THE NATIONAL INSTITUTE OF ENGINEERING MYSORE

Autonomous Institue Affliated to VTU

Department of Electronics and communication Engineering

MINOR PROJECT PHASE -2 PRESENTATION ON

“Implementation of Y86-64 processor architetecture design using hardware coding language VERILOG “

Guided by :

Dr. Narasimha kaulgud

Professor

Department Of ECE

NIE MYSORE

Sequential Y86-64 Implementations

- **Stages in Y86-64**
- Fetch
- Decode
- Execute
- Memory
- WriteBack
- PC update

1.Fetch

Algorithm for fetch Module

1.Check for Memory Error:

- If the PC value exceeds the memory boundary (e.g., 1023 in this case), set mem_error to 1 and exit.

2.Decode Instruction:

- Extract the icode and ifun from the instr bits.

3.Calculate Next PC (valP):

- Determine the next PC value based on the icode:
  - For icode values 0000, 1001, 0001: valP = PC + 1
  - For icode values 0010, 0110, 1010, 1011: valP = PC + 2
  - For icode values 0011, 0100, 0101: valP = PC + 10
  - For icode values 0111, 1000: valP = PC + 9

4.Extract Operands (if applicable):

- Based on the icode:
  - Extract rA and rB from the instr bits if applicable.
  - Extract valC from the instr bits if applicable.

5.Set instr_error:

- If the icode value is not recognized, set instr_error to 1.

2.Decode

Algorithm for Decode Stage

1.Initialize:

- Set valA and valB to 0.

2.Decode Instruction:

- Check the value of icode:
    - If icode is 6 (opq):
        - Set valA to the value of register rA.
        - Set valB to the value of register rB.
    - If icode is 2 (cmovxx):
        - Set valA to the value of register rA.
        - Set valB to 0.
    - If icode is 4 (rmmov):
        - Set valA to the value of register rA.
        - Set valB to the value of register rB.
    - If icode is 5 (mrmov):
        - Set valB to the value of register rB.
    - If icode is 10 (push A):
        - Set valA to the value of register rA.
        - Set valB to the value of register 4 (assuming %rsp is in register 4).

- If icode is 11 (pop B):
    - Set valA to the value of register 4 (assuming %rsp is in register 4).
    - Set valB to the value of register 4 (assuming %rsp is in register 4).
- If icode is 8 (call):
    - Set valB to the value of register 4 (assuming %rsp is in register 4).
- If icode is 9 (ret):
    - Set valA to the value of register 4 (assuming %rsp is in register 4).
    - Set valB to the value of register 4 (assuming %rsp is in register 4).

3.Execute

Algorithm for Execute Stage

1.Calculate Condition Flag (cnd):

- If icode is 2 (cmovxx) or 7 (jxx):
  - Determine the cnd value based on the ifun and the current flags (zf, sf, of):
    - ifun = 0000 (uc): cnd = 1 (unconditional)
    - ifun = 0001 (le): cnd = (sf ^ of) | zf
    - ifun = 0010 (l): cnd = of ^ sf
    - ifun = 0011 (e): cnd = zf
    - ifun = 0100 (ne): cnd = ~zf
    - ifun = 0101 (ge): cnd = ~(sf ^ of)
    - ifun = 0110 (g): cnd = ~(sf ^ of) & ~zf
- Otherwise, set cnd to 0.

2.Perform ALU Operations:

- Based on the icode:
  - If icode is 2 (cmovxx):
    - Perform the ALU operation for cmovxx (e.g., pass valA to valE_cmov).
  - If icode is 3 (irmov), 4 (rmmov), or 5 (mrmov):
    - Perform the ALU operation for memory moves (e.g., pass valC to valE_cb).

- If icode is 6 (opq):
  - Perform the ALU operation specified by ifun on valA and valB and store the result in valE_op.
  - Calculate cf_out based on the result of the ALU operation:
    - Set cf_out[0] (zero flag) to 1 if valE_op is 0, otherwise 0.
    - Set cf_out[1] (sign flag) to the value of the most significant bit of valE_op.
    - Set cf_out[2] (overflow flag) to the value of overflow_use from the ALU operation.
- If icode is 8 (call) or 10 (push A):
  - Perform the ALU operation for stack decrement (e.g., subtract 8 from valB and store the result in valE_sd).
- If icode is 9 (ret) or 11 (pop B):
  - Perform the ALU operation for stack increment (e.g., add 8 to valB and store the result in valE_si).

3.Select Output (valE):

- Based on the icode:
  - If icode is 2 (cmovxx): valE = valE_cmov
  - If icode is 3 (irmov), 4 (rmmov), or 5 (mrmov): valE = valE_cb
  - If icode is 6 (opq): valE = valE_op
  - If icode is 8 (call) or 10 (push A): valE = valE_sd
  - If icode is 9 (ret) or 11 (pop B): valE = valE_si
  - Otherwise, valE = 0

4.Memory

Algorithm for Memory Stage

1.Data Memory Access:

- Based on the icode:
    - If icode is 4 (rmmov) or 10 (pushq):
        - Store the value of valA in the memory location specified by valE.
    - If icode is 5 (mrmov):
        - Load the value from the memory location specified by valE into valM.
    - If icode is 11 (popq):
        - Load the value from the memory location specified by valA into valM.
    - If icode is 8 (call):
        - Store the value of valP (next instruction address) in the memory location specified by valE.
    - If icode is 9 (ret):
        - Load the value from the memory location specified by valA into valM.

2.(Optional) Data Memory Initialization:

- For debugging or testing purposes, you can initialize specific memory locations with predefined values. The example code initializes location valA with 200 and location valE with 100.

5. WRITE BACK

Algorithm for write_back Stage

1.Update Register File:

- On the negative edge of the clock:
    - Load the input register values (reg_in0 to reg_in14) into the internal register file (reg_file).
    - Based on the icode:
        - If icode is 6 (opq), 2 (cmovxx), or 3 (irmov):
            - Write the value of valE to the register specified by rB.
        - If icode is 11 (popq):
            - Write the value of valE to register 4 (%rsp).
            - Write the value of valM to the register specified by rA.
        - If icode is 8 (call), 9 (ret), or 10 (pushq):
            - Write the value of valE to register 4 (%rsp).
        - If icode is 5 (mrmov):
            - Write the value of valM to the register specified by rA.

2.Output Register Values:

- Assign the values from the internal register file (reg_file) to the output ports (reg_out0 to reg_out14).

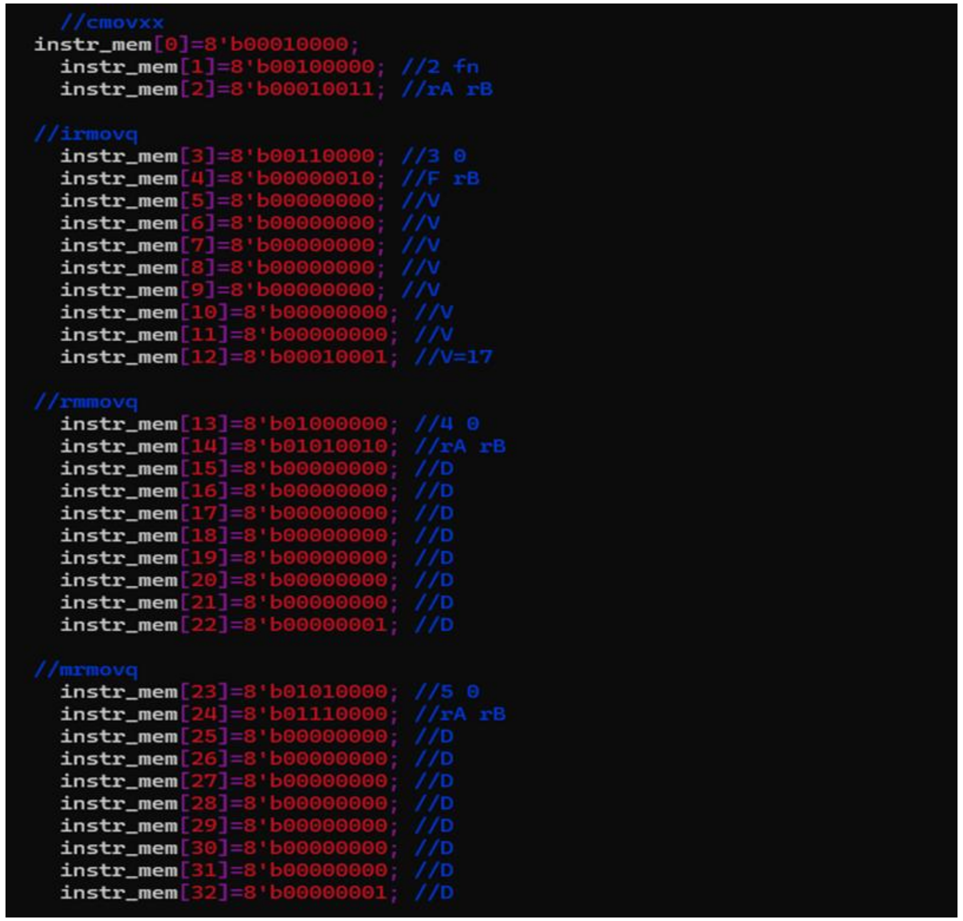
6.PC_Upadte

Algorithm for Pc update Module

1.On the Positive Edge of the Clock:

- Based on the icode:
  - If icode is 0000: Set pcnxt to 0.
  - If icode is 0111 (jxx):
    - If cnd is 1 (condition is true): Set pcnxt to valC (jump target).
    - Otherwise, set pcnxt to valP (next instruction address).
  - If icode is 1000 (call): Set pcnxt to valC (call target).
  - If icode is 1001 (ret): Set pcnxt to valM (return address).
  - For all other icode values: Set pcnxt to valP (next instruction address).

Testbench Instructions

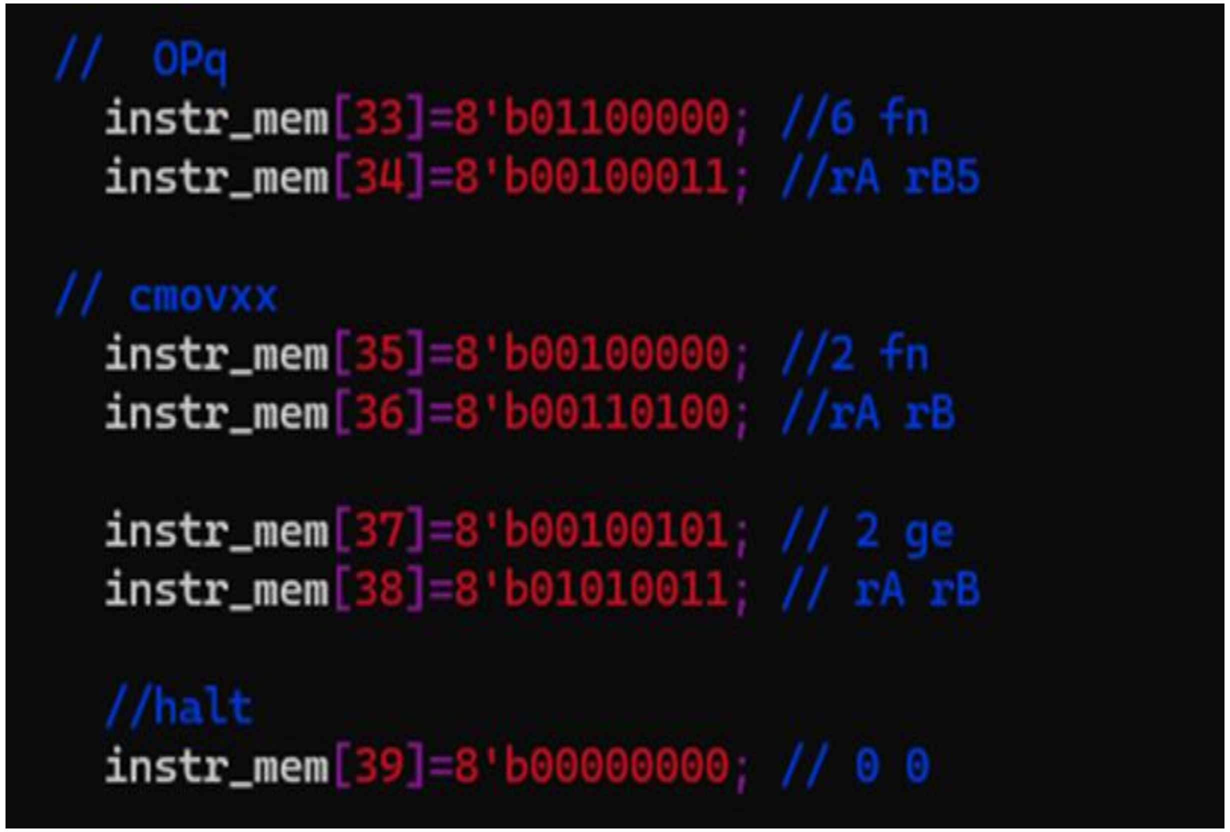


```
    //cmovxx
  instr_mem[0]=8'b00010000;
    instr_mem[1]=8'b00100000; //2 fn
    instr_mem[2]=8'b00010011; //rA rB

  //irmovq
    instr_mem[3]=8'b00110000; //3 0
    instr_mem[4]=8'b00000010; //F rB
    instr_mem[5]=8'b00000000; //V
    instr_mem[6]=8'b00000000; //V
    instr_mem[7]=8'b00000000; //V
    instr_mem[8]=8'b00000000; //V
    instr_mem[9]=8'b00000000; //V
    instr_mem[10]=8'b00000000; //V
    instr_mem[11]=8'b00000000; //V
    instr_mem[12]=8'b00010001; //V=17

  //rmmovq
    instr_mem[13]=8'b01000000; //4 0
    instr_mem[14]=8'b01010010; //rA rB
    instr_mem[15]=8'b00000000; //D
    instr_mem[16]=8'b00000000; //D
    instr_mem[17]=8'b00000000; //D
    instr_mem[18]=8'b00000000; //D
    instr_mem[19]=8'b00000000; //D
    instr_mem[20]=8'b00000000; //D
    instr_mem[21]=8'b00000000; //D
    instr_mem[22]=8'b00000001; //D

  //mrmovq
    instr_mem[23]=8'b01010000; //5 0
    instr_mem[24]=8'b01110000; //rA rB
    instr_mem[25]=8'b00000000; //D
    instr_mem[26]=8'b00000000; //D
    instr_mem[27]=8'b00000000; //D
    instr_mem[28]=8'b00000000; //D
    instr_mem[29]=8'b00000000; //D
    instr_mem[30]=8'b00000000; //D
    instr_mem[31]=8'b00000000; //D
    instr_mem[32]=8'b00000001; //D
```



```
//  OPq
  instr_mem[33]=8'b01100000; //6 fn
  instr_mem[34]=8'b00100011; //rA rB5

// cmovxx
  instr_mem[35]=8'b00100000; //2 fn
  instr_mem[36]=8'b00110100; //rA rB

  instr_mem[37]=8'b00100101; // 2 ge
  instr_mem[38]=8'b01010011; // rA rB

  //halt
  instr_mem[39]=8'b00000000; // 0 0
```