Hau Tao
005393943

Lab 1: The MIPS datapath in Verilog: The IF stage

A. Introduction

The objective of this lab is to implement and test the instruction fetch (IF) pipeline stage of the MIPS five stages pipeline. In this lab, I build up and test successfully 5 components of IF stages including program counter, multiplexer, adder, instruction memory and latch.

B. Interface

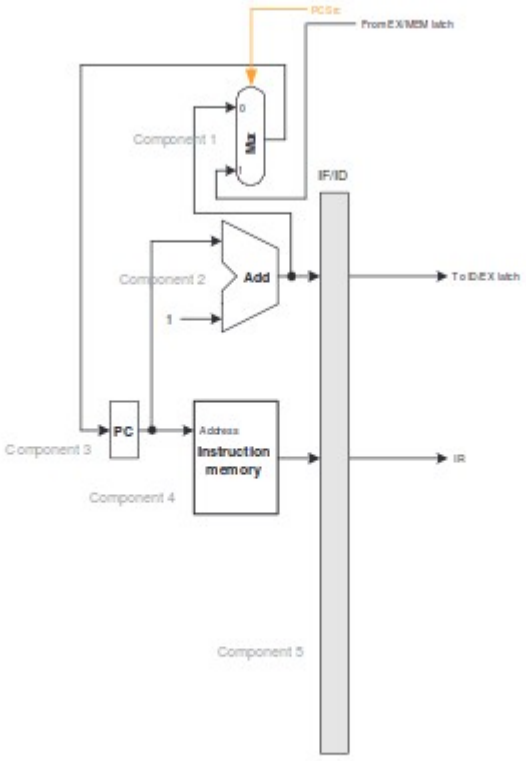


Figure 1.2: The IF stage

The next instruction is fetched from the memory address that is currently stored in the program counter (PC), and stored in the fetch register(IF_ID). At the end of the fetch operation, the PC points to the next instruction (PC +1 via Adder) that will be read at the next cycle.

C. Design

- For now, we just use the pipeline register named IF_ID in this lab, and EX_MEM is the input signal to the 2 x1 mux
- The instruction memory has 128 32-bit words. Later it will be expanded. All instructions and the PC are 32-bit wide. (Simply the 7 least significant bits (2^7 = 128) are used for the time being.)
- Implement the instruction memory, 2x1 MUX, and Incrementer-by-4 as separate modules, latch(IF/ID), and Adder (Add). For the time being consider that the 1-bit signal PCSrc comes from a 1-bit register, PC choose.
- Initialize IF_ID_IR (The instruction field of IF/ID) to 32 zeros.
- Initialize IF_ID_NPC to 32 zeros. Initialize PC choose and EX_MEM_NPC to zeros. They will not change during this simulation. Initialize the first 10 words of memory (with addresses 0, 4, 8, etc.) with the following HEX values

A00000AA
10000011
20000022
30000033
40000044
50000055
60000066
70000077
80000088
90000099

D. Implementation

- Program Counter (PC)

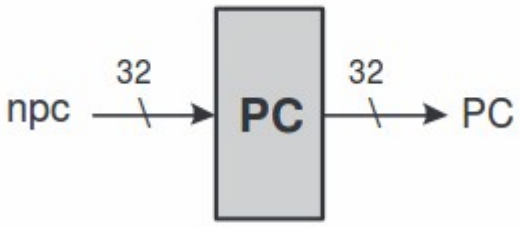


Figure 1.3: The program counter (PC)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Student: HAU TAO
// Create Date:22:01:58 10/20/2015
// Module Name:         progrmCounter
//////////////////////////////////////////////////////////////////////////////////
module programCounter(npc, pc);
    input[31:0] npc;
    output [31:0] pc;
    assign pc = npc;
endmodule
```

- The Instruction Memory

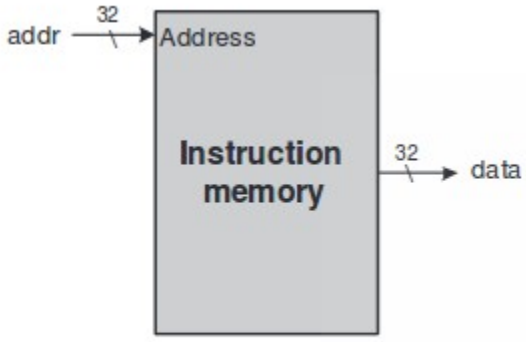


Figure 1.4: The instruction memory

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
```

```
// Create Date:10:43:29 10/21/2015
// Module Name:         instruction_memory
//////////////////////////////////////////////////////////////////////////////////

module instruction_memory(output reg[31:0] data, input wire[31:0] address);
 reg[31:0]words[0:127]; // 128 words of 32-bit memory
 initial
    begin
    words[0] <= 'hA00000AA;
    words[1] <= 'h10000011;
    words[2] <= 'h20000022;
    words[3] <= 'h30000033;
    words[4] <= 'h40000044;
    words[5] <= 'h50000055;
    words[6] <= 'h60000066;
    words[7] <= 'h70000077;
    words[8] <= 'h80000088;
    words[9] <= 'h90000099;
    end

 always@(address)
  begin
    data <= words[address];
  end
endmodule
```

- The 2x1 Multiplexer

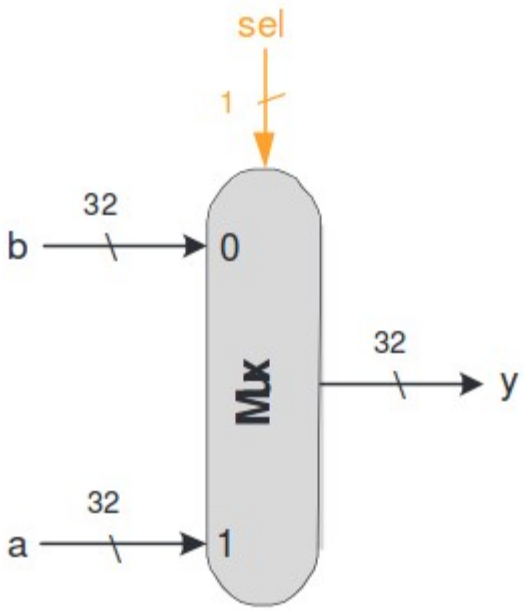


Figure 1.5: The multiplexer

```
`timescale 100ns / 10ns
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Hau Tao
// Create Date:22:01:58 10/20/2015
// Module Name:         MUX
//////////////////////////////////////////////////////////////////////////////////
module MUX( a, b, sel, y);
    input[31:0] a, b;
    input sel;
    output[31:0] y;
    assign y = sel ? a: b;
endmodule
```

- The Incrementer by 1

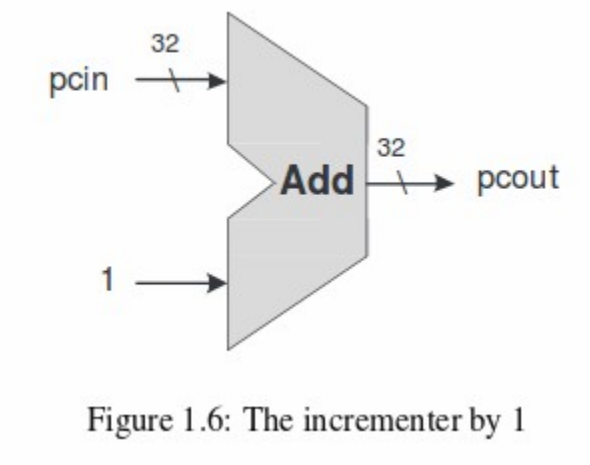


Figure 1.6: The incrementer by 1

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
//
// Create Date:16:39:09 10/07/2015
// Module Name:         INCR
//////////////////////////////////////////////////////////////////////////////////
module INCR(pcout, pcin);

    input  [31:0] pcin;
    output [31:0] pcout;
    assign  pcout  = pcin + 1;

endmodule
```

- The IF/ID register (latch)

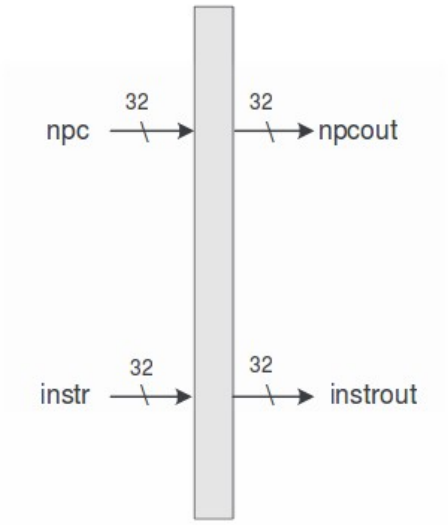


Figure 1.7: The IF/ID pipeline register (latch)

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
// Create Date:16:07:44 10/21/2015
// Module Name:         latch
//////////////////////////////////////////////////////////////////////////////////
module latch(input wire[31:0] npc, input wire[31:0] instr, output reg [31:0] npcout, output
reg[31:0] instrout
          );
    initial begin
    npcout <= npc;
   instrout <= instr;
   end

    always @*
    begin
    #1  // clock cycle of the latch
          npcout <= npc;
          instrout <= instr;
    end
endmodule
```

- The completed IF

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer: Hau Tao
//
// Create Date:22:40:13 10/27/2015
// Module Name:         IF_stage
//
//////////////////////////////////////////////////////////////////////////////////
module IF_stage(input  PCSrc, input [31:0] EX_MEM, output wire[31:0] npcout, output
wire[31:0] instrout
          );
    wire [31:0] mux_out;
    wire [31:0] pc_output;
    wire [31:0] mem_data_out;
    wire [31:0] output_incrementer;


    MUX  mux1(.a(EX_MEM), .b(output_incrementer), .sel(PCSrc), .y(mux_out));
    INCR increment1(.pcout(output_incrementer), .pcin(pc_output));
    instruction_memory mem1(.data(mem_data_out),.address(pc_output));
    programCounter pc1(.npc(mux_out),.pc(pc_output));


    latch latch1(
          .npc(output_incrementer),
          .instr(mem_data_out),
          .npcout(npcout),
          .instrout(instrout)
   );

endmodule
```

E. Test bench design

- Program counter PC

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
//
// Create Date:16:39:09 10/07/2015
// Module Name:         testPC
//////////////////////////////////////////////////////////////////////////////////
module testPC;
 wire[31:0] pcout;
 reg [31:0] pcin;
 programCounter pc(.npc(pcin),.pc(pcout));
 initial begin
 # 10
 pcin = 10;
 #10
 pcin = 20;
 #10
 pcin = 25;
#10;
end
always @(pcin)
    #2 $display("Time = %0d\tpcin =%0d\tpcout=%0d", $time, pcin, pcout);
endmodule
```

- The Instruction Memory

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
//
// Create Date:   10:50:19 10/21/2015
// Design Name:   instruction_memory
//////////////////////////////////////////////////////////////////////////////////
module test_mem;
    // Inputs
    reg [31:0] in;
    // Outputs
    wire [31:0] out;
    reg[31:0]words[0:127];
    // Instantiate the Unit Under Test (UUT)
    instruction_memory uut (
            .data(out),
            .address(in)
    );
    initial begin
            // Initialize Inputs
            #1
            words[0] = 'hA00000AA;
            in= 0;

            // Wait 10 ns for global reset to finish
            #10
            //address = 'h10000011;
            words[1] = 'h10000011;
            in = 1;
            #10
            words[2] = 'h20000022;
            in = 2;
            //address = 'h20000022;

            // Add stimulus here

    end
    always@(words or in )
    #1 $display("Time = %0d\tin=%0h\tdata=%0h", $time, in, out);
endmodule
```

- The 2x1 MUX

```
`timescale 1
s / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
// Create Date:   17:51:21 10/07/2015
// Project Name:  mux_1
//////////////////////////////////////////////////////////////////////////////////

module test_mux;

    // Verilog Test Fixture Template
    //wire ports
    wire[31:0] Y;
    //register Declarations
    reg[31:0] A, B;
    reg sel;
    MUX mux1 (A, B, sel, Y);
    initial begin
    A = 32'hAAAAAAAA;
    B = 32'h55555555;
    sel = 1'b1;
    #10;
    A = 32'h00000000;
    #10
    sel = 1'b1;
    #10;
    B = 32'hFFFFFFFF;
    #5;
    A= 32'hA5A5A5A5;
    #5;
    sel = 1'b0;
    B = 32'hDDDDDDDD;
    #5;
    sel = 1'bx;
    end
    always @(A or B or sel)

    #1 $display("At t = %0d sel =%b A =%h B=%h Y=%h",$time, sel, A, B, Y);

endmodule
```

- The Incrementer by 1

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
// Create Date:   18:08:26 10/07/2015
// Design Name:   INCR
//////////////////////////////////////////////////////////////////////////////////


module tes()t;
// Port Wires
    wire[31:0] IncrOut;

// register declaration
    reg [31:0] A;
    INCR incr1(IncrOut, A);

    initial begin
    #10
    A = 3;

    #10
    A= 15;

    #10
    A = 64;

    #5;

    end

    always @(A)
    #1 $display("Time = %0d\tA =%0d\tincrOut=%0d", $time, A, IncrOut);

endmodule //test
```

- The IF/ID (latch)

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
//
// Create Date:   22:18:47 10/27/2015
// Design Name:   latch
//////////////////////////////////////////////////////////////////////////////////

module test_latch;

    // Inputs
    reg [31:0] npc;
    reg [31:0] instr;

    // Outputs
    wire [31:0] npcout;
    wire [31:0] instrout;

    // Instantiate the Unit Under Test (UUT)
    latch uut (
            .npc(npc),
            .instr(instr),
            .npcout(npcout),
            .instrout(instrout)
    );

    initial begin
            // Initialize Inputs
            npc = 0;
            instr = 0;

            #10
            npc <= 10;
            instr <= 12;
            #10;
            $finish;
            end

            always @*
            begin
            #1 $display("Time = %0d\tnpc =%0d\tinstr=%0d\t npcout=%0d\t instrout=%0d", $time,
npc, instr, npcout, instrout);
```

```
        end
endmodule
```

- The completed IF

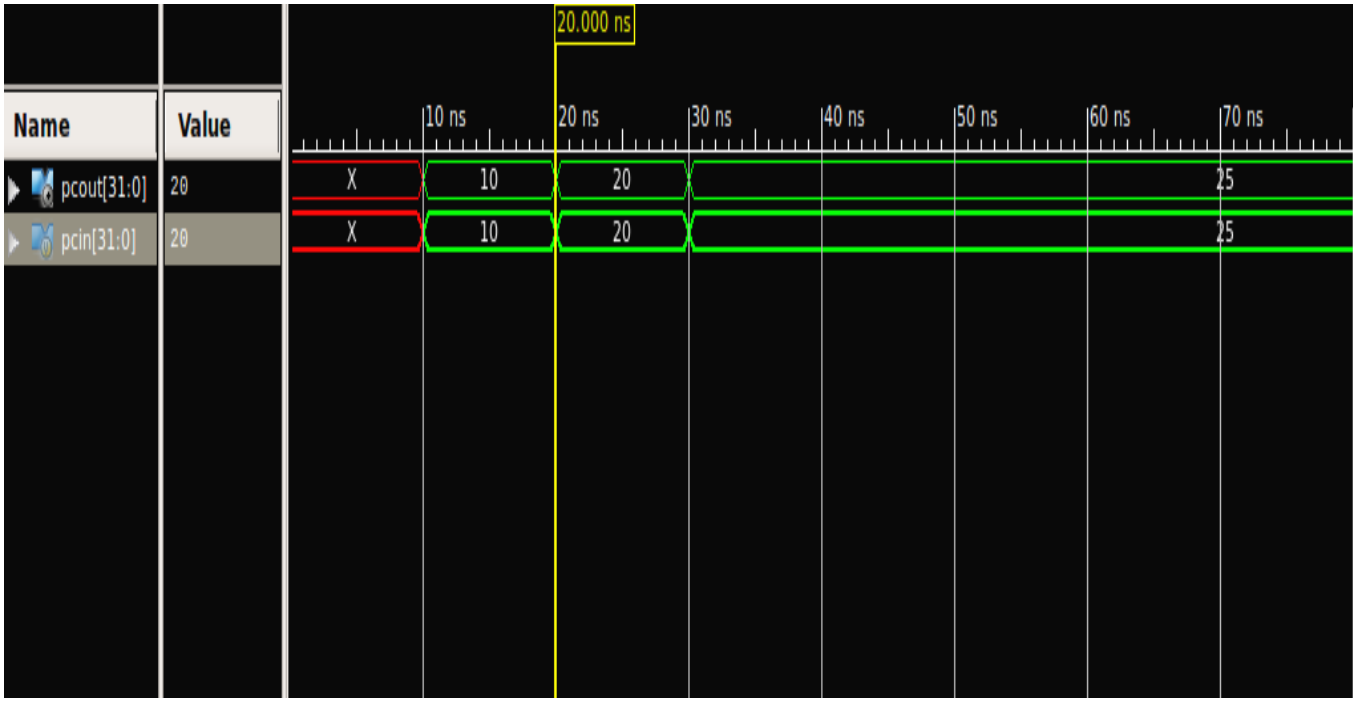
```
`timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
// Engineer: Hau Tao
// Create Date:   17:38:46 10/21/2015
// Design Name:   IF_stage
//////////////////////////////////////////////////////////////////////////////////

module test_IF;

    // Inputs
    reg PCSrc;
    reg [31:0] EX_MEM;

    // Outputs
    wire [31:0] npcout;
    wire [31:0] instrout;

    // Instantiate the Unit Under Test (UUT)
    IF_stage uut (
            .PCSrc(PCSrc),
            .EX_MEM(EX_MEM),
            .npcout(npcout),
            .instrout(instrout)
    );

    initial begin
            // Initialize Inputs
            PCSrc = 0;
            EX_MEM = 0;

            // Wait 100 ns for global reset to finish
            #6
            PCSrc = 1;
            EX_MEM =1;
            #15
            PCSrc = 1;
            EX_MEM =2;


    end
    always @(*)
            begin
            #1 $display("Time = %0d\tPCSrc =%0d\tEX_MEM=%0d\tnpcout=%0d\tinstrout=%0h\t",
$time, PCSrc,EX_MEM,npcout,instrout);
```

```
        end

endmodule
```

F. Simulation

- Program counter PC

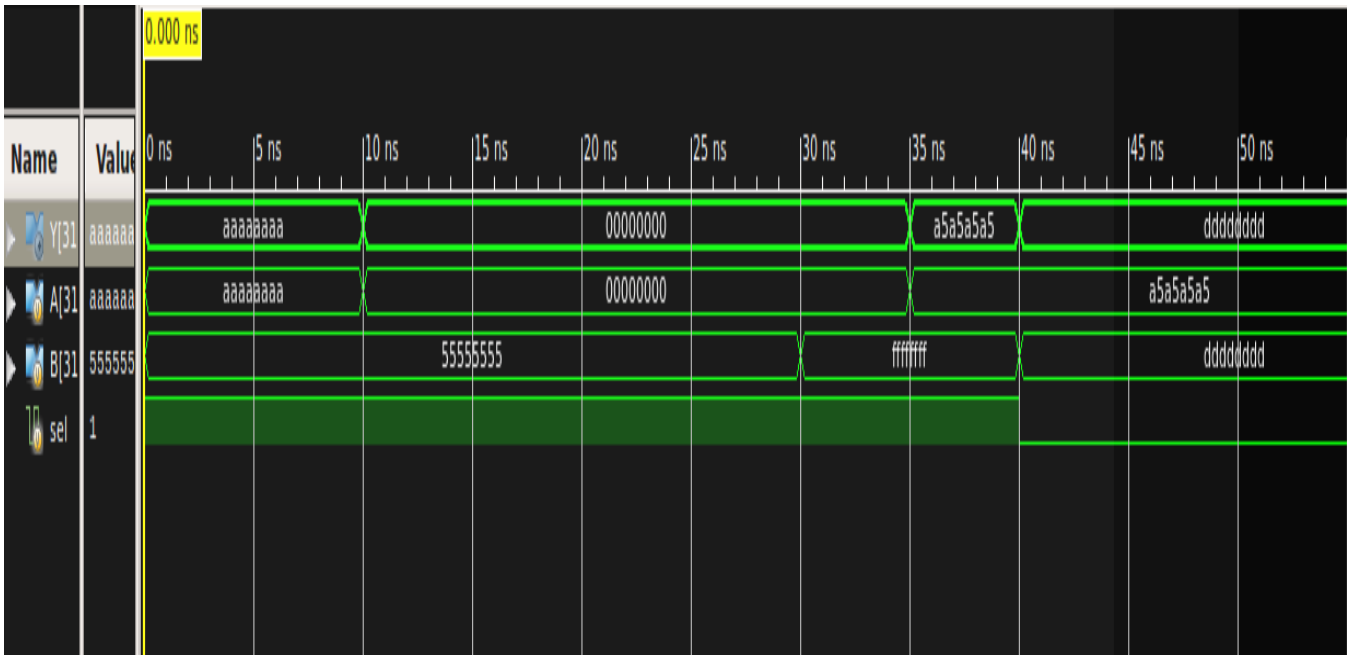



```
Simulator is doing circuit initialization
Finished circuit initialization process.
Time = 12   pcin =10     pcout=10
Time = 22   pcin =20     pcout=20
Time = 32   pcin =25     pcout=25
```

- The instruction memory

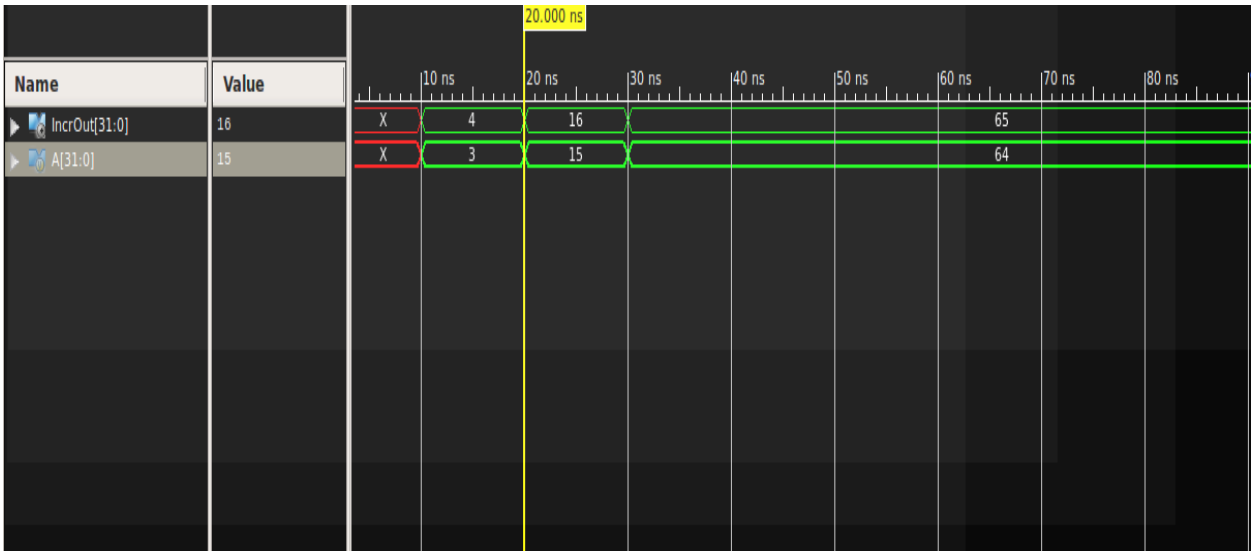

```
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Time = 2    in=0        data=a00000aa
Time = 12   in=1        data=10000011
Time = 22   in=2        data=20000022
ISim>
```

- the 2x1 MUX



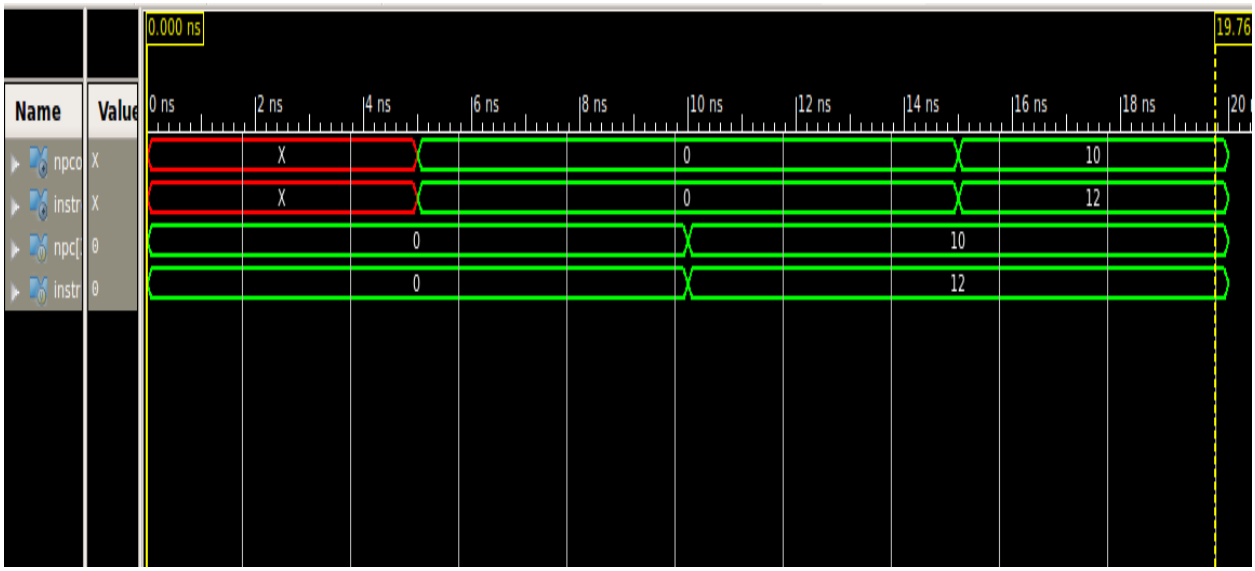

```
At t = 1 sel =1 A =aaaaaaaa B=55555555 Y=aaaaaaaa
At t = 11 sel =1 A =00000000 B=55555555 Y=00000000
At t = 31 sel =1 A =00000000 B=ffffffff Y=00000000
At t = 36 sel =1 A =a5a5a5a5 B=ffffffff Y=a5a5a5a5
At t = 41 sel =0 A =a5a5a5a5 B=dddddddd Y=dddddddd
ISim>
```

- Incrementer by 1




```
Simulator is doing circuit initialization process
Finished circuit initialization process.
Time = 11   A =3        incrOut=4
Time = 21   A =15       incrOut=16
Time = 31   A =64       incrOut=65
ISim>
```

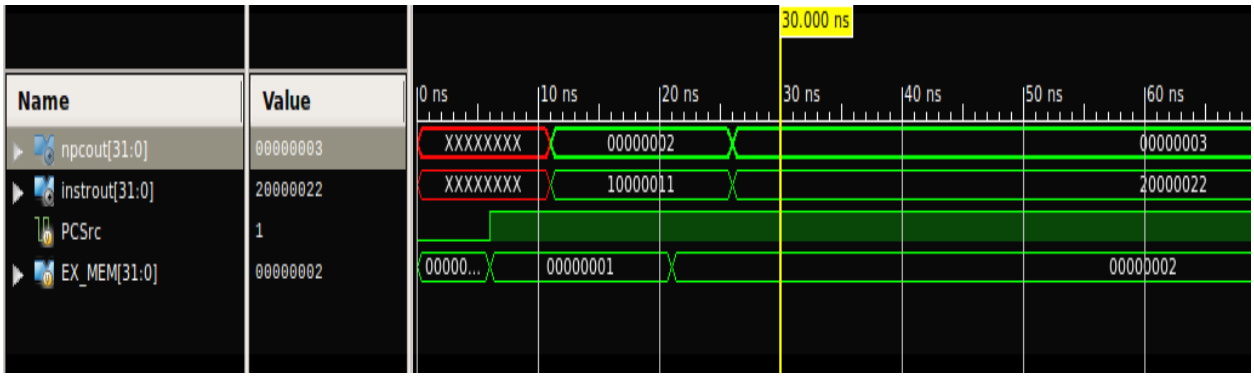
- lF(latch)




```
Time = 1    npc =0      instr=0      npcout=x   instrout=x
Time = 6    npc =0      instr=0      npcout=0   instrout=0
Time = 11   npc =10     instr=12     npcout=0   instrout=0
Time = 16   npc =10     instr=12     npcout=10  instrout=12
Stopped at time : 20 ns : File "/home/hau/Desktop/labcse401/lab1/IFMISP/test_latch.v" Line 37
```

.

- The completed IF simulation




```
Time = 1   PCSrc =0   EX_MEM=0 npcout=x   instrout=xxxxxxxx
Time = 7   PCSrc =1   EX_MEM=1 npcout=x   instrout=xxxxxxxx
Time = 12  PCSrc =1   EX_MEM=1 npcout=2   instrout=10000011
Time = 22  PCSrc =1   EX_MEM=2 npcout=2   instrout=10000011
Time = 27  PCSrc =1   EX_MEM=2 npcout=3   instrout=20000022
ISim>
```

G. Conclusion

The purpose of this project is to implement IF module. I completed it successfully and the test cases is good enough .When the clock cycle is on a positive edge the output of the IF will change regarded to input, but not change immediately. The difficulty of this lab is not implementation, but setting up the environment took me much time, change from window environment to Ubuntu . After completing all the lab, I almost lost all of them because problems with the software package, and I had to do it again