Q1.

This was a popular question and was answered by most students. The questions largely relate to book work, with part e) where you should put your knowledge into practice by writing a Verilog function (based very much on what you have seen for the Stump).

a)

The aim of this question was to identify the differences between continuous, blocking and non-blocking assignments in Verilog. I was looking for the following:

Continuous assignment - the variable is continuous updated when any variables change. You need to use the keyword "assign" (1/2 mark), i.e.

```verilog
assign w = a+b;
```

where the variable being assigned must declared as a wire, i.e. wire w; (1/2 mark).

Blocking assignment - variable must be declared as "reg" (1/2 mark) and a group of blocking assignments will be executed in order by the simulator (1/2 mark).

Non-blocking assignment - variable must be declared as "reg" (1/2 mark) and a group of non-blocking assignments will be executed in parallel (1/2 mark).

A common mistake with answering is this question was failing to identify which "type" the variable being assigned must be.

b)

Easy question - most got this correct:

```verilog
blocking - =
non-blocking - <=
```

c)

This was probably the most poorly answered question. The aim was to discuss how the simulator treats the different types of assignments, and how the events queue and non-blocking queues are used. See notes for Part 1 from page 84. The figure I was expecting to see is given on page 86 (1 mark).

The discussion should describe:

* blocking assignments are placed on the active events queue and assignments are made in the order they appear on the queue (1 mark),
* in the case of non-blocking assignment the RHS of the assignment is placed on the active events queue (1 mark),
* the LHS assignment of non-blocking assignments are placed on the non-blocking queue (1 mark),
* after all the assignments on the active events queue have been performed (so it is empty) then the assignment of non-blocking assignments is performed (1 mark)

In addition, I expected some discussion of when events are placed on the queue (1 mark), i.e. when events occur in time the simulator stops time and performs the assignments as above, and what happens once both queues are empty (1 mark), i.e. the simulator starts time again until the next event occurs.

Common mistakes were producing timing diagrams and basically explaining what was discussed in part a) - this is incorrect. The assignments are treated differently by the simulator, in the way assignments are made, but in terms of "time", these happen all at the same time. In addition, few failed to explain when events are added to the queue and what happens when the queue is empty.
d) A task can have any number of inputs and can modify any number of outputs variables. Whereas the function can only return a single Boolean value - true or false.

e) The aim of this question was to test your ability to write simple Verilog code. Common mistakes included:

* a function header is

```verilog
function testbranch;
```

the input variables are declared in the body of the function, i.e.

```verilog
function testbranch;

input [2:0] condition;
input [1:0] status_flags;

begin

end
endfunction
```

there is no always or initial block in a function.

* The output variable has the same name as the function, so in the example above you would be assigning values to a variable "testbranch", which does not need declaring.
* There was some misunderstanding of the use of blocking/non-blocking. Remember, non-blocking assignments are used when we have some time-dependent behaviour (i.e. an always block with posedge clock in the sensitivity list). Using non-blocking assignments will invariably lead to the introduction of flip-flops - the simple test branch function should be a simple combinatorial logic block, so blocking assignments should be used.
* A number of answers used syntax from other programming languages - switch, return, true, false - these would not work in Verilog!
* The answer relies of returning 0 or 1 depending on the condition and the status of the flags - a number of answers had a long list of nested if statements for this instead of a simple case statement. What's more, the if statements had no "else", so the result would be invalid (and would likely be implemented with flip-flops).
* The question "How would you use the testbranch function in your processor module?" resulted in a number o strange answers discussing the "implemented" processor - the question specifically relates to the "module". The question is asking how you would call the function in your processor module. A function returns a boolean value, so it can be used in a test, for example:

```verilog
if (testbranch(<variable list>))
```

// do something

else

// do something else

**************

Q2.

Everyone answered this question, which was strongly bookwork related. On the whole it was very well answered.

a) The question asked for a discussion of the fetch, execute and writeback stages of a typical RISC processor.
Problems included the addition of a "decode" phase, which is part of the execute phase, and a discussion of fetch, execute and memory stages, which is particular for the Stump (the aim of part b) is to discuss this as part of the difference).

b) The aim of this question was to discuss the difference between the fetch-execute-writeback stages of the general RISC processor and the fetch-execute-memory stages of the Stump. The discussion should cover

* the execute-writeback being combined into the execute phase (1 mark), and
* the addition of the extra memory phase particularly for memory access as part of load/store operations.

c) Everyone got this answer correct. R0 is hardwired to 0 (1 mark) to enable addition instructions such as NOP and MOV (1 mark).

d) The question required a discussion of the difference between Type 1 and Type 2 instructions. A number of answers presented the instruction format for both, this was unnecessary. I was after the following points being discussed:

* a type 1 instruction takes operands from 2 registers (1 mark) and writes the result back into a register/memory (1/2 mark).
* a type 2 instruction takes one operand from the register bank (1/2 mark) and the other comes from an immediate value from the instruction (1/2 mark)
* the 5-bit immediate value (1/2 mark) must sign extended to 16-bits (1 mark)

The type 1 offers an additional shifting operation as part of the instruction (1 mark).

e) The aim of this questions was to provide shaded path usage diagrams for the 3 instructions.

Typical problems:

* missing out the shaded path for the PC increment in the fetch phase.
* shading the path from the data interface to the execute unit in the fetch phase
* shading paths in the memory phase for instructions which weren't load/store (the only answer which should use the memory phase is for line 5) - 1/2 a mark was lost for cases where these were shaded when not used.

Q3 This question was intended to link the lecture material on ISAs to the microarchitecture as developed in the lab. Overall it was largely well answered.

A) Largely a 'giveaway' piece of bookwork. Generally understood and usually reasonably explained. Some answers were a bit terse for the marks available but were given the benefit of the doubt where possible.

B) Again, quite straightforward although a number of candidates appeared not to have read the words "Compare and contrast".

C) This was the main 'problem' element of the question. The instruction/ISA is probably unknown to the candidate but all the necessary information is present. In hindsight it may have been beneficial to include more explicit instructions such as "mention any non-architectural registers that are used" although being too explicit would make the question very 'wordy' and some candidates realised the benefits anyway. Too many people seemed to think that effectively repeating the question was sufficient - full marks would require more of a demonstration of understanding. The most disappointing finding was that no one enlarged on the fetch process beyond 'fetch the instruction' although, set in the context of the question, it should have been apparent that determining exactly what constituted the instruction was part of the process. An analogous process in the i8080 was discussed in the appropriate lecture and is covered in a state diagram in the notes.

D) i) The emphasis in the question was clearly insufficient for a majority the candidates. A typical answer might be "because ARM is a RISC and RISCs don't do this" - as if there were a clear definition of RISC (which ARM obeyed - single cycle operation, for example?) and some authority decreed rules which must be followed. The real restriction
is that an op. code like this wouldn't fit. (This also links back to the IA-32 fetch mechanism.

ii) This allowed more scope to bring in other knowledge. The original hope was to consider aspects such as pipelining and stalls. There were probably more mentions of 'hazards' and that such an instruction might not be reliable - odd considering that there is a high probability that if the operation didn't work neither would a desktop PC, or laptop, or …

Q4 This question bridged the subjects from a number of lectures. It was answered reasonably although there were some misconceptions apparent. A recurrent misconception was that post-layout simulation was for checking physical properties of the design. There is clearly confusion with tools such as DRC/ERC checking.

A) Most people got this okay; the key being cost.

B) Again this was largely understood, especially the first part (which should have been strongly reinforced by lab. Exercises).

C) Quite a lot of candidates did not read this question properly. It is not a repeat of the previous section: it specifically seeks performance data. A lot of candidates missed out on the concept of cycle counting at the high level and also the (deeper) issue that simulating in every detail is not possible for big functional tests.

D) A variety of answers. Gate depth featured in many answers, fewer were clear about fanout and loading.

E) Problem element of the question. A disappointing amount of confusion, including several suggestions of putting more logic between registers. The means of controlling synthesis options was intended to be challenging and was largely missed.

F) I hoped the wording here was explicit enough. In practice may candidates read this as "why will it not run" rather than "not be [choose to] run". Answers mixed.

Overall this question averaged lower marks than Q3 which quite surprised me as the 'problem' part was less. There was a significant degree of misapplying 'bookwork'.