UG Exam Performance Feedback
Second Year
2018/2019 Semester 1

COMP22111 Processor Microarchitecture

Paul Nutter
Dirk Koch

Comments Please see the attached report.

COMP22111 Exam Feedback 2018/19

The following is the feedback for the 208/19 academic year COMP22111 exam.

Q1 – PWN

This question was, on the whole, answered well. The average mark being 64%. The nature of the questions meant they tested a breadth of the material covered in the first half of the course unit.

Each of the questions is worth 2 marks.

a)

The aim of this question was to identify the errors in the path usage diagrams presented, for two different operations: a) covered an ADD fetch (the ADD doesn't mean anything in this respect as fetch is common to all instructions) and b) covered an STA execute operation.

The questions specifically asked for the errors to be highlighted by identifying paths between components, as described in the question; it is surprising how many students did not do this and made comments such as, the PC needs updating but isn't.

a) ADD fetch – there are 2 errors, each ½ mark:

- The path from component a to component e should be shaded
- The path from component e to component g should be shaded.

Most answers were correct here, so no general issues.

b) STA execute – there are 3 errors, a minimum of two must be identified for ½ mark, all 3 errors gets 1 mark:

- The path from data_in to component c should not be shaded
- The path from component g to component b should not be shaded
- The path from component c to component b should be shaded.

A common mistake here was not recognising that the PC needs updating, so the paths that take the contents of the PC through the ALU and back to the PC (listed above) need shading. There were a number of answers who suggested shading paths for a LDA instruction, i.e. c -> d, d -> e, e -> g. Be careful that you pick the right instruction!

b)

This is a simple question that any student who has looked at past papers will have seen before (although maybe asked in a different way).

Need to identify that R0 is hardwired to zero (or always zero) – 1 mark

Explain how this feature allows a compare instruction to be implemented – I expect a Stump instruction here:
SUBS R0, R1, R2

Note: destination being R0 and 'S' being set to update flags – 1 mark.

There were a number of incorrect answers:

- Talking about holes in the instruction set (would require assembler update)
- Just discussing status flags – not unique to Stump
- Stump shifting operation – unique to Stump (in some respects) but doesn't help when implementing a CMP instruction.

c)

This question asked for a description of data hazards and a discussion of possible solutions.

A data hazard is when there is a data dependency in the pipeline where one instruction sets the contents of a register that is used by a following instruction, but the update doesn't happen in time – 1 mark for a sensible description with example.

Solutions: a couple, such as register forwarding, stalling the pipeline with NOPS.

The main issue here was confusing data hazards with the other pipeline hazards, or not being clear in the discussion, not identifying an example etc.

d)

The question required you to identify what conditions of the two operands will result in an overview never being observed:

i. In the case of addition, the operands must be of different signs – 1 mark
ii. In the case of subtraction, the operands must be of the same sign – 1 mark

(assuming that the numbers were in range to begin with).

On the whole, the question was answered well. There were some issues with students clearly misunderstanding the question, or talking about adding small values to very large values, or something similar. The answer should focus on the signs of the operands.

e)

A relatively straightforward question that simply required values to be added to identify the smallest value.

i. Latency is the sum of the delays, so add them all up for the two processor designs and you find design 'a' has the shortest latency of 770ps – ½ mark for identifying the correct design and ½ mark for giving the value.
ii. Cycle time is the maximum delay time for the 5 stages plus the 20ps register delay. So all that is required is to identify for each processor design the stage with the longest delay and add 20ps. Processor 'b' has the shortest cycle time of 205ps – ½ mark for identifying the correct design and ½ mark for giving the value.

There weren't many issues with this question. Some marks were lost for not actually stating the design, or through calculating incorrect values.

Q2 – DK

This question was answered very well and 1/3 of the students got 9 or more out of the possible 10 marks.

The question required a little thinking on how Stump works, but was designed to align with the assembly programming done during the labs.

Q3 – PWN

This question focused on sequential systems design, in particular the design of a system (a vending machine) comprising of a datapath and control unit. (The clue that such a question would be asked was that I produced a large number of designs and put them in Blackboard!)

The actual question was long winded, but most of this just provides background to the design; it isn't actually needed, but is helpful to know. Overall, I was disappointed with the answers considering the amount of resources made available on Blackboard and in the lecture notes.

The average mark for the question was 54%.

a)

This question required you to identify the mistakes in a structural Verilog datapath design for the vending machine. There were 12 errors, each worth ½ mark up to a maximum mark of 5.

Errors were:

- Input buses defined as 9 bits, should be 8 bits
- Missing comma after value
- No rst
- Output bus defined as 9 bits, should be 8 bits
- Internal variables, running_total and calc_sum, should be defined as wire, not reg
- The clock input is missing in the register instantiation

- Typo – clt_total in register instantiation should be clr_total
- Pin name out should be Q in the register instantiation
- The clock should be removed from the adder instantiation
- The sum pin in the adder instantiation should be connected to calc_sum (total_sum is not used)
- The comparator instantiation is missing a name for the module
- The input pins op_A and op_B in the comparator instantiation are connected to the wrong signals – they should be swapped round.

No general problems apart from few recognised enough errors to get full marks. Just because 5 marks are available doesn't mean there are just 5 errors!

b)

Straightfoward answer – the two inputs coin and complete, must be mutually exclusive, i.e. cannot go high at the same time, otherwise the behaviour depicted will not happen - each determine a different transition out of state "wait".

c)

The aim of this question was to produce a state transition table for the state transition diagram given in figure Q3.5. The correct table is:

Inputs		current	next	Outputs		
coin	complete	state	state	clear	load	dispense
x	x	00	01	1	0	0
0	0	01	01	0	0	0
1	x		10	0	0	0
x	1		00	0	0	1
x	x	10	01	0	1	0

Marks were awarded as follows:

- Specifying the state vectors – 1 mark
- Correct state transitions – 1 mark
- Correct inputs – 2 marks
- Correct outputs – 3 marks

Issues:

- Drawing the state transition diagram? You are not asked to do this.
- A large number produced a state transition table that listed when outputs go high under a single heading – this is not a valid format for the table and does not indicate when outputs are 0.
- Putting don't care conditions for outputs – outputs must always be set to 0 or 1 – they should never be set to don't care as they could then take on any value, which may impact the design of the system.

- Not providing values for the state vectors – a large number of answers did this, even though the question specifically asks you to define them

d)

This question required the missing Verilog code for the FSM module to be produced. The correct answer (or one of many) and marking scheme is shown below:

```
// internal signal declarations                          - 1 mark
// looking for correctly defined

reg [1:0]  current_state, next_state

// next_state logic                                      - 2 marks
// looking correct function (minor syntax errors are fine) and a
// a default

always @(*)
  case(current_state)
    2'b00: next_state = 2'b01;
    2'b01: if(coin)            next_state = 2'b10;
           else if (complete)  next_state = 2'b00;
           else                next_state = 2'b01;
    2'b10: next_state = 2'b00;
    default: next_state = 2'b00;
  endcase

// state assignment                                      - 1 mark
// looing for correct reset action - minor syntax errors are fine

always @(posedge clock, posedge reset)
  if(reset)
    current_state = 2'b00;
  else
    current_state = next_state;

// output logic                                          - 3 marks
// looking for the outputs to be correctly assigned in each case
// for all states - whole marks only - so the answer must be
// correct

always @(*)
begin
  clear = 0;
  load = 0;
  dispense = 0;
  if(current_state == 2'b00)
    clear = 1;
  else if(current_state = 2'b10)
    load = 1;
  else if(current_state = 2'b01 && complete == 1)
```

```
    dispense = 1;
end
```

Issues:

- Incorrect implementation of the reset action (be this in the wrong place or not asynchronous in nature)
- Incorrect state transitions (particularly out of state wait – 01)
- Incorrect outputs assignments (the above is just one example)
- Incorrect sensitivity lists
- A significant number of answers attempted a single always block answer, which is prone to error
- Although I wasn't particularly concerned about slight syntax errors, a number of examples demonstrated significant Verilog misunderstandings.

Q4 – DK

This question was poorly answered with 1/3 of the students not trying the question at all. This is a bit surprising as the core of this question was directly aligned to the lecture on "Specialised Processing Architectures".

Given was an algorithm that takes some load & store, multiplication and accumulation. Even without the knowledge what the blocks exactly do, it would have been possible to answer most of the questions by treating those blocks as a black box with whatever cost. The question was deliberately open and any sensible answer got marks awarded.

a)

Students were asked to estimate CPU cycles when implementing the FIR function on a RISC CPU. A CPU executes sequentially, has only a limited register file size and I/O takes commonly a clock cycle longer than data manipulation instructions. Then there maybe some loop-overhead (or a hint that everything can be unrolled). The number of cycles can be somewhere between 50 and 5000 depending on the assumptions taken to answer the question.

b)

What changes when using a DSP? A DSP provides a multiply-accumulate instruction so that makes it a little faster (~half the time for the actual arithmetic). This was worth 2 out of the three marks. The final mark was for pointing at a DSP feature like zero-overhead loops or something on I/O.

c)

What changes when using VLIW? A characteristic asked for here is that VLIW can perform multiple different operations in one cycle. For example, a load, a multiplication and an addition in each cycle, which will make a VLIW correspondingly faster. Depending on the assumptions, this has some limitations on I/O (e.g., we cannot read and write to memory

simultaneously).

d)

What changes when using SIMD? This is a little similar to VLIW, but this time we perform multiple identical operations in a single clock cycle. It is important to understand that the number of operations is somehow limited (by the size of the SIMD vectors)

e)

What changes when using FPGAs? We can execute everything in one long (slow) cycle (assuming sufficient resources are provided). As an alternative answer, we can implement the algorithm pretty much exactly as shown in the figure provided in a pipelined manner (there are different ways how to do this that were all accepted). The delay chain is nothing else as a shift register, but other assumptions had been accepted.

Considering that this is just asking some basic understanding of different processing architectures, the outcome of this question is a bit worrying.