

UNIVERSITY OF WATERLOO

ECE 455

---

## TASK 3

### REGISTER FILE AND EXECUTE STAGE

---

Stephan Van Den Heuval  
Ravil Baizhiyenov  
David Janssen  
Ben Ridder

sjvanden  
rbaizhiy  
dajjanss  
brridder

July 12, 2012

# 1 Output

Bubble sort is not working correctly. There is an issue with the sorting part of it most likely tied to the SLL operation. We ran out of time to debug this issue as the assembly is difficult to work through line by line.

Listing 1: Bubble Sort Output

```
# opened file src_files/BubbleSort.srec
#
# Initializing memory
# Initializing Fetch module
# Initializing memory
# read in file
#
# initial register file state
# Stack at 0000200, data = 00000000
# Stack at 00001fc, data = 00000000
# Stack at 00001f8, data = 00000000
# Stack at 00001f4, data = 00000000
# Stack at 00001f0, data = 00000000
# Stack at 00001ec, data = 00000000
# Stack at 00001e8, data = 00000000
# Stack at 00001e4, data = 00000000
# Stack at 00001e0, data = 00000000
# Stack at 00001dc, data = 00000000
# Stack at 00001d8, data = 00000000
# Stack at 00001d4, data = 00000000
# Stack at 00001d0, data = 00000000
# Stack at 00001cc, data = 00000000
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000002
# Register 3, value: 00000003
# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
```

```

# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f
# final register file state
# Stack at 00000200, data = 00000000
# Stack at 000001fc, data = 0000001f
# Stack at 000001f8, data = 0000001e
# Stack at 000001f4, data = 00000078
# Stack at 000001f0, data = 00000063
# Stack at 000001ec, data = 0000000c
# Stack at 000001e8, data = 0000000c
# Stack at 000001e4, data = 0000000c
# Stack at 000001e0, data = 0000000c
# Stack at 000001dc, data = 0000000c
# Stack at 000001d8, data = 0000000c
# Stack at 000001d4, data = 00000000
# Stack at 000001d0, data = 00000000
# Stack at 000001cc, data = 00000008
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000000
# Register 3, value: 00000008
# Register 4, value: 800201d8
# Register 5, value: 00000008
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b

```

```
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f
```

Listing 2: Simple Add Output

```
# opened file src_files/SimpleAdd.srec
#
# Initializing memory
# Initializing Fetch module
# Initializing memory
# read in file
#
# initial register file state
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000002
# Register 3, value: 00000003
# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f
# final register file state
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000005 +++
# Register 3, value: 00000003 +++
```

```

# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f

```

Listing 3: Simple If Output

```

# opened file src_files/SimpleIf.srec
#
# Initializing memory
# Initializing Fetch module
# Initializing memory
# read in file
#
# initial register file state
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000002
# Register 3, value: 00000003
# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c

```

```
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f
# final register file state
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000007 +++
# Register 3, value: 00000005 +++
# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
```

```
# Register 31, value: 0000001f
```

Listing 4: Sum Array Output

```
# opened file srec_files/SumArray.srec
#
# Initializing memory
# Initializing Fetch module
# Initializing memory
# read in file
#
# initial register file state
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 00000002
# Register 3, value: 00000003
# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f
# final register file state
# Register 0, value: 00000000
# Register 1, value: 00000001
# Register 2, value: 0000002d +++
# Register 3, value: 00000024 +++
# Register 4, value: 00000004
# Register 5, value: 00000005
# Register 6, value: 00000006
```

```
# Register 7, value: 00000007
# Register 8, value: 00000008
# Register 9, value: 00000009
# Register 10, value: 0000000a
# Register 11, value: 0000000b
# Register 12, value: 0000000c
# Register 13, value: 0000000d
# Register 14, value: 0000000e
# Register 15, value: 0000000f
# Register 16, value: 00000010
# Register 17, value: 00000011
# Register 18, value: 00000012
# Register 19, value: 00000013
# Register 20, value: 00000014
# Register 21, value: 00000015
# Register 22, value: 00000016
# Register 23, value: 00000017
# Register 24, value: 00000018
# Register 25, value: 00000019
# Register 26, value: 0000001a
# Register 27, value: 0000001b
# Register 28, value: 0000001c
# Register 29, value: 80020200
# Register 30, value: 0000001e
# Register 31, value: 0000001f
```

## 2 Source Code

Listing 5: Memory Stage Module

```
//
// mem_stage.v
//
// Memory Stage
//

`include "control.vh"

module mem_stage(
    clock,
    address,
    data_in,
    address_out,

    mem_data_out,
    data_out,
    control,
    control_out,

    rdIn,
```



```

    rdOut,

    print_stack
);

input wire clock;
input wire[0:31] address;
input wire[0:31] data_in;
input wire[0:'CONTROL_REG_SIZE-1] control;
input wire[0:4] rdIn;
input wire print_stack;

output reg[0:31] address_out;
output reg[0:31] data_out;
output reg[0:31] mem_data_out;
output reg[0:'CONTROL_REG_SIZE-1] control_out;
output reg[0:4] rdOut;
wire[0:31] mcu_data_out;
wire wren_mem;

assign wren_mem = control['MEM_WE'];

mem_controller mcu1(
    .clock (clock),
    .address (address),
    .wren (wren_mem),
    .data_in (data_in),
    .data_out (mcu_data_out),
    .print_stack (print_stack) // Debugging
);

always @(posedge clock)
begin
    rdOut <= rdIn;
end

always @(posedge clock)
begin
    mem_data_out = mcu_data_out;
end

always @(posedge clock)
begin
    //$display("TIME: %d, address %X, wren %b,mem_data_in %X, mem_data out : %X", $time,
        address, wren_mem,data_in, mem_data_out);
    address_out = address;
end

always @(posedge clock)

```

```

//hehehe I know this is terrible but it was a really easy way to
//implement JAL ...
begin
    if (control['LINK'] == 1) begin
        data_out <= data_in;
    end
    else begin
        data_out <= address;
    end

    //$display("TIME: %d, data_out = %X", $time, data_out);
end

always @(posedge clock)
begin
    control_out = control;
end

endmodule

```

Listing 6: Writeback Module

```

//
// writeback_stage.v
//
// Writeback Stage
//

`include "control.vh"

module writeback_stage(
    clock,
    rdIn,
    rdDataIn,
    memDataIn,
    control,
    rdOut,
    regWriteEnable,
    writeBackData
);

    input wire clock;
    input wire[0:4] rdIn;
    input wire[0:31] rdDataIn;
    input wire[0:31] memDataIn;
    input wire[0:'CONTROL_REG_SIZE-1] control;

    output reg[0:4] rdOut;
    output reg regWriteEnable;
    output reg[0:31] writeBackData;

    always @(posedge clock)

```

```

begin
    regWriteEnable <= control['REG_WE'];
    if (control['MEM_WB']) begin
        writeBackData <= memDataIn;
    end
    else begin
        writeBackData <= rdDataIn;
    end
    rdOut <= rdIn;
end
endmodule

```

Listing 7: Writeback Module Test Bench

```

//
// writeback_stage_tb.v
//
// Testbench for the writeback stage
// includes all previous stages
//

`include "alu_func.vh"

module writeback_stage_tb;
    reg clock;

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

    wire[0:31] mcu_address;
    wire mcu_wren;
    wire[0:31] mcu_data_in;
    wire[0:31] mcu_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;
    reg[0:31] fetch_pc_in;
    reg fetch_jump;
    reg[0:31] fetch_pc_in_buff_0;
    reg fetch_jump_buff_0;
    reg[0:31] fetch_pc_in_buff_1;
    reg fetch_jump_buff_1;

```

```

wire[0:31] decode_rs_data;
wire[0:31] decode_rt_data;
wire[0:4] decode_rd_in;
wire[0:31] decode_pc_out;
wire[0:31] decode_ir_out;
wire[0:31] decode_write_back_data;
wire decode_reg_write_enable;
wire[0:'CONTROL_REG_SIZE-1] decode_control;
wire[0:4] decode_rd_out;
reg decode_dump_regs;

reg[0:31] alu_rs_data_res;
reg[0:31] alu_rt_data_res;
reg[0:31] alu_insn;
wire[0:31] alu_output;
wire alu_bt;
wire[0:31] alu_rt_data_out;
wire[0:31] alu_insn_out;
reg [0:31] alu_pc_res;
wire[0:'CONTROL_REG_SIZE-1] alu_control_out;
wire[0:4] alu_rd_in;
wire[0:4] alu_rd_out;

wire[0:31] mem_stage_address;
wire[0:31] mem_stage_address_out;
wire[0:31] mem_stage_data_in;
wire[0:31] mem_stage_data_out;
wire[0:'CONTROL_REG_SIZE-1] mem_stage_control_in;
wire[0:'CONTROL_REG_SIZE-1] mem_stage_control_out;
wire[0:31] mem_stage_insn;
wire[0:31] mem_stage_insn_out;
wire[0:31] mem_stage_mem_data_out;
reg[0:'CONTROL_REG_SIZE-1] mem_stage_srec_read_control;
wire[0:4] mem_stage_rd_in;
wire[0:4] mem_stage_rd_out;
reg mem_stage_print_stack;
reg[0:7] cycle_count;

wire[0:31] bytes_read;
integer byte_count;
reg instruction_valid;
reg terminate_signal;

event terminate_sim;

srec_parser #("srec_files/BubbleSort.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)
    );

    mem_controller mcu(
        .clock (clock),
        .address (mcu_address),
        .wren (mcu_wren),
        .data_in (mcu_data_in),
        .data_out (mcu_data_out)
    );

    fetch F0(
        .clock (clock),
        .address (fetch_address),
        .insn (fetch_data_in),
        .insn_decode (fetch_data_out),
        .pc (fetch_pc),
        .wren (fetch_wren),
        .stall (fetch_stall),
        .pcIn (fetch_pc_in),
        .jump (fetch_jump)
    );

    decode U1(
        .clock (clock),
        .insn (fetch_data_out),
        .insn_valid (instruction_valid),
        .pc (fetch_address),
        .rsData (decode_rs_data),
        .rtData (decode_rt_data),
        .rdIn (decode_rd_in),
        .pcOut (decode_pc_out),
        .irOut (decode_ir_out),
        .writeBackData (decode_write_back_data),
        .regWriteEnable (decode_reg_write_enable),
        .control (decode_control),
        .rdOut (decode_rd_out),
        .dumpRegs (decode_dump_regs)
    );

    alu alu(
        .clock (clock),
        .rsData (decode_rs_data),
        .rtData (decode_rt_data),
        .control (decode_control),
        .control_out (alu_control_out),
        .outData (alu_output),
        .bt (alu_bt),

```

```

        .insn (decode_ir_out),
        .insn_out(alu_insn_out),
        .rtDataOut(alu_rt_data_out),
        .pc (decode_pc_out),
        .rdIn (decode_rd_out),
        .rdOut (alu_rd_out)
    );

    mem_stage DUT(
        .clock (clock),
        .address (mem_stage_address),
        .data_in (mem_stage_data_in),
        .address_out (mem_stage_address_out),
        .mem_data_out (mem_stage_mem_data_out),
        .data_out (mem_stage_data_out),
        .control (mem_stage_control_in),
        .control_out (mem_stage_control_out),
        .rdIn (alu_rd_out),
        .rdOut (mem_stage_rd_out),
        .print_stack (mem_stage_print_stack)
    );

    writeback_stage wbs(
        .clock (clock),
        .rdIn (mem_stage_rd_out),
        .rdDataIn (mem_stage_data_out),
        .memDataIn (mem_stage_mem_data_out),
        .control (mem_stage_control_out),
        .rdOut (decode_rd_in),
        .regWriteEnable (decode_reg_write_enable),
        .writeBackData (decode_write_back_data)
    );

    assign mcu_address = srec_done ? fetch_address : srec_address;
    assign mcu_wren = srec_done ? fetch_wren : srec_wren;
    assign mcu_data_in = srec_done ? fetch_data_in : srec_data_in;
    assign fetch_data_in = mcu_data_out;

    assign mem_stage_address = srec_done ? alu_output : srec_address;
    assign mem_stage_data_in = srec_done ? alu_rt_data_out : srec_data_in;
    assign mem_stage_control_in = srec_done ? alu_control_out : mem_stage_srec_read_control;

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

    // Initial setup
    initial begin

```

```

    clock = 1;
    fetch_stall = 1;
    instruction_valid = 1'b0;
    mem_stage_srec_read_control = 0;
    mem_stage_srec_read_control['MEM_WE] = 1;
    terminate_signal = 1'b0;
end

// Clock process
always begin
    #5 clock = !clock;
end

//wait for the last JR RA
always @ (posedge clock) begin
    if (alu_insn_out[26:31] == 'JR && alu_output == 31) begin
        terminate_signal <= 1;
    end
end

// Debug dump
initial begin
    $dumpfile("writeback_stage_tb.vcd");
    $dumpvars;
end

// Register feed process
always @ (posedge clock) begin
    alu_insn <= fetch_data_out;
    alu_pc_res <= decode_pc_out;
    alu_rs_data_res <= decode_rs_data;
    alu_rt_data_res <= decode_rt_data;
    // a little ugly... is there a better way of doing it?
    fetch_pc_in_buff_0 <= alu_output;
    fetch_jump_buff_0 <= alu_bt;
    fetch_pc_in_buff_1 <= fetch_pc_in_buff_0;
    fetch_jump_buff_1 <= fetch_jump_buff_0;
    fetch_pc_in <= fetch_pc_in_buff_1;
    fetch_jump <= fetch_jump_buff_1;
end

// Process to control the length of execution and
// ensuring that there is only one insn in the pipeline at a time
initial begin
    @ (posedge srec_done);
    decode_dump_regs = 1'b1;
    @ (posedge clock);
    decode_dump_regs = 1'b0;
    byte_count = bytes_read + 4;
    instruction_valid = 1'b0;
    fetch_stall = 0;
end

```

```

cycle_count = 0;

//byte count breaks when we have loops ...
while (terminate_signal == 0) begin
    @ (posedge clock);
    //$display("Fetch Address %X", fetch_address);
    if (fetch_address < 32'h8002_0000) begin
        instruction_valid = 1'b0;
    end else begin
        instruction_valid = 1'b1;
    end

    // stall the pipeline such that there is only one insn at a time in it
    if (cycle_count == 0) begin
        byte_count = byte_count - 4;
        fetch_stall <= 1;
    end
    if (cycle_count == 4) begin
        cycle_count <= 0;
        fetch_stall <= 0;
    end
    else begin
        cycle_count <= cycle_count + 1;
    end

    //$display("ALU address out: %X", alu_output);
end

// The decode runs one clock cycle behind the fetch
@ (posedge clock);
// allow the last alu op to run
@ (posedge clock);
@ (posedge clock);
->terminate_sim;
end // initial begin

// Debug process for outputting values
initial begin
    @ (posedge srec_done);
    @ (posedge clock);

    while (terminate_signal == 0) begin
        @ (posedge clock);
        if (cycle_count == 2) begin
            //decode_dump_regs = 1'b1;
            $display("Time: %d, Inp insn: %X, Inp PC: %X, Inp RS: %x, Inp RT: %x,
                ALU_RESULT: %x",
                $time, alu_insn_out, alu_pc_res, alu_rs_data_res, alu_rt_data_res,
                alu_output);
            $display("Memory stage data in : %X", alu_rt_data_out);
        end
    end
end

```



```

    if (cycle_count == 3) begin
        $display("Time: %d, memory location: %X, memory write enabled: %b, memory data:
            %X", $time, mem_stage_address_out, mem_stage_control_out['MEM_WE'],
            mem_stage_mem_data_out);
        mem_stage_print_stack = 1;
        decode_dump_regs = 1;
    end
    if (cycle_count == 4) begin
        mem_stage_print_stack = 0;
        decode_dump_regs = 0;
        //$display("");
    end

    // $display("WB register: %d WB data: %d", decode_rd_in, decode_write_back_data
    );
    // $display("decode rd_out: %d", decode_rd_out);
end
// allow the last decode to run
@ (posedge clock);
decode_dump_regs = 1'b1;
@ (posedge clock);
decode_dump_regs = 1'b0;
$display("Time: %d, Inp insn: %X, Inp PC: %X, Inp RS: %d, Inp RT: %d, ALU_RESULT: %d",
    $time, alu_insn_out, alu_pc_res, alu_rs_data_res, alu_rt_data_res, alu_output)
    ;

end

endmodule

```