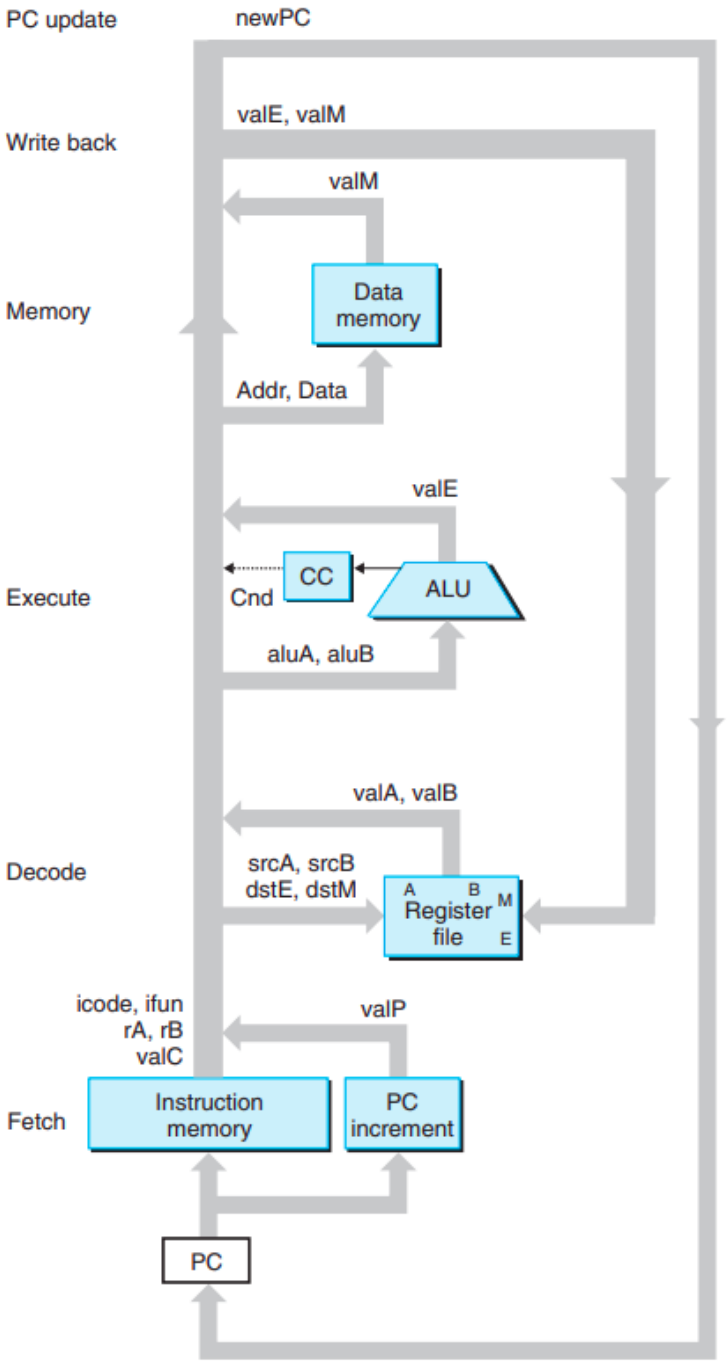
Project Report

22/02/22

Srujana Vanka - 2020102005

Shreeya Singh - 2020102011

To implement a Y86-64 processor, we describe a processor called SEQ (for "sequential" processor). On each clock cycle, SEQ performs all the steps required to process a complete instruction. This would require a very long cycle time, however, and so the clock rate would be unacceptably low. Our purpose in developing SEQ is to provide a first step toward our ultimate goal of implementing an efficient pipelined processor.

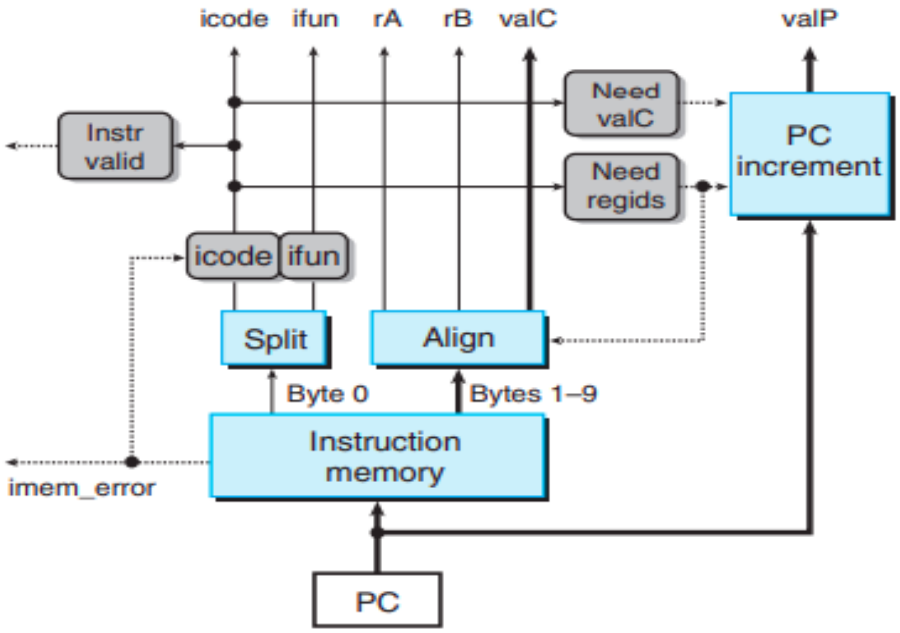


The processor architecture can be broadly divided into 5 main modules described below:

- Fetch
- Decode and Writeback
- Execute
- Memory
- PC update

Fetch module

SEQ Fetch stage:



- The fetch stage reads the bytes of an instruction from memory, using the program counter (PC) as the memory address.
- Now, when this step ends, the PC points to the next instruction to be read in the next cycle and the cycle continues like this to fetch all the instructions.
- From the instruction it extracts the two 4-bit portions of the instruction specifier byte, referred to as icode (the instruction code) and ifun (the instruction function).

- It possibly fetches a register specifier byte, giving one or both of the register operand specifiers rA and rB. It also possibly fetches an 8-byte constant word valC. It computes valP to be the address of the instruction following the current one in sequential order. That is, valP equals the value of the PC plus the length of the fetched instruction.The instruction memory reads the bytes of an instruction using the program counter register as an address.

```
`timescale 1ns / 1ps
module fetch(
 clk,pc,
 icode,ifun,rA,rB,valC,valP,instr_valid,imem_error,hlt
);

 input clk;
 input [63:0] pc;
 output reg [3:0] icode, ifun, rA, rB;
 output reg [63:0] valC, valP;
 output reg instr_valid, hlt, imem_error;
 reg [7:0] instruction_memory[0:2047];
 reg [0:79] instr;

 always@(posedge clk)
 begin
   imem_error=0;
   if(pc>2047)imem_error=1;
   instr=
   {
     instruction_memory[pc],
     instruction_memory[pc+1],
```

```
    instruction_memory[pc+2],
    instruction_memory[pc+3],
    instruction_memory[pc+4],
    instruction_memory[pc+5],
    instruction_memory[pc+6],
    instruction_memory[pc+7],
    instruction_memory[pc+8],
    instruction_memory[pc+9]
  };

  icode= instr[0:3];
  ifun= instr[4:7];
  instr_valid=1'b1;

  if(icode==4'b0000) //halt
  begin
    hlt=1;
    valP=pc+64'd1;
  end
  else if(icode==4'b0001) valP=pc+64'd1; //nop
  else if(icode==4'b0010) //cmovxx
  begin
    rA=instr[8:11];
    rB=instr[12:15];
    valP=pc+64'd2;
  end
  else if(icode==4'b0011) //irmovq
```

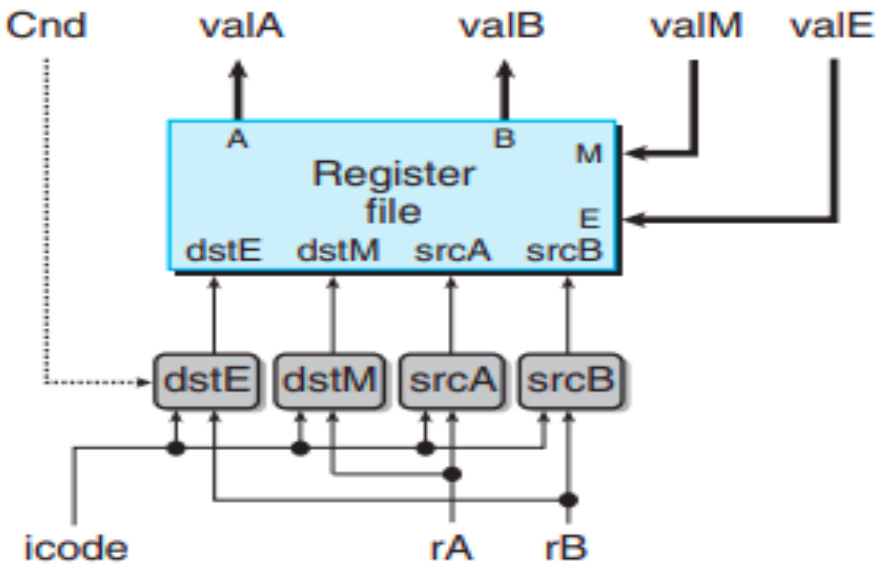
```
begin
  rA=instr[8:11];
  rB=instr[12:15];
  valC=instr[16:79];
  valP=pc+64'd10;
end
else if(icode==4'b0100) //rmmovq
begin
  rA=instr[8:11];
  rB=instr[12:15];
  valC=instr[16:79];
  valP=pc+64'd10;
end
else if(icode==4'b0101) //mrmovq
begin
  rA=instr[8:11];
  rB=instr[12:15];
  valC=instr[16:79];
  valP=pc+64'd10;
end
else if(icode==4'b0110) //OPq
begin
  rA=instr[8:11];
  rB=instr[12:15];
  valP=pc+64'd2;
end
else if(icode==4'b0111) //jxx
```

```
  begin
    valC=instr[8:71];
    valP=pc+64'd9;
  end
  else if(icode==4'b1000) //call
  begin
    valC=instr[8:71];
    valP=pc+64'd9;
  end
  else if(icode==4'b1001) valP=pc+64'd1;//ret
  else if(icode==4'b1010) //pushq
  begin
    rA=instr[8:11];
    rB=instr[12:15];
    valP=pc+64'd2;
  end
  else if(icode==4'b1011) //popq
  begin
    rA=instr[8:11];
    rB=instr[12:15];
    valP=pc+64'd2;
  end
  else  instr_valid=1'b0;
 end
endmodule
```

Decode and Writeback module

- The decode module is used to interpret the encoded instruction that is provided in the instruction register at a given clock cycle.

SEQ Decode and Writeback stage:



- These two stages (decode and write-back) are combined because they both access the register file.
- The instruction fields are decoded to generate register identifiers for four addresses (two read and two write) used by the register file.

```
module regfile(
 clock,icode,rA,rB,cnd,valA,valB,valE,valM,mem0,mem1,mem2,mem3,mem4,mem5,
 mem6,mem7,mem8,mem9,mem10,mem11,mem12,mem13,mem14);
// Reset is used to set all registers to 0
 input reset, clock, cnd;
 input [3:0] icode,rA,rB;
 output reg [63:0] valA, valB;
```

```
 input [63:0] valE, valM;
 output reg [63:0]
mem0,mem1,mem2,mem3,mem4,mem5,mem6,mem7,mem8,mem9,mem10,mem11,mem12,mem13,
mem14;
 reg [63:0] mem[0:14];


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


 //This is the decode module
always@(*) begin
     case(icode)
     4'b0010:valA=mem[rA];
     4'b0100:begin
       valA=mem[rA];
       valB=mem[rB];
     end
     4'b0101:valB=mem[rB];
     4'b0110:begin
       valA=mem[rA];
       valB=mem[rB];
     end
     4'b1000: valB=mem[4];
     4'b1001:begin
```

```
      valA=mem[4];
      valB=mem[4];
    end
    4'b1010:begin
      valA=mem[rA];
      valB=mem[4];
    end
    4'b1011:begin
      valA=mem[4];
      valB=mem[4];
    end
    endcase

mem0=mem[0];mem1=mem[1];mem2=mem[2];mem3=mem[3];mem4=mem[4];mem5=mem[5];me
m6=mem[6];mem7=mem[7];

mem8=mem[8];mem9=mem[9];mem10=mem[10];mem11=mem[11];mem12=mem[12];mem13=me
m[13];mem14=mem[14];
 end
  //This is the write-back module
 always@(negedge clock) begin
   if(icode==4'b0010 || cnd==1'b1) mem[rB]=valE;//cmovxx
   else if(icode==4'b0011)mem[rB]=valE; //irmovq
   else if(icode==4'b0101)mem[rA]=valM; //mrmovq
   else if(icode==4'b0110)mem[rB]=valE; //OPq
   else if(icode==4'b1000)mem[4]=valE; //call
   else if(icode==4'b1001) mem[4]=valE;//ret
   else if(icode==4'b1010)mem[4]=valE; //pushq
```

```
   else if(icode==4'b1011) //popq
   begin
      mem[4]=valE;
      mem[rA]=valM;
   end

mem0=mem[0];mem1=mem[1];mem2=mem[2];mem3=mem[3];mem4=mem[4];mem5=mem[5];me
m6=mem[6];mem7=mem[7];

mem8=mem[8];mem9=mem[9];mem10=mem[10];mem11=mem[11];mem12=mem[12];mem13=me
m[13];mem14=mem[14];
 end
endmodule
```

Results

```
❯ ./a.out
clock=0 icode=xxxx  rA= x rB= x,valA=                x,valB=                x,valE=                x,valM=
      x
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                 10,mem11=               11,mem12=               12,mem13=               13,mem14=
     14

clock=1 icode=xxxx  rA= x rB= x,valA=                x,valB=                x,valE=                x,valM=
      x
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                 10,mem11=               11,mem12=               12,mem13=               13,mem14=
     14

clock=0 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                 10,mem11=                0,mem12=               12,mem13=               13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                 10,mem11=                0,mem12=               12,mem13=               13,mem14=
     14
```

```
clock=0 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14

clock=0 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14
```

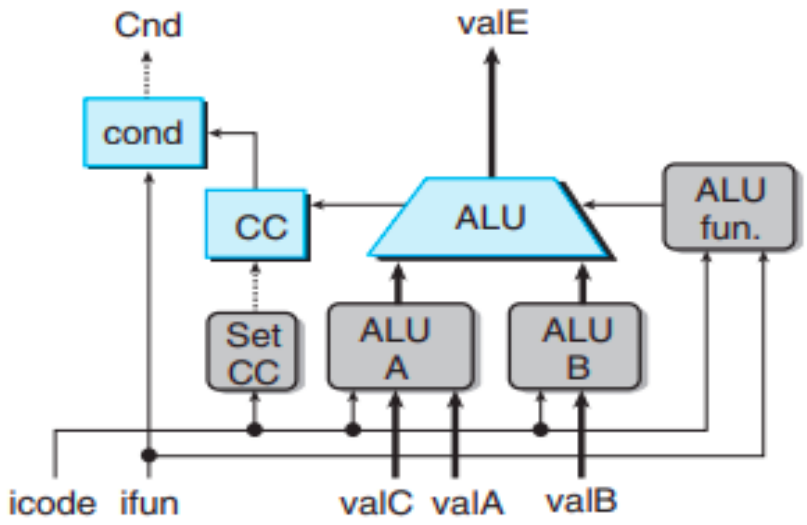
```
clock=0 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14

clock=1 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14

clock=0 icode=0110  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14
```

```
clock=1 icode=0101  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                2,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14

clock=0 icode=0101  rA= 2 rB=11,valA=                2,valB=                0,valE=                0,valM=
      0
,mem0=                  0,mem1=                1,mem2=                0,mem3=                3
,mem4=                  4,mem5=                5,mem6=                6,mem7=                7,mem8=
 8,mem9=                  9
,mem10=                10,mem11=                0,mem12=                12,mem13=                13,mem14=
     14


 ~/acad/ipa
❯
```

Execute module

- The execute module carries out the functional operations which are implemented by ALU for different purposes according to the instruction type.
- The decoded data is sent as control signals to the functional units and then the required instructions are carried out before being transferred to the ALU.
- The ALU output becomes the signal valE.

SEQ Execute stage:




```
`timescale 1ns / 1ps

`include "./ALU/ALU.v"


module execute(

 clk,icode,ifun,valA,valB,valC,valE,Cnd,zero_flag,sign_flag,overflow_flag

);


 input clk;

 input [3:0] icode, ifun;
```

```
input [63:0] valA, valB, valC;


output reg [63:0] valE;
output reg Cnd, zero_flag, sign_flag, overflow_flag;


always@(*)
begin
  if(icode==4'b0110 && clk==1)
  begin
    zero_flag=(y==1'b0);
    sign_flag=(y<1'b0);
    overflow_flag=(a<1'b0==b<1'b0)&&(y<1'b0!=a<1'b0);
  end
end


initial begin zero_flag=0; sign_flag=0; overflow_flag=0; end


reg [63:0]temp, a, b;
reg [1:0] control;


wire [63:0]y;
wire overflow;


ALU alu1(
 control, a, b, y
);
```

```
reg r1, r2, p1, p2,q1, q2, s;
wire rout, pout, qout, sout;
xor g1(rout,r1,r2);
or g2(qout,p1,p2);
and g3(pout,q1,q2);
not g4(sout,s);

initial begin control=2'b00; a = 64'b0; b = 64'b0; end
 always@(*)
begin
    Cnd=0;
    if(icode==4'b0010) //cmovxx
    begin
      if(ifun==4'b0000)Cnd=1;//rrmovq
      else if(ifun==4'b0001)//cmovle
      begin
        r1=sign_flag; // (sign_flag^overflow_flag)||zero_flag
        r2=overflow_flag;
        if(rout)Cnd=1;
        else if(zero_flag)Cnd=1;
      end
      else if(ifun==4'b0010)//cmovl
      begin
        r1=sign_flag; // sign_flag^overflow_flag
        r2=overflow_flag;
        if(rout)Cnd=1;
      end
```

```
    else if(ifun==4'b0011)//cmove
    begin
      if(zero_flag)Cnd=1; // zero_flag
    end
    else if(ifun==4'b0100)//cmovne
    begin
      s=zero_flag; // !zero_flag
      if(sout)Cnd=1;
    end
    else if(ifun==4'b0101)//cmovge
    begin
      r1=sign_flag;
      r2=overflow_flag;
      s=rout;
      if(sout)Cnd=1;
    end
    else if(ifun==4'b0110)//cmovg
    begin
      r1=sign_flag;
      r2=overflow_flag;
      s=rout;
      if(sout)
      begin
        s=zero_flag;
        if(sout) Cnd=1;
      end
    end
```

```
    valE=64'd0+valA;
  end
  else if(icode==4'b0011) valE=64'd0+valC; //irmovq
  else if(icode==4'b0100) valE=valB+valC;//rmmovq
  else if(icode==4'b0101)valE=valB+valC; //mrmovq
  else if(icode==4'b0110) //OPq
  begin
    if(ifun==4'b0000) //add
    begin control=2'b00; a = valA;b = valB;end
    else if(ifun==4'b0001) //sub
    begin  control=2'b01;   a = valB;  b = valA; end
    else if(ifun==4'b0010) //and
    begin control=2'b10; a = valA;  b = valB;
    end
    else if(ifun==4'b0011) //xor
    begin control=2'b11; a = valA; b = valB; end
    assign temp=y;
    valE=temp;
  end
  if(icode==4'b0111) //jxx
  begin
    if(ifun==4'b0000)Cnd=1;//jmp
    else if(ifun==4'b0001)//jle
    begin
      r1=sign_flag;
      r2=overflow_flag;
      if(rout) Cnd=1;
```

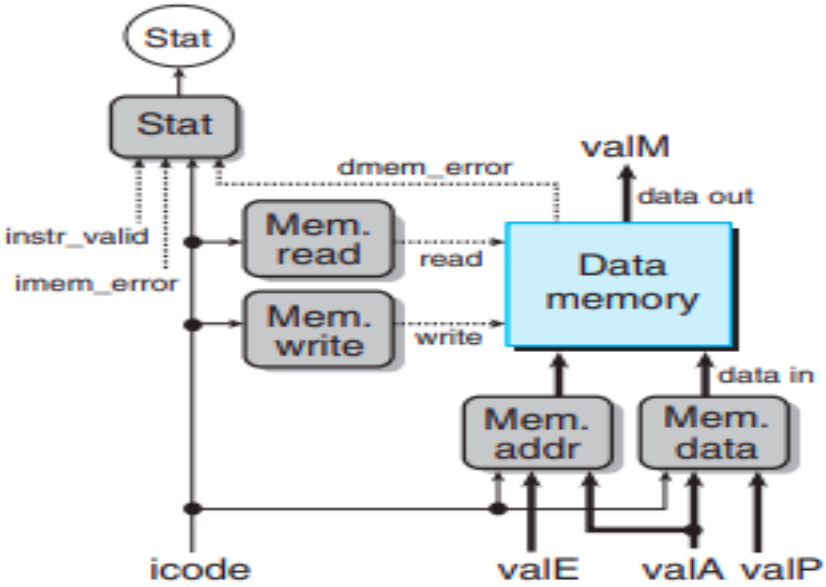
```
      else if(zero_flag) Cnd=1;
    end
    else if(ifun==4'b0010)
    begin
      r1=sign_flag;
      r2=overflow_flag;
      if(rout)Cnd=1;
    end
    else if(ifun==4'b0011)if(zero_flag)Cnd=1;
    else if(ifun==4'b0100)
    begin
      s=zero_flag;
      if(sout)Cnd=1;
    end
    else if(ifun==4'b0101)
    begin
      r1=sign_flag;
      r2=overflow_flag;
      s=rout;
      if(sout)Cnd=1;
    end
    else if(ifun==4'b0110)
    begin
      r1=sign_flag;
      r2=overflow_flag;
      s=rout;
      if(sout)
```

```
        begin
          s=zero_flag;
          if(sout)Cnd=1;
        end end end
    if(icode==4'b1000)valE=-64'd8+valB; //call
    if(icode==4'b1001)valE=64'd8+valB; //ret
    if(icode==4'b1010) valE=-64'd8+valB;//pushq
    if(icode==4'b1011)valE=64'd8+valB; //popq
 end
endmodule
```

Memory module

- This is the final module used while developing this processor design.
- The memory stage has the task of either reading or writing program data.
- Two control blocks generate the values for the memory.
- Two other blocks generate the control signals indicating whether to perform a read or a write operation

SEQ memory stage:




```
`timescale 1ns / 1ps

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

   input clk;

   input [3:0] icode,valA,valB,valE,valP;

   output reg [63:0] valM, datamem;

   reg [63:0] mem[0:1023];

   always@(*) begin

            if(icode==4'b0100)mem[valE] = valA;

            else if(icode==4'b0101) valM = mem[valE];

            else if(icode==4'b1000) mem[valE] = valP;

            else if(icode==4'b1001) valM = mem[valA];

            else if(icode==4'b1010) mem[valE] = valA;

            else if(icode==4'b1011)valM = mem[valA];

        datamem = mem[valE];

   end

endmodule
```

Results

```
❯ iverilog memory.v memory_test.v
❯ ./a.out
VCD info: dumpfile memory_test.vcd opened for output.
                0clk=1, icode= 4,valA= 4,valE= 2,valP= 6,valM=                 x,datamem=                 4
                5clk=1, icode= 4,valA=14,valE=14,valP= 6,valM=                 x,datamem=                14
               10clk=1, icode= 9,valA= 2,valE=14,valP= 6,valM=                 4,datamem=                14
 ~/acad/ipa
```

PC update

- Inputs to this module are clk,icode,cnd,valC,valM,valP.
- This module outputs a PC updated signal.
- Depending on the type of instruction the program counter value is updated using valC, valM, valP and it is shown at the output.
- For some instructions like jump it will check with cnd signal and then depending on the signal it updates the program counter.

```
`timescale 1ns / 1ps

module pc_update(

 clk,PC,Cnd,f_icode,valC,valM,valP,

 updated_pc

);

 input clk, Cnd;

 input [3:0] f_icode;

 input [63:0] valC,valP,valM,PC;

 output reg [63:0] updated_pc;

 always@(*)

 begin

   if(f_icode==4'b0111) //jxx

   begin
```

```
    if(Cnd==1'b1)updated_pc=valC;
    else updated_pc=valP;
  end
  else if(f_icode==4'b1000)updated_pc=valC; //call
  else if(f_icode==4'b1001)updated_pc=valM; //ret
  else  updated_pc=valP;
end
endmodule
```

Results

```
❯ iverilog pc-update.v pc_update_testbench.v
❯ ./a.out
VCD info: dumpfile pc_update_test1.vcd opened for output.
icode=1,Cnd= 2,valC=0,valM=                   x,valP=                   x,updated_pc=                  10                  10
icode=1,Cnd= 9,valC=0,valM=                   x,valP=                  20,updated_pc=                  12                  20
icode=1,Cnd= 8,valC=0,valM=                  13,valP=                  25,updated_pc=                  15                  13
icode=1,Cnd= 7,valC=0,valM=                   6,valP=                  19,updated_pc=                   9                   9
icode=1,Cnd= 7,valC=1,valM=                   6,valP=                  19,updated_pc=                   9                   6
```