COMP 22111

Two hours

UNIVERSITY OF MANCHESTER
SCHOOL OF COMPUTER SCIENCE

Processor Microarchitecture

Date: Wednesday 16th January 2019

Time: 09:45 - 11:45

Please answer all FOUR Questions.

Use a SEPARATE answerbook for EACH Question.

This is a CLOSED book examination

The use of electronic calculators is permitted provided they are not programmable and do not store text

[PTO]

1.

a) Figure Q1.1 illustrates the datapath for the MU0 processor where a student has attempted to shade the path usage for a) the fetch phase of an ADD instruction, and b) the execute phase of a STA instruction. However, in both cases a number of mistakes have been made. Identify the errors made by the student. (2 marks)

Note: to identify the errors, simply state the path in error, for example "the path from component f to component a should not be shaded" or "the path from data_in to component d should be shaded".

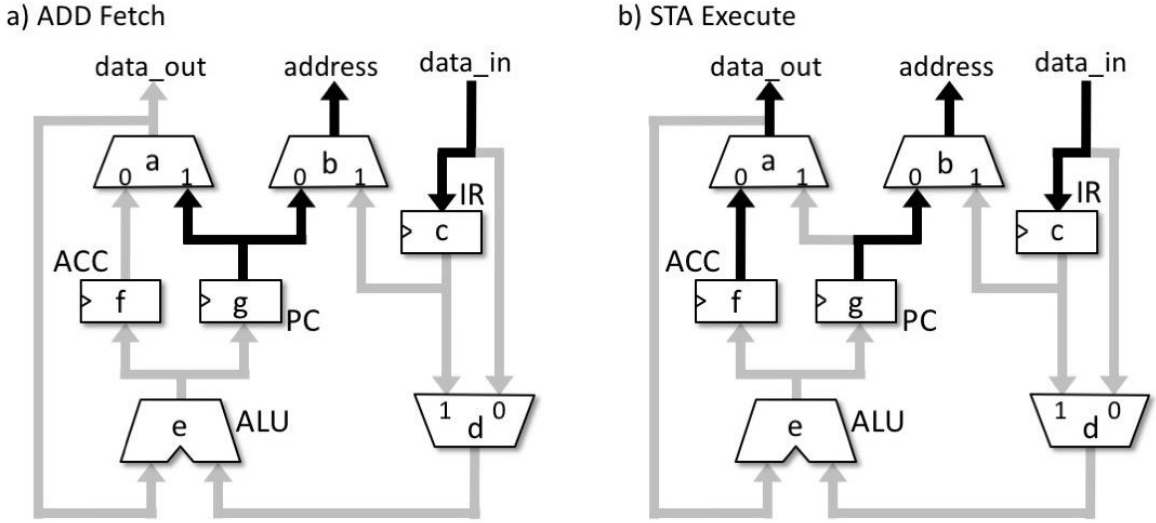

Figure Q1.1

b) The Stump processor has a limited instruction set. What Stump design feature enables additional instructions to be realised? Explain how this feature can be used to implement a compare instruction in Stump. (2 marks)

c) In a pipelined design, explain what a data hazard is. What solution can be applied to prevent such a hazard from occurring? (2 marks)

d) The processor overflow flag, V, gives an indication of when the result of an arithmetic operation produces a result that is outside the range of numbers that can be represented. Discuss for the cases of

i. addition, and (1 mark)
ii. subtraction (1 mark)

what conditions of the input operands will result in an overflow condition <u>never</u> being observed.

e) A RISC processor design has 5 stages in the datapath RTL implementation. Two designs, labelled a and b, have been produced, based on this processor design, that exhibit different latencies over each of the stages in the datapath, as shown in Figure Q1.2.

i. In the case of a non-pipelined implementation, which processor design, a or b, has the shortest latency when executing an instruction (assuming all stages are used)? What is the latency for this design? (1 mark)

ii. In the case of a pipelined implementation, which processor design, a or b, has the shortest cycle time? What is the cycle time for this design? You can assume that the register added at each stage introduces an additional delay of 20ps per stage? (1 mark)

Design	Fetch	Decode	Execute	Memory	Writeback
a	160ps	190ps	180ps	120ps	120ps
b	150ps	185ps	175ps	125ps	140ps

Figure Q1.2

[PTO]

2. The Stump architecture has no instructions to invoke a subroutine (like the BL instruction in ARM).

   a) The following (not necessarily useful) code fragment is provided in ARM assembly:

```
loop:      BL   my_mul2      ; call subroutine
           B    loop
           ...

my_mul2:   ADD  R1, R1, R1   ; R1=2xR1
           MOV  PC, LR       ; return from subr.
```

   Rewrite this code fragment in Stump assembly. You do not need to be worried that ARM uses 32-bit registers or that registers may have to be saved on stack. However, you should use a subroutine. (4 marks)

   b) You are asked to add an instruction similar to BL to the Stump ISA. Using the datapath microarchitecture given in Figure Q2.1, describe a sequence of internal operations which could perform this. You can assume that R6 can be used as the link register (LR). (5 marks)

   c) What is the difference in latency between calling a subroutine without (as considered in part a)) and with BL instruction? Explain your answer. (1 mark)

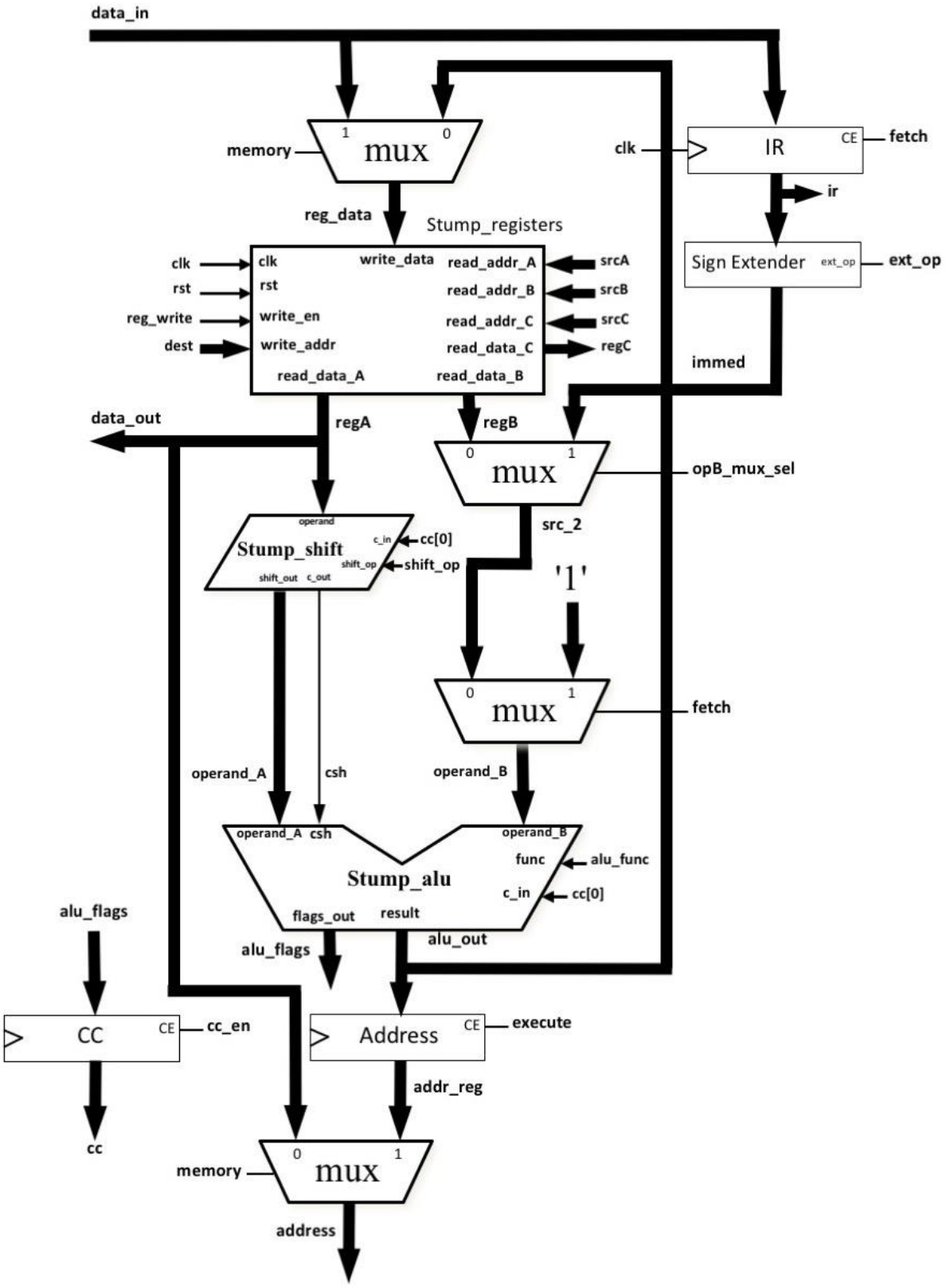

Figure Q2.1

[PTO]

3. Figure Q3.1 shows the block diagram of a controller module designed to be used as part of a vending machine design.

As the user enters coins into the vending machine the controller calculates a running total of the amount entered and raises an output signal when enough money has been entered to cover the cost of the item being dispensed. The controller then identifies the amount of change that needs returning to the user.

The controller has the following inputs:

- coin_value – an 8-bit value representing the monetary value of the coin entered by the user
- item_cost – an 8-bit value representing the cost of the item selected by the user
- coin_in – a signal that goes high when a coin has been entered
- rst – an external reset
- clk – an external clock

along with the following outputs:

- dispense – a signal that goes high when the amount of money entered covers the cost of the item being purchased
- change – an 8-bit value representing the total amount of change to be returned to the user.

The controller design consists of a finite state machine, FSM, to control the operation of the controller, and a datapath to keep a running total of the value of the coins entered by the user.

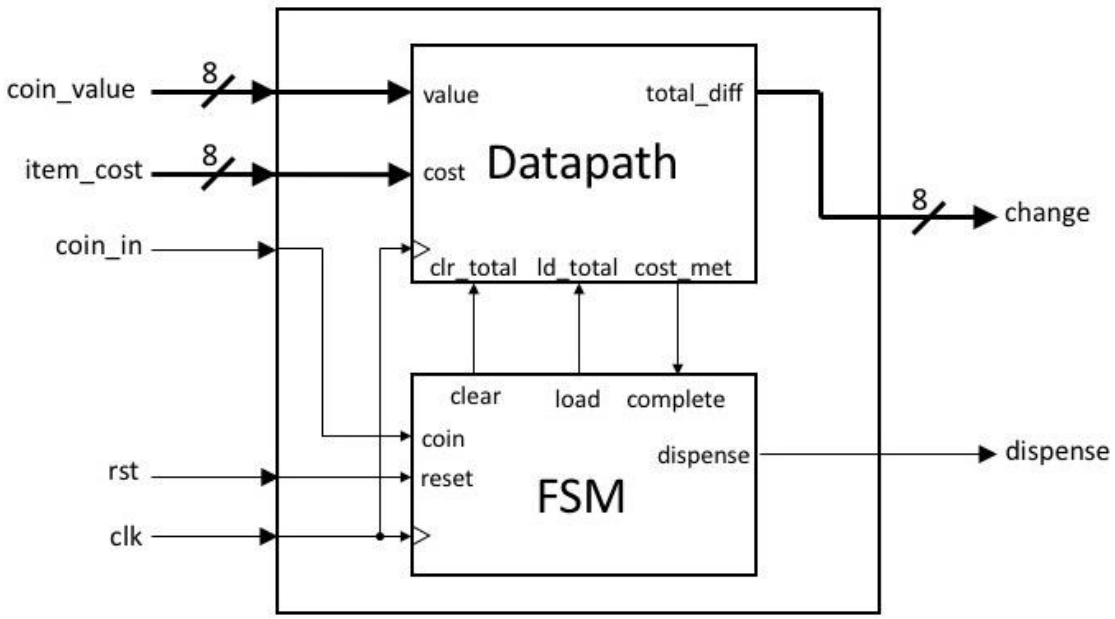

Figure Q3.1

A datapath design for the system has been produced, as shown in Figure Q3.2, which consists of three functional elements: a register, **total**, to hold the running total of the coins entered, an 8-bit adder, **adder**, to add the value of the latest coin entered to the running total, and a comparator, **comp**, that compares the running total with the cost of the item being purchased.

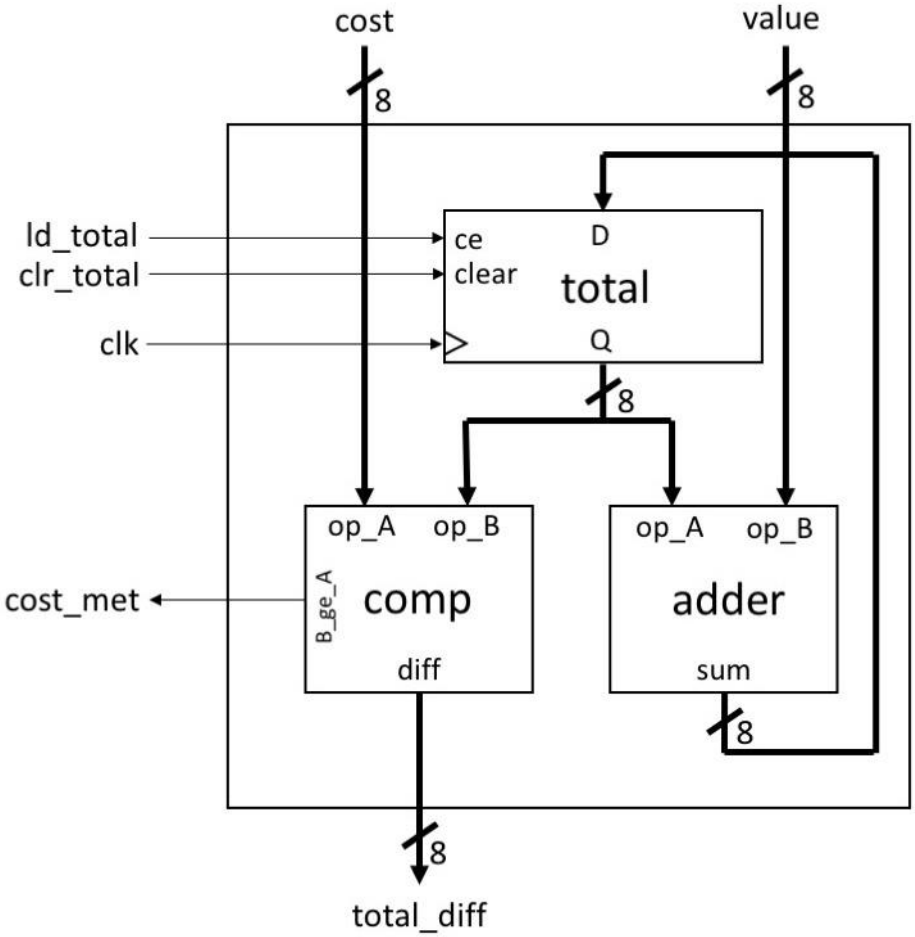

Figure Q3.2

```
module register_8bit(input [7:0]    D,
                     input          clock, clear, ce,
                     output [7:0]   Q);

module adder_8bit   (input [7:0]    op_A, op_B,
                     output [7:0]   sum);

module comp_8bit    (input [7:0]    op_A, op_B,
                     output         B_ge_A,
                     output [7:0]   diff);
```

Figure Q3.3

[PTO]

a) The modules for the three functional components in the datapath have been designed elsewhere, with the module headers given in Figure Q3.3.

A designer has attempted to produce a structural Verilog representation of the datapath as given in Figure Q3.4.

However, the module design for the datapath contains a number of errors. List the errors in the design, clearly indicating what the error is and how it can be corrected. (5 marks)

```
module vend_datapath(input [8:0]    cost, value
                     input          clk, rst,
                     input          clr_total, ld_total,
                     output [8:0]   total_diff,
                     output         cost_met);


// internal signal declaration

reg [7:0]  running_total, calc_sum;

// instantiate register

register_8bit total ( .D(calc_sum),
                      .clear(clt_total),
                      .ce(ld_total),
                      .out(running_total));

// instantiate adder

adder_8bit adder ( .op_A(running_total),
                   .op_B(value),
                   .clock(clk),
                   .sum(total_sum));

// instantiate comparator

comp_8bit ( .op_A(running_total),
            .op_B(cost),
            .B_ge_A(cost_met),
            .diff(total_diff));

// end of datapath module
endmodule
```

Figure Q3.4

b) Figure Q3.5 illustrates the state transition diagram for the vending machine controller FSM, which has been designed as a Mealy machine.

Assuming the state transition diagram is functionally correct, what can you assume about the input signals **complete** and **coin**? (1 mark)

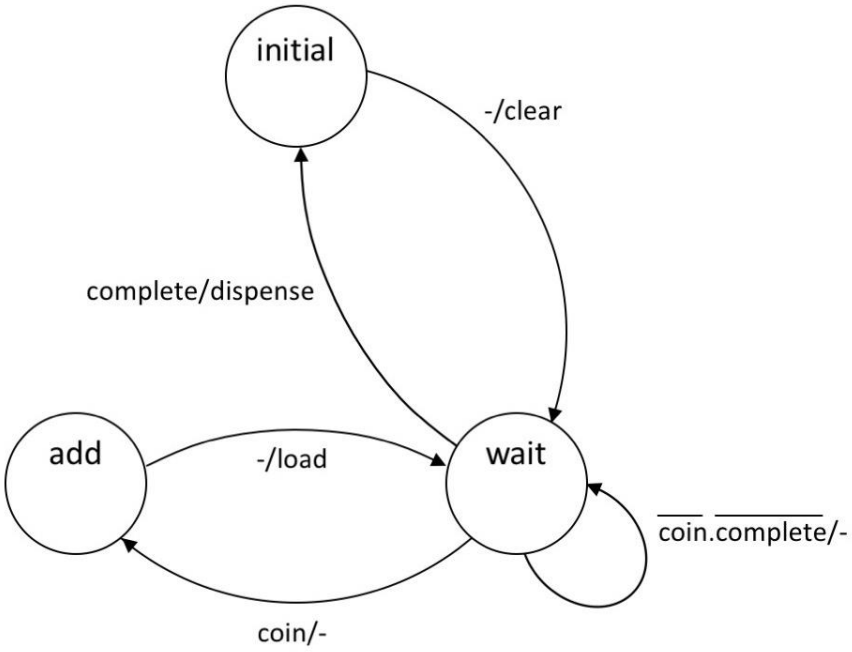

Figure Q3.5

c) Identify suitable state vectors for each state in the state transition diagram in Figure Q3.5 and complete a state transition table for the controller. (7 marks)

d) Produce a Verilog implementation of the FSM using the module header shown in Figure Q3.6. The reset action is assumed to be asynchronous and takes you back to the initial state. Provide just the missing code internal to the module in your answer. (7 marks)

```
module vend_FSM(input        clock, reset,
                input        coin, complete,
                output reg   clear, load, dispense);

// put your code here, including any definitions

endmodule
```

Figure Q3.6

[PTO]

4. Figure Q4.1 shows a generic FIR filter that is widely used for signal processing applications. It is not important to understand the theory behind the filter for this question, as we want to look into the characteristics when implementing the filter on different compute substrates (CPU, DSP, SIMD and FPGA).

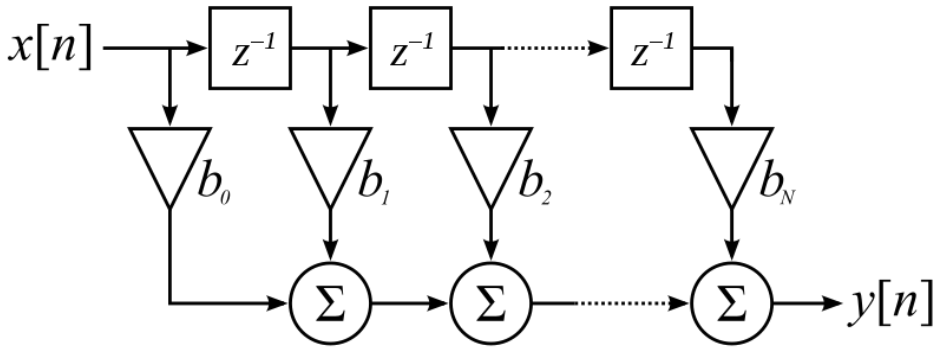

Figure Q4.1

The operation of the filter is described as follows: *x[n]* is the input stream (such as data samples from a microphone); the samples get delayed in the z^{-1} blocks, then weighted by the coefficients b_0 ... b_N and summed together to produce the output stream *y[n]* (to drive a bass speaker).

You may assume for this question that we have to implement a filter with 10 taps (having the coefficients b_0 ... b_9 and 9 delay elements).

a) Assuming that you implement the filter on a RISC processor architecture, such as ARM, discuss how many CPU cycles it roughly takes before the next sample can be processed. Make sensible assumptions and justify your estimates. (7 marks)

b) Now, assuming the use of a DSP architecture for the filter implementation instead of the ARM, discuss how many DSP cycles it takes before the next sample can be processed. You can answer this question by highlighting what would be different over the ARM implementation when using a DSP. Make sensible assumptions and justify your estimates. (3 marks)

c) Now, assuming the use of a VLIW processor (as introduced in the lectures) for the filter implementation instead of the ARM, discuss how many VLIW CPU cycles it takes before the next sample can be processed. You can answer this question by highlighting what would be different over the ARM when using a VLIW CPU. Make sensible assumptions and justify your estimates. (3 marks)

d) Now, assuming the use of a SIMD unit (such as the ARM NEON) for the filter implementation instead of the ARM, discuss how many CPU/SIMD cycles it takes before the next sample can be processed. You can answer this question by highlighting what would be different over the ARM when using a SIMD unit. Make sensible assumptions and justify your estimates. (3 marks)

e) Finally, assuming the use of an FPGA for the filter implementation instead of the ARM, discuss how many cycles it takes before the next sample can be processed. Explain your answer and illustrate, using a sketch, how the filter could be implemented using DSP (multiply-add) blocks that are available on literally all modern FPGAs. You don't have to draw the entire 10 taps, just provide enough detail to make it clear what the circuit will look like. (4 marks)

END OF EXAMINATION