University Of Waterloo

ECE 429

# Task 2
# Fetching and Decoding Instructions


| | |
|---|---|
| Stephan Van Den Heuval | sjvanden |
| Ravil Baizhiyenov | rbaizhiy |
| David Janssen | dajjanss |
| Ben Ridder | brridder |


May 28, 2012

# 1 Output

Listing 1: Bubble Sort Output

```
# opened file srec_files/BubbleSort.srec
#
# Initializing memory
# read in file
#
# 80020000: 27bdffc8
# 80020004: afbf0034
# 80020008: afbe0030
# 8002000c: 03a0f021
# 80020010: 2402000c
# 80020014: afc20010
# 80020018: 24020009
# 8002001c: afc20014
# 80020020: 24020004
# 80020024: afc20018
# 80020028: 24020063
# 8002002c: afc2001c
# 80020030: 24020078
# 80020034: afc20020
# 80020038: 24020001
# 8002003c: afc20024
# 80020040: 24020003
# 80020044: afc20028
# 80020048: 2402000a
# 8002004c: afc2002c
# 80020050: 27c20010
# 80020054: 00402021
# 80020058: 24050008
# 8002005c: 0c008020
# 80020060: 00000000
# 80020064: 00001021
# 80020068: 03c0e821
# 8002006c: 8fbf0034
# 80020070: 8fbe0030
# 80020074: 27bd0038
# 80020078: 03e00008
# 8002007c: 00000000
# 80020080: 27bdffe8
# 80020084: afbe0014
# 80020088: 03a0f021
# 8002008c: afc40018
# 80020090: afc5001c
# 80020094: afc00000
# 80020098: 0800805f
# 8002009c: 00000000
# 800200a0: 24020001
# 800200a4: afc20004
# 800200a8: 08008055
```

```
# 800200ac: 00000000
# 800200b0: 8fc20004
# 800200b4: 2442ffff
# 800200b8: 00021080
# 800200bc: 8fc30018
# 800200c0: 00621021
# 800200c4: 8c430000
# 800200c8: 8fc20004
# 800200cc: 00021080
# 800200d0: 8fc40018
# 800200d4: 00821021
# 800200d8: 8c420000
# 800200dc: 0043102a
# 800200e0: 10400019
# 800200e4: 00000000
# 800200e8: 8fc20004
# 800200ec: 2442ffff
# 800200f0: 00021080
# 800200f4: 8fc30018
# 800200f8: 00621021
# 800200fc: 8c420000
# 80020100: afc20008
# 80020104: 8fc20004
# 80020108: 2442ffff
# 8002010c: 00021080
# 80020110: 8fc30018
# 80020114: 00621021
# 80020118: 8fc30004
# 8002011c: 00031880
# 80020120: 8fc40018
# 80020124: 00831821
# 80020128: 8c630000
# 8002012c: ac430000
# 80020130: 8fc20004
# 80020134: 00021080
# 80020138: 8fc30018
# 8002013c: 00621021
# 80020140: 8fc30008
# 80020144: ac430000
# 80020148: 8fc20004
# 8002014c: 24420001
# 80020150: afc20004
# 80020154: 8fc3001c
# 80020158: 8fc20000
# 8002015c: 00621823
# 80020160: 8fc20004
# 80020164: 0043102a
# 80020168: 1440ffd1
# 8002016c: 00000000
# 80020170: 8fc20000
# 80020174: 24420001
```

```
# 80020178: afc20000
# 8002017c: 8fc30000
# 80020180: 8fc2001c
# 80020184: 0062102a
# 80020188: 1440ffc5
# 8002018c: 00000000
# 80020190: 03c0e821
# 80020194: 8fbe0014
# 80020198: 27bd0018
# 8002019c: 03e00008
# 800201a0: 00000000
```

Listing 2: Simple Add Output

```
# opened file srec_files/SimpleAdd.srec
#
# Initializing memory
# read in file
#
# 80020000: 27bdffe8
# 80020004: afbe0014
# 80020008: 03a0f021
# 8002000c: 24020003
# 80020010: afc20000
# 80020014: 24020002
# 80020018: afc20004
# 8002001c: afc00008
# 80020020: 8fc30000
# 80020024: 8fc20004
# 80020028: 00621021
# 8002002c: afc20008
# 80020030: 8fc20008
# 80020034: 03c0e821
# 80020038: 8fbe0014
# 8002003c: 27bd0018
# 80020040: 03e00008
# 80020044: 00000000
```

Listing 3: Simple If Output

```
# opened file srec_files/SimpleIf.srec
#
# Initializing memory
# read in file
#
# 80020000: 27bdffe8
# 80020004: afbe0014
# 80020008: 03a0f021
# 8002000c: 24020003
# 80020010: afc20000
# 80020014: 24020002
# 80020018: afc20004
```

```
# 8002001c: afc00008
# 80020020: 8fc20000
# 80020024: 28420005
# 80020028: 10400007
# 8002002c: 00000000
# 80020030: 8fc30000
# 80020034: 8fc20004
# 80020038: 00621021
# 8002003c: afc20000
# 80020040: 08008016
# 80020044: 00000000
# 80020048: 8fc30000
# 8002004c: 8fc20004
# 80020050: 00621023
# 80020054: afc20000
# 80020058: 8fc30000
# 8002005c: 8fc20004
# 80020060: 00621021
# 80020064: afc20008
# 80020068: 8fc20008
# 8002006c: 03c0e821
# 80020070: 8fbe0014
# 80020074: 27bd0018
# 80020078: 03e00008
# 8002007c: 00000000
```

## 2 Source Code

Listing 4: Fetch Module

```
//
// fetch.v
//
//

module fetch (
    clock,
    address,
    insn,
    insn_decode,
    pc,
    wren,
    stall
);

    input wire clock; // Clock signal for module. Make module sensitive to the positive edge
        of the clock
    input wire[0:31] insn; // this receives the instruction word from the main memory
        associated with the supplied PC
```

```
    input wire stall; // When asserted, the fetch effectively does a NOP. Data supplied to
        any of the outputs do not change, and the PC is not incremented by 4.

    output reg[0:31] address; // Output address supplied to the address input of the main
        memory. This signal transmits the PC for the instruction we are going to fetch
    output reg[0:31] insn_decode; // This transmits the instruction received from the main
        memory from insn
    output reg[0:31] pc; // transmits PC to the decode stage
    output reg wren; // indicates whether the fetch stage is performing a read or write to
        the main memory. Should always be asserted to a read for the fetch stage.

    reg[2:0] stage;

    initial
    begin
        $display("Initializing Fetch module");
        pc = 32'h8002_0000;
        wren = 1'b0;
    end

    always @(posedge clock)
    begin
        if (stall) begin
            address = pc;
            insn_decode = insn;
            pc = pc +4;
        end
    end

endmodule
```

Listing 5: Fetch Module Test Bench

```
//
// fetch_tb.v
//
// Fetch test bench
//

module fetch_tb;
    reg clock;

    wire[0:31] address;
    wire wren;
    wire[0:31] data_in;
    wire[0:31] data_out;

    wire[0:31] fetch_address;
    wire fetch_wren;
    wire[0:31] fetch_data_in;
    wire[0:31] fetch_data_out;
```

```
wire[0:31] fetch_insn_decode;
wire[0:31] fetch_pc;
reg fetch_stall;

wire[0:31] srec_address;
wire srec_wren;
wire[0:31] srec_data_in;
wire[0:31] srec_data_out;

wire srec_done;

reg[0:31] tb_address;
reg tb_wren;
reg[0:31] tb_data_in;
wire[0:31] tb_data_out;


wire[0:31] bytes_read;
integer byte_count;
integer read_word;
reg[0:31] fetch_word;

reg instruction_valid;

mem_controller mcu(
    .clock (clock),
    .address (address),
    .wren (wren),
    .data_in (data_in),
    .data_out (data_out)
);

fetch DUT(
    .clock (clock),
    .address (fetch_address),
    .insn (fetch_data_in),
    .insn_decode (fetch_data_out),
    .pc (fetch_pc),
    .wren (fetch_wren),
    .stall (fetch_stall)
);

srec_parser #("srec_files/SimpleIf.srec") U0(
    .clock (clock),
    .mem_address (srec_address),
    .mem_wren (srec_wren),
    .mem_data_in (srec_data_in),
    .mem_data_out (srec_data_out),
    .done (srec_done),
    .bytes_read(bytes_read)
);
```

```
decode U1(
    .clock (clock),
    .insn (fetch_word),
    .insn_valid (instruction_valid)
);

assign address = srec_done ? (fetch_stall ? tb_address : fetch_address) : srec_address;
assign wren = srec_done ? (fetch_stall ? tb_wren : fetch_wren) : srec_wren;
assign tb_data_out = data_out;
assign fetch_data_in = data_out;
assign data_in = srec_done ? (fetch_stall ? tb_data_in : fetch_data_in) : srec_data_in;

// Specify when to stop the simulation
event terminate_sim;
initial begin
    @ (terminate_sim);
    #10 $finish;
end

initial begin
    clock = 1;
    fetch_stall = 1;
    instruction_valid = 1'b0;
end

always begin
    #5 clock = !clock;
end

initial begin
    $dumpfile("fetch_tb.vcd");
    $dumpvars;
end

initial begin
    @(posedge srec_done);
    @(posedge clock);
    byte_count = bytes_read;
    tb_address = 32'h8002_0000;
    tb_wren = 1'b0;
    while (byte_count  0) begin
        @(posedge clock);
            instruction_valid = 1'b0;
            read_word = tb_data_out;

        fetch_stall = 0;
        @(posedge clock);
        @(posedge clock)
        fetch_stall = 1;
        fetch_word = fetch_data_out;
```

```
            if (tb_address != fetch_address
                && read_word != fetch_word
                && fetch_pc == byte_count) begin
                $display("2: %8X :: %8X :: %8X :: %8X :: %h", tb_address,
                              fetch_address, read_word, fetch_word, data_out);

            end
            //$display("a: %b, ", fetch_word[31:26] );
           $display("PC: %X, Instruction: %b", fetch_address, fetch_word);
           instruction_valid = 1'b1;


                tb_address = tb_address + 4;
                byte_count = byte_count - 4; // 27 = 0010 011
        end // while (byte_count  0)
        //
        // signal to end the simulation
        - terminate_sim;
    end

endmodule
```

Listing 6: Decode Module

```
//
// decode.v
//
//

module decode (
    clock,
    insn,
    insn_valid,
    pc
);

    input wire[0:31] insn;
    input wire [0:31] pc;
    input wire clock;
    input wire insn_valid;

    reg[0:1] insn_type;

    wire[0:5] opcode;
    wire[4:0] rs;
    wire[4:0] rt;
    wire[4:0] rd;
    wire[4:0] shift_amount;
    wire[5:0] funct;
    wire[15:0] immediate;
    wire[25:0] j_address;
    wire[4:0] base;
```

```
wire[15:0] offset;

// Instruction types
parameter I_TYPE = 0;
parameter J_TYPE = 1;
parameter R_TYPE = 2;
parameter INVALID_INS = 3;

assign opcode = insn[0:5];
assign rs = insn[6:10];
assign base = insn[6:10];
assign rt = insn[11:15];
assign rd = insn[16:20];
assign shift_amount = insn[20:25];
assign funct = insn[26:31];
assign immediate = insn[16:31];
assign offset = insn[16:31];
assign j_address = insn[6:31];

always @(posedge clock)
begin
        if(insn_valid) begin
        //opcode = insn[26:31];
        // DEBUG
        $display("Got opcode (decode.v) %b", opcode);
        case(opcode)
                // R-TYPE
            6'b000000: begin
                    case(funct)
                    6'b100000: //ADD
                          $display("ADD rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100001: //ADDU
                          $display("ADDU rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100010: //SUB
                          $display("SUB rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100011: //SUBU
                          $display("SUBU rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b101010: //SLT
                          $display("SLT rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b101011: //SLTU
                          $display("SLTU rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b000000: //SLL
                          $display("SLL/NOP(sa = 0) sa: %d rt: %d rd: %d", shift_amount, rt,
                                rd);
                        6'b000010: //SRL
                          $display("SRL sa: %d rt: %d rd: %d", shift_amount, rt, rd);
                        6'b000011: //SRA
                          $display("SRA sa: %d rt: %d rd: %d", shift_amount, rt, rd);
                        6'b100100: //AND
                          $display("AND rs: %d rt: %d rd: %d", rs, rt, rd);
                        6'b100101: //OR
```

```
            $display("OR rs: %d rt: %d rd: %d", rs, rt, rd);
          6'b100110: //XOR
            $display("XOR rs: %d rt: %d rd: %d", rs, rt, rd);
          6'b100111: //NOR
            $display("NOR rs: %d rt: %d rd: %d", rs, rt, rd);

          default:
            $display("unimplemented ADD type intruction");
      endcase // case (funct)
    end // case: 6'b000000

    //I-TYPE
    6'b001001:  //ADDIU
      $display("ADDIU rs: %d rt: %d immediate: %d", rs, rt, immediate);
    6'b001010:  //SLTI
      $display("SLTI rs: %d rt: %d immediate: %d", rs, rt, immediate);
    6'b100011:  //LW
      $display("LW base: %d rt: %d offset: %d", base, rt, offset);
    6'b101011:  //SW
      $display("SW base: %d rt: %d offset: %d", base, rt, offset);
    6'b001111:  //LUI
      $display("LUI rt: %d immediate: %d", rt, immediate);
    6'b001101:  //ORI
      $display("ORI rs: %d rt: %d immediate: %d", rs, rt, immediate);

    //J-TYPE
    6'b000010:  //J
      $display("J instr_index: %d", j_address);
    6'b000100:  //BEQ
      $display("BEQ rs: %d rt: %d offset: %d", rs, rt, offset);
    6'b000101:  //BNE
      $display("BNE rs: %d rt: %d offset: %d", rs, rt, offset);
    6'b000111:  //BGTZ
      $display("BGTZ rs: %d offset: %d", rs, offset);
    6'b000110:  //BLEZ
      $display("BLEZ rs: %d offset: %d", rs, offset);

    6'b000001: //REGIMM instructions
    begin
          case(rt)
            5'b00000: //BLTZ
              $display("BLTZ rs: %d offset: %d", rs, offset);
            5'b00001: //BGEZ
              $display("BGEZ rs: %d offset: %d", rs, offset);
            default:
              $display("REGIMM not implemented");
          endcase // case (rt)
    end
default:
     $display("unimplemented/incorrect intruction");
//insn_type = INVALID_INS;
```

```
            endcase // case (insn[26:31])

            end // if (insn_valid = 1'b1)

    end // always @ (posedge clock)

endmodule
```

Listing 7: Decode Module Test Bench

```
//
// decode_tb.v
//
// Decode test bench
//

module decode_tb;
    reg clock;
    reg i_valid;
    reg[0:31] insn;
    reg [0:31] pc;

    wire[0:1] insn_type;
    wire[5:0] opcode;
    wire[4:0] rs;
    wire[4:0] rt;
    wire[4:0] rd;
    wire[4:0] shift_amount;
    wire[5:0] funct;
    wire[15:0] immediate;
    wire[25:0] j_address;

    decode DUT(
        .clock (clock),
        .insn (insn),
            .insn_valid (i_valid),
        .pc (pc)
    );

    // initialization of the test bench
    initial begin
        clock = 1;
    end

    // clock signal
    always begin
        #5 clock = !clock;
    end

    // Specify when to stop the simulation
    event terminate_sim;
```

```
initial begin
    @ (terminate_sim);
    #10 $finish;
end

// TEST CASES BEGIN HERE
initial begin

    @ (posedge clock);
    i_valid = 1'b1;

    //ADD
    //insn = 32'b10000000000000100001100001000000;
    // | op || rs|| rt|| rd|| sh|| fu |
    insn = 32'b00000000101110000011001010100000;
    @ (posedge clock);

    //ADDU
    //insn = 32'b10000100000000100001100001000000;
    // | op || rs|| rt|| rd|| sh|| fu |
    insn = 32'b00000000001000110001000000100001;
    @ (posedge clock);

    //SUB
    insn = 32'b10001000000000100001100001000000;
    @ (posedge clock);

    //SUBU
    insn = 32'b10001100000000100001100001000000;
    @ (posedge clock);

    //SLT
    insn = 32'b10101000000000100001100001000000;
    @ (posedge clock);

    //SLTU
    insn = 32'b10101100000000100001100001000000;
    @ (posedge clock);

     //SLL
    insn = 32'b00000000000000100001100001000000;
    @ (posedge clock);

     //SRL
    insn = 32'b00001000000000100001100001000000;
    @ (posedge clock);

    //SRA
    insn = 32'b00001100000000100001100001000000;
    @ (posedge clock);
```

```
 //AND
insn = 32'b10010000000000100001100001000000;
@ (posedge clock);

//OR
insn = 32'b10010100000000100001100001000000;
@ (posedge clock);

 //XOR
insn = 32'b10011000000000100001100001000000;
@ (posedge clock);

//NOR
insn = 32'b10011100000000100001100001000000;
@ (posedge clock);

//ADDIU
        // 001001 11101111011111111111101000
insn = 32'b00100111101111011111111111101000;

//insn = 32'b00100100000000100001100001001001;
@ (posedge clock);

//SLTI
insn = 32'b00100100000000100001100001001010;
@ (posedge clock);

//LW
insn = 32'b00100100000000100001100001100011;
@ (posedge clock);

//SW
insn = 32'b00100100000000100001100001101011;
@ (posedge clock);

//LUI
insn = 32'b00100100000000100001100001001111;
@ (posedge clock);

//ORI
insn = 32'b00100100000000100001100001001101;
@ (posedge clock);

//J
insn = 32'b00100100000000100001100001000010;
@ (posedge clock);

//BEQ
insn = 32'b00100100000000100001100001000100;
@ (posedge clock);
```

```
        //BNE
        insn = 32'b00100100000000100001100001000101;
        @ (posedge clock);

        //BGTZ
        insn = 32'b00100100000000100001100001000111;
        @ (posedge clock);

        //BLEZ
        insn = 32'b00100100000000100001100001000110;
        @ (posedge clock);

        //BLTZ
        insn = 32'b00100100000000100000000001000001;
        @ (posedge clock);

        //BGEZ
        insn = 32'b00100100000000100000100001000001;
        @ (posedge clock);

        //
        // signal to end the simulation
        - terminate_sim;

    end

endmodule // decode_tb
```