COL718 (FALL, 2019)

HIGH PERFORMANCE COMPUTING

Midterm Solutions

The solutions highlight the key-points

Total Marks: 20

1 `Ms.SuperFast` **decides to get rid of register renaming. She says that the renaming logic consumes too much power. She also says that instead of using register renaming, she will simply use a larger number of architectural registers (the same size as the physical register file).**
What are some advantages of this design?
What are some disadvantages of this design?
Overall, is this design a good or a bad idea, what do you think?

Total Marks: 5

Advantages include: slightly simpler hardware (no RAT, no free list), more flexibility to the compiler to assign register names to the temporary values/variables.
The most significant disadvantage includes: If **WAW** and **WAR** dependencies exist in the code, then the processor cannot resolve them automatically. Instead, it will have to stall on such dependencies resulting in poorer utilization of the register file. This is especially critical for OOO superscalar processors with branch prediction, where the compiler may not be able to predict which registers may be used simultaneously in the pipeline. Register renaming would have resolved WAW/WAR dependencies at run-time, whereas doing this at compile time without knowing which branches are likely taken is much harder and prone to mistakes.
Bad design for the above reasons.

marking scheme:

- correctly pointing out advantages: $+1$
- concluding that the design is **bad**: $+1$
- correctly pointing out disadvantage(s): $+3$
  - The crux is the trouble in handling **WAW**, and **WAR** dependencies in OOO superscalar processor by the compiler which lack run-time dynamic information. If this is explained/mentioned: $+3$
  - If the aforementioned point is omitted, and other secondary disadvantages are specified, then partial marks: $+1.5$

2 `Mr.TooWorried` **thinks that the burden of supporting precise exceptions is too much for the architecture because the architect has to maintain extra state for rollback. He decides to get rid of precise exception semantics (meaning that the software writer cannot assume that the architecture supports precise exceptions). Is this a good or a bad idea? Why or why not?**

Total Marks: 3.5

Bad idea. Precise exceptions make the software abstractions simple and easy to reason about. Imprecise exceptions are a huge cause of mistakes and software bugs.
Moreover removing precise exceptions hardly simplifies the architecture because the hardware still needs to deal with branch mis-predictions which have identical handling logic to precise exceptions, and are much more frequent than exceptions. Thus removing precise exceptions will not simplify the hardware–

we still need the extra state.

marking scheme:

- correct conclusion: +0.5
- Explanation: Why Bad? +3
  - If the issue of correctness, mistakes, bugs, simplified S/W abstractions are specified: +1.5
  - If the implication in H/W is also pointed out, and it is analyzed that there is no simplification in H/W, then an additional +1.5

3 What is the significance of the Global History Register? Why is it a good idea? Explain succinctly but clearly.

Total Marks: 3.5

GHR captures the global history of branch taken/not-taken decisions and allows prediction based on the correlated branches in the program, which happen to be common. E.g., if branch 1 is taken, with a high probability, branch 2 is not-taken.

marking scheme:
Most of the students did answer this question correctly and also provide examples. No fine-grained division in marks is introduced in the marking policy.

4 Explain how SMT handles the tension between the goal of "fastest possible single-threaded execution" and "fairness and deadlock freedom in multi-threaded execution" simultaneously. Does it require any OS support? If so, what?

Total Marks: 4

By using two modes : single-thread (0 and 1), and multi-thread.
It requires the OS to use the `hlt` instruction to indicate that one thread is currently inactive (which it anyways should be using to save power even with SMT support).

marking scheme:

- The answer to first 'how': +1
- Does it require OS support? correct answer: +1
- What support from OS is required? +2

5 Explain why the handling of def-use and use-def chains is different in the SLP algorithm? What is the difference?

Total Marks: 4

Because one definition can have multiple uses, but one use can only have one definition.
Thus, in the former case, we choose the most profitable packing from multiple options, while in the latter case there is only one packing to consider.

marking scheme:

- Why difference? $+2$
- What difference? $+2$