IB Computer Design - Supervision 1

Nandor Licker `<nl364@cl.cam.ac.uk>`

Due noon before the day of supervision

Q1 Attempt 2011 Paper 5 Question 1

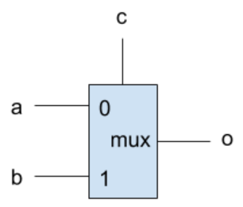
Q2 Attempt 2014 Paper 5 Question 1

Q3 Explain the function and draw the circuits which would be synthesised from the given Verilog blocks. You should use various functional modules, such as D-type flip-flops, decoders, multiplexers, adders, incrementers, etc. Draw the wires and registers as they are declared, performing no optimisations.

As an example, the following circuit:

```
module EX(input logic a, input logic b, input logic c, output logic d);
  always_comb d = c ? a : b;
endmodule
```

Is a multiplexer and synthesises to:



1.
```
module EX(input logic clk, input logic rst, output logic o);
  logic[2:0] a, b, c;
  always_comb o <= c >= 3 ? 1 : 0;
  always_ff @(posedge clk) begin
    if (rst) begin
      a <= 0; b <= 0; c <= 0;
    end else begin
      if (a == 5) begin
        a <= 0;
        if (b == 5) begin
          b <= 0;
          if (c == 5) begin
            c <= 0;
          end else c <= c + 1;
        end else b <= b + 1;
      end else a <= a + 1;
    end
  end
endmodule
```

2.
```
module EX(input logic clk, input logic rst, output logic o);
  logic[3:0] r;
  always_ff @(posedge clk) begin
    if (rst) begin
      o <= 0; r <= 0;
    end else begin
      r <= r >= 9 ? 0 : (r + 1);
      o <= r == 0 ? 1 : 0;
    end
  end
endmodule
```

3.
```
module EX(input logic clk, input logic rst, output logic[5:0] o);
  logic[1:0] a, b, c;
  always_ff @(posedge clk) begin
    if (rst) begin
      a <= 0; b <= 0; c <= 0; o <= 0;
    end else begin
      a <= a == 3 ? 0 : (a + 1);
      b <= a == 2 ? (b + 1) : b;
      c <= b == 3 ? (c + 1) : c;
      o = c == 3;
    end
  end
endmodule
```

4.
```
module EX(input logic clk, input logic rst, output logic[5:0] o);
  logic[1:0] a, b, c;
  always_ff @(posedge clk) begin
    if (rst) begin
      a <= 0; b <= 0; c <= 0; o <= 0;
    end else begin
      a = a == 3 ? 0 : (a + 1);
      b <= a == 2 ? (b + 1) : b;
      c <= b == 3 ? (c + 1) : c;
      o = c == 3;
    end
  end
endmodule
```

5.
```
module EX(input logic clk, input logic rst, output logic[5:0] o);
  logic[1:0] a, b, c;
  always_ff @(posedge clk) begin
    if (rst) begin
      a <= 0; b <= 0; c <= 0; o <= 0;
    end else begin
      a = a == 3 ? 0 : (a + 1);
      b = a == 2 ? (b + 1) : b;
      c <= b == 3 ? (c + 1) : c;
      o = c == 3;
    end
  end
endmodule
```

Q4 This exercise ties together content from various lectures and some of the upcoming supervisions. You can attempt it at any time you feel ready and send it over for marking. It will be discussed during one of the upcoming supervisions, ideally before pipelining is introduced.

Design, on paper, a simple processor with the prescribed set of features. Draw a diagram including major components and the wiring between them, including the width of various buses.

- 8 bit data bus, 16 bit address bus
- 8Kb EEPROM for program code and 64Kb RAM for data
- 8 I/O ports
- Stack-based, with a single accumulator register
- Variable-length, multi-cycle instructions
- 8-bit ALU, based on the SN74181[1]
- state machine for control

Approximate the maximum clock rate. Discuss 3 strategies for implementing the state machine.

The machine should at least be able to execute the following instructions. Describe their encoding and the control signals during each cycle of their execution.

- `call` *addr16*: Pushes the return address and jumps to the target.
- `ret` Pops the return address and jumps to it.
- `load` *addr16*: Loads a value from a direct 16-bit address.
- `add`: Adds a value popped from the stack to the accumulator.
- `shl`: Shifts the accumulator to the left by 1 bit.

Provide syntax and encoding of a reasonable and useful set of instructions this machine should execute:

- control flow: conditional and unconditional jumps
- memory access: absolute and register-indirect loads
- arithmetic
- I/O

The previously designed machine has program code burnt into EEPROM. This is fine for microcontrollers, but more generic machines should be able to load instructions into memory and execute them. Briefly answer the following questions:

a How would you modify the machine to use the RAM for both instruction and data storage?

b Show how you would wire an 8Kb EEPROM to store a BIOS for boot.

c How would the BIOS load an operating system into memory?

d How would you reclaim the 8Kb of address space after the BIOS is loaded?

[1] Refer to Table 2 (Active-High Data) of `http://www.ti.com/lit/ds/symlink/sn54ls181.pdf`