Lab 12: Full Non-Pipelined Datapath

Matthew Carrano and Breana Leal

April 16, 2018

1 Introduction

The goal of the lab is to integrate all five stages of the ARM's datapath into one module. Previously, each stage was implemented as a separate lab assignment and tested for accuracy. The stages are compared with an expected results table, configured with binary instruction sets. As a final test, a new binary data file was created to perform a simple division.

2 Interface

This section should identify the inputs and outputs of each stage. To do this, rather than explaining them in paragraph form, please take the datapath diagram in Figure 1 and add your signal names to the diagram. This will give you a graphical representation of your system that can be quickly evaluated to determine the meaning of each signal. For any additional signals that appear on your simulation results, put the signals in a table with a short description of that signal.

3 Design

See Figure 2

4 Implementation

Listing 1: Verilog code for testing the fetch stage.

```
`include "definitions.vh"

module datapath;

   wire clk;
   reg rst;
   //reg pc_src;
```

Figure 1: Full Non-Pipelined Datapath

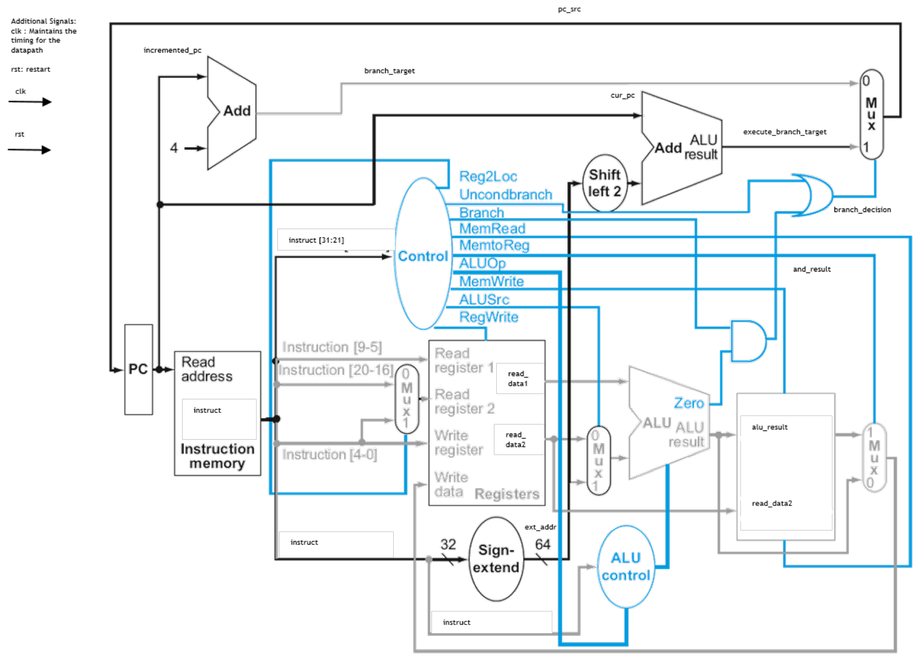
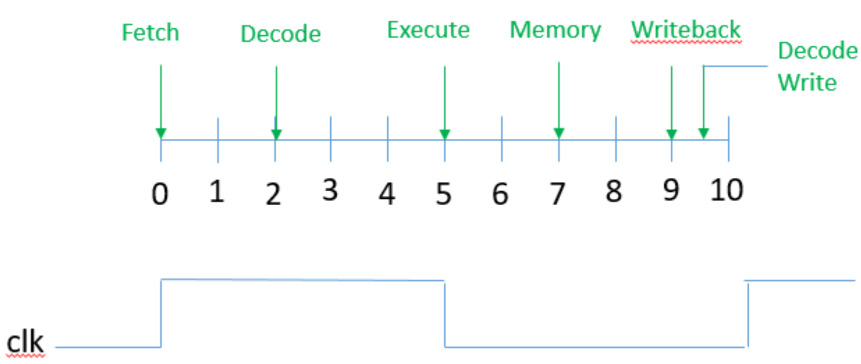


Figure 2: Timing Diagram



```
//reg [`WORD-1:0] branch_target;
wire [`WORD-1:0] write_data;
wire [`WORD-1:0] read_data1;
wire [`WORD-1:0] read_data2;
wire [`WORD-1:0] incremented_pc;
wire [`WORD-1:0] cur_pc;
wire uncondbranch;
wire branch;
wire mem_read;
wire mem_to_reg;
wire [1:0] alu_op;
wire mem_write;
wire alu_src;
wire [`WORD-1:0] ext_addr;
wire [`INSTR_LEN-1:0] instruct;
wire [`WORD-1:0] execute_branch_target;
wire [`WORD-1:0] alu_result;
wire [`WORD-1:0] read_data;
//wire [`WORD-1:0] write_back;
wire zero;
wire branch_decision;

 oscillator clk_gen(clk);

 iFetch fetch(
 .clk(clk),
 .reset(rst),
 .pc_src(branch_decision),
 .branch_target(execute_branch_target),
 .incremented_pc(incremented_pc),
 .instruction(instruct), //The only wire that connects iFetch and iDecode
 .cur_pc(cur_pc)
 );

 iDecode decode(
 .clk(clk_plus_2),
 .read_clk(clk_plus_3),
 .write_clk(clk_plus_9),
 .instr(instruct),
 .write_data(write_data),
 .read_data1(read_data1),
 .read_data2(read_data2),
 .uncondbranch(uncondbranch),
 .branch(branch),
 .mem_read(mem_read),
 .mem_to_reg(mem_to_reg),
```

```
.alu_op(alu_op),
.mem_write(mem_write),
.alu_src(alu_src),
.ext_addr(ext_addr)
);

iExecute execute(
  .pc_in(cur_pc),
  .read_data1(read_data1),
  .read_data2(read_data2),
  .sign_extended_output(ext_addr),
  .branch_target(execute_branch_target),
  .alu_result(alu_result),
  .zero(zero),
  .alu_op(alu_op),
  .opcode(instruct[31:21]),
  .alu_src(alu_src)
 );

 iMemory memory(
  .mem_read(mem_read),
  .mem_write(mem_write),
  .zero(zero),
  .branch(branch),
  .uncond_branch(uncondbranch),
  .clk(clk_plus_7),
  .alu_result(alu_result),
  .read_data2(read_data2),
  .or_result(branch_decision),
  .read_data(read_data)
  );

 iWrite_Back writeback(
  .read_data_mem(read_data),
  .alu_result(alu_result),
  .mem_to_reg(mem_to_reg),
  .result(write_data)
  );


delay clk_delay_1(
    .a(clk),
    .a_delayed(clk_plus_1)
);

delay clk_delay_2(
```

```
        .a(clk_plus_1),
        .a_delayed(clk_plus_2)
    );

    delay clk_delay_3(
        .a(clk_plus_2),
        .a_delayed(clk_plus_3)
    );

        delay clk_delay_4(
        .a(clk_plus_3),
        .a_delayed(clk_plus_4)
    );

    delay clk_delay_5(
        .a(clk_plus_4),
        .a_delayed(clk_plus_5)
    );

    delay clk_delay_6(
        .a(clk_plus_5),
        .a_delayed(clk_plus_6)
    );

        delay clk_delay_7(
        .a(clk_plus_6),
        .a_delayed(clk_plus_7)
    );


      delay clk_delay_8(
      .a(clk_plus_7),
      .a_delayed(clk_plus_8)
    );


      delay clk_delay_9(
      .a(clk_plus_8),
      .a_delayed(clk_plus_9)
    );

initial
        begin
            // wait for the rising clock edge
           // #(`CYCLE/2);
            rst <= 1;
```

```
        //branch_target <= 0;
        #(`CYCLE/1.5);
        rst <= 0;
        /* write_data <= `WORD'd20;
        #(`CYCLE);
        write_data <= `WORD'd30; // Addition result
        #(`CYCLE);
        write_data <= `WORD'd0; // Subtraction Result
        #(`CYCLE);
        write_data <= `WORD'd0; // N/A
        #(`CYCLE);
        write_data <= `WORD'd0; // N/A
        #(`CYCLE);
        write_data <= `WORD'd0; // N/A
        #(`CYCLE);
        write_data <= `WORD'd0; // N/A
        #(`CYCLE);
        write_data <= `WORD'd0; // N/A
        #(`CYCLE);
        write_data <= `WORD'd30; // ORR result
        #(`CYCLE);
        write_data <= `WORD'd14; // Writing
        */
    end

endmodule
```

5 Test

1. Assembly code for the "Division Problem" C Code For x = X10, y = X11, z = X12, one = X13, Base address of A = X22

   LDUR X10, [X22, #0]

   LDUR X11, [X22, #8]

   LDUR X12, [X22, #16]

   LDUR X13, [X22, #24]

   SUB X10, X10, X11

   ADD X12, X12, X13

   B -3

   STUR X12, [X22,#16]

2. Instruction Data File for the Division Problem

   See Figure 3

Figure 3: Instruction Data

```
11111000010000000000001011001010
11111000010000010000001011001011
11111000010000100000001011001000
11111000010000110000001011001101
10110100000000000000000010001010
11001011000010110000000101001010
10001011000011010000000110001100
00010111111111111111111111111101
01111100000000010000001011001011
```

3. Register Data File for the Division Problem

   See Figure 4

4. Memory Data File for the Division Problem

   See Figure 5

5. Simulation Results for the Division Problem

   See Figure 6

6 Conclusions

Our datapath is working successfully based on our results from the division code. The code divided 10 by 2. The program looped through six times, attaining a correct final alu_result of 5. The value was then stored in A[2].

Figure 4: Ram Data

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000001010
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000010
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000001
```

Figure 5: Register File

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000001000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

Figure 6: Division Simulation