LAB 1

The MIPS datapath in Verilog: The IF stage

Objective: To implement and test the Instruction Fetch (IF) pipeline stage of the MIPS five stage pipeline.

The series of labs in this manual has ultimate objective to implement and simulate in Verilog the MIPS pipeline datapath Figure 6.30 in Paterson and Hennessy's textbook [4]. The model will be structural (as opposed to behavioral), but with one exception: basic units, such as multiplexors and ALU's, may implemented as behavioral models. This approach reinforces the object-oriented style of programming, while at the same time relieving from the burden of structurally defining the basic units, which can be quite tedious, time consuming, and beyond the scope of this lab series.

The slightly revised MIPS datapath to be implemented is in figure 1.1 on page Lab 1–2.

For this week, you will implement the IF stage and test the fetching of instructions from memory. The IF stage isolated from the rest of the datapath can be seen in figure 1.2 on page Lab 1–2.

- The names of the pipeline registers are IF ID, ID EX, EX MEM, MEM WB. For now, you will need only IF ID and EX MEM.
- 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. 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:

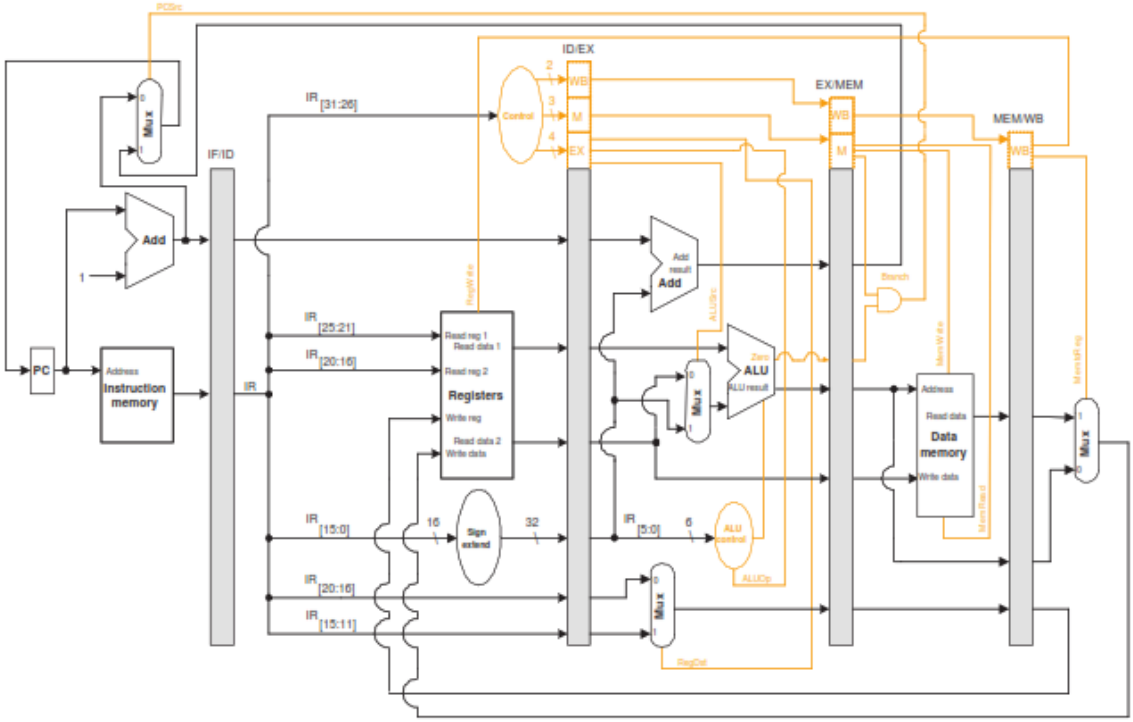

Figure 1.1: The revised MIPS datapath

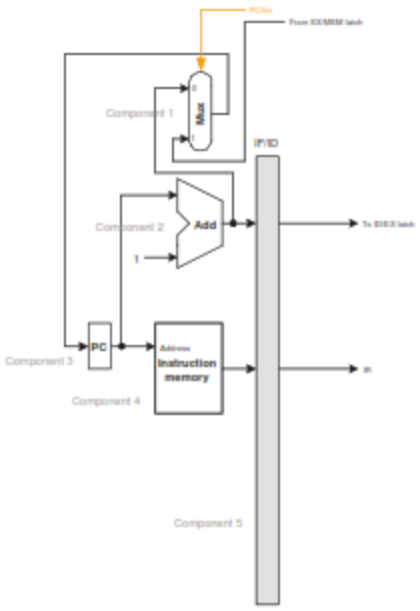

Figure 1.2: The IF stage

```
module mux (a,b,sel,y);
  input [31:0] a, b;
  input  sel;
  output [31:0] y;
   assign       y = sel ? a : b;
endmodule
```

Listing 1.1: Verilog code for the multiplexer.

```
A00000AA
10000011
20000022
30000033
40000044
50000055
60000066
70000077
80000088
90000099
```

- Turn in the source code and the printout of the clock cycle number, the contents of the PC (in decimal), IF ID IR (in hex), and IF ID NPC (in decimal) for 10 cycles of simulation. Be ready to demonstrate.

Note: The code in listing 1.1 implements the multiplexer in the IF stage as a combinational circuit.

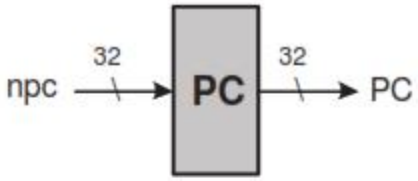

Figure 1.3: The program counter (PC)

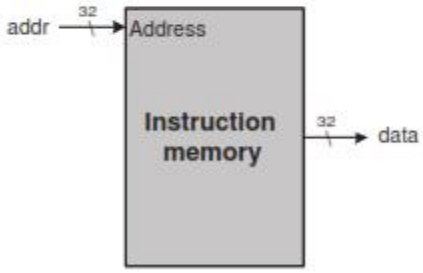

Figure 1.4: The instruction memory

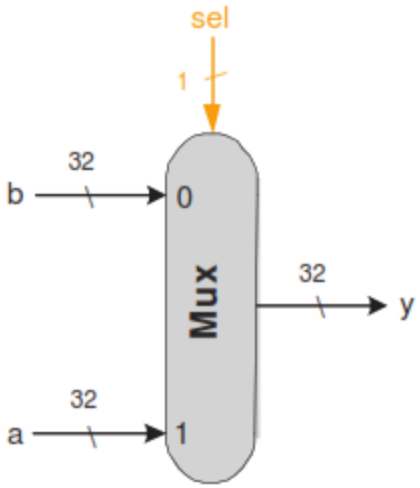

Figure 1.5: The multiplexer

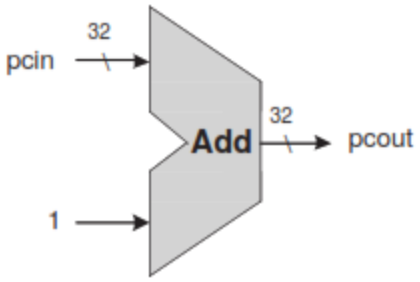

Figure 1.6: The incrementer by 1

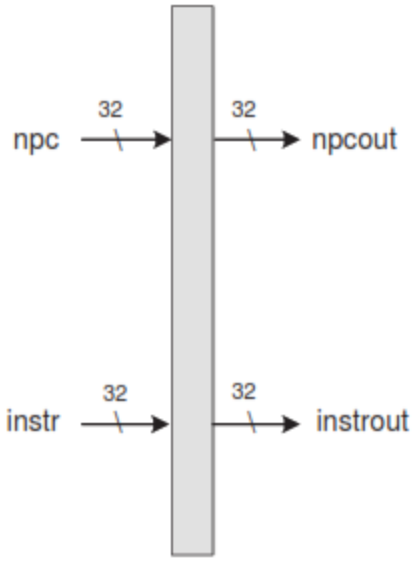

Figure 1.7: The IF/ID pipeline register (latch)

1.1 Testbenches

Testbenches help us verify that the design is correct. In this subsection we show two testbenches: One in listing 1.2 on page Lab 1–6 for the multiplexer of figure 1.5 on page Lab 1–4 and one in listing 1.3 on page Lab 1–7 for the incrementer of figure 1.6 on page Lab 1–4 . The results of the running the testbench are in figure 1.8 and in figure 1.9, respectively. In the latter, and in other testbench runs in the labs that follow, the standard messages of the runs will be largely omitted.

```
Beginning Compile
Beginning Phase I
Compiling source file: muxtest.v
Compiling included source file 'mux.v'
Continuing compilation of source file 'muxtest.v' Finished
Phase I
Entering Phase II... Finished
Phase II Entering Phase III...
Finished Phase III
Highest level modules: test_mux
Compile Complete
. Running...
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
At t = 46 sel = x A = a5a5a5a5 B = dddddddd Y = XXXXXXXX
0 Errors, 0 Warnings
Compile time = 0.00000, Load time = 0.00000, Execution time = 0.00000

Normal exit
```

Figure 1.8: The output when running the testbench for the multiplexer (listing 1.2 on page Lab 1–6)

```
Running...
Time = 11 A=3 IncrOut=4
Time = 21 A=15 IncrOut=16
Time = 31 A=64 IncrOut=65
```

Figure 1.9: The output when running the testbench for the incrementer (listing 1.3 on page Lab 1–7)

```
// Filename         : test_mux.v
// Description      : Testing the 32bit_mux module
//                    of the IF stage of the pipeline.

`include "mux.v"
module test_mux;

// Wire Ports
   wire [31:0] Y;

// Register Declarations
   reg  [31:0] A, B;
   reg         sel;

   MUX mux1 (Y, A, B, sel);  //instantiate the mux

   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  // test
```

```
// Filename         : test-incr.v
// Description      : Test for incr.v, an incrementer by 1
//                    (32-bit input)

`include "incr.v"

module test ();

// Port Wires
   wire [31:0] IncrOut;

// Register Declarations
   reg  [31:0] A;

   INCR incr1 (IncrOut, A);  //instantiate the incrementer

   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
```

Listing 1.3: The testbench for the incrementer in figure 1.6 on page Lab 1–4