Advanced Computer Architecture

CS-470

24th June 2023, 9:15am–11:15am, Room CE6

- *The exam lasts exactly **2 hours**.*
- ***No books**, notes, or references are allowed.*
- *Read the questions carefully—they may not be asking what you think at first.*
- *Answer **all** the questions **in the space provided** (you are encouraged to use more sheets to develop many of the solutions, but these should **not** be handed in).*
- *Try to be as **succinct and to the point** as possible: **long and vague answers will be treated at a disadvantage!***
- *When required, explain the answer given—a correct answer without an appropriate explanation is **worthless**.*

Identification

Surname and name: . . . . . .

SCIPER no.: . . . . . .

1 Out-of-Order (OoO) Execution

a. Consider a processor which can issue a single instruction per cycle and whose functional units take exactly one cycle to execute any instruction. How much advantage would you expect by implementing the logic for OoO execution? Explain your answer.

No. The purpose of OoO is to execute later instructions ahead of time to hide the latency of early instructions, so that the processor is always busy. For a single-issue processor with all instructions taking one cycle to finish, dependencies cannot delay any instruction from being issued.

b. Which structures of OoO processors are responsible of changing the order of execution of instructions?

Reservation stations and the load-store queue.

c. A Reorder Buffer (ROB) is one of the key components for an OoO processor to implement precise exceptions and speculative execution. The basic idea is to hold back state updates until the execution of an instruction is fully confirmed (no exceptions, no mispredictions, etc.). Why might the ROB contain the address in memory of the instruction (informally sometimes called PC)?

The PC would be needed in case of exception, to load appropriately the EPC.

d. Suppose that a processor has 32 architectural registers and an implementation has a 32-entry ROB. (a) What would be the maximum number of physical registers that would make sense for it? (b) Would it make sense to have fewer physical registers than that? What might happen in that case? Explain.

(a) 64. More would never be used at once. (b) Yes, but then decoding might need to stop if there are no free physical registers.

2 Register Copy Elimination

The decode stage of modern processors can execute directly copy instructions such as `move rY, rX`, where the value of register `rX` is copied to the register `rY`. The idea is to implement the copy by simply mapping the architectural register `rY` to the same physical register to which `rX` is mapped.

a. Consider the following processor resources:

   - reorder buffer,
   - reservation stations, and
   - physical register file.

   For each of these three resources, mention if you expect them to be on average (i) less occupied, (ii) more occupied, or essentially (iii) left unaffected by this modification. Briefly explain.

   - Reorder buffer: Unaffected, because the instruction should still be tracked for sequential ordering.
   - Reservation stations: Less occupied, because no actual functional unit is used.
   - Physical register file: Less occupied, because no additional physical register is allocated.

b. Consider a normal register renaming scheme such as the one implemented in the MIPS R10000. What are the two situations when physical registers are freed and made available for a new use?

   There are two cases:

   - When an instruction is committed, the old physical register mapping to the dest architectural register should be recycled.
   - When there is a rollback (e.g., exception, or branch misprediction), all in-flight instruction destination physical registers will be recycled.

c. If more than a single architectural register points to the same physical register, it will be more complicated to figure out when the physical register can be freed. Suggest an idea on how this problem could be solved.

   An additional entry is needed for every physical register to track how many architectural registers are mapped to it (i.e., a counter of references). During the decode stage, when copy elimination happened, the reference counter is increased. Before attempting to free a physical register, the reference counter is decremented and actually freed only if the counter is null.

3 Memory Order Speculation

a. Both OoO processors and VLIW processors reorder memory instructions to improve parallelism. What are the two ways used in Itanium to reorder memory instructions?

- Speculative load, when a load instruction is moved before a branch instruction. It requires the user to check the load poison bit before using the result, in case an exception happened.
- Advanced load, when a load instruction is moved before a store instruction. It requres the user to check the load poison bit before using the result, in case the store and the load accessed the same memory location.

b. For each of the two ways mentioned above, name the mechanisms implementing the corresponding memory reordering in an OoO processor.

- Branch prediction and speculation achieve the same effect as an speculative load.
- Dependency prediction and speculation achieve the same effect as a advanced load.

c. Why do both designs only reorder load instruction, rather than reorder both load and store instructions?

- In both VLIW and OoO processors, store instructions do not return any data and thus cannot create any register dependency; hence, there is little motivation to reorder.
- In an OoO processor, store instructions are only executed when committed, because they alter the machine state irreversibly. Thus, they are executed in program order.

d. Do you think it might be possible to conduct a Meltdown attack on Itanium? Explain.

No. Meltdown makes an illegal access to memory. Although the returned value of the load is never made architecturally visible, of course, Meltdown exploits speculative execution to leave an imprint of the value in the cache before the speculated instruction are cancelled. Since Itanium has no way to hide a returned value (there is no commit phase), any load, including advanced and speculative loads, must only return legitimate values or return garbage—in fact, a speculative load returns NaN in Itanium.

4 Dynamic Binary Translation

As an intermediate layer between traditional software and hardware, dynamic binary translation is an effective tool to solve many problems at binary level.

a. What problem of VLIW processors is mitigated by using dynamic binary translation?

Binary incompatibility.

b. Many dynamic binary translation engines (e.g., Transmeta Crusoe, Intel Pin, DynamoRIO) use a translation cache to improve performance. Name one program property that makes a translation cache beneficial. Explain.

In general, temporal locality as for all caches. This may occur because of loops or because of frequently used functions and routines.

c. A particular form of dynamic binary translation is dynamic code optimization, which can generate more efficient code than a static compiler by exploiting profiling information at runtime. This enables the construction of traces, that is, chains of basic blocks frequently executed as a sequence and efficiently scheduled together.

Both trace scheduling and predication remove branch instructions. What is the main difference between the two optimizations?

Traces are a form of static speculation and only contain instructions from one path (e.g., they assume a branch is taken or not taken), while predicated execution involves instructions from both paths and is not really speculative.

d. What is the equivalent way in an OoO processor to achieve the same goal that trace scheduling achieves in a VLIW?

Dynamic branch prediction and speculative execution.

5 Cache Side-Channel Attack

You want to perform the cache attack of Lab 4 on an Apple M-series processor. However, this processor has two important relevant characteristics:

- For the sake of security, it does not provide a high-resolution timer for user-space applications.
- By default, it does not provide an effective cache flush instruction

Assume that you have the full control of the victim functionality (i.e., the same way as in Lab 4).

a. How would you build a highly precise timer on an M-series multicore processor.

Utilize a separate core to build a timer by executing instructions that increment the value of a counter in memory. The attacker thread can read the value of the counter to get an accurate measure of time (probably in the range of 10 ns, due to the coherence protocol).

b. How would you implement cache flushing? Which information would you need in order to make your plan work?

One can only use eviction to mimic the behavior of flushing. To fully fill the cache with one's own content, one needs to know all parameters of the cache, such as size, associativity, set selection function, replacement policy, victim cache, etc. Also one needs to know something about the mapping between virtual and physical addresses.

6 Spectre Attack

a. One of the key factors to successfully perform a Spectre attack is to train the branch predictor. Explain why you need to train the branch predictor to perform a Spectre attack.

Because Spectre relies on leaking through the cache information obtained in a malicious memory access performed after a branch misprediction, where the branch typically was intended to prevent the access. Training the branch predictor is needed to guarantee that a branch misprediction would happen when attacking.

b. Here is an idea for training the branch predictor:

```
for (int i = 0; i < 10; i++) {
    // [...] omitted code to prepare the side channel attack
    int input = LEGAL_INPUT;
    if (i == 9) {
        input = MALICIOUS_INPUT;
    }
    victim_function(input);
    // [...] omitted code for counting
}
```

As you may know, modern processors use a combination of global branch history and branch address to query the Pattern History Table (PHT). Thus, there are sound reasons to believe that this might not work. Why?

This global branch history is different in the loop iteration when the malicious input is presented, thus updating a one particular entry in the PHT during the preparation and using a different during for the attack.

c. Suppose that indeed the attack fails. By inserting a simple loop before calling `victim_function`, one would create a uniform history and the problem would be altogether avoided. Following this idea, one could reverse-engineer the maximum global history a processor can hold. Explain a possible experiment to get this information.

Each iteration of the simple loop would shift by one position in the global history the problematic branch outcome of the test loading the input with the malicious input. While that branch outcome remains in the global history, the attack fails. Hence, the minimum number of simple loop iterations that make the attack succeed represents the size of the global history shift register.

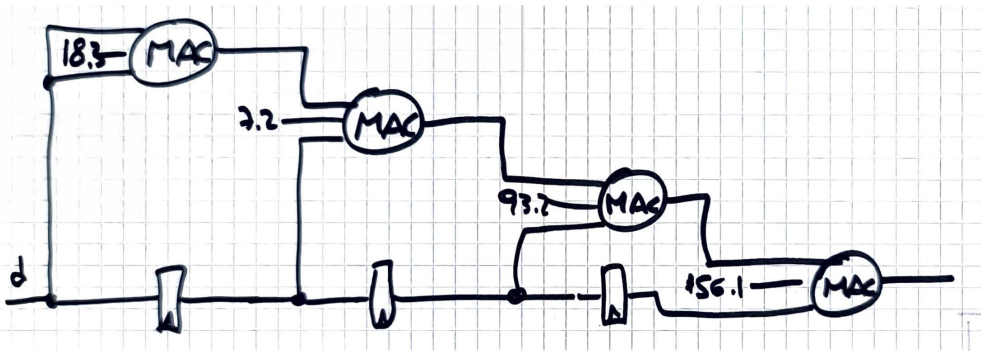
7 Statically Scheduled HLS

Consider the following kernel:

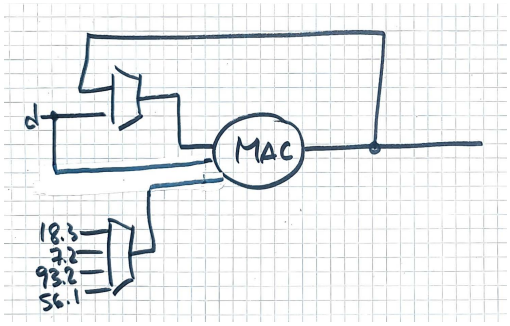
```
fp16 polynomial(fp16 d) {
  return (((((((d + 18.3) * d + 7.2) * d) + 93.2) * d) + 56.1) * d);
}

fp64 accumulation(fp16 A[N]) {
  fp64 acc = 0.0;
  for (int i = 0; i < 1000; i++) {
    fp16 elem = A[i];
    acc += polynomial(elem);
  }
  return acc;
}
```

The `fp16` and `fp64` types are, respectively, 16-bit and 64-bit custom floating-point formats. In `fp16`, there are only three available components: (i) a three-input component performing the addition of two inputs and a multiplication of the result with the third component, (ii) a two-input component performing a comparison, and (iii) a load unit. All have latency of one cycle and include a register before the output. In `fp64`, the only available component is an adder with a 4-cycle latency; it has four pipeline registers internally, including one before the output. All components can start a new operation every cycle.

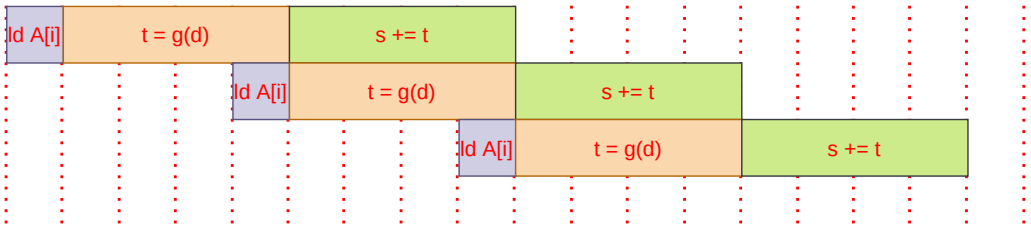
a. Draw the datapath that would be generated by a static HLS tool with a target initiation interval (II) of 1 for the function `polynomial`. Do not detail control signals, if any.



b. Now draw the datapath that would be generated by a static HLS tool with a target II $= 4$ for the same function `polynomial`. Do not detail control signals, if any.



c. Draw the best schedule for function `accumulation`. What is the II? Which of the previous implementations of `polynomial` is most adapted? Why?



The II is 4 due to the accumulation and the latency of the adder. Hence, there is no point on using the implementation of bullet (a); the one at bullet (b) is cheaper and equally effective.

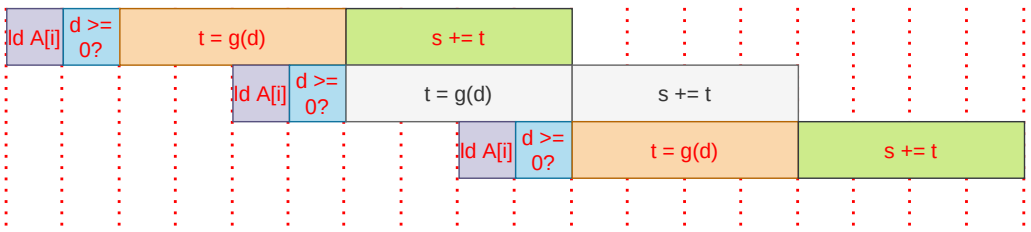
d. If there were an even cheaper implementation of `polynomial` with II $= 8$, would a good HLS tool use it? Explain.

No. It would slow down execution. [The question was actually a mistake, because it was meant to ask about the influence of a higher latency, not higher II....]

e. Now consider a different algorithm with the following loop (the rest is unmodified):

```
for (int i = 0; i < 1000; i++) {
  fp16 elem = A[i];
  if (elem >= 0) {
    acc += polynomial(elem);
  }
}
```

Show the modified schedule. Assume, for instance, that the first three elements of `A[]` are positive, negative, and positive, respectively.



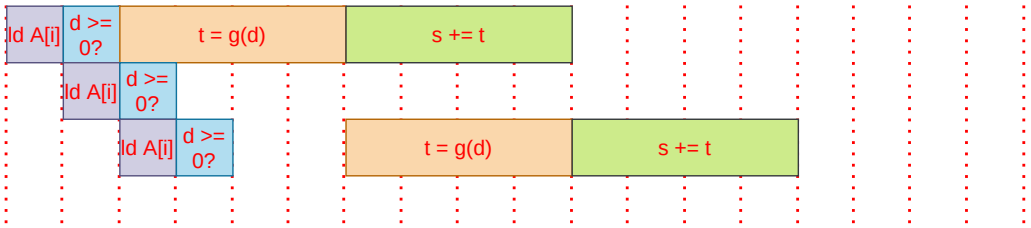
8 Hybrid Statically and Dynamically Scheduled HLS

a. What overhead does dynamically scheduled HLS add over statically scheduled HLS? Explain.

The FSM controlling the datapath is replaced by handshake management logic. This is usually more expensive (more area) and may creates a longer critical path (slower).

b. Now consider the code at the end of Question 7 (the version with the modified loop). Would this code offer opportunities for dynamically scheduled HLS to achieve better performance than statically scheduled HLS? Why? Justify your positive answer by showing the superior schedule of the dynamically scheduled circuit or explain why it would make no difference in this case.

The loop carried dependendence now exists only when the values in the array are positive or null. A dynamically scheduled circuit would not waste time for the accumulation when unnecessary.



c. Can statically scheduled HLS be a better choice than dynamically scheduled HLS, in some cases? Explain.

Statically scheduled HLS is better when the optimal schedule for the circuit does not depend from runtime information, as in FIR filters or in (dense) matrix multiply.

d. An idea could then be to implement some parts of the design as statically scheduled circuits and limit the dynamic handshaking protocol only to those parts of the design which can really profit of that. Explain which parts of the algorithm of Question 7 (again, the version with the modified loop) you would synthesize with statically scheduled HLS and which parts with dynamically scheduled HLS. Explain.

- The loop should be implemented as a dynamically scheduled circuit because the condition and the existence of a loop-carried dependence depends on runtime information; the function `polynomial`, on the other hand, is a simple arithmetic operation that can be implemented with static HLS.