# Intro to Processor Architecture

## Y86-64 Sequential + Pipelined processor design

### Project Report

**Team Members:**

**Name:** V.S.S.Pradhith

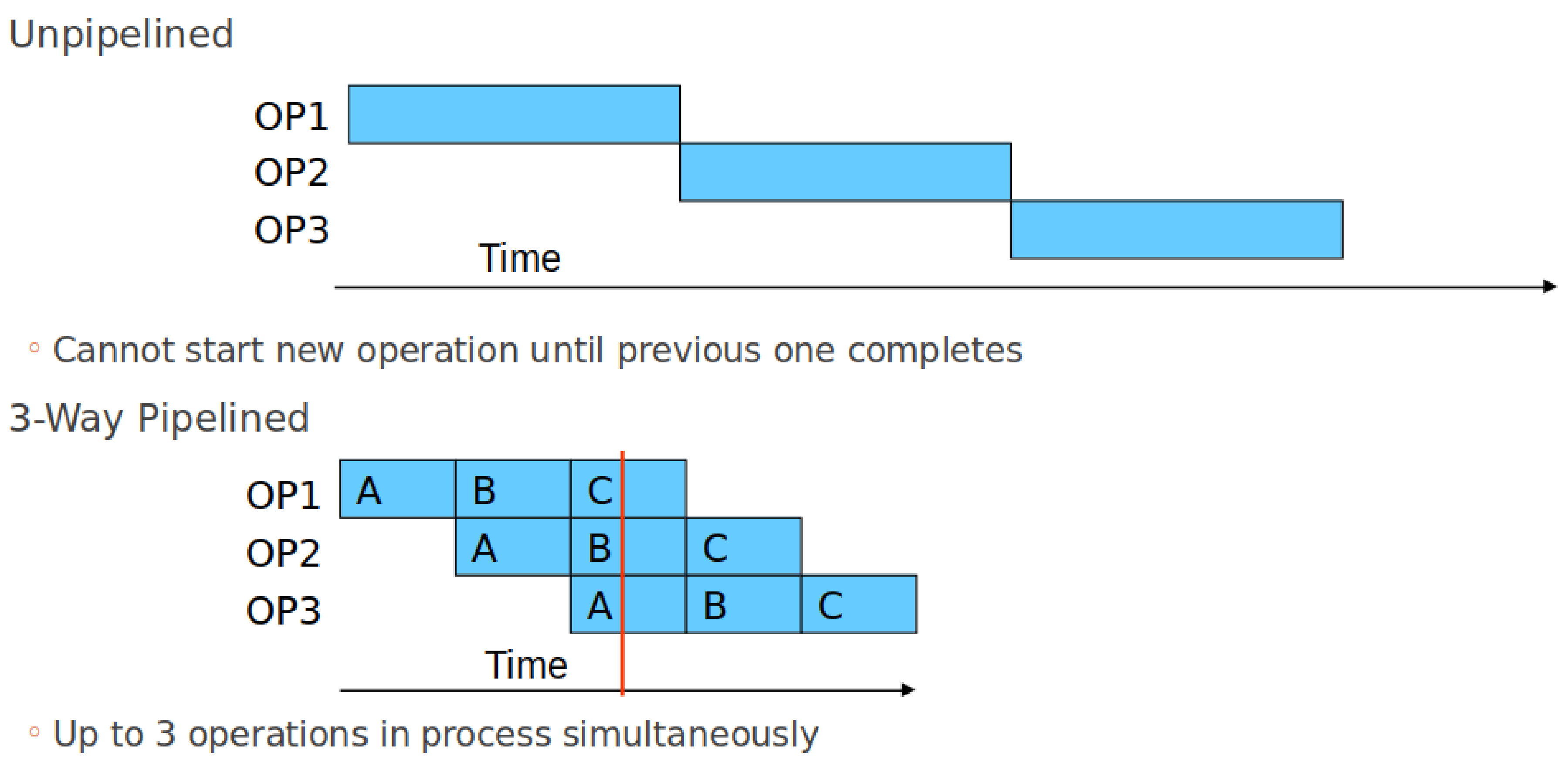
**Roll no. :** 2021102015

**Name:** K. Vishnu

**Roll no. :** 2021102034

## Overview:

Our goal is to implement a sequential Y86-64 processor as well as a pipelined Y86-64 processor where the sequential processor is implemented in 6 stages (F, D, E, M, WB, PC) and the pipelined one is implemented in 5 stages (F, D, E, M, WB) only. The basic difference between a sequential implementation and a pipelined implementation is as follows:


Unpipelined
OP1
OP2
OP3
Time
◦ Cannot start new operation until previous one completes
3-Way Pipelined
OP1
OP2
OP3
A
B
C
A
B
C
A
B
C
Time
◦ Up to 3 operations in process simultaneously

where OP1, OP2 and OP3 are three operations which has to be implemented respectively.

This is how we can reduce the delay as well as increase the throughput (defined as no. of instructions sent per second) of the design in pipelined model. But still there can be few problems which can be encountered while pipeline architecture is in use such as data dependencies, data hazards, etc which needs to be handled immediately.

In general, we can modify the sequential hardware implementation into pipelined hardware implementation by making few changes by inserting pipeline registers at the end of every stage and excluding the PC update stage as well.

## SEQ and PIPE Hardware:

SEQ (sequential processor) and the PIPE (pipelined processor) hardware implementations are almost same as they almost follow similar stages of implementation in common but there are few additional register pipelines in the PIPE hardware. The hardware for both of these implementations are as follows:

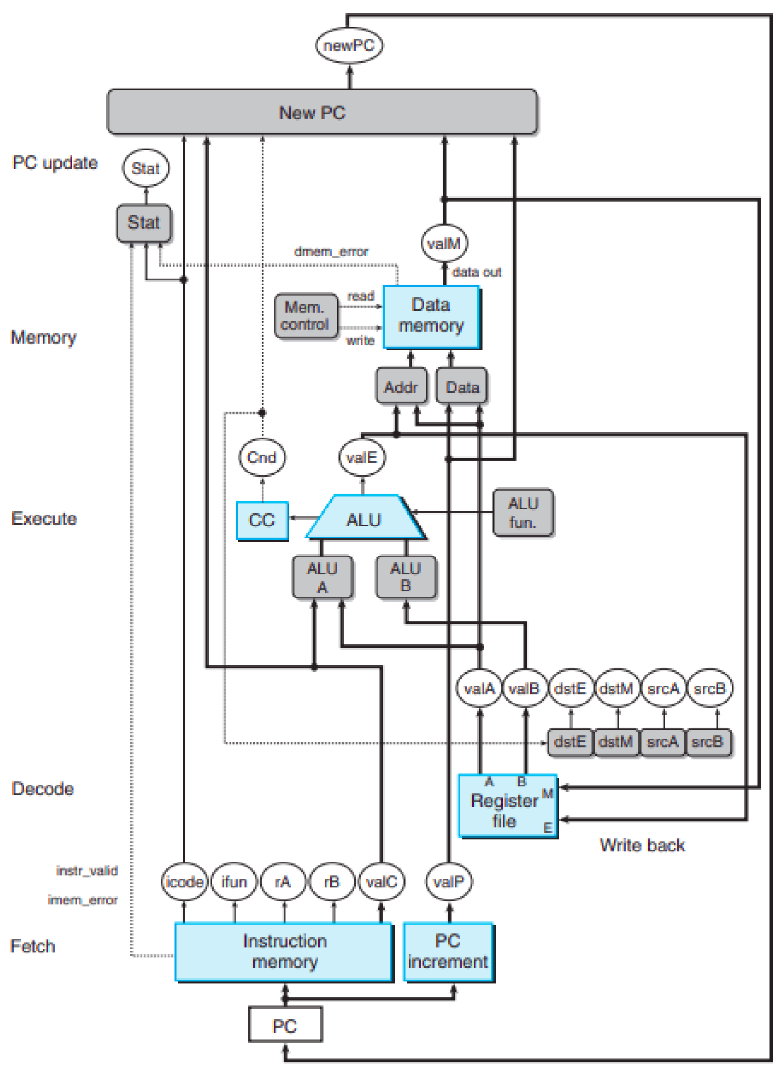


Fig: SEQ hardware implementation

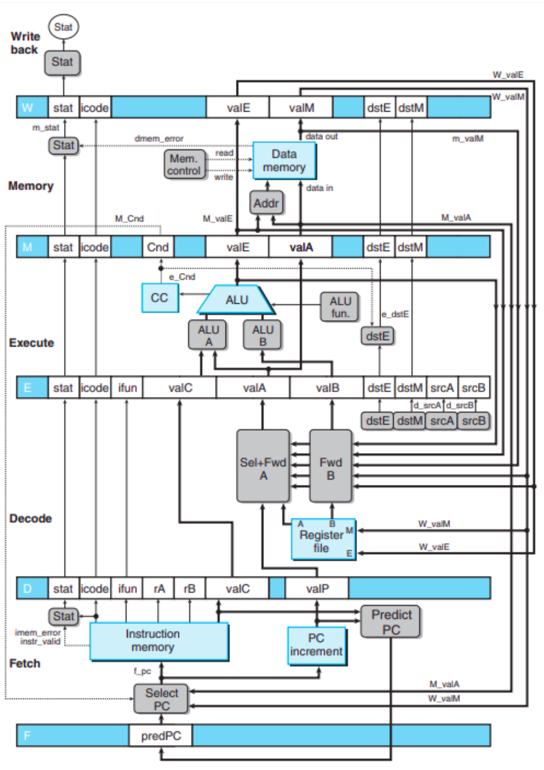


Fig: PIPE hardware implementation

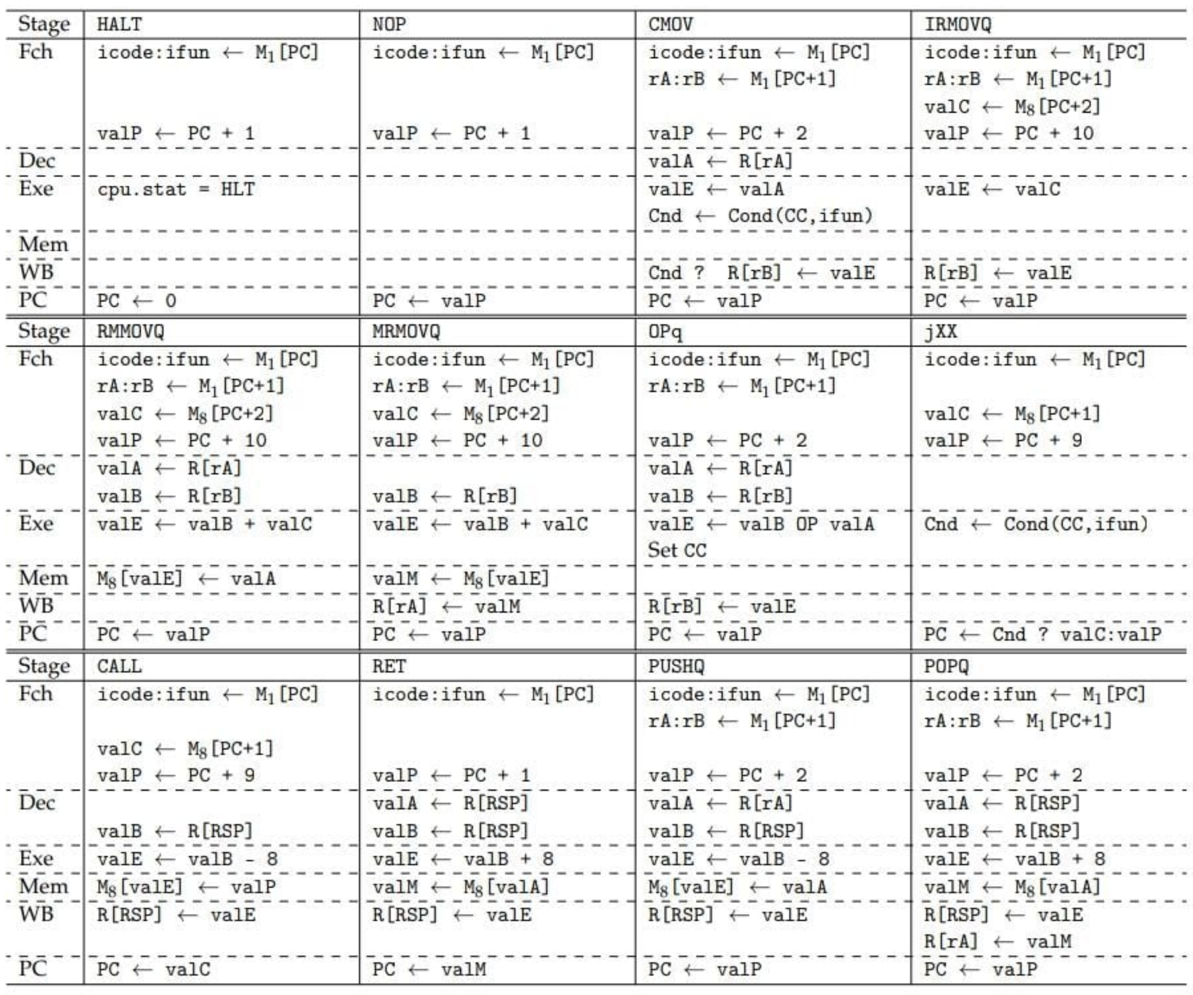
## Different instruction stages in SEQ and PIPE processor:

Each instruction goes through following common stages:

- Fetch : Reads instruction from memory
- Decode : Reads program registers
- Execute : Compute value or addresses
- Memory : Read or write data
- Write-back : Write program registers
- PC update (only in SEQ processor) : Updates program counter

Only PC update stage is not available in PIPE design, whereas rest all stages are common in both type of architectures.

The working of above instruction stages for SEQ architecture can be summarised as follows:



| Stage | HALT | NOP | CMOV | IRMOVQ |
|---|---|---|---|---|
| Fch | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] |
| | | | rA:rB ← $M_1$[PC+1] | rA:rB ← $M_1$[PC+1] |
| | | | | valC ← $M_8$[PC+2] |
| | valP ← PC + 1 | valP ← PC + 1 | valP ← PC + 2 | valP ← PC + 10 |
| Dec | | | valA ← R[rA] | |
| Exe | cpu.stat = HLT | | valE ← valA | valE ← valC |
| | | | Cnd ← Cond(CC,ifun) | |
| Mem | | | | |
| WB | | | Cnd ? R[rB] ← valE | R[rB] ← valE |
| PC | PC ← 0 | PC ← valP | PC ← valP | PC ← valP |

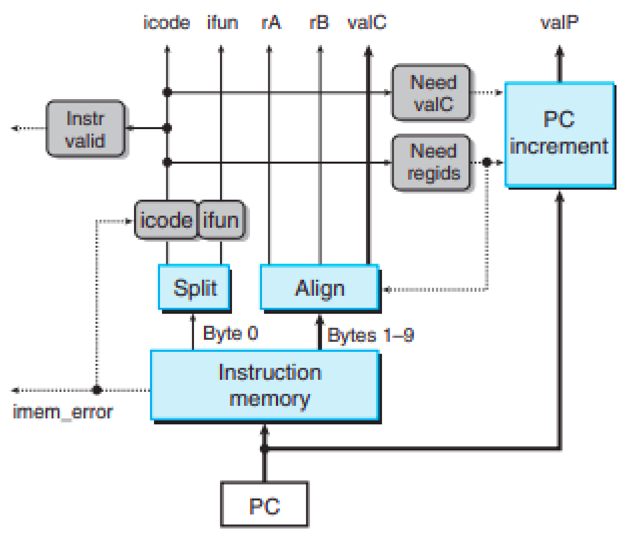
| Stage | RMMOVQ | MRMOVQ | OPq | jXX |
|---|---|---|---|---|
| Fch | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] |
| | rA:rB ← $M_1$[PC+1] | rA:rB ← $M_1$[PC+1] | rA:rB ← $M_1$[PC+1] | |
| | valC ← $M_8$[PC+2] | valC ← $M_8$[PC+2] | | valC ← $M_8$[PC+1] |
| | valP ← PC + 10 | valP ← PC + 10 | valP ← PC + 2 | valP ← PC + 9 |
| Dec | valA ← R[rA] | | valA ← R[rA] | |
| | valB ← R[rB] | valB ← R[rB] | valB ← R[rB] | |
| Exe | valE ← valB + valC | valE ← valB + valC | valE ← valB OP valA | Cnd ← Cond(CC,ifun) |
| | | | Set CC | |
| 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] | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] | icode:ifun ← $M_1$[PC] |
| | | | rA:rB ← $M_1$[PC+1] | rA:rB ← $M_1$[PC+1] |
| | valC ← $M_8$[PC+1] | | | |
| | valP ← PC + 9 | valP ← PC + 1 | valP ← PC + 2 | valP ← PC + 2 |
| Dec | | valA ← R[RSP] | valA ← R[rA] | valA ← R[RSP] |
| | valB ← R[RSP] | valB ← R[RSP] | valB ← 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 |

## Design logic for each stage in both type of architectures:

### a) Fetch stage:

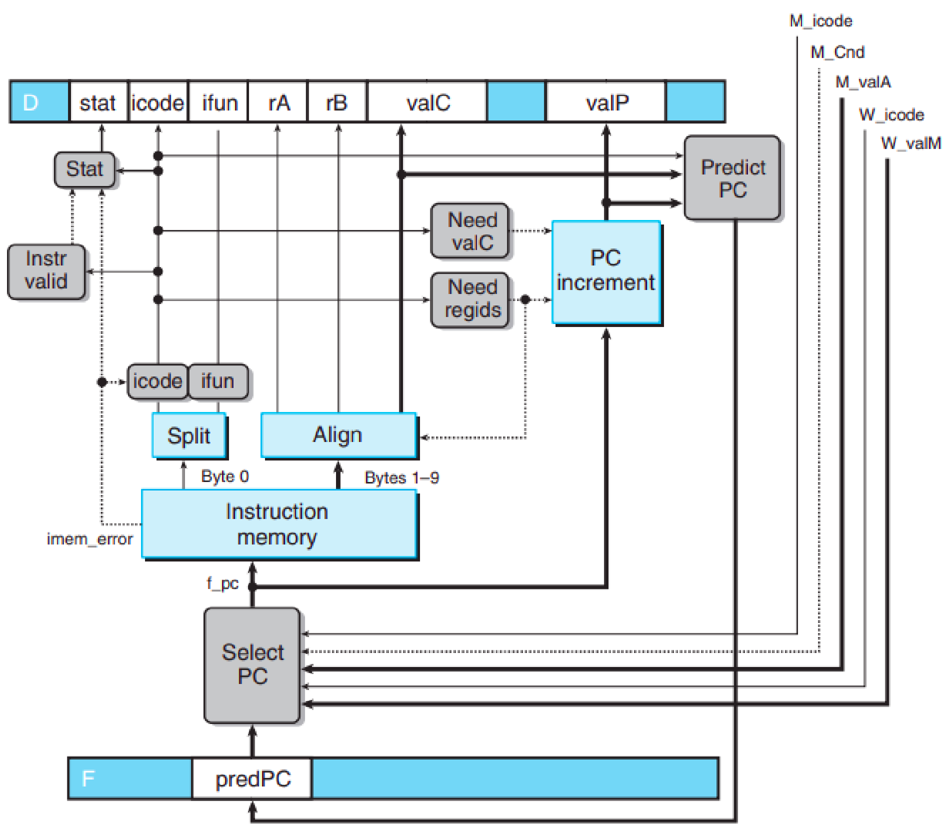
Consider the SEQ fetch stage:



Six bytes are read from the instruction memory using the PC as the starting address. From these bytes, we generate the different instruction fields. The PC increment block computes signal valP.

In fetch stage the processor reads at most 10 bytes of instruction and then it partitions the bytes into 1 byte for **icode** and **ifun** and 1 bytes for registers **rA, rB** and remaining 8 bytes as **valC** and sets PC value as **valP**.

Consider the PIPE fetch stage:



The fundamental building block of PIPE fetch stage is similar to that of the entire SEQ fetch stage. Additionally, there are fetch and decode pipelined registers for storing predicted PC and the signals (valP, valC, rA, rB, etc.) generated in the fetch stage respectively.

As we are not performing PC update stage in PIPE implementation, we are predicting the next PC value and are updating it directly in fetch stage only and this process is called PC prediction.

The value of predicted PC depends on parameters such as **icode, valC** and **valP**.  Within the one cycle time limit, the processor can only predict the address of the next instruction.

**PC prediction strategy :**

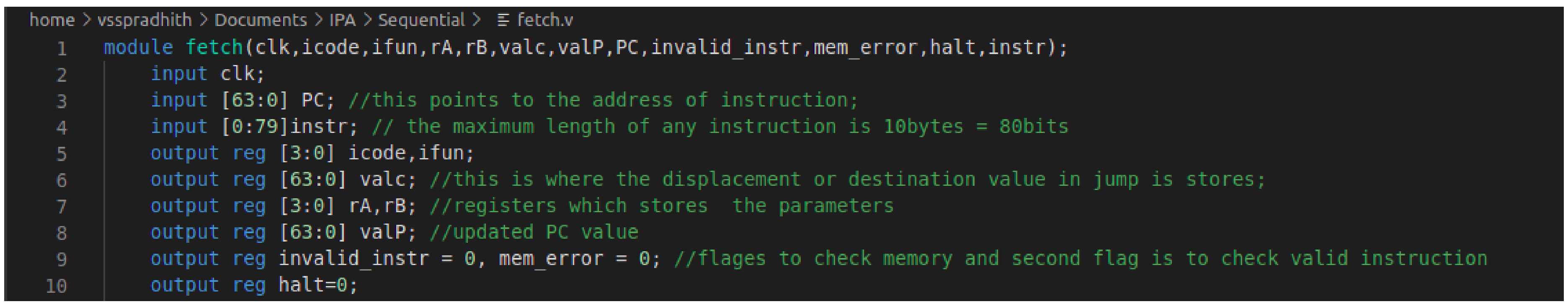
i) Instructions that don't transfer control : Predict next PC to valP. This is always reliable.

ii) Call and unconditional jumps : Predict next PC to be valC (destination). This also always reliable.

iii) Conditional jumps : Predict next PC to be valC (destination). It's only correct if branch is taken. Typically right 60% of time

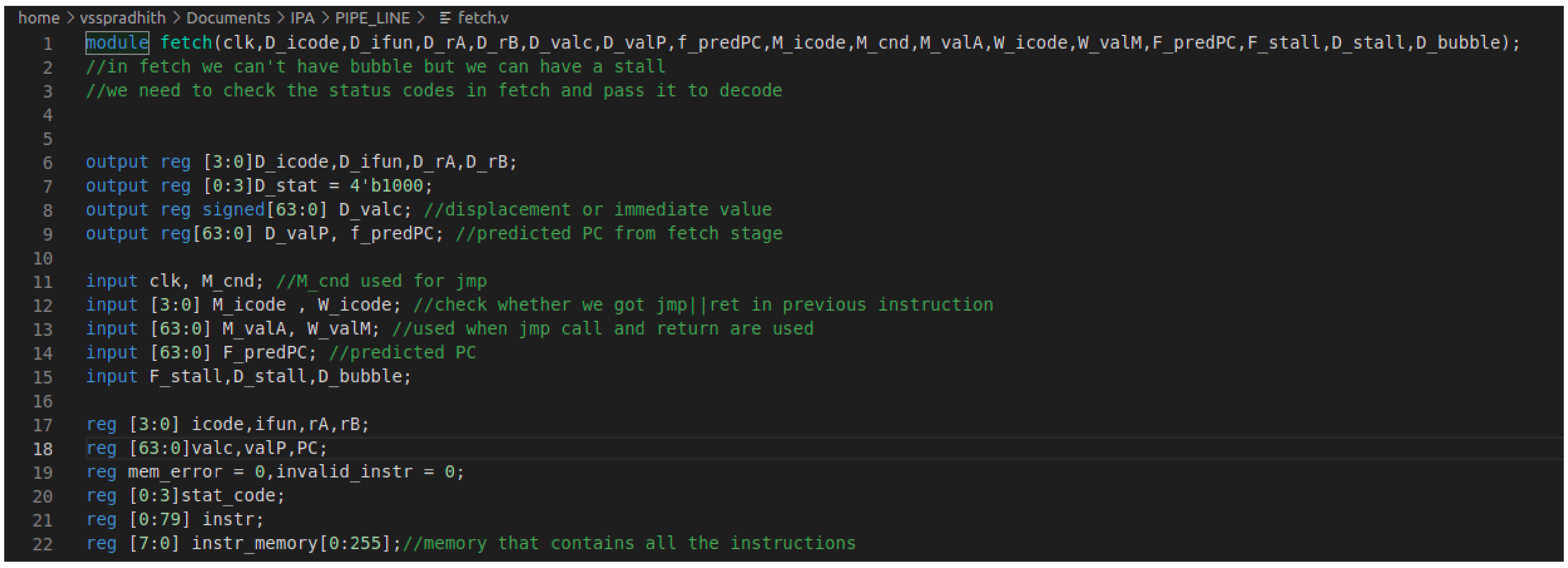
iv) Return instruction : Don't try to predict.

We have referred to the above block diagrams while writing the fetch modules in verilog and the modules are defined as follows:


home > vsspradhith > Documents > IPA > Sequential > fetch.v

```
module fetch(clk,icode,ifun,rA,rB,valc,valP,PC,invalid_instr,mem_error,halt,instr);
    input clk;
    input [63:0] PC; //this points to the address of instruction;
    input [0:79]instr; // the maximum length of any instruction is 10bytes = 80bits
    output reg [3:0] icode,ifun;
    output reg [63:0] valc; //this is where the displacement or destination value in jump is stores;
    output reg [3:0] rA,rB; //registers which stores  the parameters
    output reg [63:0] valP; //updated PC value
    output reg invalid_instr = 0, mem_error = 0; //flages to check memory and second flag is to check valid instruction
    output reg halt=0;
```

SEQ fetch


home > vsspradhith > Documents > IPA > PIPE_LINE > fetch.v

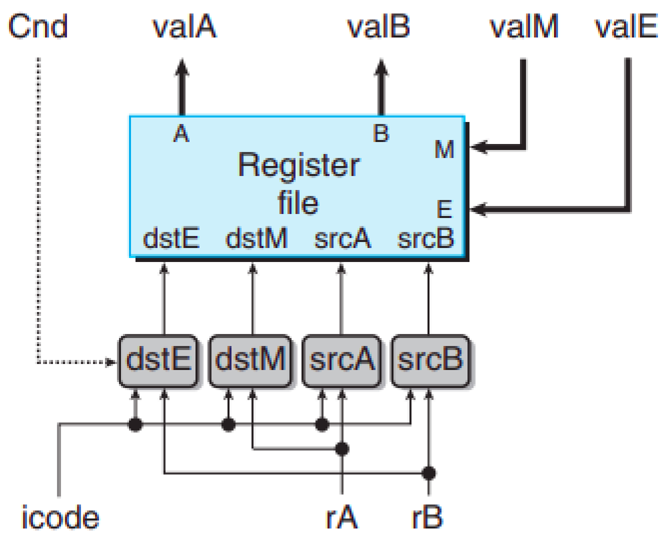
```
module fetch(clk,D_icode,D_ifun,D_rA,D_rB,D_valc,D_valP,f_predPC,M_icode,M_cnd,M_valA,W_icode,W_valM,F_predPC,F_stall,D_stall,D_bubble);
//in fetch we can't have bubble but we can have a stall
//we need to check the status codes in fetch and pass it to decode


output reg [3:0]D_icode,D_ifun,D_rA,D_rB;
output reg [0:3]D_stat = 4'b1000;
output reg signed[63:0] D_valc; //displacement or immediate value
output reg[63:0] D_valP, f_predPC; //predicted PC from fetch stage

input clk, M_cnd; //M_cnd used for jmp
input [3:0] M_icode , W_icode; //check whether we got jmp||ret in previous instruction
input [63:0] M_valA, W_valM; //used when jmp call and return are used
input [63:0] F_predPC; //predicted PC
input F_stall,D_stall,D_bubble;

reg [3:0] icode,ifun,rA,rB;
reg [63:0]valc,valP,PC;
reg mem_error = 0,invalid_instr = 0;
reg [0:3]stat_code;
reg [0:79] instr;
reg [7:0] instr_memory[0:255];//memory that contains all the instructions
```

PIPE fetch

### b) Decode + Write-back stage:

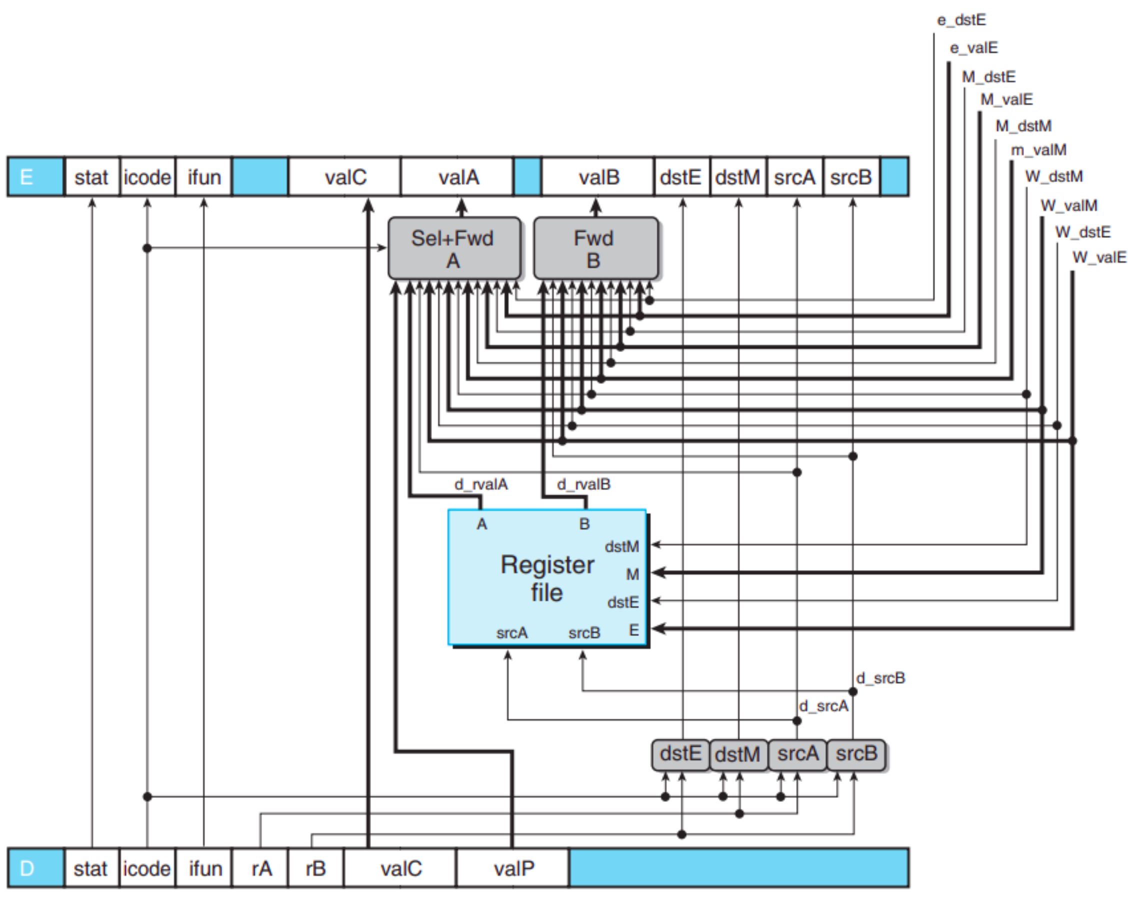
In both SEQ and PIPE implementations, we are implementing decode and write-back stages collectively as we are dealing with register files which contain registers for reading (decode stage) and writing (write-back stage) the data.

Consider the SEQ decode + write-back stage:



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

Now, consider the PIPE decode + write-back stage:

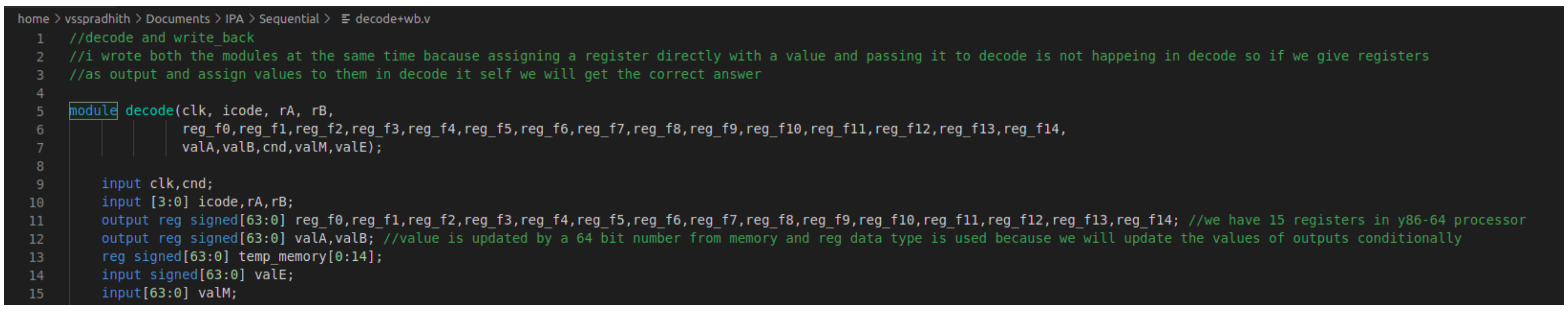


The fundamental building block of PIPE decode + write-back stage is similar to that of the entire SEQ decode + write-back stage. Additionally, there are decode and execute pipelined registers for storing the signals (**D_valP, D_valC, rA, rB**, etc.) generated in the fetch stage and storing the signals (**E_valP, E_valC, E_dstM, E_dstE, E_srcA, E_srcB**, etc.) generated in the decode + write-back stage respectively.

No instruction requires both valP and the value read from register port A, and so these two can be merged to form the signal valA for later stages. The block labeled "**Sel+Fwd A**" performs this task and also implements the forwarding logic for source operand valA. The block labeled "**Fwd B**" implements the forwarding logic for source operand valB.

The register write locations are specified by the **dstE** and **dstM** signals from the write-back stage rather than from the decode stage, since it is writing the results of the instruction currently in the write-back stage.

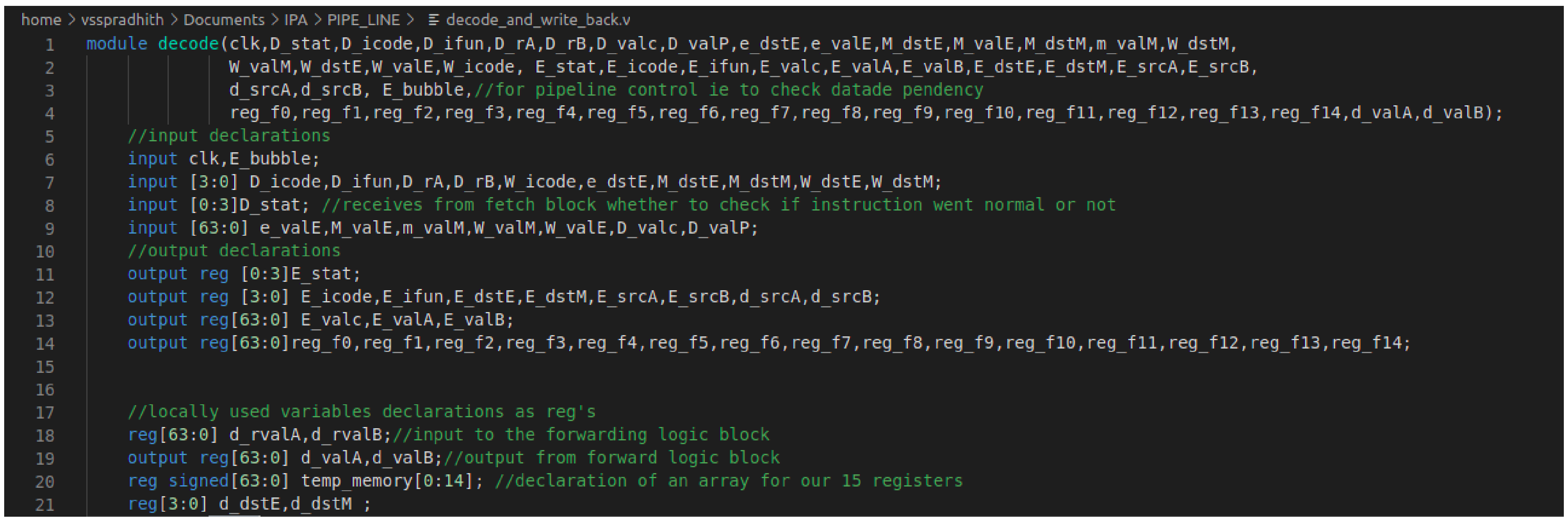
We have referred to the above block diagrams while writing the decode+write-back modules in verilog and we defined our modules as follows:



home > vsspradhith > Documents > IPA > Sequential > decode+wb.v

```
//decode and write_back
//i wrote both the modules at the same time bacause assigning a register directly with a value and passing it to decode is not happeing in decode so if we give registers
//as output and assign values to them in decode it self we will get the correct answer

module decode(clk, icode, rA, rB,
                reg_f0,reg_f1,reg_f2,reg_f3,reg_f4,reg_f5,reg_f6,reg_f7,reg_f8,reg_f9,reg_f10,reg_f11,reg_f12,reg_f13,reg_f14,
                valA,valB,cnd,valM,valE);

    input clk,cnd;
    input [3:0] icode,rA,rB;
    output reg signed[63:0] reg_f0,reg_f1,reg_f2,reg_f3,reg_f4,reg_f5,reg_f6,reg_f7,reg_f8,reg_f9,reg_f10,reg_f11,reg_f12,reg_f13,reg_f14; //we have 15 registers in y86-64 processor
    output reg signed[63:0] valA,valB; //value is updated by a 64 bit number from memory and reg data type is used because we will update the values of outputs conditionally
    reg signed[63:0] temp_memory[0:14];
    input signed[63:0] valE;
    input[63:0] valM;
```

**SEQ decode+write-back**

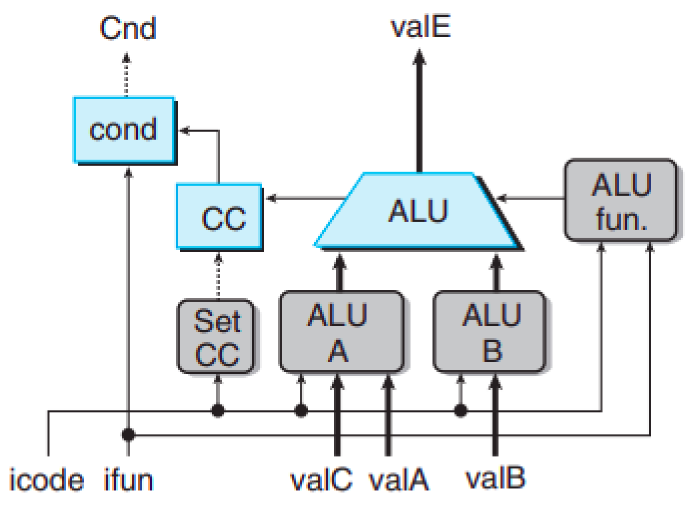

home > vsspradhith > Documents > IPA > PIPE_LINE > decode_and_write_back.v

```verilog
module decode(clk,D_stat,D_icode,D_ifun,D_rA,D_rB,D_valc,D_valP,e_dstE,e_valE,M_dstE,M_valE,M_dstM,m_valM,W_dstM,
              W_valM,W_dstE,W_valE,W_icode, E_stat,E_icode,E_ifun,E_valc,E_valA,E_valB,E_dstE,E_dstM,E_srcA,E_srcB,
              d_srcA,d_srcB, E_bubble,//for pipeline control ie to check datade pendency
              reg_f0,reg_f1,reg_f2,reg_f3,reg_f4,reg_f5,reg_f6,reg_f7,reg_f8,reg_f9,reg_f10,reg_f11,reg_f12,reg_f13,reg_f14,d_valA,d_valB);
    //input declarations
    input clk,E_bubble;
    input [3:0] D_icode,D_ifun,D_rA,D_rB,W_icode,e_dstE,M_dstE,M_dstM,W_dstE,W_dstM;
    input [0:3]D_stat; //receives from fetch block whether to check if instruction went normal or not
    input [63:0] e_valE,M_valE,m_valM,W_valM,W_valE,D_valc,D_valP;
    //output declarations
    output reg [0:3]E_stat;
    output reg [3:0] E_icode,E_ifun,E_dstE,E_dstM,E_srcA,E_srcB,d_srcA,d_srcB;
    output reg[63:0] E_valc,E_valA,E_valB;
    output reg[63:0]reg_f0,reg_f1,reg_f2,reg_f3,reg_f4,reg_f5,reg_f6,reg_f7,reg_f8,reg_f9,reg_f10,reg_f11,reg_f12,reg_f13,reg_f14;


    //locally used variables declarations as reg's
    reg[63:0] d_rvalA,d_rvalB;//input to the forwarding logic block
    output reg[63:0] d_valA,d_valB;//output from forward logic block
    reg signed[63:0] temp_memory[0:14]; //declaration of an array for our 15 registers
    reg[3:0] d_dstE,d_dstM ;
```

PIPE decode+write-back

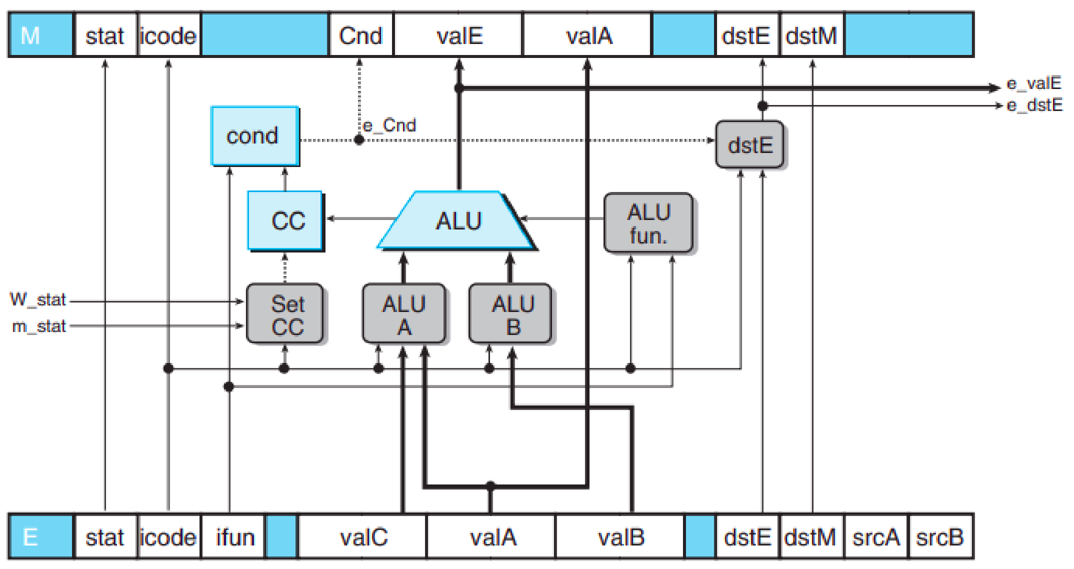
## c) Execute stage:

Consider the SEQ execute stage:



The ALU either performs the operation for an integer operation instruction or acts as an adder. The condition code registers are set according to the ALU value. The condition code values are tested to determine whether a branch should be taken.

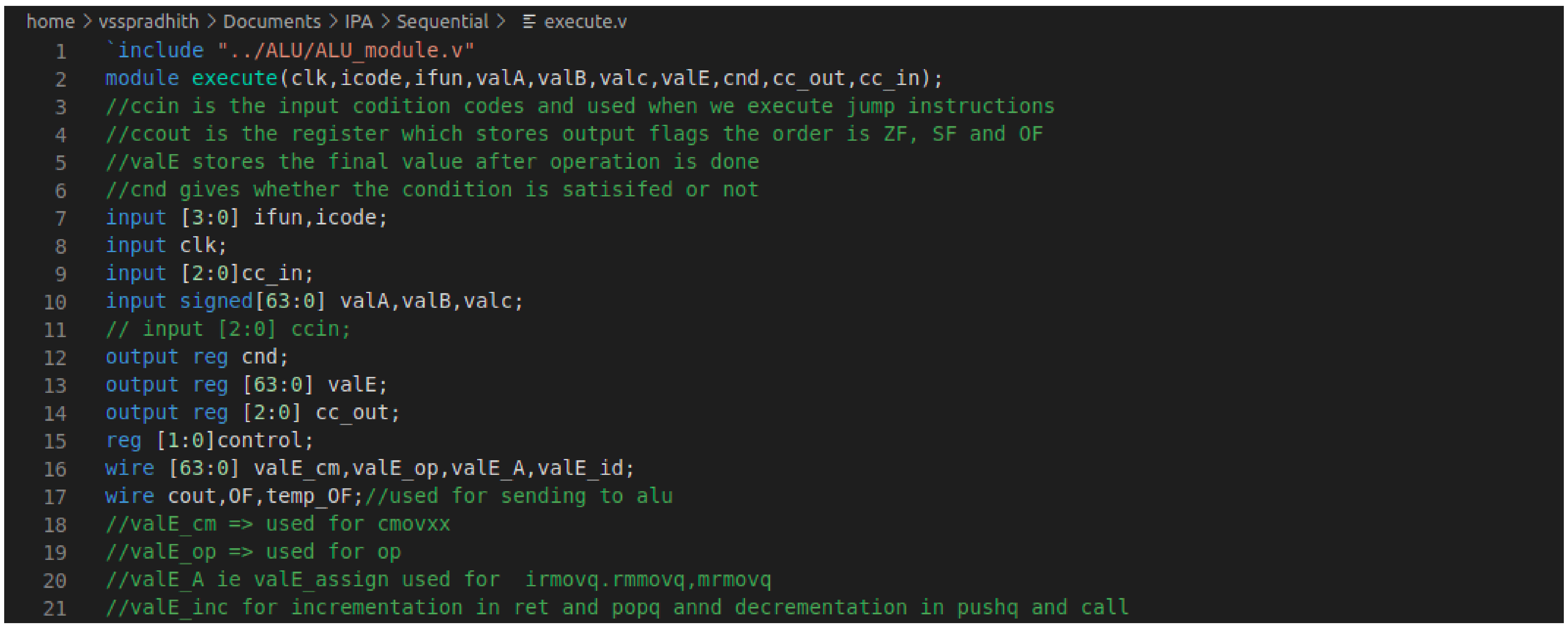
Consider the PIPE execute stage:



The hardware units and the logic blocks are identical to those in SEQ, with an appropriate renaming of signals. We can see the signals **e_valE** and **e_dstE** directed toward the decode stage as one of the forwarding sources. One difference 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.

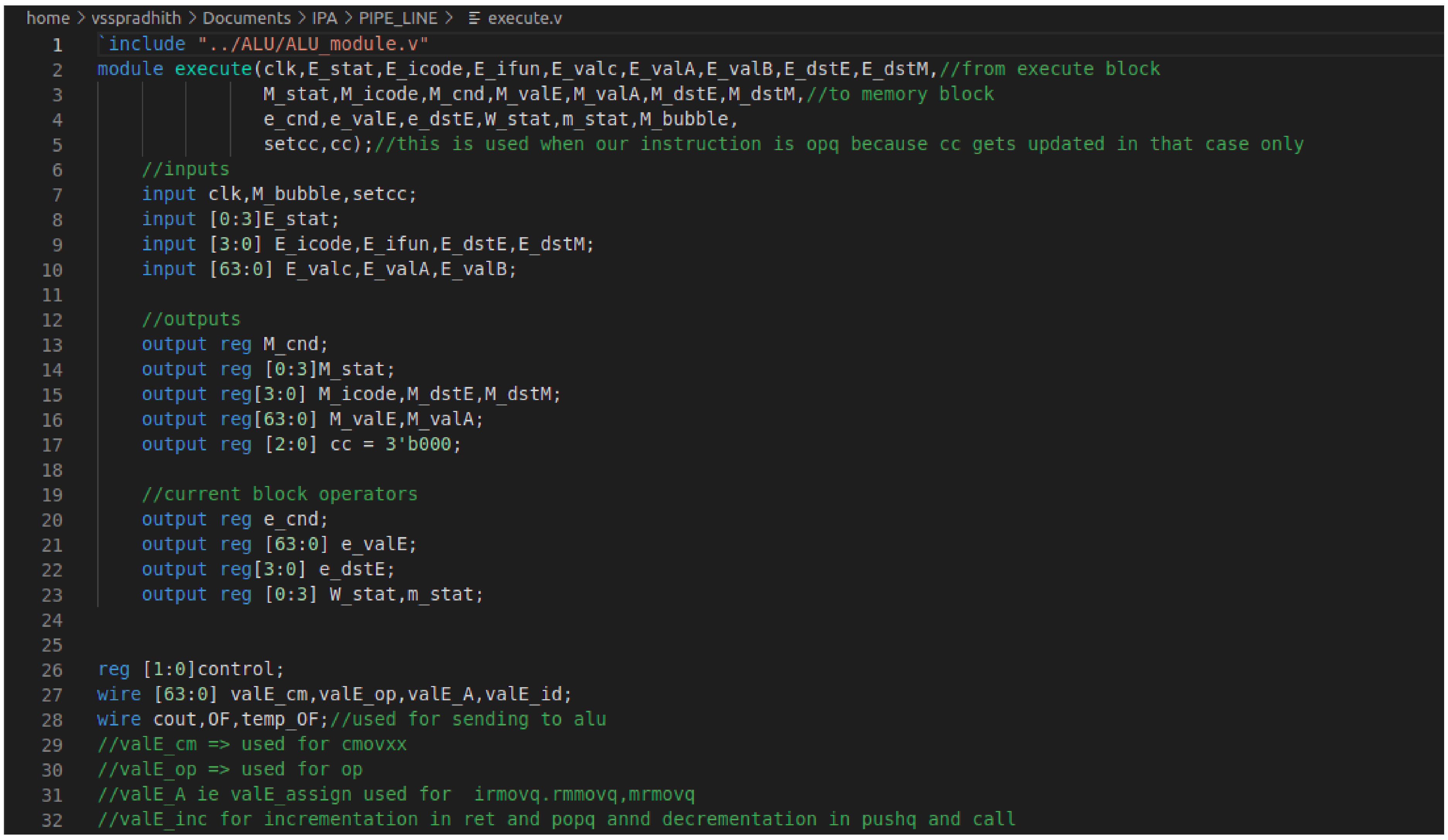
These signals are used to detect cases where an instruction  causing an exception is passing through later pipeline stages, and therefore any updating of the condition codes should be suppressed.

We have referred to the above block diagrams while writing the execute modules in verilog and we defined our modules as follows:


home > vsspradhith > Documents > IPA > Sequential > execute.v

```verilog
`include "../ALU/ALU_module.v"
module execute(clk,icode,ifun,valA,valB,valc,valE,cnd,cc_out,cc_in);
//ccin is the input codition codes and used when we execute jump instructions
//ccout is the register which stores output flags the order is ZF, SF and OF
//valE stores the final value after operation is done
//cnd gives whether the condition is satisifed or not
input [3:0] ifun,icode;
input clk;
input [2:0]cc_in;
input signed[63:0] valA,valB,valc;
// input [2:0] ccin;
output reg cnd;
output reg [63:0] valE;
output reg [2:0] cc_out;
reg [1:0]control;
wire [63:0] valE_cm,valE_op,valE_A,valE_id;
wire cout,OF,temp_OF;//used for sending to alu
//valE_cm => used for cmovxx
//valE_op => used for op
//valE_A ie valE_assign used for  irmovq.rmmovq,mrmovq
//valE_inc for incrementation in ret and popq annd decrementation in pushq and call
```

SEQ execute

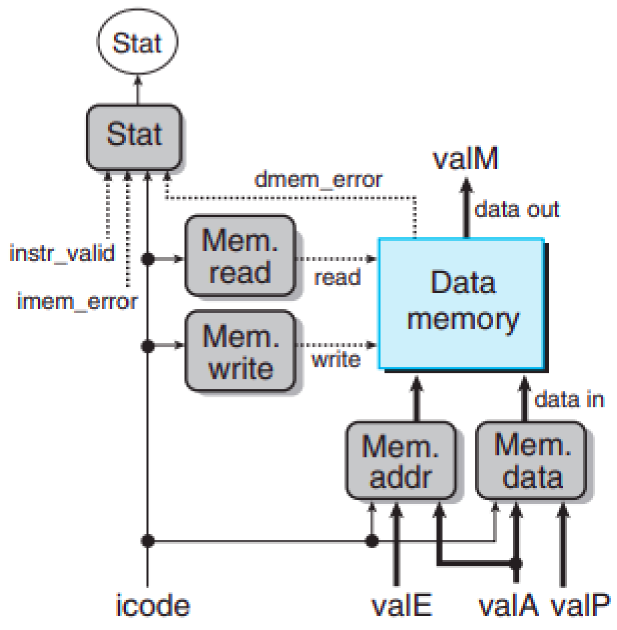

home > vsspradhith > Documents > IPA > PIPE_LINE > execute.v

```verilog
`include "../ALU/ALU_module.v"
module execute(clk,E_stat,E_icode,E_ifun,E_valc,E_valA,E_valB,E_dstE,E_dstM,//from execute block
               M_stat,M_icode,M_cnd,M_valE,M_valA,M_dstE,M_dstM,//to memory block
               e_cnd,e_valE,e_dstE,W_stat,m_stat,M_bubble,
               setcc,cc);//this is used when our instruction is opq because cc gets updated in that case only
    //inputs
    input clk,M_bubble,setcc;
    input [0:3]E_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;
    output reg[63:0] M_valE,M_valA;
    output reg [2:0] cc = 3'b000;

    //current block operators
    output reg e_cnd;
    output reg [63:0] e_valE;
    output reg[3:0] e_dstE;
    output reg [0:3] W_stat,m_stat;


reg [1:0]control;
wire [63:0] valE_cm,valE_op,valE_A,valE_id;
wire cout,OF,temp_OF;//used for sending to alu
//valE_cm => used for cmovxx
//valE_op => used for op
//valE_A ie valE_assign used for  irmovq.rmmovq,mrmovq
//valE_inc for incrementation in ret and popq annd decrementation in pushq and call
```

PIPE execute

### d) Memory stage:

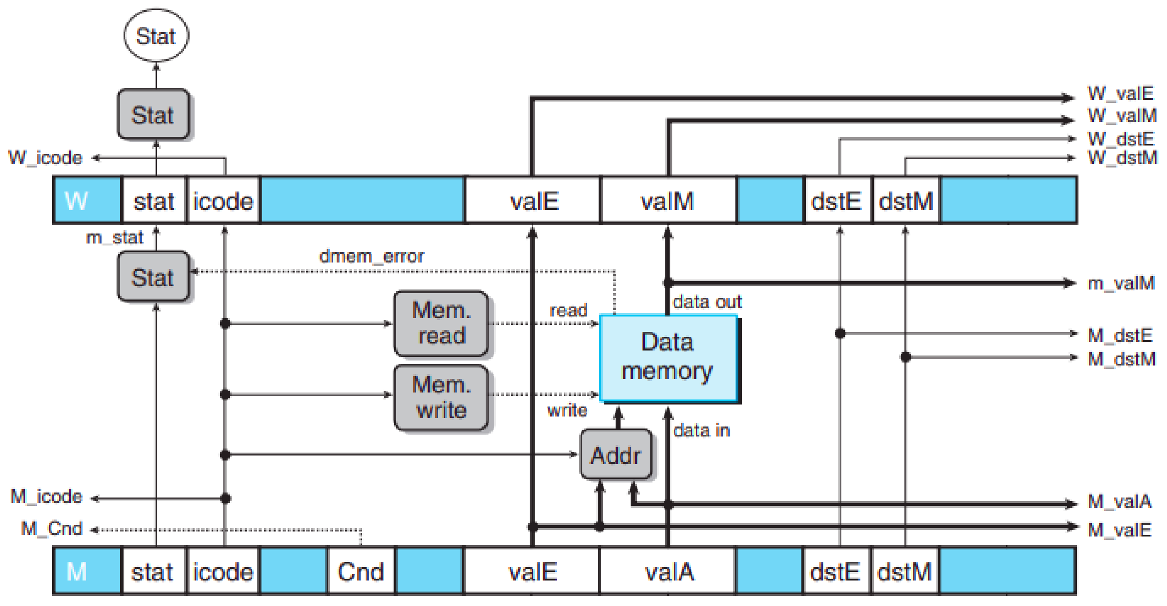
Consider, SEQ memory stage:



The data memory can either write or read memory values. The value read from
memory forms the signal **valM**.

The memory stage has the task of either reading or writing program data. As shown in above figure, two control blocks generate the values for the memory address and the memory input data (for write operations). Two other blocks generate the control signals indicating whether to perform a read or a write operation. When a read operation is performed, the data memory generates the value **valM**.

Consider, PIPE memory stage:

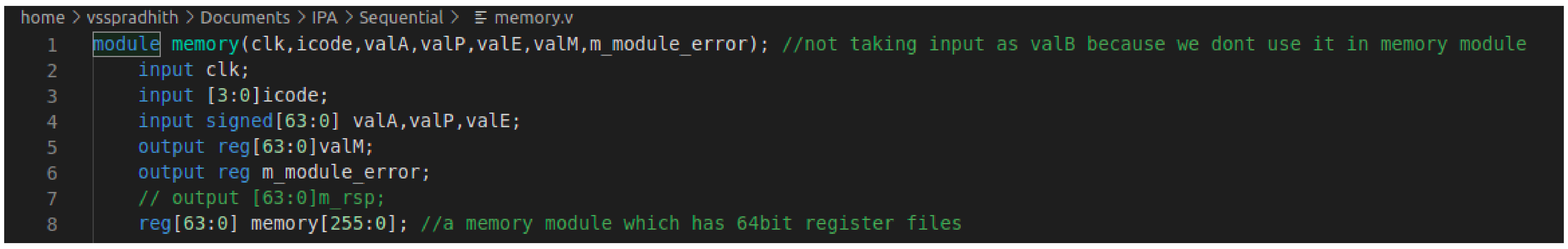


Many of the signals from pipeline registers M and W are passed down to earlier stages to provide write-back results, instruction addresses, and forwarded results.

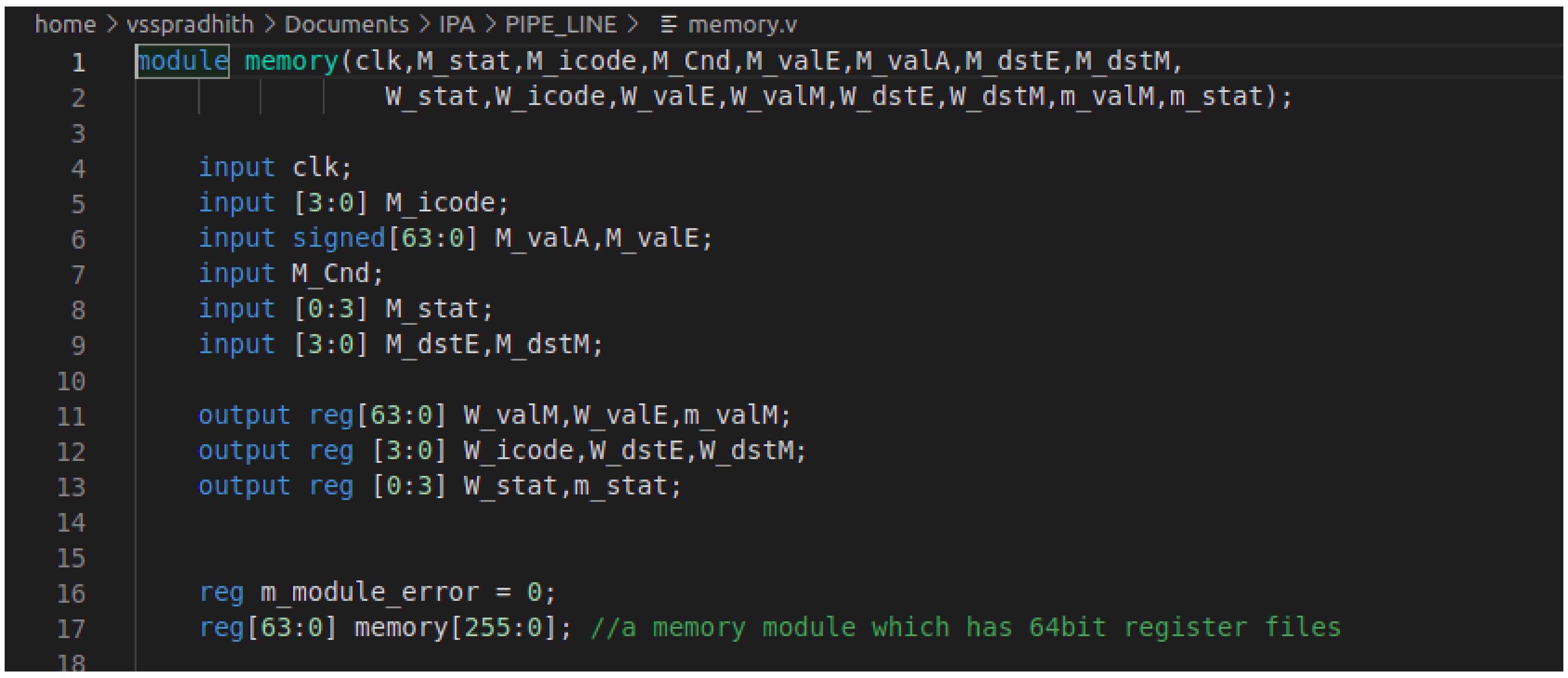
Comparing this to the memory stage for SEQ, we see that, as noted before, the block labeled “**Mem. data**” in SEQ is not present in PIPE. This block served to select between data sources **valP** (for call instructions) and **valA**, but this selection is now performed by the block labeled “**Sel+Fwd A**” in the decode stage. Most other blocks in this stage are identical to their counterparts in SEQ, with an appropriate renaming of the signals.

In this figure, you can also see that many of the values in pipeline registers and M and W are supplied to other parts of the circuit as part of the
forwarding and pipeline control logic.

We have referred to the above block diagrams while writing the memory modules in verilog and we defined our modules as follows:


```
home > vsspradhith > Documents > IPA > Sequential > memory.v
module memory(clk,icode,valA,valP,valE,valM,m_module_error); //not taking input as valB because we dont use it in memory module
    input clk;
    input [3:0]icode;
    input signed[63:0] valA,valP,valE;
    output reg[63:0]valM;
    output reg m_module_error;
    // output [63:0]m_rsp;
    reg[63:0] memory[255:0]; //a memory module which has 64bit register files
```

SEQ memory

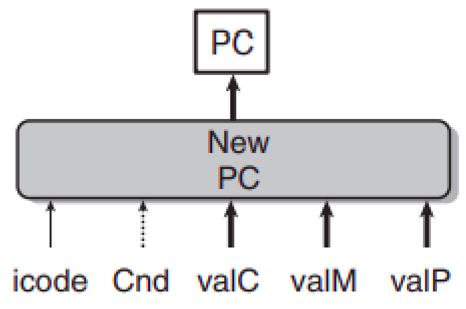

```
home > vsspradhith > Documents > IPA > PIPE_LINE > memory.v
module memory(clk,M_stat,M_icode,M_Cnd,M_valE,M_valA,M_dstE,M_dstM,
                W_stat,W_icode,W_valE,W_valM,W_dstE,W_dstM,m_valM,m_stat);

    input clk;
    input [3:0] M_icode;
    input signed[63:0] M_valA,M_valE;
    input M_Cnd;
    input [0:3] M_stat;
    input [3:0] M_dstE,M_dstM;

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


    reg m_module_error = 0;
    reg[63:0] memory[255:0]; //a memory module which has 64bit register files
```

PIPE memory

### e) PC update stage:

This stage is only present in SEQ implementation because in PIPE implementation, we are predicting PC in the fetch stage and thus reducing the delay.

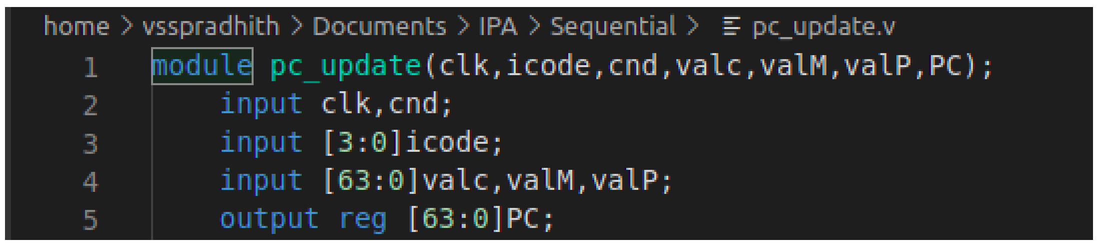
Consider the SEQ PC update stage:



The next value of the PC is selected from among the signals **valC**, **valM**, and **valP**, depending on the instruction code and the branch flag. The control logic for PC update is as follows:

```
word new_pc = [
        # Call.  Use instruction constant
        icode == ICALL : valC;
        # Taken branch.  Use instruction constant
        icode == IJXX && Cnd : valC;
        # Completion of RET instruction.  Use value from stack
        icode == IRET : valM;
        # Default: Use incremented PC
        1 : valP;
];
```

We have referred to the above block diagram while writing the PC update modules in verilog and we defined our module as follows:



home > vsspradhith > Documents > IPA > Sequential > pc_update.v

```
module pc_update(clk,icode,cnd,valc,valM,valP,PC);
    input clk,cnd;
    input [3:0]icode;
    input [63:0]valc,valM,valP;
    output reg [63:0]PC;
```

SEQ PC update

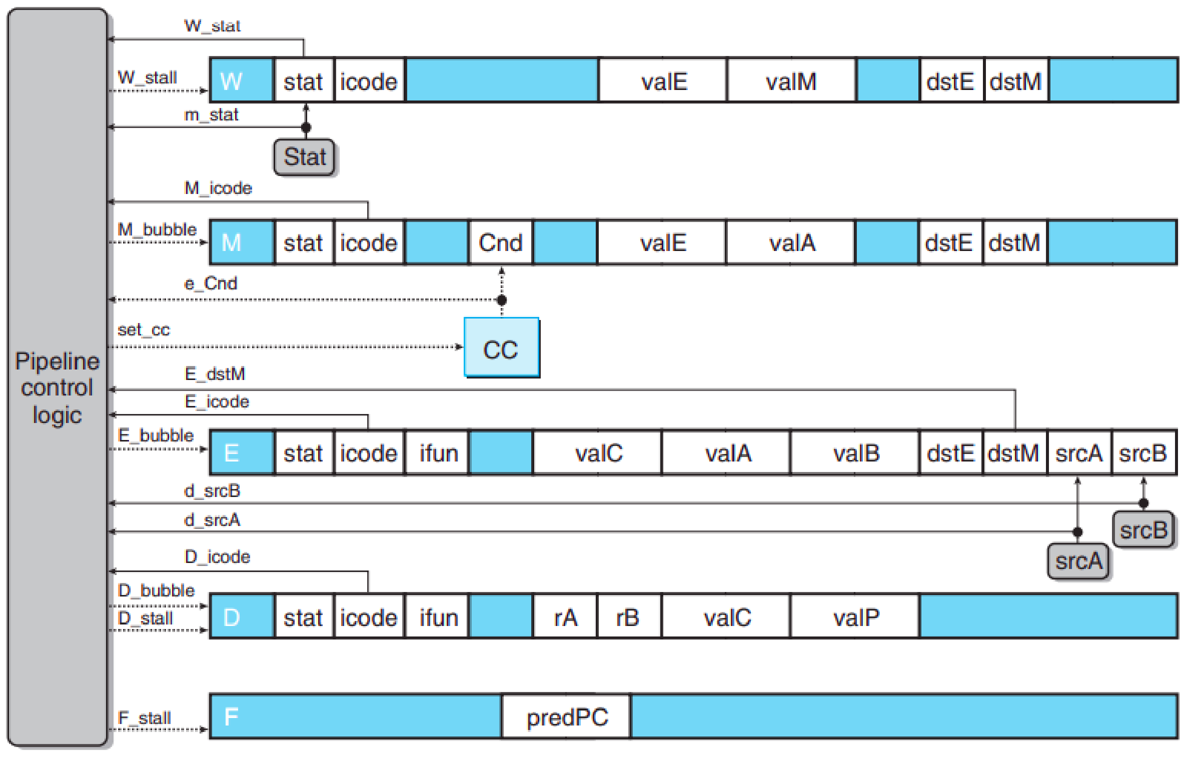
## Pipeline controlling in PIPE implementation:

As discussed earlier, we may face some problems in the pipeline implementation. That can be data hazards, PC misprediction, data dependencies, processing ret, etc.

These all can be handled by doing pipeline controlling. We should implement some control logic that handles this issue. This logic must handle the following four control cases for which other
mechanisms, such as data forwarding and branch prediction, do not suffice:

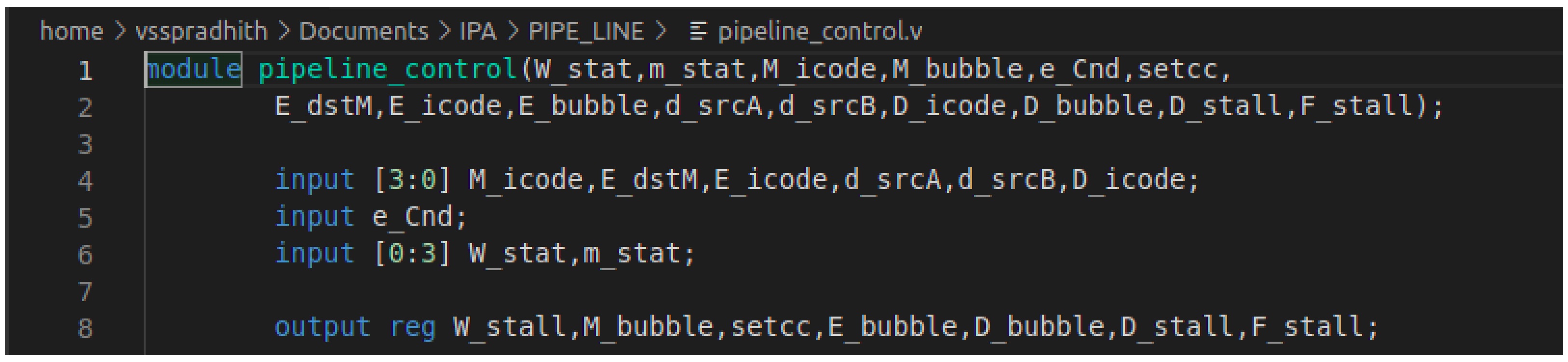
- **Load/use hazards:** The pipeline must stall for one cycle between an instruction that reads a value from memory and an instruction that uses this value.
- **Processing ret:** The pipeline must stall until the ret instruction reaches the write-back stage.
- **Mispredicted branches:** By the time the 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 cancelled, and fetching should begin at the instruction following the jump instruction.
- **Exceptions:** When an instruction causes an exception, we want to disable the updating of the programmer-visible state by later instructions and halt execution once the excepting instruction reaches the write-back stage.

Consider the implementation PIPE pipeline control logic:


Pipeline control logic
W_stat
W_stall
W stat icode valE valM dstE dstM
m_stat
Stat
M_icode
M_bubble
M stat icode Cnd valE valA dstE dstM
e_Cnd
set_cc
CC
E_dstM
E_icode
E_bubble
E stat icode ifun valC valA valB dstE dstM srcA srcB
d_srcB
d_srcA
srcB
srcA
D_icode
D_bubble
D_stall
D stat icode ifun rA rB valC valP
F_stall
F predPC

Based on signals from the pipeline registers and pipeline stages, the control logic generates stall and bubble control signals for the pipeline registers and also determines whether the condition code registers should be updated.

We have referred to the above block diagram while writing the pipeline control logic in verilog and we defined our module as follows:



home > vsspradhith > Documents > IPA > PIPE_LINE > pipeline_control.v

```
module pipeline_control(W_stat,m_stat,M_icode,M_bubble,e_Cnd,setcc,
        E_dstM,E_icode,E_bubble,d_srcA,d_srcB,D_icode,D_bubble,D_stall,F_stall);

        input [3:0] M_icode,E_dstM,E_icode,d_srcA,d_srcB,D_icode;
        input e_Cnd;
        input [0:3] W_stat,m_stat;

        output reg W_stall,M_bubble,setcc,E_bubble,D_bubble,D_stall,F_stall;
```

Pipeline control logic

The control logic for pipe control is as follows:

```
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 }
       # but not condition for a load/use hazard
       && !(E_icode in { IMRMOVQ, IPOPQ }
            && E_dstM in { d_srcA, d_srcB });
```

```
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 };
```

## Testbench for Sequential and Pipelined architecture:

Consider the following set of instructions:

```
instr_memory[1]  = 8'h10; //nop

instr_memory[2] = 8'h20; //rrmovq
instr_memory[3] = 8'h12;
```

```
instr_memory[4] = 8'h30;//irmovq
instr_memory[5] = 8'hF2;
instr_memory[6] = 8'h00;
instr_memory[7] = 8'h00;
instr_memory[8] = 8'h00;
instr_memory[9] = 8'h00;
instr_memory[10] = 8'h00;
instr_memory[11] = 8'h00;
instr_memory[12] = 8'h00;
instr_memory[13] = 8'b00000010;

instr_memory[14] = 8'h40;//rmmovq
instr_memory[15] = 8'h24;
{instr_memory[16],instr_memory[17],instr_memory[18],instr_memory[19],instr_memory[20],instr_memory[21],instr_memory[22],instr_memory[2
3]} = 64'd1;

instr_memory[24] = 8'h40;//rmmovq
instr_memory[25] = 8'h53;
{instr_memory[26],instr_memory[27],instr_memory[28],instr_memory[29],instr_memory[30],instr_memory[31],instr_memory[32],instr_memory[3
3]} = 64'd0;

instr_memory[34] = 8'h50;//mrmovq
instr_memory[35] = 8'h53;
{instr_memory[36],instr_memory[37],instr_memory[38],instr_memory[39],instr_memory[40],instr_memory[41],instr_memory[42],instr_memory[4
3]} = 64'd0;

instr_memory[44] = 8'h60;
instr_memory[45] = 8'h9A;

instr_memory[46] = 8'h73;
{instr_memory[47],instr_memory[48],instr_memory[49],instr_memory[50],instr_memory[51],instr_memory[52],instr_memory[53],instr_memory[5
4]} = 64'd56;

instr_memory[55] = 8'h00;

instr_memory[56] = 8'hA0;
instr_memory[57] = 8'h9F;

instr_memory[58] = 8'hB0;
instr_memory[59] = 8'h9F;

instr_memory[60] = 8'h80;
{instr_memory[61],instr_memory[62],instr_memory[63],instr_memory[64],instr_memory[65],instr_memory[66],instr_memory[67],instr_memory[6
8]} = 64'd80;

instr_memory[69] = 8'h60;
instr_memory[70] = 8'h56;

instr_memory[71] = 8'h70;
{instr_memory[72],instr_memory[73],instr_memory[74],instr_memory[75],instr_memory[76],instr_memory[77],instr_memory[78],instr_memory[7
9]} = 64'd46;

instr_memory[80] = 8'h30;//irmovq
instr_memory[81] = 8'hF2;
instr_memory[82] = 8'h00;
instr_memory[83] = 8'h00;
instr_memory[84] = 8'h00;
instr_memory[85] = 8'h00;
instr_memory[86] = 8'h00;
instr_memory[87] = 8'h00;
instr_memory[88] = 8'h00;
instr_memory[89] = 8'b00000010;

instr_memory[90] = 8'h60;
instr_memory[91] = 8'h9A;
instr_memory[92] = 8'h10;

instr_memory[93] = 8'h90;
```

For the first instruction we will be implementing a **nop,**

now we will be implementing some basic instructions such as **rrmovq,irmovq,rmmovq,mrmovq** until we reach instruction number **44**.

Now we will set condition codes using the **opq** operation. where we use 2 register values **rA = 9, rB = A.**

we have **valA = -12345** and **valB = 12345**.

we will get **valE = 0** in execute stage and the condition codes gets updated to **CC = 001 =⇒ OF,SF,ZF**.

now after **opq** operation we will be having our condition codes updated and we wrote a **je** instruction which has **icode** = **7** & **ifun = 3**. because of condition codes the jump condition gets satisfied. so the next instruction is updated to 56 instead of 55.

now we have **pushq** in **56**th instruction which pushes value into memory stack and the value of stack is at 254**(rsp).** Because of **pushq**, the **rsp** is reduced to **253** and the value from register 9 is pushed to memory. now we wiil be moving to instruction number **58**.

we have popq in **58**th instruction. **rsp** would be updated to **254** again and the value present at memory location **253** is pushed into the register **9**.

call instruction is implemented in **instr_60** and we wiil be transferred to a **PC** value of **80**. so call is like a function which shifts operation from current instruction **60** to **80** instead of **69**.

so we will be implementing functions at **80** such as **irmovq, opq,nop** and we will be currently lying at **PC** value **93**.

Now **93** is return function which takes last updated value in stack and this value was updated by call. when we call the instruction, the original progress namely **valP** is updated to memory stack and the **rsp** is located to it.

because we have encountered a return instruction we will be updated with a value of **69** as **valM** which is taken from memory.

now we have reached instruction number **69** which contains **opq** and this updates the condition as **CC = 010**.

after **opq** we have out **PC** at out jump instruction and since its an unconditional jump we would be directly jumping to instruction number **46**.

now because of our new condition codes our **je** condition at instruction number **46** is failed and it is moved to **55** which is a halt instruction.

order of execution is as follows;

**1→2→4→14→24→34→44→46→56→58→60→**

**80→some call instruction→93→69→71→46→55(halt)**

**Following results are for the testbench of Pipeline processor:**

```
VCD info: dumpfile processor.vcd opened for output.
F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =      1 clk=1 F_predPC=                   1 f_predPC=                   2
 d_srcA =  x d_srcB =  x
e_cnd = x e_valE =                    0 E_valA =                    x E_valB =                    x E_dstM =  x E_dstE =  x e_dstE =  x
 OF = 0 SF = 0 ZF = 0
D_icode=xxxx D_ifun=xxxx D_rA=xxxx D_rB=xxxx,valC=                   x
 W_valM =                    x
rsp =                    x and m_valM =                    x M_valA =                    x M_valE =                    x
d_valA =                    x d_valB =                    x
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =      2 clk=1 F_predPC=                   2 f_predPC=                   4
 d_srcA = 15 d_srcB = 15
e_cnd = x e_valE =                    0 E_valA =                    x E_valB =                    x E_dstM =  x E_dstE =  x e_dstE =  x
 OF = 0 SF = 0 ZF = 0
D_icode=0001 D_ifun=0000 D_rA=xxxx D_rB=xxxx,valC=                   x
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                    x M_valE =                    0
d_valA =                    x d_valB =                    x
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =      3 clk=1 F_predPC=                   4 f_predPC=                  14
 d_srcA =  1 d_srcB = 15
e_cnd = x e_valE =                    0 E_valA =                    x E_valB =                    x E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0010 D_ifun=0000 D_rA=0001 D_rB=0010,valC=                   x
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                    x M_valE =                    0
d_valA =                   10 d_valB =                    0
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =      4 clk=1 F_predPC=                  14 f_predPC=                  24
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                   10 E_valA =                   10 E_valB =                    0 E_dstM = 15 E_dstE =  2 e_dstE =  2
 OF = 0 SF = 0 ZF = 0
D_icode=0011 D_ifun=0000 D_rA=1111 D_rB=0010,valC=                   2
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                    x M_valE =                    0
d_valA =                   10 d_valB =                    0
───────────────────────────────────────────────────────────────────────────────────────────
```

```
hi1 F_stalled
hi3
F_stall = 1 D_stall = 1 D_bubble = 0 setcc = 1 cycle =     5 clk=1 F_predPC=                  24 f_predPC=                  34
 d_srcA =  2 d_srcB =  4
e_cnd = 1 e_valE =                    2 E_valA =                   10 E_valB =                    0 E_dstM = 15 E_dstE =  2 e_dstE =  2
 OF = 0 SF = 0 ZF = 0
D_icode=0100 D_ifun=0000 D_rA=0010 D_rB=0100,valC=                   1
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                   10 M_valE =                   10
d_valA =                    2 d_valB =                  254
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     6 clk=1 F_predPC=                  24 f_predPC=                  34
 d_srcA =  2 d_srcB =  4
e_cnd = 1 e_valE =                  255 E_valA =                    2 E_valB =                  254 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0100 D_ifun=0000 D_rA=0010 D_rB=0100,valC=                   1
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                   10 M_valE =                    2
d_valA =                    2 d_valB =                  254
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     7 clk=1 F_predPC=                  34 f_predPC=                  44
 d_srcA =  5 d_srcB =  3
e_cnd = 1 e_valE =                  255 E_valA =                    2 E_valB =                  254 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0100 D_ifun=0000 D_rA=0101 D_rB=0011,valC=                   0
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                    2 M_valE =                  255
d_valA =                   50 d_valB =                    3
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     8 clk=1 F_predPC=                  44 f_predPC=                  46
 d_srcA = 15 d_srcB =  3
e_cnd = 1 e_valE =                    3 E_valA =                   50 E_valB =                    3 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0101 D_ifun=0000 D_rA=0101 D_rB=0011,valC=                   0
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                    2 M_valE =                  255
d_valA =                   50 d_valB =                    3
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     9 clk=1 F_predPC=                  46 f_predPC=                  56
 d_srcA =  9 d_srcB = 10
e_cnd = 1 e_valE =                    3 E_valA =                   50 E_valB =                    3 E_dstM =  5 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0110 D_ifun=0000 D_rA=1001 D_rB=1010,valC=                   0
 W_valM =                    x
rsp =                  254 and m_valM =                    x M_valA =                   50 M_valE =                    3
d_valA =               -12345 d_valB =                12345
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    10 clk=1 F_predPC=                  56 f_predPC=                  58
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                    0 E_valA =               -12345 E_valB =                12345 E_dstM = 15 E_dstE = 10 e_dstE = 10
 OF = 0 SF = 0 ZF = 1
D_icode=0111 D_ifun=0011 D_rA=1001 D_rB=1010,valC=                  56
 W_valM =                    x
rsp =                  254 and m_valM =                   50 M_valA =                   50 M_valE =                    3
d_valA =                   55 d_valB =                12345
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    11 clk=1 F_predPC=                  58 f_predPC=                  60
 d_srcA =  9 d_srcB =  4
e_cnd = 1 e_valE =                    0 E_valA =                   55 E_valB =                12345 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 1
D_icode=1010 D_ifun=0000 D_rA=1001 D_rB=1111,valC=                  56
 W_valM =                   50
rsp =                  254 and m_valM =                   50 M_valA =               -12345 M_valE =                    0
d_valA =               -12345 d_valB =                  254
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    12 clk=1 F_predPC=                  60 f_predPC=                  80
 d_srcA =  4 d_srcB =  4
e_cnd = 1 e_valE =                  253 E_valA =               -12345 E_valB =                  254 E_dstM = 15 E_dstE =  4 e_dstE =  4
 OF = 0 SF = 0 ZF = 1
D_icode=1011 D_ifun=0000 D_rA=1001 D_rB=1111,valC=                  56
 W_valM =                   50
rsp =                  254 and m_valM =                   50 M_valA =                   55 M_valE =                    0
d_valA =                  253 d_valB =                  253
───────────────────────────────────────────────────────────────────────────────────────────

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    13 clk=1 F_predPC=                  80 f_predPC=                  90
 d_srcA = 15 d_srcB =  4
e_cnd = 1 e_valE =                  254 E_valA =                  253 E_valB =                  253 E_dstM =  9 E_dstE =  4 e_dstE =  4
 OF = 0 SF = 0 ZF = 1
D_icode=1000 D_ifun=0000 D_rA=1001 D_rB=1111,valC=                  80
```

```
 W_valM =                   50
rsp =                  254 and m_valM =                   50 M_valA =               -12345 M_valE =                  253
d_valA =                   69 d_valB =                  254
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     14 clk=1 F_predPC=                  90 f_predPC=                  92
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                  253 E_valA =                   69 E_valB =                  254 E_dstM = 15 E_dstE =  4 e_dstE =  4
 OF = 0 SF = 0 ZF = 1
D_icode=0011 D_ifun=0000 D_rA=1111 D_rB=0010,valC=                   2
 W_valM =                   50
rsp =                  254 and m_valM =               -12345 M_valA =                  253 M_valE =                  254
d_valA =                  254 d_valB =                    0
___________________________________________________________________________________________

hi1 F_stalled
hi3
F_stall = 1 D_stall = 1 D_bubble = 0 setcc = 1 cycle =     15 clk=1 F_predPC=                  92 f_predPC=                  93
 d_srcA =  9 d_srcB = 10
e_cnd = 1 e_valE =                    2 E_valA =                  254 E_valB =                    0 E_dstM = 15 E_dstE =  2 e_dstE =  2
 OF = 0 SF = 0 ZF = 1
D_icode=0110 D_ifun=0000 D_rA=1001 D_rB=1010,valC=                   2
 W_valM =               -12345
rsp =                  253 and m_valM =               -12345 M_valA =                   69 M_valE =                  253
d_valA =               -12345 d_valB =                    0
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     16 clk=1 F_predPC=                  92 f_predPC=                  93
 d_srcA =  9 d_srcB = 10
e_cnd = 1 e_valE =               -12345 E_valA =               -12345 E_valB =                    0 E_dstM = 15 E_dstE = 10 e_dstE = 10
 OF = 0 SF = 1 ZF = 0
D_icode=0110 D_ifun=0000 D_rA=1001 D_rB=1010,valC=                   2
 W_valM =               -12345
rsp =                  254 and m_valM =               -12345 M_valA =                  254 M_valE =                    2
d_valA =               -12345 d_valB =               -12345
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     17 clk=1 F_predPC=                  93 f_predPC=                  93
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =               -24690 E_valA =               -12345 E_valB =               -12345 E_dstM = 15 E_dstE = 10 e_dstE = 10
 OF = 0 SF = 1 ZF = 0
D_icode=0001 D_ifun=0000 D_rA=1001 D_rB=1010,valC=                   2
 W_valM =               -12345
rsp =                  253 and m_valM =               -12345 M_valA =               -12345 M_valE =               -12345
d_valA =               -12345 d_valB =                    0
___________________________________________________________________________________________

hi2
hi2
F_stall = 1 D_stall = 0 D_bubble = 1 setcc = 1 cycle =     18 clk=1 F_predPC=                  93 f_predPC=                  93
 d_srcA =  4 d_srcB =  4
e_cnd = 1 e_valE =                    0 E_valA =               -12345 E_valB =                    0 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=1001 D_ifun=0000 D_rA=1001 D_rB=1010,valC=                   2
 W_valM =               -12345
rsp =                  253 and m_valM =               -12345 M_valA =               -12345 M_valE =               -24690
d_valA =                  253 d_valB =                  253
___________________________________________________________________________________________

hi2
F_stall = 1 D_stall = 0 D_bubble = 1 setcc = 1 cycle =     19 clk=1 F_predPC=                  93 f_predPC=                  93
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                  254 E_valA =                  253 E_valB =                  253 E_dstM = 15 E_dstE =  4 e_dstE =  4
 OF = 0 SF = 0 ZF = 0
D_icode=0001 D_ifun=0000 D_rA=0000 D_rB=0000,valC=                   0
 W_valM =               -12345
rsp =                  253 and m_valM =               -12345 M_valA =               -12345 M_valE =                    0
d_valA =                  253 d_valB =                  253
___________________________________________________________________________________________

hi2
F_stall = 1 D_stall = 0 D_bubble = 1 setcc = 1 cycle =     20 clk=1 F_predPC=                  93 f_predPC=                  93
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                    0 E_valA =                  253 E_valB =                  253 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0001 D_ifun=0000 D_rA=0000 D_rB=0000,valC=                   0
 W_valM =               -12345
rsp =                  253 and m_valM =                   69 M_valA =                  253 M_valE =                  254
d_valA =                  253 d_valB =                  253
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =     21 clk=1 F_predPC=                  93 f_predPC=                  71
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                    0 E_valA =                  253 E_valB =                  253 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0001 D_ifun=0000 D_rA=0000 D_rB=0000,valC=                   0
 W_valM =                   69
```

```
rsp =                  253 and m_valM =                   69 M_valA =                  253 M_valE =                    0
d_valA =                  253 d_valB =                  253
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    22 clk=1 F_predPC=                  71 f_predPC=                  46
 d_srcA =  5 d_srcB =  6
e_cnd = 1 e_valE =                    0 E_valA =                  253 E_valB =                  253 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 0 ZF = 0
D_icode=0110 D_ifun=0000 D_rA=0101 D_rB=0110,valC=                   2
 W_valM =                   69
rsp =                  254 and m_valM =                   69 M_valA =                  253 M_valE =                    0
d_valA =                   50 d_valB =                 -143
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    23 clk=1 F_predPC=                  46 f_predPC=                  56
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                  -93 E_valA =                   50 E_valB =                 -143 E_dstM = 15 E_dstE =  6 e_dstE =  6
 OF = 0 SF = 1 ZF = 0
D_icode=0111 D_ifun=0000 D_rA=0101 D_rB=0110,valC=                  46
 W_valM =                   69
rsp =                  254 and m_valM =                   69 M_valA =                  253 M_valE =                    0
d_valA =                   80 d_valB =                 -143
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    24 clk=1 F_predPC=                  56 f_predPC=                  58
 d_srcA = 15 d_srcB = 15
e_cnd = 1 e_valE =                    0 E_valA =                   80 E_valB =                 -143 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 1 ZF = 0
D_icode=0111 D_ifun=0011 D_rA=0101 D_rB=0110,valC=                  56
 W_valM =                   69
rsp =                  254 and m_valM =                   69 M_valA =                   50 M_valE =                  -93
d_valA =                   55 d_valB =                 -143
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    25 clk=1 F_predPC=                  58 f_predPC=                  60
 d_srcA =  9 d_srcB =  4
e_cnd = 0 e_valE =                    0 E_valA =                   55 E_valB =                 -143 E_dstM = 15 E_dstE = 15 e_dstE = 15
 OF = 0 SF = 1 ZF = 0
D_icode=1010 D_ifun=0000 D_rA=1001 D_rB=1111,valC=                  56
 W_valM =                   69
rsp =                  254 and m_valM =                   69 M_valA =                   80 M_valE =                    0
d_valA =               -12345 d_valB =                  254
___________________________________________________________________________________________

F_stall = 0 D_stall = 0 D_bubble = 0 setcc = 1 cycle =    26 clk=1 F_predPC=                  60 f_predPC=                  55
 d_srcA =  4 d_srcB =  4
e_cnd = 0 e_valE =                  253 E_valA =               -12345 E_valB =                  254 E_dstM = 15 E_dstE =  4 e_dstE =  4
 OF = 0 SF = 1 ZF = 0
D_icode=1011 D_ifun=0000 D_rA=1001 D_rB=1111,valC=                  56
 W_valM =                   69
rsp =                  254 and m_valM =                   69 M_valA =                   55 M_valE =                    0
d_valA =                  253 d_valB =                  253
___________________________________________________________________________________________

halt encountered
```

**Similarly, following results are for the testbench of sequential processor:**

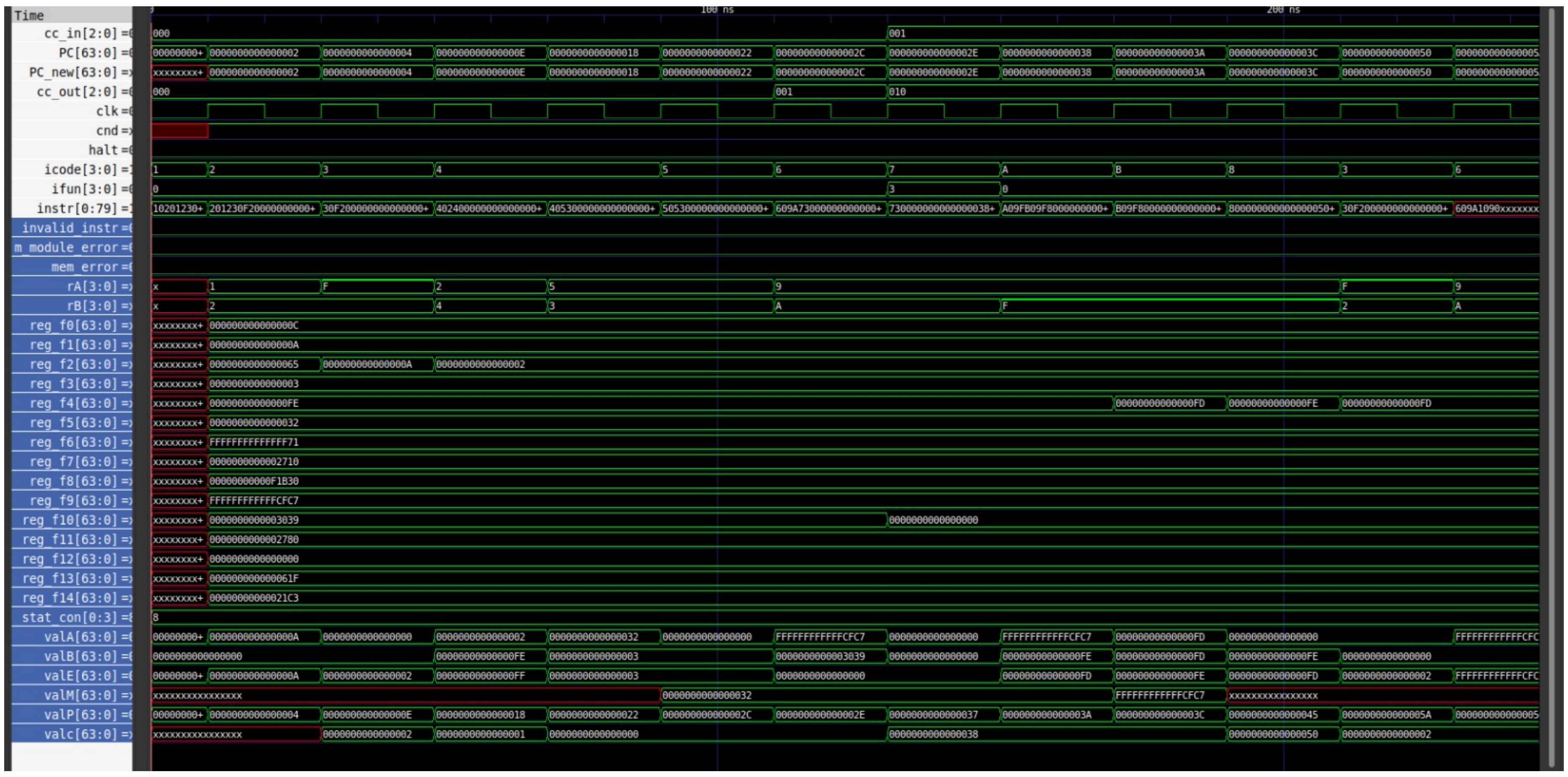
```
VCD info: dumpfile decode_tb.vcd opened for output.
1)rsp =                    x,PC =                    1 clk=1

2)icode=1 ifun=0 rA= x rB= x,valc=                   x,valP=                   2,

3)valA =                    0 and valB =                    0
valE =                    0 valM =                    x
ccodes OF,SF,ZF cc_in = 000 || cc_out=000 and cnd = x

1)rsp =                  254,PC =                    2 clk=1

2)icode=2 ifun=0 rA= 1 rB= 2,valc=                   x,valP=                   4,

3)valA =                   10 and valB =                    0
valE =                   10 valM =                    x
ccodes OF,SF,ZF cc_in = 000 || cc_out=000 and cnd = 1

1)rsp =                  254,PC =                    4 clk=1
```

```
2)icode=3 ifun=0 rA=15 rB= 2,valc=                   2,valP=                  14,

3)valA =                    0 and valB =                    0
valE =                    2 valM =                    x
ccodes OF,SF,ZF cc_in = 000 || cc_out=000 and cnd = 1

1)rsp =                  254,PC =                   14 clk=1

2)icode=4 ifun=0 rA= 2 rB= 4,valc=                   1,valP=                  24,

3)valA =                    2 and valB =                  254
valE =                  255 valM =                    x
ccodes OF,SF,ZF cc_in = 000 || cc_out=000 and cnd = 1

1)rsp =                  254,PC =                   24 clk=1

2)icode=4 ifun=0 rA= 5 rB= 3,valc=                   0,valP=                  34,

3)valA =                   50 and valB =                    3
valE =                    3 valM =                    x
ccodes OF,SF,ZF cc_in = 000 || cc_out=000 and cnd = 1

1)rsp =                  254,PC =                   34 clk=1

2)icode=5 ifun=0 rA= 5 rB= 3,valc=                   0,valP=                  44,

3)valA =                    0 and valB =                    3
valE =                    3 valM =                   50
ccodes OF,SF,ZF cc_in = 000 || cc_out=000 and cnd = 1

1)rsp =                  254,PC =                   44 clk=1

2)icode=6 ifun=0 rA= 9 rB=10,valc=                   0,valP=                  46,

3)valA =               -12345 and valB =                12345
valE =                    0 valM =                   50
ccodes OF,SF,ZF cc_in = 000 || cc_out=001 and cnd = 1

1)rsp =                  254,PC =                   46 clk=1

2)icode=7 ifun=3 rA= 9 rB=10,valc=                  56,valP=                  55,

3)valA =                    0 and valB =                    0
valE =                    0 valM =                   50
ccodes OF,SF,ZF cc_in = 001 || cc_out=010 and cnd = 1

1)rsp =                  254,PC =                   56 clk=1

2)icode=a ifun=0 rA= 9 rB=15,valc=                  56,valP=                  58,

3)valA =               -12345 and valB =                  254
valE =                  253 valM =                   50
ccodes OF,SF,ZF cc_in = 001 || cc_out=010 and cnd = 1

1)rsp =                  253,PC =                   58 clk=1

2)icode=b ifun=0 rA= 9 rB=15,valc=                  56,valP=                  60,

3)valA =                  253 and valB =                  253
valE =                  254 valM =               -12345
ccodes OF,SF,ZF cc_in = 001 || cc_out=010 and cnd = 1

1)rsp =                  254,PC =                   60 clk=1

2)icode=8 ifun=0 rA= 9 rB=15,valc=                  80,valP=                  69,

3)valA =                    0 and valB =                  254
valE =                  253 valM =                    x
ccodes OF,SF,ZF cc_in = 001 || cc_out=010 and cnd = 1

1)rsp =                  253,PC =                   80 clk=1

2)icode=3 ifun=0 rA=15 rB= 2,valc=                   2,valP=                  90,

3)valA =                    0 and valB =                    0
valE =                    2 valM =                    x
ccodes OF,SF,ZF cc_in = 001 || cc_out=010 and cnd = 1

1)rsp =                  253,PC =                   90 clk=1

2)icode=6 ifun=0 rA= 9 rB=10,valc=                   2,valP=                  92,

3)valA =               -12345 and valB =                    0
valE =               -12345 valM =                    x
ccodes OF,SF,ZF cc_in = 001 || cc_out=010 and cnd = 1
```

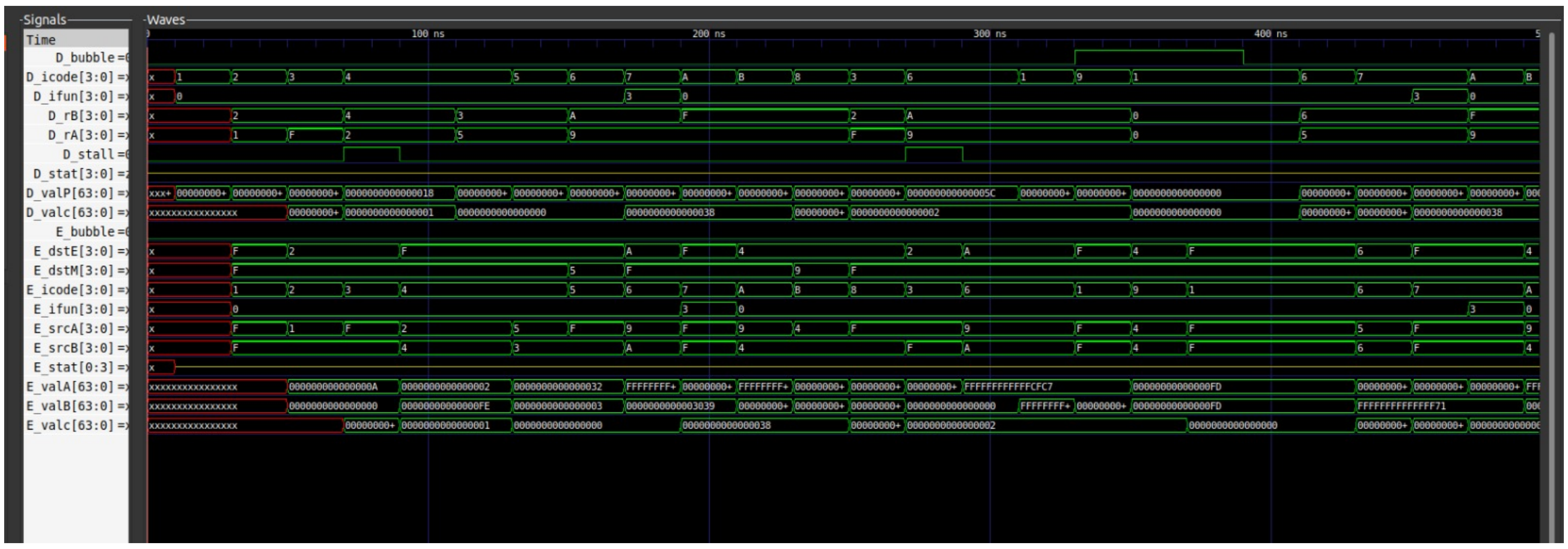
```
1)rsp =                  253,PC =                   92 clk=1

2)icode=1 ifun=0 rA= 9 rB=10,valc=                   2,valP=                   93,

3)valA =                    0 and valB =                    0
valE =                    0 valM =                    x
ccodes OF,SF,ZF cc_in = 010 || cc_out=000 and cnd = 1

1)rsp =                  253,PC =                   93 clk=1

2)icode=9 ifun=0 rA= 9 rB=10,valc=                   2,valP=                   94,

3)valA =                  253 and valB =                  253
valE =                  254 valM =                   69
ccodes OF,SF,ZF cc_in = 010 || cc_out=000 and cnd = 1

1)rsp =                  254,PC =                   69 clk=1

2)icode=6 ifun=0 rA= 5 rB= 6,valc=                   2,valP=                   71,

3)valA =                   50 and valB =                 -143
valE =                  -93 valM =                    x
ccodes OF,SF,ZF cc_in = 010 || cc_out=010 and cnd = 1

1)rsp =                  254,PC =                   71 clk=1

2)icode=7 ifun=0 rA= 5 rB= 6,valc=                  46,valP=                   80,

3)valA =                    0 and valB =                    0
valE =                    0 valM =                    x
ccodes OF,SF,ZF cc_in = 010 || cc_out=010 and cnd = 1

1)rsp =                  254,PC =                   46 clk=1

2)icode=7 ifun=3 rA= 5 rB= 6,valc=                  56,valP=                   55,

3)valA =                    0 and valB =                    0
valE =                    0 valM =                    x
ccodes OF,SF,ZF cc_in = 010 || cc_out=010 and cnd = 0

halt instruction
```

## GTKwave plots:

### a) Sequential processor:



**b) Pipeline processor:**


Signals
Waves
Time
100 ns
200 ns
300 ns
400 ns
D_bubble
D_icode[3:0]
D_ifun[3:0]
D_rB[3:0]
D_rA[3:0]
D_stall
D_stat[3:0]
D_valP[63:0]
D_valC[63:0]
E_bubble
E_dstE[3:0]
E_dstM[3:0]
E_icode[3:0]
E_ifun[3:0]
E_srcA[3:0]
E_srcB[3:0]
E_stat[0:3]
E_valA[63:0]
E_valB[63:0]
E_valC[63:0]