8'b00000001; //D

  //OPq
   instr_mem[30]=8'b01100000; //6 fn
   instr_mem[31]=8'b00100011; //rA rB

   //halt
   instr_mem[32]=8'b00000000;
```

Output:

```
clk=0 f_predPC=              10 F_predPC=               0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=              x,rdx=              x,rbx=              x,rcx=              x
clk=1 f_predPC=              20 F_predPC=              10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              20 F_predPC=              10 D_icode= 3,E_icode= x, M_icode= x, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              30 F_predPC=              20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              30 F_predPC=              20 D_icode= 3,E_icode= 3, M_icode= x, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              32 F_predPC=              30 D_icode= 5,E_icode= 3, M_icode= 3, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              32 F_predPC=              30 D_icode= 5,E_icode= 3, M_icode= 3, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              32 F_predPC=              32 D_icode= 6,E_icode= 5, M_icode= 3, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=0 f_predPC=              32 F_predPC=              32 D_icode= 6,E_icode= 5, M_icode= 3, ifun= 0,rax=              0,rdx=              2,rbx=              3,rcx=              1
clk=1 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 6, M_icode= 5, ifun= 0,rax=              0,rdx=              2,rbx=             12,rcx=              1
clk=0 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 6, M_icode= 5, ifun= 0,rax=              0,rdx=              2,rbx=             12,rcx=              1
clk=1 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=              0,rdx=             16,rbx=             12,rcx=              1
clk=0 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 0, M_icode= 6, ifun= 0,rax=              0,rdx=             16,rbx=             12,rcx=              1
clk=1 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=              x,rdx=             16,rbx=             12,rcx=              1
clk=0 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=              x,rdx=             16,rbx=             12,rcx=              1

Halting
processor.v:50: $finish called at 150 (1s)
clk=1 f_predPC=              32 F_predPC=              32 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=              x,rdx=             16,rbx=             28,rcx=              1
```

Test Case 9 `call,return,rrmovq with Conditional and Unconditional Jumps`

```
//////////////////////////// test9_pipe ////////////////////////
//call, return, rrmovq along with conditional and unconditional jumps

// call fun
    instr_mem[0]=8'h80; // 8 0
    instr_mem[1]=8'h00;
    instr_mem[2]=8'h00;
    instr_mem[3]=8'h00;
    instr_mem[4]=8'h00;
    instr_mem[5]=8'h00;
    instr_mem[6]=8'h00;
    instr_mem[7]=8'h00;
    instr_mem[8]=8'h0a;
// ret
    instr_mem[9]=8'h90; // 9 0
// fun(%rdx,%rbx)
// fun:
//irmovq $0x0, %rax
    instr_mem[10]=8'b00110000; //3 0
    instr_mem[11]=8'b00000000; //F rB=0
    instr_mem[12]=8'b00000000;
    instr_mem[13]=8'b00000000;
    instr_mem[14]=8'b00000000;
    instr_mem[15]=8'b00000000;
    instr_mem[16]=8'b00000000;
    instr_mem[17]=8'b00000000;
    instr_mem[18]=8'b00000000;
    instr_mem[19]=8'b00000000; //V=0
// irmovq $0x10 %rdx
    instr_mem[20]=8'h30; //3 0
    instr_mem[21]=8'hF2; //F rB=2
    instr_mem[22]=8'h00;
    instr_mem[23]=8'h00;
    instr_mem[24]=8'h00;
    instr_mem[25]=8'h00;
    instr_mem[26]=8'h00;
    instr_mem[27]=8'h00;
    instr_mem[28]=8'h00;
    instr_mem[29]=8'h10; //V=16
    //jmp t1
    instr_mem[30]=8'b01110000; //7 fn
    instr_mem[31]=8'b00000000; //Dest
    instr_mem[32]=8'b00000000; //Dest
    instr_mem[33]=8'b00000000; //Dest
    instr_mem[34]=8'b00000000; //Dest
    instr_mem[35]=8'b00000000; //Dest
    instr_mem[36]=8'b00000000; //Dest
    instr_mem[37]=8'b00000000; //Dest
    instr_mem[38]=8'h27; //Dest=39
```

```
352     // t1:
353     // addq %rax, %rbx
354     instr_mem[39]=8'b01100000; //5 fn
355     instr_mem[40]=8'b00000011; //rA=0 rB=3
356     // je t2
357     instr_mem[41]=8'b01110011; //7 fn=3
358     instr_mem[42]=8'b00000000; //Dest
359     instr_mem[43]=8'b00000000; //Dest
360     instr_mem[44]=8'b00000000; //Dest
361     instr_mem[45]=8'b00000000; //Dest
362     instr_mem[46]=8'b00000000; //Dest
363     instr_mem[47]=8'b00000000; //Dest
364     instr_mem[48]=8'b00000000; //Dest
365     instr_mem[49]=8'h33; //Dest=51
366     //halting
367     instr_mem[50]=8'b00000000;
368     // t2:
369     // rrmovq %rdx, %rcx
370     instr_mem[51]=8'b00100000; //2 fn=0
371     instr_mem[52]=8'b00100001; //rA=2 rB=1
372     // ret
373     instr_mem[53]=8'h90; // 9 0
```

Output:

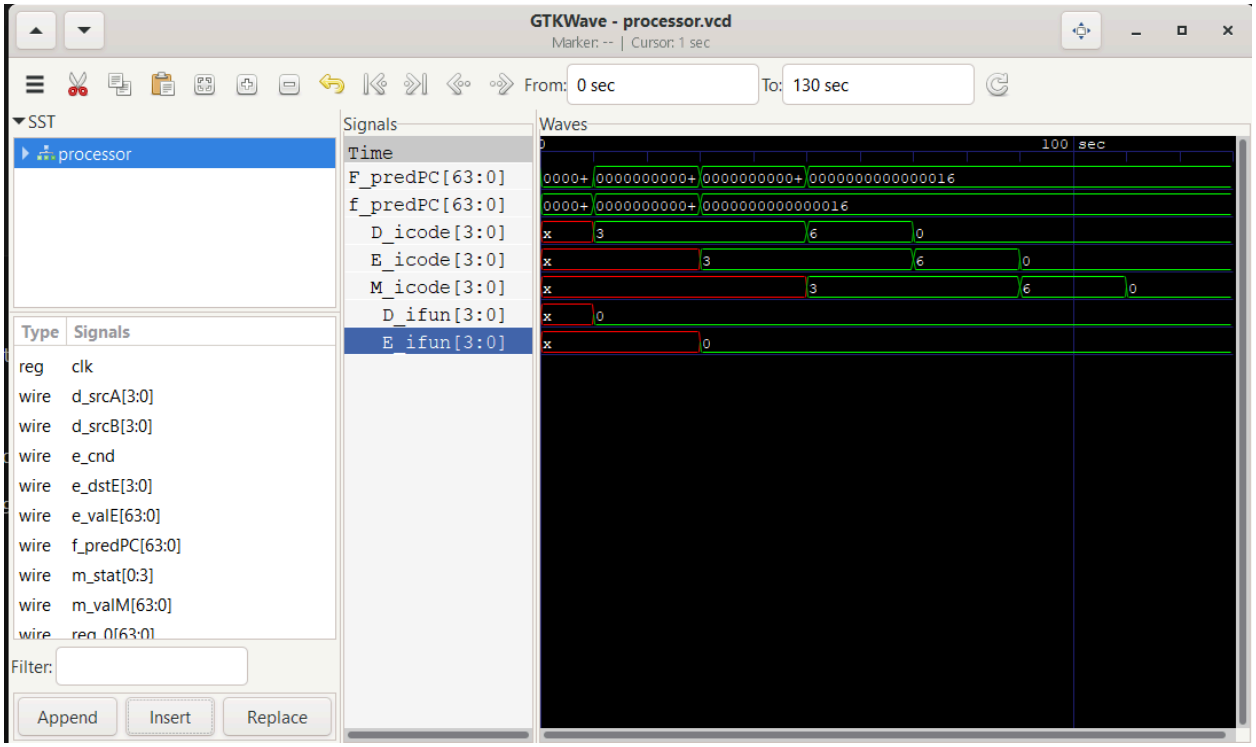
```
VCD info: dumpfile processor.vcd opened for output.
clk=0 f_predPC=          10 F_predPC=           0 D_icode= x,E_icode= x, M_icode= x, ifun= x,rax=          x,rdx=          x,rbx=          x,rcx=          x
clk=1 f_predPC=          20 F_predPC=          10 D_icode= 8,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          20 F_predPC=          10 D_icode= 8,E_icode= x, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          30 F_predPC=          20 D_icode= 3,E_icode= 8, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          30 F_predPC=          20 D_icode= 3,E_icode= 8, M_icode= x, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          39 F_predPC=          30 D_icode= 3,E_icode= 3, M_icode= 8, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          39 F_predPC=          30 D_icode= 3,E_icode= 3, M_icode= 8, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          41 F_predPC=          39 D_icode= 7,E_icode= 3, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          41 F_predPC=          39 D_icode= 7,E_icode= 3, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          51 F_predPC=          41 D_icode= 6,E_icode= 7, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          51 F_predPC=          41 D_icode= 6,E_icode= 7, M_icode= 3, ifun= 0,rax=          0,rdx=          2,rbx=          3,rcx=          1
```

```
clk=1 f_predPC=          53 F_predPC=          51 D_icode= 7,E_icode= 6, M_icode= 7, ifun= 3,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=0 f_predPC=          53 F_predPC=          51 D_icode= 7,E_icode= 6, M_icode= 7, ifun= 3,rax=          0,rdx=          2,rbx=          3,rcx=          1
clk=1 f_predPC=          53 F_predPC=          53 D_icode= 2,E_icode= 7, M_icode= 6, ifun= 0,rax=          0,rdx=         16,rbx=          3,rcx=          1
clk=0 f_predPC=          53 F_predPC=          53 D_icode= 2,E_icode= 7, M_icode= 6, ifun= 0,rax=          0,rdx=         16,rbx=          3,rcx=          1
```

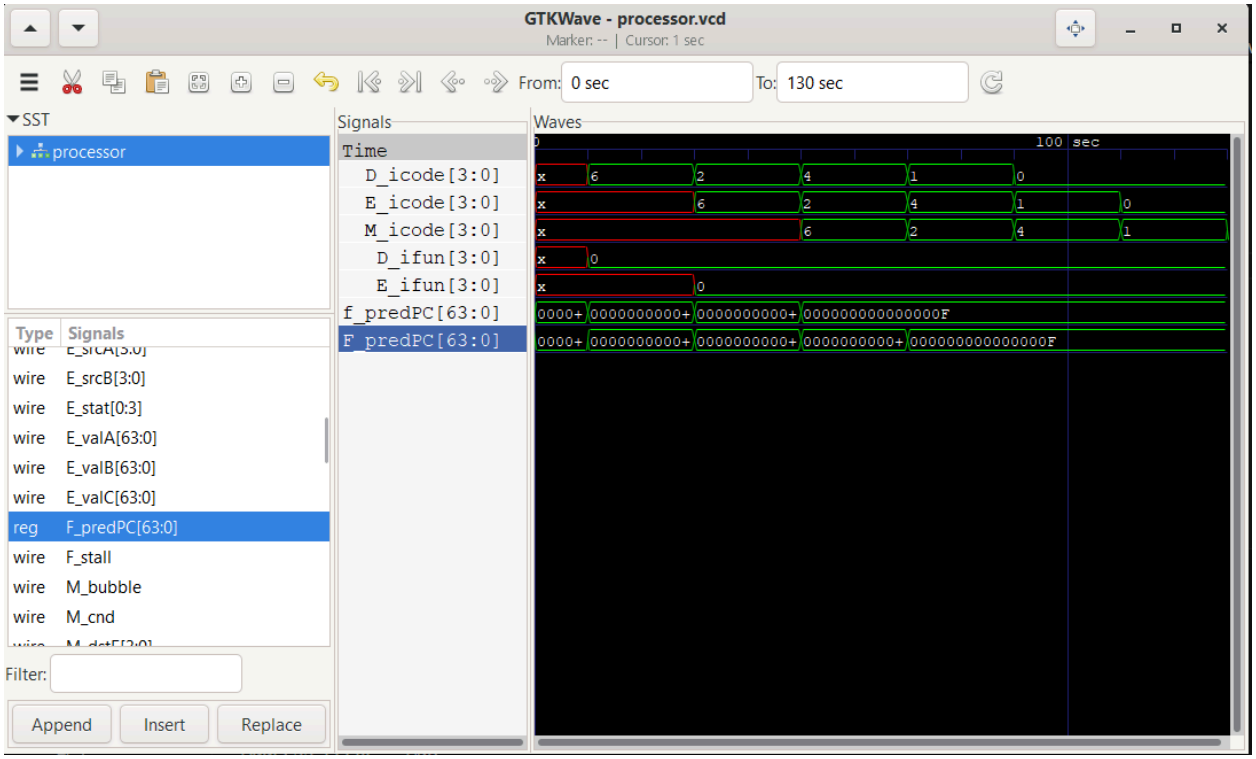
```
clk=1 f_predPC=          50 F_predPC=          53 D_icode= 1,E_icode= 1, M_icode= 7, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=0 f_predPC=          50 F_predPC=          53 D_icode= 1,E_icode= 1, M_icode= 7, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=1 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 1, M_icode= 1, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=0 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 1, M_icode= 1, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=1 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 0, M_icode= 1, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=0 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 0, M_icode= 1, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=1 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
clk=0 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1

Halting
processor.v:50: $finish called at 230 (1s)
clk=1 f_predPC=          50 F_predPC=          50 D_icode= 0,E_icode= 0, M_icode= 0, ifun= 0,rax=          0,rdx=          16,rbx=          3,rcx=          1
```

GTK Wave for Pipeline Implementation:

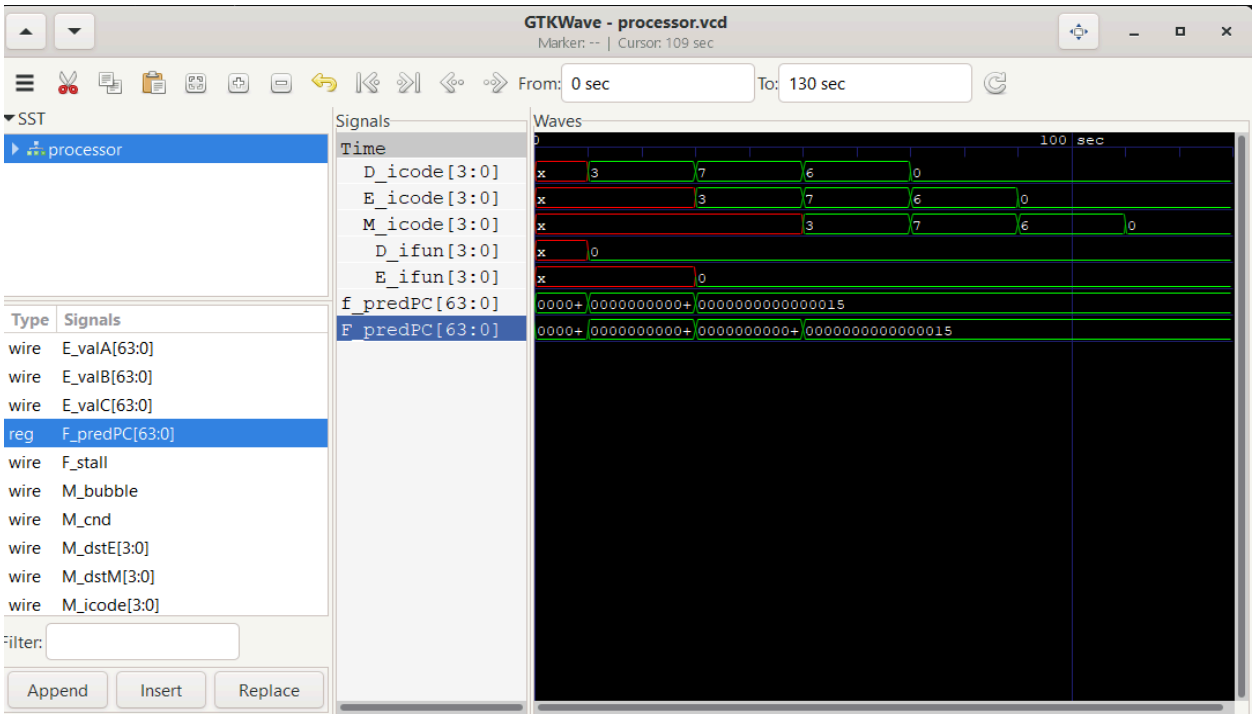
Test Case 1 `opq , halt along with irmovq:`



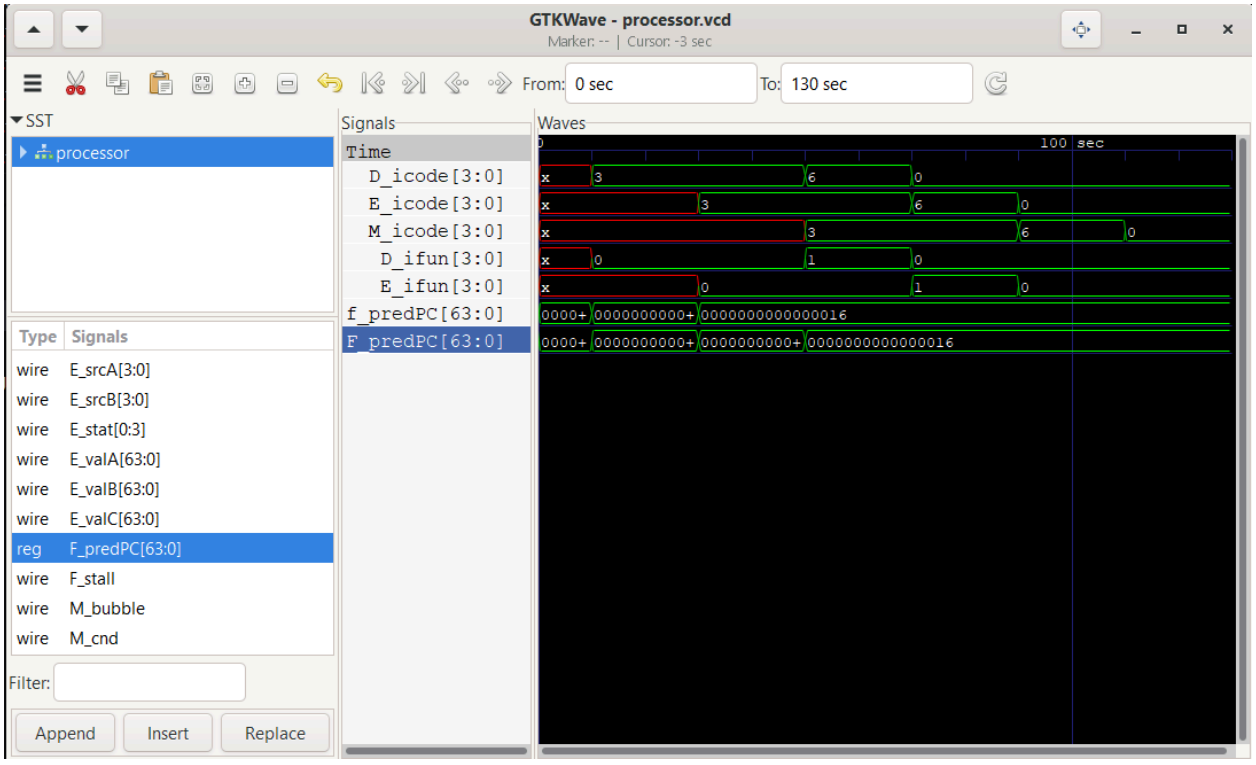
Test Case 2 `rrmovq,rmmovq,nop,halt with opq:`



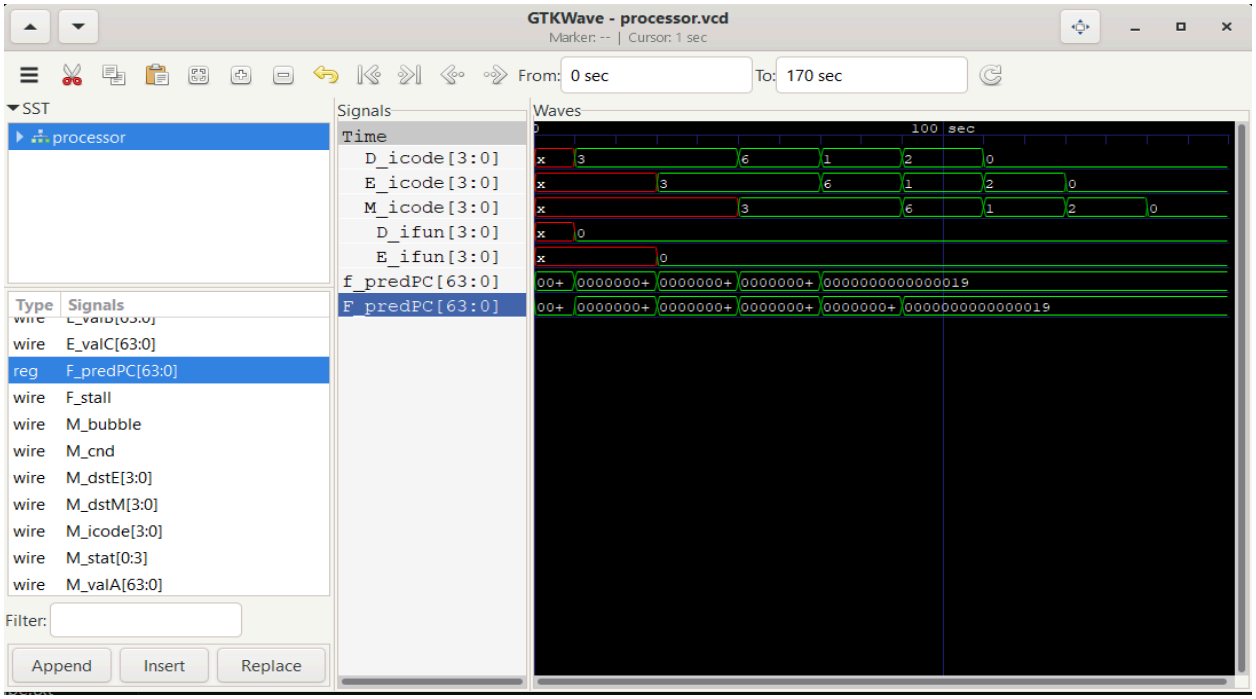
Test Case 3 `jump halt with irmovq`



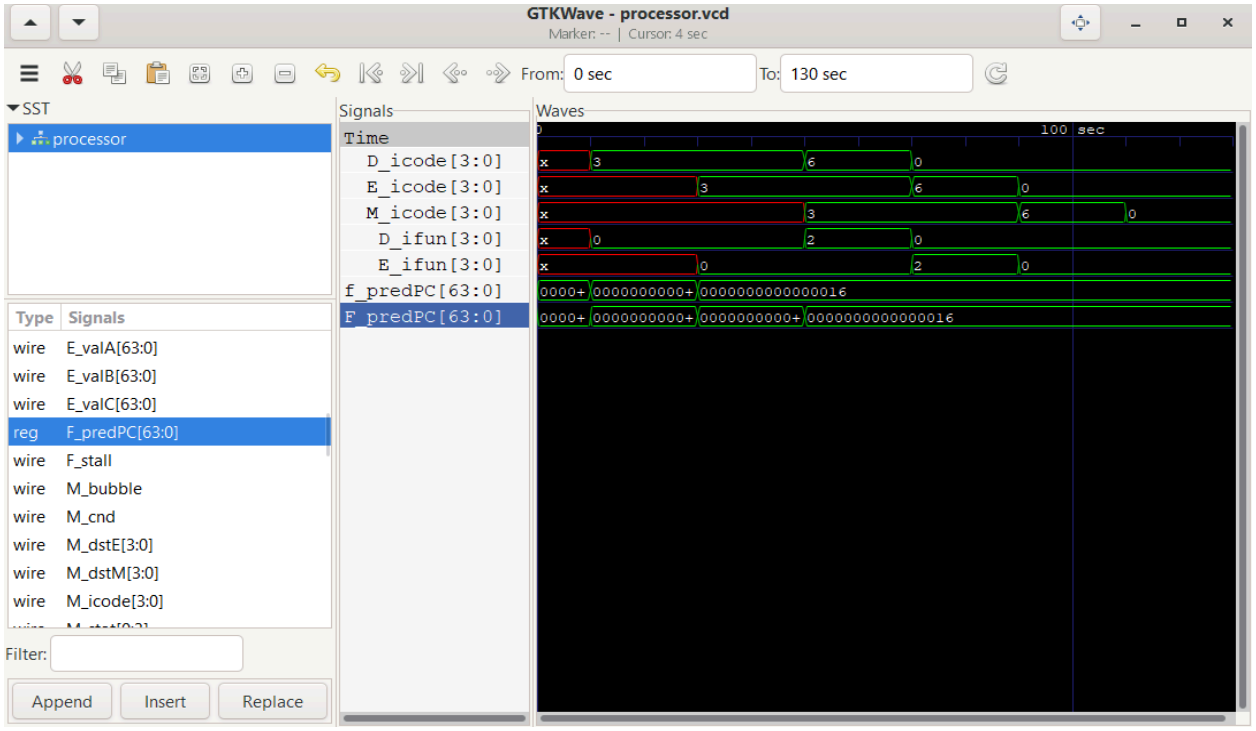
Test Case 4 `opq,halt with irmovq`



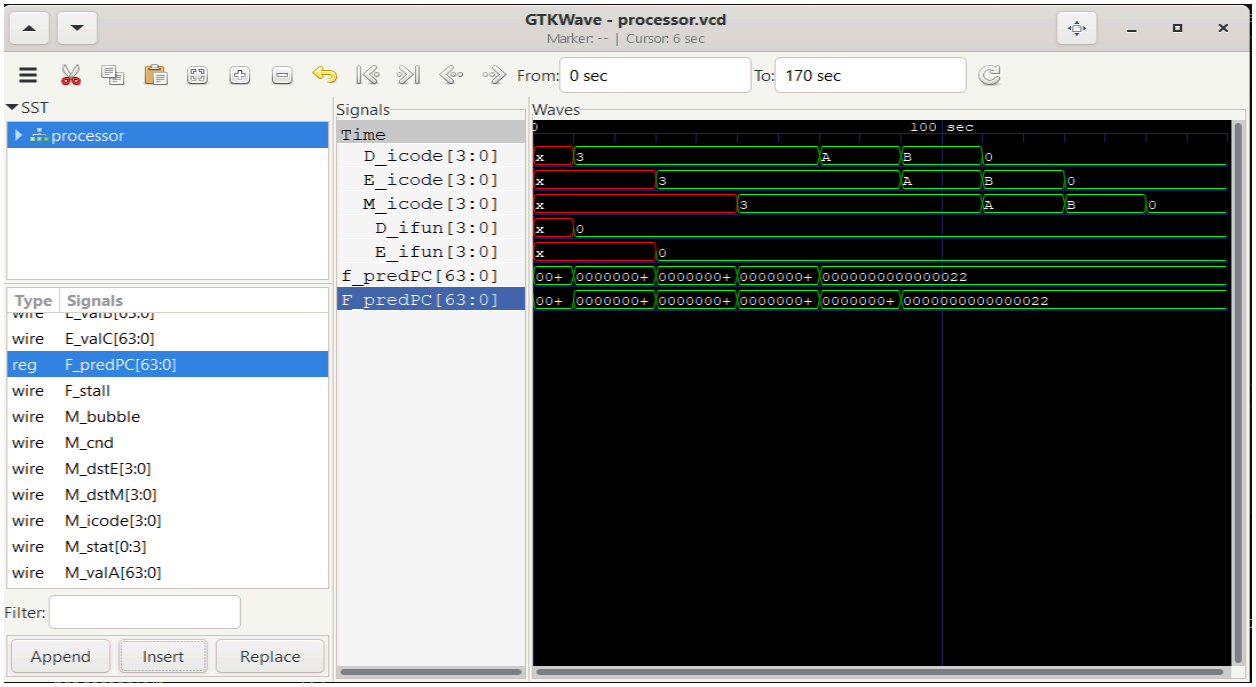
Test Case 5 `cmovxx,halt irmovq with opq`



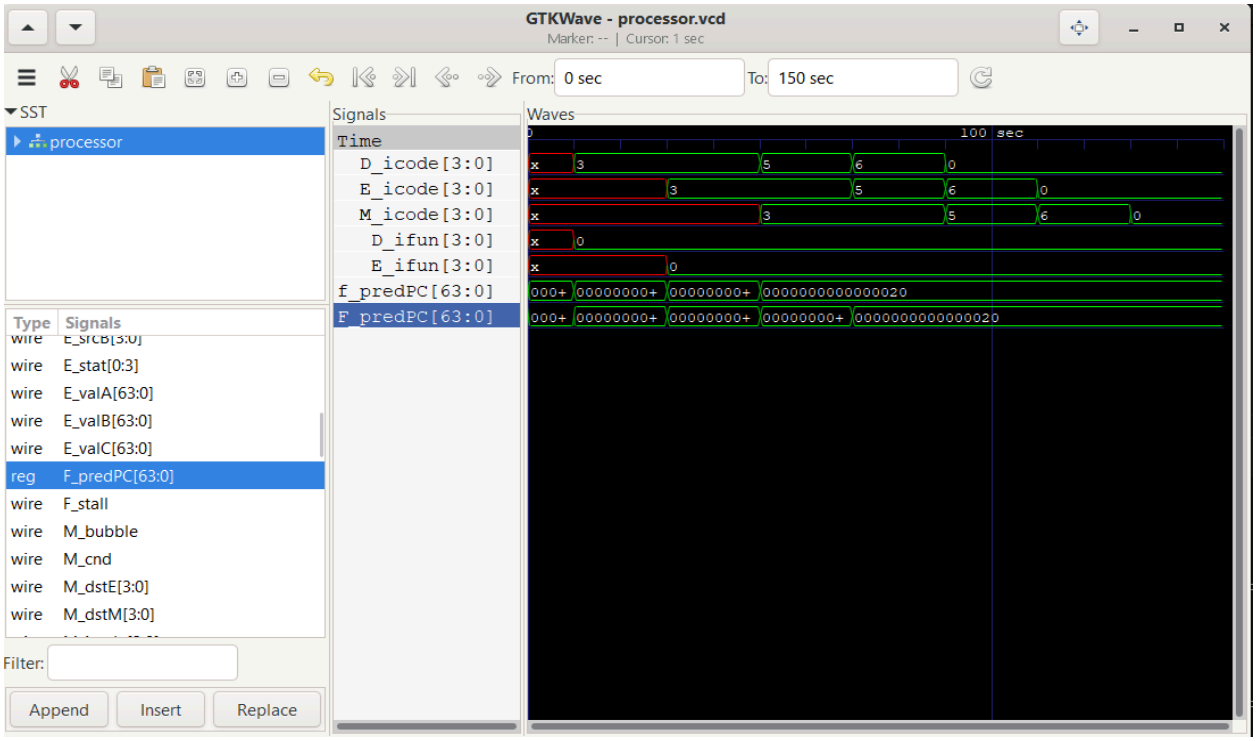
Test Case 6 `irmovq,halt with opq`



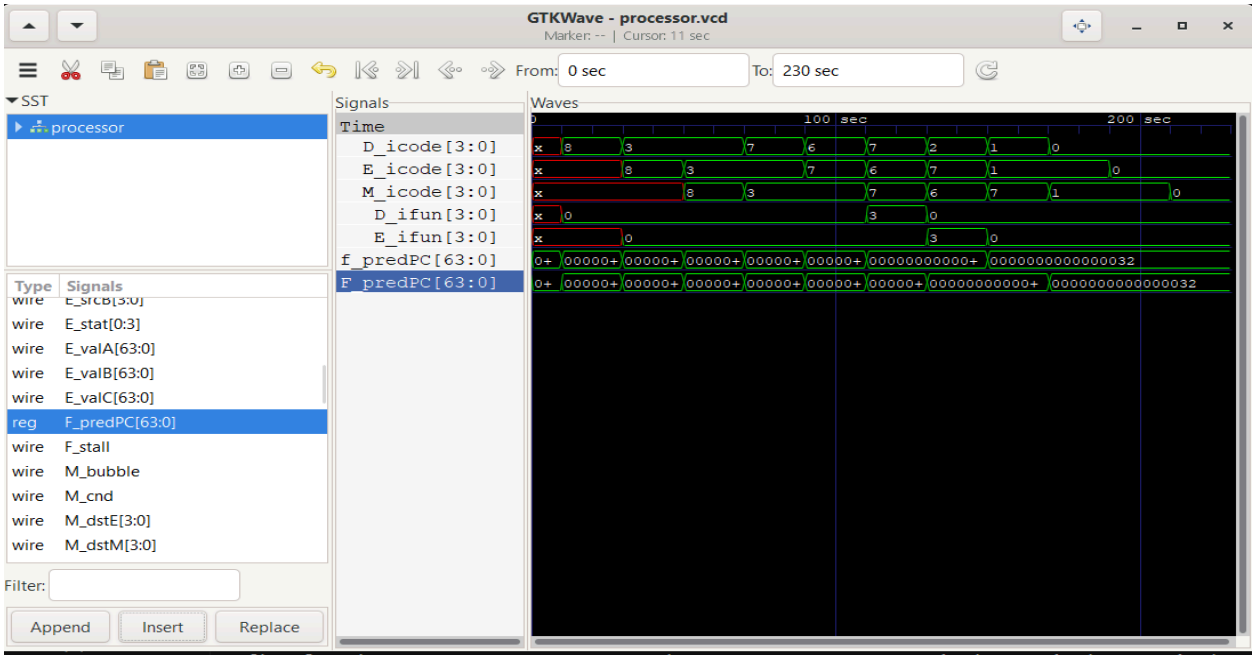
Test Case 7 `push,pop,halt with irmovq`



Test Case 8 `irmovq, mrmovq,opq with halt`



Test Case 9 `Call,Return,rrmovq with Conditional and unconditional jumps`



Challenges Encountered

- Implementation of all sequential stages in a single clock cycle was difficult to monitor
- Some of the test cases had a impedance state due to wrong execution
- Faced difficulty in understanding the idea of parallelism of different stages
- Implementing Data forwarding was difficult
- Testing cases including Conditional Jumps , Call , Return
- Handling Data Hazards and Control Logic

Conclusion

Sequential and Pipeline Implementation of Y86-64 processor for performing stages such as Fetch , Decode , Execute , Memory , Write back and PC update are executed successfully for various instructions and verified. The supported instructions are

- halt
- nop
- cmovXX
- irmovq
- rmmovq
- opq
- jXX
- call
- ret
- pushq
- popq

The processor supports all instructions in the Y86-64 instruction set