COMP22111 Exam Feedback 2019-20

Q1. PWN

There were two parts to this question: 1) showing (proving) that the the k-stage pipeline results in a speed improvement of k over the unpipelined design, this was worth 2 marks, and 2) a discussion of a balanced pipeline (one mark).

The proof for the k-stage pipeline is given in the notes as:

If there are k stages in the pipeline, then the first instruction will take kt seconds to complete, where t is the clock period. Once the pipeline is full, then one instruction is executed per clock cycle, so in the case of n instructions, this will take

kt + (n-1)t seconds.

In the unpipelined case it will take nkt seconds.

The pipeline speedup is given by the ratio of the two, i.e.

(nkt)/(kt + (n - 1)t)

If n is large, such that n >> k and (n-1) >> k then this simplifies to k (this assumption is required for full marks!).

A number of answers incorrectly discussed that if the delay of the unpipelined case in n, then the pipelined is n/k then this is why we see an improvement of k.

If a useful discussion was provided without a proof, then I was generous and awarded a mark, even though this is not what was asked for. I was not after a discussion of what a pipeline actually is.

In the case of a “balanced” pipeline each stage has the same delay/critical path. On the whole, most answers recognised this.

Average 1.51/3 (50%)

Q2. PWN

This question focuses on identifying hazards in pipelines. I wasn’t necessary after code examples, however, they proved useful if provided! One mark was awarded for a suitable example and description of one of the hazards discussed in the lectures:

- Data – where the data required by an instruction depends on the result of a prior instruction

- Structural – when different instructions try to access the same hardware at the same time (fetch instruction and store/load access to memory at the same time).
- Control – where a branch is to be followed but the following instructions in the stream have started execution.

With respect to a common approach to overcome, there are a few possible answers depending on the type of hazard, such as:

- stall the pipeline with NOPs, I expected the answer to mention both stall and NOPs in this case
- use of register forwarding, I expected the answer to discuss what is needed, such as extra hardware, data movement etc.
- use of instruction reordering, I expected the answer to discuss where the code reordering generally occurs (compiler)

On the whole this question was answered well.

Average 1.63/2 (81%)

Q3. PWN

Book work:

Cycle time is associated with the time of each stage in the pipeline, which is associated with the longest stage delay. The latency is the time it takes for the instruction to pass through all stages in the pipeline for the pipelined design or is the same as the cycle time in the unpipelined design. In the unpipelined case the cycle time and latency are the same.

I did not except answers that were not clear enough in the description – so answers such as “the cycle time being the time of each clock cycle”, or “the time to complete a single cycle” were not accepted without relating this directly to each stage in the pipeline and stage with the longest delay.

As the depth increases you would expect the latency to increase – the initial delay for the first instruction in the pipelined case. I would expect a reason why, reasons being the period of the clock being restricted to the delay of the slowest stage, or the addition of extra hardware.

Mixed answers to this question, as a number of answers were ambiguous/unclear, with a lack of focus with respect to pipelining designs.

Average 1.28/2 (64%)

Q4. PWN

You were required to identify the labelled elements of the test bench, these are:

a – test data/stimulus
b – design under test
c – golden test data
d – any reported errors

There was no flexibility in the marking. For example, I wouldn't accept "result" or "output" for d, as here "result" or "output" are ambiguous: is it the result of the simulation or the result of the comparison?

On the whole the question was answered well.

Average 1.38/2 (69%)

Q5. PWN

This question tests your understanding of the different testing strategies.

A basic description of each was required:

- Unit – testing individual components of a design
- Integration testing – testing of the design when components are connected together – largely testing connectivity
- Regression testing – testing that a design still works after modification

Unit testing is performed first as you would test the components before putting them together in a complete system.

The question was on the whole answered well. Most marks were lost by not answering the second part of the question regarding which type of test comes first!

Average 0.85/1 (85%)

Q6. CM

The question aimed at a basic understanding of what the main structural blocks are. It needed detail (not just mention ALUs, but, also examples of arithmetic/logic components, e.g. adder)

Issues:

- There was some confusion in this questions. The answers needed were at the RTL level abstraction (e.g. adders, multipliers, shifters, FIFOs, RAM, as taught in the slides)

Average 1.24/2 (62%)

Q7. CM

The question aimed at the main understanding of instruction set architectures as the interface between software and hardware and that it is a standardised description of how a chip works at the most basic level and provides with instructions for writing software to run on it. Common industrial ISAs are the ARM ISA and Intel x86 ISA. Some people went on to describe with more detail RISC and CISC, which was very good.

Issues:

- Although people mainly got this right, very short descriptions were provided which did not include enough detail, on what an ISA is.
- Mentioning only that an ISA is an abstract functionality layer or like an assembly language, is not enough.

Average 1.71/2 (86%)

Q8. CM

The question aimed at identifying knowledge of a main open-source hardware instruction set architecture (RISC-V) and the main benefits of such an approach, namely, being able to work without royalties and contractual constraints.

Issues:

- Most people mainly got this mostly right, but they often did not mention the RISC-V ISA explicitly as an example.

Average 1.48/2 (74%)

Q9. CM

This aimed to show understanding of how FPGAs work by exploring the main advantage/disadvantage of using LUTs.

Issues:

- The answers to the first part of this question mainly focused on LUTs used for logic functions. In this context, they correspond to truth tables. This was accepted as a fully correct answer. It's worth noting, nevertheless, that, while LUTs are used for logic, this is by no means their only use. They are a general way to translate an input pattern to an output pattern (e.g. can be used to describe complex mathematical functions).
- The advantage / disadvantage of the FPGAs approach (using LUTs to implement logic in hardware instead of utilising physical logic gates) was often missed, however.
- It was mentioned that LUTs provide a faster and cheaper way to implement logic. However, the correct answer is that it is indeed very convenient (and cheaper) as it

has a constant access time and with high precision (as much as we put in). It can be seen as faster in relation to having used logic gates hardware with a long critical path, but you have to take into consideration of how it scales (see next point).
- On the other hand, the disadvantage of the approach was almost completely missing from the discussions. It is true that using LUTs is versatile in that the same hardware block can emulate any 2, 3-input combinatorial function. However, it is usually large and slow. It can get very big and does not scale well.

Average 1/2 (50%)

Q10. CM

This questioned aimed at basic understanding of shifters.

Issues:

- There was some occasional lack of clarity on stating what a shifter is but the answers mostly got it right.

Average 1.13/2 (57%)

Q11. CM

This question aimed to understanding on the necessity of using specialised floating point units (FPUs), with regards to cost, footprint, efficiency and whether it is critical for the application or not. E.g. you would not need to use on in a lift controller.

Issues:

- A lot of answers missed the opportunity to provide detail on what an FPU is and to mention some typical operations like addition, subtraction, multiplication, division, etc.
- few answers mentioned explicitly that you need floating point to represent very large and very small values in a hardware efficient/economic way.
- Very few answers mentioned all possibilities for doing FP operations (co-processor as add-on or integrated in the main CPU and an FPU emulator/library).

Average 3.13/5 (63%)

Q12. CM

This question aimed to a basic understanding of the normalized form.

Issues:

- Most answers addressed part of it. That the normalized form provides with a standard form to the floating point values is a key message.

Average 2.37/4 (59%)

Q13. CM

This question aimed to test a basic conversion to floating point.
Issues:

- The steps here are convert to binary form, normalize the number, get the 3 parts (sign, exponent, mantissa), put all the bits together and fill out the rest of the mantsissa with 0s, and then convert to hex (by converting each group of 4 to a hex digit).

Average 0.63/2 (32%)

Q14. CM

Issues:

- The steps here are convert the hexadecimal to binary, identify the 3 parts in the IEEE format (sign, mantissa, exponent) and we then convert the resulting binary to the decimal equivalent (e.g. by powers of 2). The floating point formula would have been useful here.

Average 0.57/2 (29%)

Q15. CM

This was somewhat challenging and aimed to explore conversions when the mantissa/significand and exponent specifications are given (but are not in one of the IEEE formats).

Issues:

- The mantissa/significand and exponent were mostly readily identified. But the hint (2's complement form) was very often not taken onboard.
- The result comes from correctly identifying the mantissa and the exponent, according to the specifications given, realizing the mantissa is a negative number, working out the 2's complement, shifting the binary point (according to the exponent) and converting to decimal.

Average 0.43/2 (22%)

Q16. CM

This was aimed to explore basic understanding of machine learning accelerators, and, in particular how the TPU works. The question proved challenging, although most people managed to answer part of it.

Issues:

- the activation unit was very often forgotten as one of the main components of the Tensor Processing Unit.

- the unified buffer (and the need for registers) was also -less- often omitted.
- While a lot of answers described the systolic array architecture, the name of the, Matrix Multiplier Unit (MXU) was often omitted.
- On the question why we need ML accelerators, there needed to be emphasis both on the fact that ML methods are very compute-intensive as well as that the data sets are increasing in size.

Average 2.72/5 (54%)

Q17. PWN

Questions 17-22 focus on a fictitious RISC processor design with similar features to the Stump processor we looked at extensively in lectures and you completed in the lab. The design has been devised for the question and may have flaws that make it impractical to implement. It is not meant to be Stump or even in anyway related to Stump.

In this question you were required to recognise the key features of a load/store architecture, particularly with respect to the inclusion of a register bank. I was looking at the answer identifying that in a load/store architecture operations are only performed on data held in local registers, no data is stored in memory, with the result is always written back to a local register. So, the presence of a local register bank fulfils these requirements.

In the answer I was specifically looking for:

- operands come from a local register in the register bank
- the result of an operation is written to a register (not necessarily in the register bank, but I wouldn't have marked you down for not recognising this)

I wasn't looking for a discussion of how load/store instructions allow data to be transferred from/to memory. The discussion MUST relate to the use of the register bank when performing operations. A discussion of the use of registers to store data temporarily from memory, whilst not technically incorrect, isn't a true definition of a load/store architecture. Key is that the use of local memory makes it faster/more efficient.

Marks were mainly lost for not recognising that the result of an operation is written back to a register – as this is an important feature of a load/store architecture - or marks were lost due to a lack of clarity in the answer.

Very much a mixed bag of answers, with too much of a focus on moving data from memory to enable operations to take place.

Average 0.54/1 (54%)

Q18.PWN

The question focuses on your understanding of the operation of a RISC processor datapath and, in particular the use of the address bus, which is used for two different purposes.

The mux is used to select between

- the address in the PC in the fetch cycle and
- the address used for memory access when performing a load/store instruction – must be clearly stated. (Not branching!)

Both must be specifically identified for the mark.

Just pointing out that the address in the PC needs to go to memory in order to fetch the next instruction is not enough for the full marks, otherwise, why have the multiplexer? Why not make the connection permanent! I was not after a discussion of the instruction fetch operation.

Almost everyone recognised the need to connect the PC to the address bus for the instruction fetch. However, a large number failed to recognise the other use of the address bus in load/store. There was a lack of clarity in some answers.

Average 0.65/1 (65%)

Q19. PWN

The add1 block is simply there to perform the PC increment so the PC points to the next instruction. The update is made during the fetch phase.

One mark for incrementing the PC, the other mark for recognising this happens in the fetch phase. The discussion needs to specifically highlight that it is used to increment (add 1) to the PC.

Any lost marks were related to not identifying that the PC increment happens in the fetch phase. However, on the whole the question was answered well.

Average 1.81/2 (91%)

Q20. PWN

The multiplexer mux1 is used to select where the final value of the PC comes from:

- from the add1 block in fetch (when calculating the address of the next instruction), which ALWAYS happens, or
- the PC is overwritten with an address to branch to, from the ALU, if a branch is taken.

Both are needed for the mark. I was harsh in cases where the discussion refers to the an "or" action. The PC is ALWAYS incremented in the fetch phase, it is further overwritten if the branch is taken – clarity in the answer was important. The key thing is the PC is updated for a branch.

Average 0.73 (73%)

Q21. PWN

This question required you to do some Verilog coding (which you have done plenty of in the labs).

You are required to produce a Verilog module using behavioural Verilog. There are number of ways this can be done: using assign or an always block, concatenation etc, and you are free to choose your own approach, along with selecting appropriate names for the input and output. However, the module name MUST be called *extender*.

Examples:

```
module extender(input      [5:0]    value_in,
                output reg [15:0]   value_out);
always (*)
  value_out = {{10{value_in[5]}},value_in[5:0]};

endmodule
```

or

```
module extender(input  [5:0]    value_in,
                output [15:0]   value_out);

assign value_out = {{10{value_in[5]}},value_in[5:0]};

endmodule
```

or

```
module extender(input  [5:0]    value_in,
                output [15:0]   value_out);
always (*)
begin
  value_out[15] = value_in[5];
  value_out[14] = value_in[5];
   . /// repetition
   .
  value_out[5:0] = value_in[5:0];
end
```

```
endmodule
```

or any variant where the sign bit is copied across the most significant bits.

Marks were awarded as follows, 1 mark each:

- correctly defined header (½ mark if the output is not defined as reg if not using assign or reg when using assign)
- correct functionality – it either works or doesn't
- no syntax errors (even if not correct operation) - design would compile and synthesize without error.

It is important that the value is sign-extended, so padding with 0s or 1s is incorrect – there were a number of examples like this, in which case the functionality is wrong.

I was not interested in commenting, just syntax and functionality.

Average 1.90/3 (63%)

Q22. PWN

This question tests your ability to write structural Verilog through the translation of the schematic design for a processor datapath into a structural Verilog form.

You should use the net definitions provided in the circuit diagram, and the names of modules given. I was not concerned what you called the instances. I wasn't looking for commenting when marking.

The correct answer is:

```
wire [15:0] net1, net2, net3, net4, net5, net6, net7;

//instantiate PC mux controlled by m1_sel

mux_2to1 mux1(.Din0(net1),
              .Din1(net3),
              .sel(m1_sel),
              .Dout(net2));

//instantiate reg_bank mux controlled by m2_sel

mux_2to1 mux2(.Din0(data_in),
              .Din1(net3),
              .sel(m2_sel),
              .Dout(net4));
```

```
//instantiate add1

increment add1(.num_in(net7)),
              .inc_out(net1));

//instantiate PC

reg_16bit PC(.data_in(net2),
             .clk(clock),
             .rst(reset),
             .enable(PC_en),
             .data_out(net7));

//instantiate IR

reg_16bit IR(.data_in(data_in),
             .clk(clock),
             .rst(reset),
             .enable(IR_en),
             .data_out(IR_out));

//instantiate reg_bank

registers reg_bank(.D_in(net4),
                   .wr_dest(dest),
                   .rd_src(src_rd),
                   .st_src(src_st),
                   .clk(clock),
                   .rst(reset),
                   .wr_en(RB_en),
                   .D_out(net6),
                   .S_out(data_out));

//instantiate sign_ext

extender sign_ext(.value_in(IR_out[5:0]),
                  .value_out(net5));

//instantiate proc_ALU

ALU proc_ALU(.srcA(net5),
             .srcB(net6),
             .ALU_sel(ALU_con),
             .ALU_out(net3));

//instantiate addr mux controller by m3_sel

mux_2to1 mux3(.Din0(net7),
```

```
    .Din1(net3),
    .sel(m3_sel),
    .Dout(addr));
```

Marks were awarded as follows:

- correctly defined internal variables,
- correctly instantiating each component (1 mark for each). If there is a minor mistake, such as a single typo, then ½ mark. Significant errors, such as a missing connection, resulted in the loss of a mark.
- using the stated wire names throughout
- overall, full-working (that compiles and synthesizes) design.

There was some flexibility in this marking scheme due to the variation in incorrect answers.

Issues that resulted in the loss of marks (not an exhaustive list):

- defining IR_out – this is defined already in the module header
- not defining the internal variables, or not defining all
- not defining the internal variables as 16-bit buses
- defining internal variables as reg
- not using the net names given in the schematic
- getting the order of name and instantiation name wrong in the module instantiation – it should be module name followed by instantiation name.
- adding “module” before the instantiation
- incorrectly defining the input for the sign extender (not as a 6-bit bus)
- getting the order wrong when defining the inputs/outputs in the module instantiation – should be .pin_name(net_name)
- not instantiating all modules
- typos in pin/net names

I was really impressed with the quality of the answers for this question, with every answer using the explicit representation! It was very well-answered overall, with a number of answers getting full marks. Clearly it was too easy!

Average 9.63/12 (80%)