<u>Intro to Processor Architecture</u>

Project report

T. Abhishek 2021102010 N. Raviteja 2021102009

Sequential:

We have six modules in sequential implementation of Y86-64 processor . the stages are as follows.

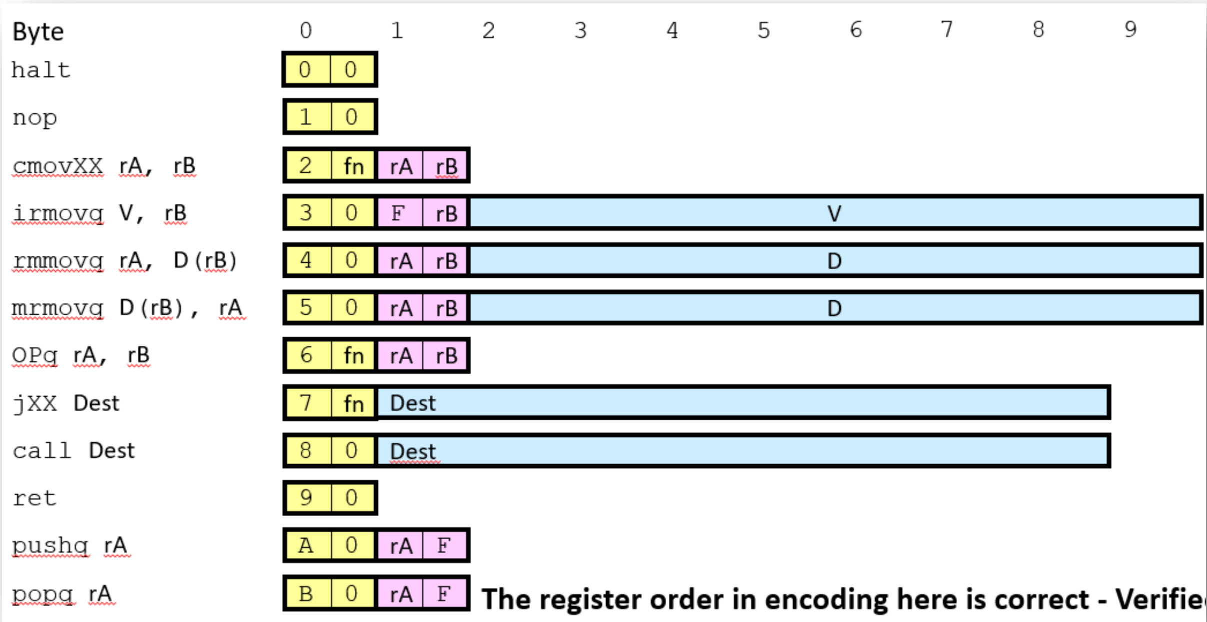
Fetch:

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

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

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

Fetch block uses instruction array which may each have 10 bytes to compute icode , ifun , rA, rB, valC, valP, instruction error and halting first 1 byte represent icode : ifun where icode is of 4bits and ifun is of 4 bits. Second byte represents the registers rA, rB, then after all the bits represents destination offset



ifun of the instruction represents the condition required to execute the respective instruction. Instruction is set to 0 if the instruction given is wrong.

Imem error is set to 0 if the pc value stays within 1023. And when halt is encountered halt is set to 1.

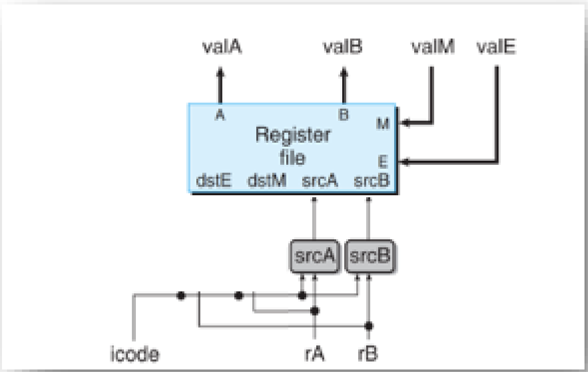
Code

```
module fetch (clk,PC,instruct,icode,ifun,ra,rb,valC,valP,mem_err,instruct_err);
    input clk;                          // clock
    input [63:0] PC;                    // program counter
    input [0:79] instruct;              // instruction

    output reg [3:0] icode,ifun,ra,rb;  // instruction code, function code, register A, register B
    output reg [63:0] valC,valP;        // Constant(Displacemnet/value), next PC
    output reg mem_err,instruct_err;    // memory error, instruction error
```

This is how fetch is implemented . example given below are for rmmovq and mrmovq.

```
else if(icode==4'h4)                    //rmmovq
    begin
        {ra,rb,valC}=instruct[8:79];
        valP=PC+10;
    end
else if(icode==4'h5)                    //mrmovq
    begin
        {ra,rb,valC}=instruct[8:79];
        valP=PC+10;
    end
```

Decode



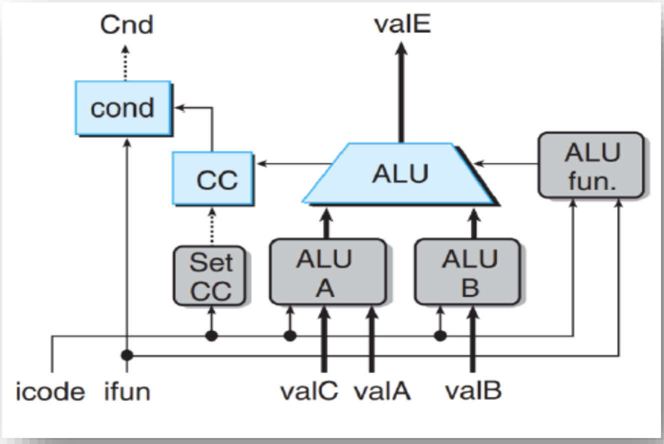
We have 15 registers in the Y86-64 processor which are to be accessed in the decode stage. The register file represents those 15 registers.

Reading and assigning the values of valA and valB are done in the decode stage . the registers are represented here as register file .

In the decode phase, if else statements are used to calculate valA and valB in accordance with the instructions.

```
module decode (clk,icode,ra,rb,valA,valB);

    input clk;                          //clock
    input [3:0] icode,ra,rb;            //instruction code, register A, register B

    output reg [63:0] valA,valB;        //value A, value B

    reg [63:0] register[0:14];          //register
```

execute:



the output of the execute block are Cnd and valE and its inputs are icode , ifun , valC, valA and valB.It works on if else and decides which execution to be done on the respective instruction.

The execute block's functions include carrying out the command, using the ALU to compute the effective address or value, and setting the necessary condition codes.

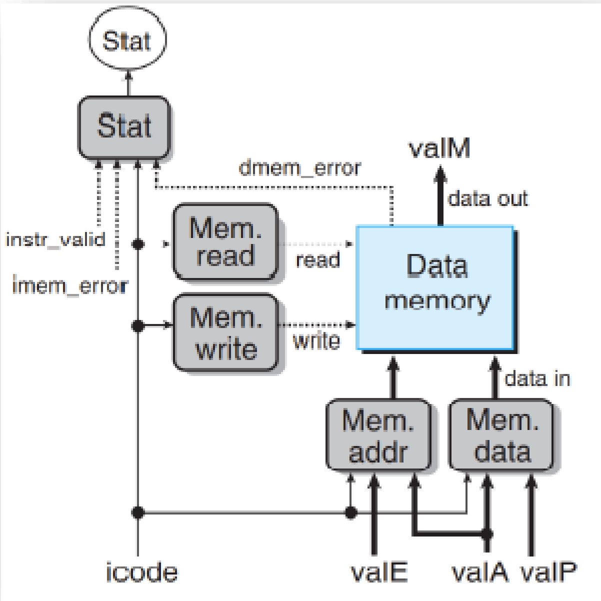
Condition codes are set for jXX and CMOV. We also include two different condition codes CC_in and CC_out. CC_in represents the condition code of previous instruction whereas CC_out represent the condition code of the on going instruction.

CODE

```
module execute(clk,icode,ifun,valA,valB,valC,valE,cond,CC_in,CC_out,Z_F,S_F,O_F);

input clk;
input[3:0] icode,ifun;
input[2:0] CC_in;
input signed [63:0] valA,valB,valC;

output reg cond,Z_F,S_F,O_F;
output reg[2:0] CC_out;
output reg[63:0] valE;

reg[1:0] control;
reg signed [63:0] aluA,aluB,ans;
wire signed [63:0] out;
wire overflow;
```

Memory:

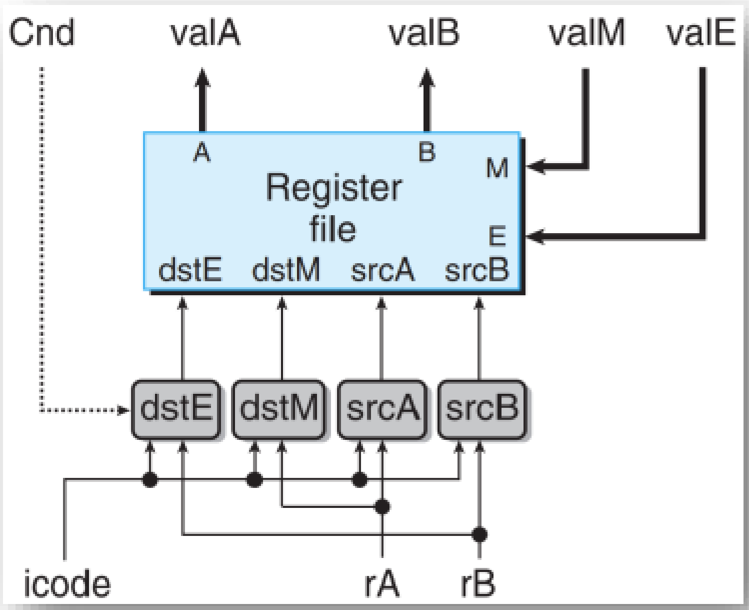
The memory block has icode, valE, valA, and valP as input values and valM as an output value. The memory block's function is to receive and write values from memory. Although the data mem in the processor is generally available everywhere, we have set it here as an output for testbench's verification purposes.

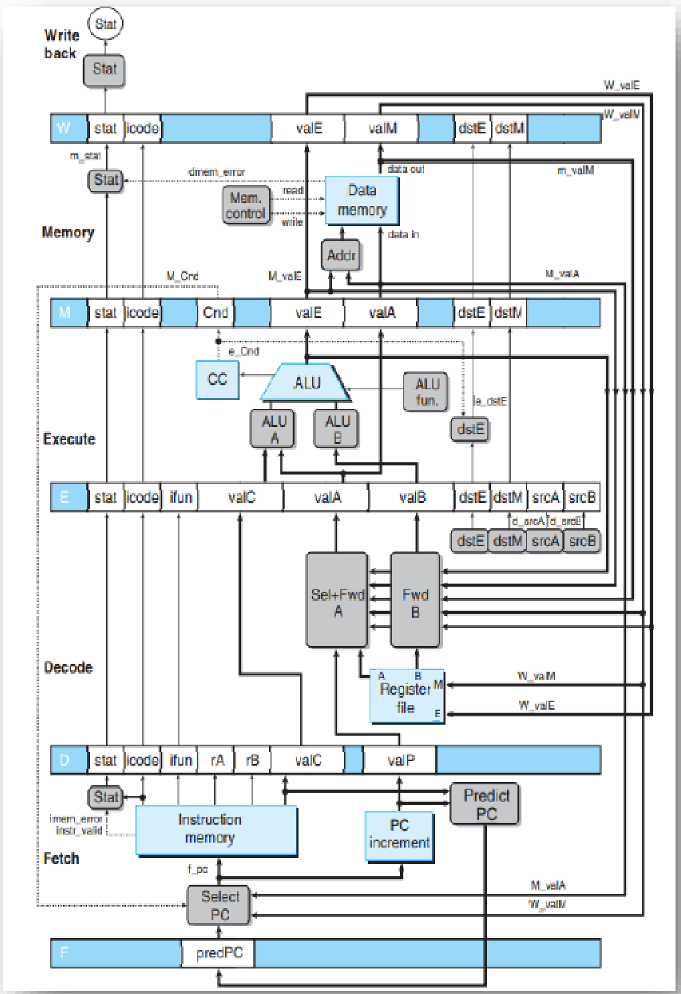



```
module memory (clk,icode,valP,valA,valB,valM,valE,mem_err);

    input clk;                          //clock
    input [3:0] icode;                  //instruction code
    input [63:0] valP,valA,valB,valE;   //constant, next PC, value A, value B, value E

    // output regm memo_arr;                   //memory array
    output reg mem_err;                 //memory error
    output reg [63:0] valM;             //value M

    // output reg [63:0] memory;     //memory

    reg [63:0] memo_arr[0:1023];     //temporary memory for storing and assigning values
```

Writeback:

In write back the values are written back into registers which are computed till memory stage.





Pipelined implementation

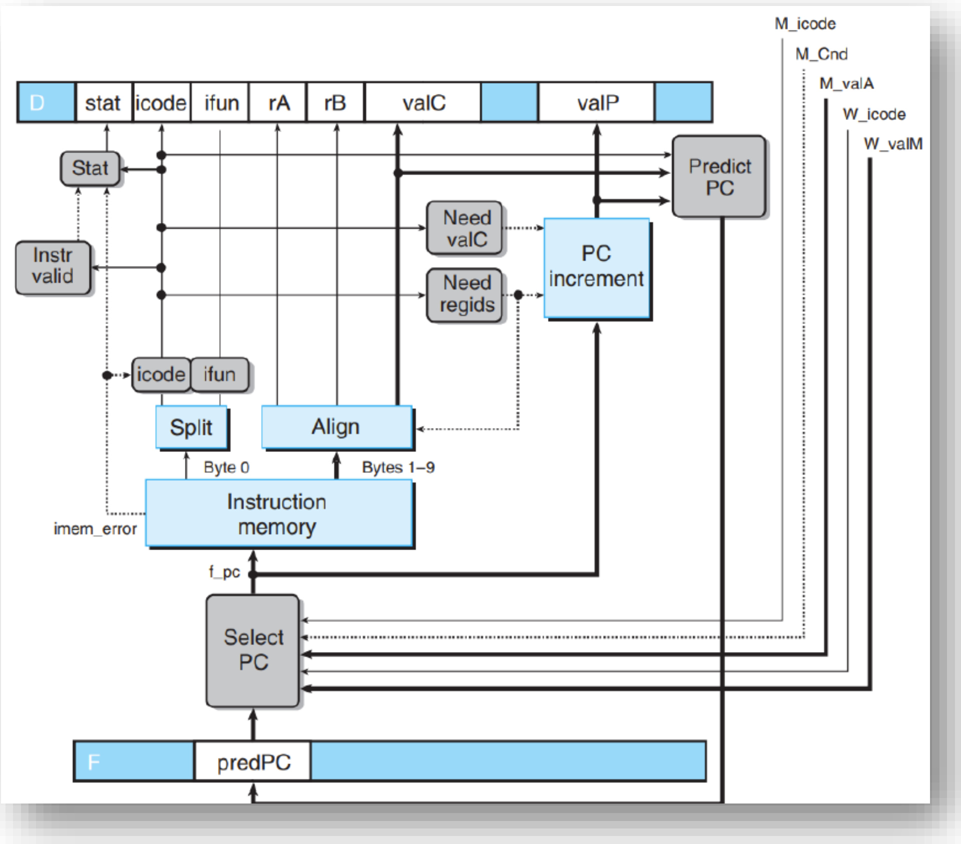
The processor Y86-64's implementation uses the same modules as its sequential implementation, with the addition of pipelined registers, a slight modification to the fetch and decode blocks, also addition of data forwarding and PC prediction to boost performance, and pipeline control logic to eliminate pipeline hazards.

64's implementation uses the same modules as its sequential implementation, with the addition of pipelined registers, a slight modification to the fetch and decode blocks, also addition of data forwarding and PC prediction to boost performance, and pipeline control logic to eliminate pipeline hazards.

As we want to fetch the next instruction constantly without having to wait for the PC update stage of the previous instruction to finish had it been at the end of the cycle, we should move the PC update stage to the beginning of the cycle for the pipelined implementation. Circuit retiming is the term for this. This modifies the circuit's overall state while having no impact on its local behaviour. Additionally, it enables us to manage the delays in the pipelined system between phases. In a pipelined version, some hardware and signals in the SEQ implementation are moved around, and pipeline registers are added in between each step.

The stages are given below.

Fetch:



The starting value of PC from the processor.v block is used to create the field f pc in this case.

The values of stat, icode, ifun, rA, rB, valC, and valP are computed using the PC as the input, and since they will be sent into the decode register, they will be given the names D_icode, D_ifun, D rA, D_rB, valC, and D_stat.

The fetch portion operates in the same way as the sequential portion, but the inclusion of the sequential block boosts the processor's performance.

Predict PC block capability will be sent to the retrieve register in the processor.v block after the Predict PC block is added.

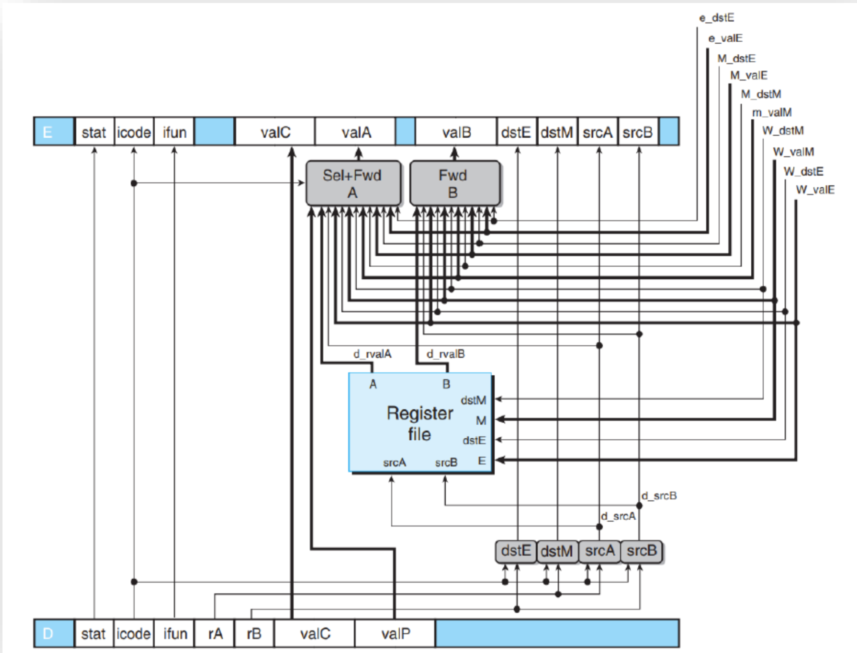
New pc will be updated after each positive edge of the clock.

```
module fetch(clk,M_icode,W_icode,M_valA,W_valM,M_cnd,F_PC_in,F_stall,D_stall,D_bubble,D_icode,D_ifun,D_rA,D_rB,D_valC,D_valP,D_stat,F_PC_out);
    input clk,M_cnd,F_stall,D_stall,D_bubble;
    input [3:0] M_icode,W_icode;
    input [63:0] M_valA,W_valM,F_PC_in;

    reg [3:0] icode, ifun,ra,rb;  //local values
    reg [63:0] PC,valC,valP;
    reg mem_err=0, instruct_err=0;
    reg [0:3] stat_code;
    reg [0:79] instruct;
    reg [7:0] inst_arr[0:1024];
```

The input for this block are the clock signal and M_icode, M_cnd, M valA, W_icode, W_valM which are present for the calculation of the next predicted PC i.e.output PC and input PC in which is used to calculate the values of the D_icode, D_ifun, D_rA, D_rB, D_valC, D stat and D valP.

Once the clock's positive edge is reached, the register is changed and the output for D icode, D ifun, D rA, D rB, D valC, D stat, and D valP is made accessible as a register output.

Decode and writeback:



Inputs :-
D_icode, D_ifun, D_rA, D rB, D_valC, D_stat, D_valP

Inputs for data forwarding :- e_dstE, e_valE, M_dstE, M_valE, M_dstM, m_valM, W_dstM, W_dstE, and W_valE are used in this.

Along with the earlier blocks, there are two more blocks in this. These blocks, which are represented by the Sel+Fwd A and Fwd B that directly provide the value for valA or valB from the execute, memory, or writeback stages, aid in the forwarding of data.

Data forwarding:

```
## What should be the A value?
int d_valA = [
  # Use incremented PC
    D_icode in { ICALL, IJXX } : D_valP;
  # Forward valE from execute
    d_srcA == e_dstE : e_valE;
  # Forward valM from memory
    d_srcA == M_dstM : m_valM;
  # Forward valE from memory
    d_srcA == M_dstE : M_valE;
  # Forward valM from write back  d_srcA
== W_dstM : W_valM;
  # Forward valE from write back
    d_srcA == W_dstE : W_valE;
  # Use value read from register file
    1 : d_rvalA;
];
```

CODE:

```
// Forwarding A
if(D_icode==4'h7 | D_icode == 4'h8)
begin
    d_valA = D_valP;
end
else if(d_srcA==e_dstE & e_dstE!=4'hF)
begin
    d_valA = e_valE;
end
else if(d_srcA==M_dstM & M_dstM!=4'hF)
begin
    d_valA = m_valM;
end
else if(d_srcA==W_dstM & W_dstM!=4'hF)
begin
    d_valA = W_valM;
end
else if(d_srcA==M_dstE & M_dstE!=4'hF)
begin
    d_valA = M_valE;
end
else if(d_srcA==W_dstE & W_dstE!=4'hF)
begin
    d_valA = W_valE;
end
```

```
// Forwarding B
if(d_srcB==e_dstE & e_dstE!=4'hF)
begin
    d_valB = e_valE;
end
else if(d_srcB==M_dstM & M_dstM!=4'hF)
begin
    d_valB = m_valM;
end
else if(d_srcB==W_dstM & W_dstM!=4'hF)
begin
    d_valB = W_valM;
end
else if(d_srcB==M_dstE & M_dstE!=4'hF)
begin
    d_valB = M_valE;
end
else if(d_srcB==W_dstE & W_dstE!=4'hF)
begin
    d_valB = W_valE;
end
```

Execute:

When the inputs E_stat, E_ifun, E_icode, E_valA, E_valB, E_valC, E_dstE, and E_dstM are transmitted through this block, M_stat, M_icode, Cnd, M_valE, M_valA, M_dstE, M_dstM, and e_valE are computed as the outputs, similarly to how sequential computations would.

Based on the value of e Cnd, the value of e_dstE is determined, and it will either be E_dstE or an empty register. When a command like halt is executed, W stat and m stat take caution to avoid changing the conditional codes.

Conditions for stall:

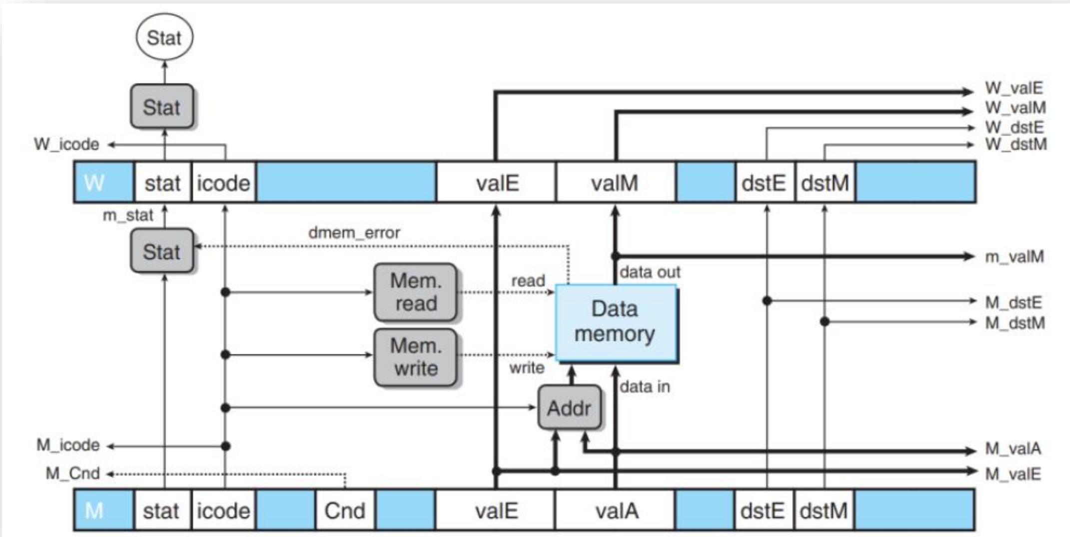
```
bool F_stall =
      # Conditions for a load/use hazard
      E_icode in { IMRMOVQ, IPOPQ } && E_dstM in { d_srcA, d_srcB } ||
      # Stalling at fetch while ret passes through pipeline
      IRET in { D_icode, E_icode, M_icode };

bool D_stall =
      # Conditions for a load/use hazard
      E_icode in { IMRMOVQ, IPOPQ } && E_dstM in { d_srcA, d_srcB };

bool D_bubble =
      # Mispredicted branch
      (E_icode == IJXX && !e_Cnd) ||
      # Stalling at fetch while ret passes through pipeline
       IRET in { D_icode, E_icode, M_icode };

bool E_bubble =
      # Mispredicted branch
      (E_icode == IJXX && !e_Cnd) ||
      # Load/use hazard
      E_icode in { IMRMOVQ, IPOPQ } && E_dstM in { d_srcA, d_srcB };
```

```
module execute(clk,E_stat,E_icode,E_ifun,E_valC,E_valA,E_valB,E_dstE,E_dstM,e_cnd,W_stat,m_stat,
               set_cc,
               M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,e_valE,e_dstE);

input clk;
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;

input set_cc;

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

Memory:

This stage functions similar to the Memory part of Sequential except the variables names renamed as M_icode,M_ifun,M_dstE,M_dstM,M_valP,M_valA,M_valE;

```
module memory (clk,M_stat,M_cnd,M_dstE,M_dstM,W_stat,W_icode,W_valE,W_valM,W_dstE,W_dstM,m_stat,M_icode,M_valA,m_valM,M_valE);

    input clk,M_cnd;                          //clock
    input [3:0] M_icode,M_dstE,M_dstM;                   //instruction code
    input [63:0] valP,M_valA,M_valE;   //constant, next PC, value A, value B, value E
    input [0:3] M_stat;

    reg [63:0] memo_arr[0:1023];     //temporary memory for storing and assigning values

    // output regm memo_arr;                   //memory array

    reg mem_err;                        //memory error

    output reg [63:0] m_valM,W_valE,W_valM;             //value M
    output reg [0:3] W_stat,m_stat;
    output reg [3:0] W_icode,W_dstE,W_dstM;

    // output reg [63:0] memory;     //memory
```



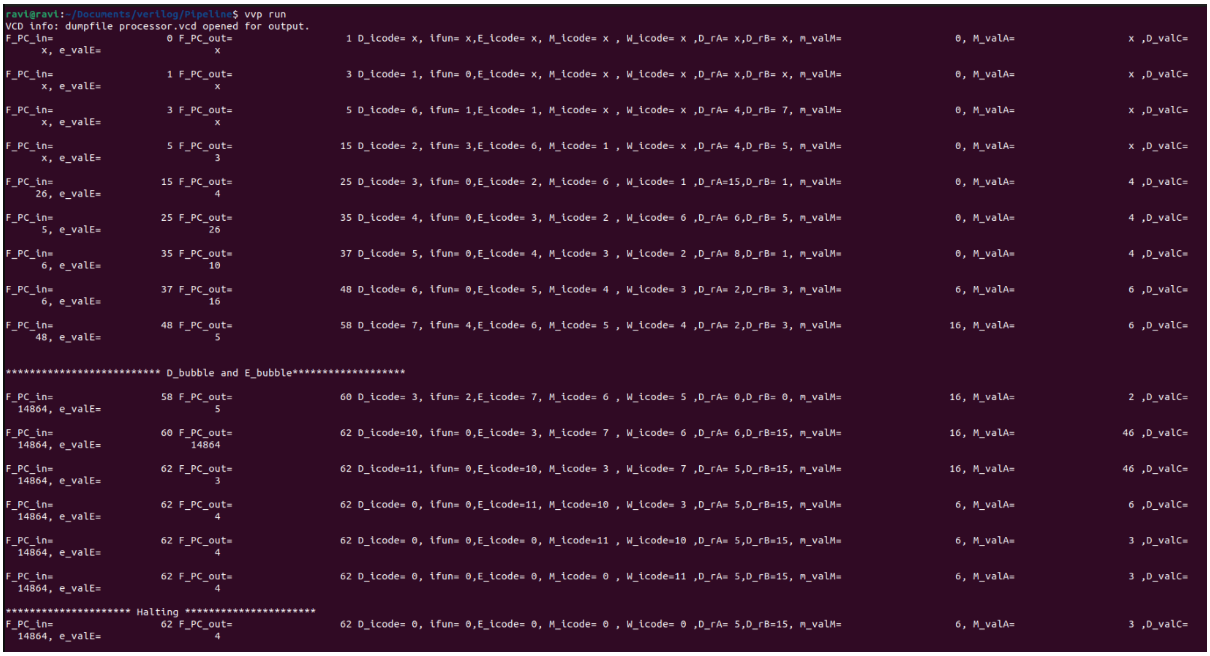
Instruction set:

```
inst_arr[0]=8'h10; //nop

inst_arr[1]=8'h61; //subtraction
inst_arr[2]=8'h47; //ra and rb

inst_arr[3]=8'h23; //cmov
inst_arr[4]=8'h45; // ra=4 and rb=5

inst_arr[5]=8'h30; //irmovq
inst_arr[6]=8'hF1; //F and rB
inst_arr[7]=8'h00; //
inst_arr[8]=8'd00;
inst_arr[9]=8'h00;
inst_arr[10]=8'h00;
inst_arr[11]=8'h00;
inst_arr[12]=8'h00;
inst_arr[13]=8'h00;
inst_arr[14]=8'd26;

inst_arr[15]=8'h40; //rmmovq
inst_arr[16]=8'h65; //ra and rb
inst_arr[17]=8'h00; //
inst_arr[18]=8'd00;
inst_arr[19]=8'd00; //0 0

inst_arr[20]=8'h00; //3 0
inst_arr[21]=8'h00; //F rB=7
inst_arr[22]=8'h00;
inst_arr[23]=8'h00;
inst_arr[24]=8'd5; //D value
inst_arr[25]=8'h50;   //mrmovq
inst_arr[26]=8'h81;   // ra rb
```

```
inst_arr[23]=8'h00;
inst_arr[24]=8'd5; //D value
inst_arr[25]=8'h50;   //mrmovq
inst_arr[26]=8'h81;   // ra rb
inst_arr[27]=8'h00;
inst_arr[28]=8'h00;
inst_arr[29]=8'd00;
inst_arr[30]=8'h00;
inst_arr[31]=8'h00;
inst_arr[32]=8'h00;
inst_arr[33]=8'h00;
inst_arr[34]=8'd6; //D=6

inst_arr[35]=8'h60;  //OPq addition
inst_arr[36]=8'h23;  //ra and rb

inst_arr[37]=8'h74;  // Jump if equal
inst_arr[38]=8'h00;
inst_arr[39]=8'h00;
inst_arr[40]=8'h00;
inst_arr[41]=8'h00;
inst_arr[42]=8'h00;
inst_arr[43]=8'h00;
inst_arr[44]=8'h00;
inst_arr[45]=8'd48; //Jump to 48
inst_arr[46]=8'h60;
inst_arr[47]=8'h45;

inst_arr[48]=8'h32; //call
inst_arr[49]=8'h00;
inst_arr[50]=8'h00;
inst_arr[51]=8'h00;
inst_arr[52]=8'h00;
inst_arr[53]=8'h00;
inst_arr[54]=8'h00;
inst_arr[55]=8'h00;
inst_arr[56]=8'd58; //goto 58
```

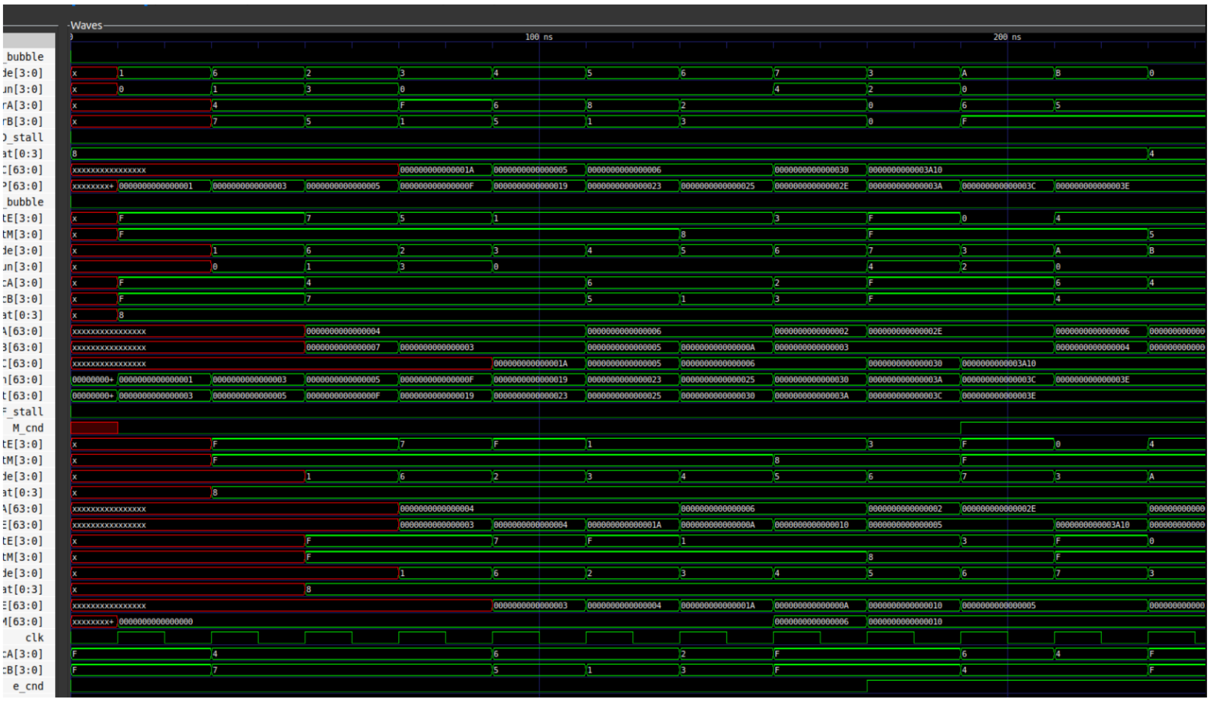
```
inst_arr[40]=8'h00;
inst_arr[41]=8'h00;
inst_arr[42]=8'h00;
inst_arr[43]=8'h00;
inst_arr[44]=8'h00;
inst_arr[45]=8'd48; //Jump to 48
inst_arr[46]=8'h60;
inst_arr[47]=8'h45;

inst_arr[48]=8'h32; //call
inst_arr[49]=8'h00;
inst_arr[50]=8'h00;
inst_arr[51]=8'h00;
inst_arr[52]=8'h00;
inst_arr[53]=8'h00;
inst_arr[54]=8'h00;
inst_arr[55]=8'h00;
inst_arr[56]=8'd58; //goto 58
inst_arr[57]=8'h10;



inst_arr[58]=8'hA0; //pushq
inst_arr[59]=8'h6F; //ra and F

inst_arr[60]=8'hB0; //popq
inst_arr[61]=8'h5F; //ra and F

inst_arr[62]=8'h00; //halt
```

Final output:

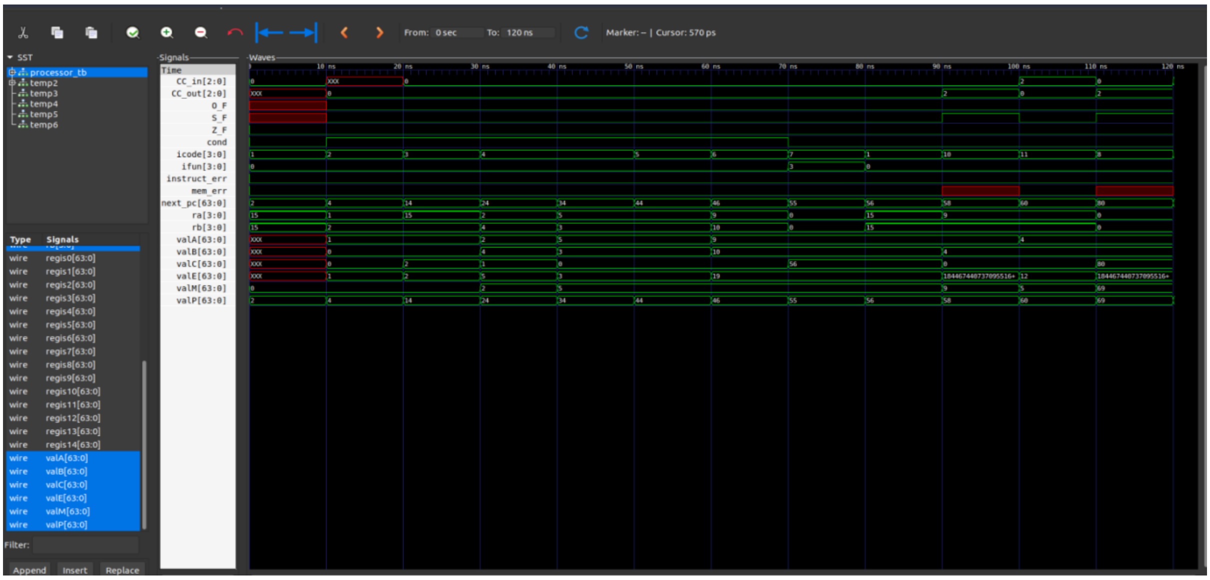


```
ravi@ravi:~/Documents/verilog/Pipeline$ vvp run
VCD info: dumpfile processor.vcd opened for output.
F_PC_in=              0 F_PC_out=              1 D_icode= x, ifun= x,E_icode= x, M_icode= x , W_icode= x ,D_rA= x,D_rB= x, m_valM=              0, M_valA=              x ,D_valC=
      x, e_valE=              x

F_PC_in=              1 F_PC_out=              3 D_icode= 1, ifun= 0,E_icode= x, M_icode= x , W_icode= x ,D_rA= x,D_rB= x, m_valM=              0, M_valA=              x ,D_valC=
      x, e_valE=              x

F_PC_in=              3 F_PC_out=              5 D_icode= 6, ifun= 1,E_icode= 1, M_icode= x , W_icode= x ,D_rA= 4,D_rB= 7, m_valM=              0, M_valA=              x ,D_valC=
      x, e_valE=              x

F_PC_in=              5 F_PC_out=             15 D_icode= 2, ifun= 3,E_icode= 6, M_icode= 1 , W_icode= x ,D_rA= 4,D_rB= 5, m_valM=              0, M_valA=              x ,D_valC=
      x, e_valE=              3

F_PC_in=             15 F_PC_out=             25 D_icode= 3, ifun= 0,E_icode= 2, M_icode= 6 , W_icode= 1 ,D_rA=15,D_rB= 1, m_valM=              0, M_valA=              4 ,D_valC=
     26, e_valE=              4

F_PC_in=             25 F_PC_out=             35 D_icode= 4, ifun= 0,E_icode= 3, M_icode= 2 , W_icode= 6 ,D_rA= 6,D_rB= 5, m_valM=              0, M_valA=              4 ,D_valC=
      5, e_valE=             26

F_PC_in=             35 F_PC_out=             37 D_icode= 5, ifun= 0,E_icode= 4, M_icode= 3 , W_icode= 2 ,D_rA= 8,D_rB= 1, m_valM=              0, M_valA=              4 ,D_valC=
      6, e_valE=             10

F_PC_in=             37 F_PC_out=             48 D_icode= 6, ifun= 0,E_icode= 5, M_icode= 4 , W_icode= 3 ,D_rA= 2,D_rB= 3, m_valM=              6, M_valA=              6 ,D_valC=
      6, e_valE=             16

F_PC_in=             48 F_PC_out=             58 D_icode= 7, ifun= 4,E_icode= 6, M_icode= 5 , W_icode= 4 ,D_rA= 2,D_rB= 3, m_valM=             16, M_valA=              6 ,D_valC=
     48, e_valE=              5


************************* D_bubble and E_bubble******************

F_PC_in=             58 F_PC_out=             60 D_icode= 3, ifun= 2,E_icode= 7, M_icode= 6 , W_icode= 5 ,D_rA= 0,D_rB= 0, m_valM=             16, M_valA=              2 ,D_valC=
  14864, e_valE=              5

F_PC_in=             60 F_PC_out=             62 D_icode=10, ifun= 0,E_icode= 3, M_icode= 7 , W_icode= 6 ,D_rA= 6,D_rB=15, m_valM=             16, M_valA=             46 ,D_valC=
  14864, e_valE=          14864

F_PC_in=             62 F_PC_out=             62 D_icode=11, ifun= 0,E_icode=10, M_icode= 3 , W_icode= 7 ,D_rA= 5,D_rB=15, m_valM=             16, M_valA=             46 ,D_valC=
  14864, e_valE=              3

F_PC_in=             62 F_PC_out=             62 D_icode= 0, ifun= 0,E_icode=11, M_icode=10 , W_icode= 3 ,D_rA= 5,D_rB=15, m_valM=              6, M_valA=              6 ,D_valC=
  14864, e_valE=              4

F_PC_in=             62 F_PC_out=             62 D_icode= 0, ifun= 0,E_icode= 0, M_icode=11 , W_icode=10 ,D_rA= 5,D_rB=15, m_valM=              6, M_valA=              3 ,D_valC=
  14864, e_valE=              4

F_PC_in=             62 F_PC_out=             62 D_icode= 0, ifun= 0,E_icode= 0, M_icode= 0 , W_icode=11 ,D_rA= 5,D_rB=15, m_valM=              6, M_valA=              3 ,D_valC=
  14864, e_valE=              4

******************** Halting *********************
F_PC_in=             62 F_PC_out=             62 D_icode= 0, ifun= 0,E_icode= 0, M_icode= 0 , W_icode= 0 ,D_rA= 5,D_rB=15, m_valM=              6, M_valA=              3 ,D_valC=
  14864, e_valE=              4
```

GTKwave - pipeline

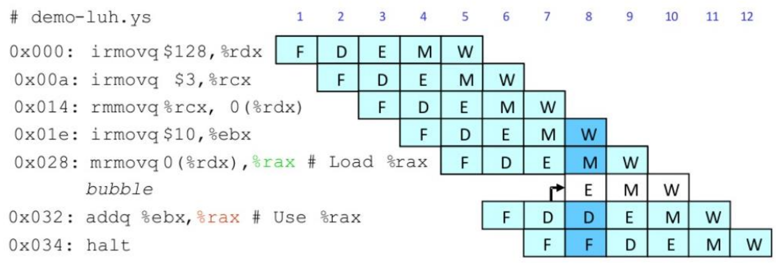


GTKwave sequential


From: 0 sec
To: 120 ns
Marker: -- | Cursor: 570 ps
SST
processor_tb
temp2
temp3
temp4
temp5
temp6
Signals
Waves
Time
CC_in[2:0]
CC_out[2:0]
O_F
S_F
Z_F
cond
icode[3:0]
ifun[3:0]
instruct_err
mem_err
next_pc[63:0]
ra[3:0]
rb[3:0]
valA[63:0]
valB[63:0]
valC[63:0]
valE[63:0]
valM[63:0]
valP[63:0]
Type
Signals
regis0[63:0]
regis1[63:0]
regis2[63:0]
regis3[63:0]
regis4[63:0]
regis5[63:0]
regis6[63:0]
regis7[63:0]
regis8[63:0]
regis9[63:0]
regis10[63:0]
regis11[63:0]
regis12[63:0]
regis13[63:0]
regis14[63:0]
valA[63:0]
valB[63:0]
valC[63:0]
valE[63:0]
valM[63:0]
valP[63:0]
Filter:
Append
Insert
Replace

HAZARDS

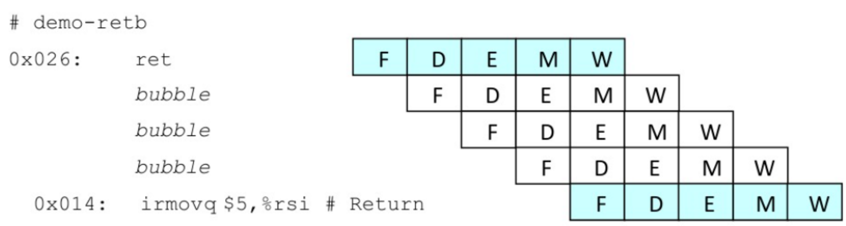
Control for Load/Use Hazard



- Stall instructions in fetch and decode stages
- Inject bubble into execute stage

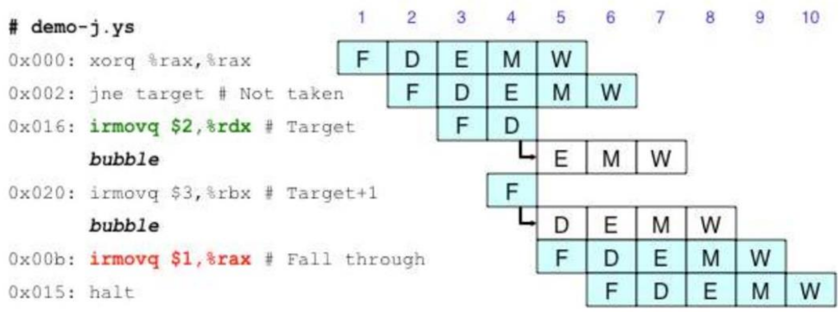
Condition	F	D	E	M	W
Load/Use Hazard	stall	stall	bubble	normal	normal

Control for Return



Condition	F	D	E	M	W
Processing `ret`	stall	bubble	normal	normal	normal

Control for Misprediction



Condition	F	D	E	M	W
Mispredicted Branch	normal	bubble	bubble	normal	normal

Hazard control

```verilog

    output reg F_stall,D_stall,D_bubble,E_bubble,set_cc;

always@(*)
    begin
        F_stall=0;D_stall=0;D_bubble=0;E_bubble=0;set_cc=1;//initially all the stalls and bubbles are 0 and condition is true
        //Processing ret
        if(E_icode == 4'h9 || M_icode == 4'h9 || D_icode == 4'h9)
        begin
            $display("\n*************** Fstall and D_bubble******************\n");
            F_stall = 1;
            D_bubble = 1;
        end
        else if((E_icode == 4'h5 || E_icode == 4'hB) && (E_dstM==d_srcA || E_dstM==d_srcB))//Load/use Hazard
        begin
            $display("\n************************ Fstall and D_stall and E_bubble********************\n");
            F_stall = 1;
            E_bubble = 1;
            D_stall = 1;
        end
        else if(E_icode==4'h7 && !e_cnd)
        begin
            $display("\n************************* D_bubble and E_bubble******************\n");
            D_bubble=1;
            E_bubble=1;
        end
        else if(E_icode==4'h0 || m_stat!=4'b1000 || W_stat!=4'b1000)  //If halting condition in Execute stage or memory stat and writeback stats are " NOT OK " then set_cc=0;
```

Challenges faced

1) Implementing control logic for this pipelined processor was challenging.
2) Data forwarding was challenging in pipeline implementation.
3) Working of call and return was challenging.