[comp228-w16] review for the final, Thu Apr 15, H-507, 5:45 p.m.

David K. Probst probst at cse.concordia.ca
Wed Apr 13 20:49:41 EDT 2016

- Previous message: [comp228-w16] extra POD sessions are available (even for people who work!)
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Lecture 9  Review                                         DKP/Winter 2016
_________

Lecture 4 remarks

1) Connective syntax

Most people write binary (sentential) connectives in _infix_ fashion, i.e.,
they put the connective between its two arguments.  Thus,

p \/ q        , or

(p --> q) \/ (r + s)

We connect simple variables or more complex formulas.

Of course, we could write in _prefix_ fashion, although this would confuse
most people.  Thus,

\/pq          , or

\/ (p --> q) (r + s)

But logicians always write ternary (indeed, n-ary) connectives in prefix
fashion.  (Could it be otherwise?).  Thus, if '$' is a ternary connective
(its semantics is immaterial),

$pqr          , or

$ p (p --> q) (r + s)

This is _not_ sum-of-products syntax.  In particular, 'pqr' lists the three
arguments of '$'; it is not the product of 'p', 'q', and 'r'.

If you leave a little white space, you are not forced to use parentheses
(although this is not strictly legal or even user friendly).  Thus,

$ p p --> q r + s

is reasonably unambiguous.  Use your judgement.

2) Combinational logic
```

Combinational circuits compute Boolean functions if their inputs are held stable; for this reason, we often combine them with state elements in sequential circuits, such as the instruction-execution pipeline.

3) Modular composition

We often design circuits hierarchically: wire gates into subcircuits, wire subcircuits into larger subcircuits, and so on.

4) Settling time

A circuit settles as the longest delay from any of its inputs to its output.

5) De Morgan's laws

These are good to know.

Lecture 5 remarks

1) RISC dominance

Simple, uniform RISC instructions allow fast pipelining. Moreover, they simplify pipeline control, allowing microprogramming to be replaced by hard-wired control.

2) Load-store architectures

In these architectures, no machine instruction combines memory referencing and arithmetic.

3) Pipeline stages

These are good to know.

4) Pipelining

For us, this is injecting a new instruction into the instruction-execution pipeline on each and every cycle. The novel behavior is that we may have all the pipeline stages active simultaneously, even if each stage is working on a different instruction.

Lecture 6 remarks

1) Dependences

Data and control dependences are properties of programs. Running a particular program on a particular pipeline may give rise to _hazards_, which are situations where it is necessary to stall the pipeline, or to repeat a stage, in order to compute the correct result. Responding to hazards slows the pipeline down.

2) Data and control dependences

Between two instructions, there may be a data or a control dependence. You should know the difference.

3) Multicycle operations

In floating-point arithmetic, we often use multiple x's in space-time diagrams to indicate that the latency of a floating-point operation is greater than the latency of an integer operation. We do this primarily to make the space-time diagram more realistic. The floating-point stage functions as an embedded pipeline in the instruction-execution pipeline.

4) The Three Walls (not on the final)

- Focusing on single-thread performance rather than on the collective performance of a large number of threads. (The former has hit a wall).

- Increasing the clock frequency is forbidden by power dissipation.

- Large robot caches are _not_ a general solution to memory latency. (We need other mechanisms).

Alas, industry has learned lesson 2, but continues to ignore lessons 1 and 3. This is the reason why multicore is progressing so slowly.

5) Nonlinear pipelines (not on the final)

Industry did try making part of the pipeline controlled by dataflow ordering rather than strict control-flow ordering, and this postponed the RISC collapse by 10 to 15 years, before the multicore inflection point in 2003. The problem is that industry currently uses nonlinear pipelines in their multicore designs in 2016. Good lord!

Lecture 7 remarks

1) Lecture 7 should be understood in the framework of Lecture 7.5.

2) There are two ideas: i) robot caches are not the only way of dealing with memory latency, and ii) if caches improve the performance of cache-friendly programs, they degrade the performance of cache-unfriendly programs.

3) Cache friendliness is determined by the presence or absence of temporal or spatial locality.

Lecture 3 remarks

1) If a register has a bit length that is a multiple of 4, it may easily be represented by a sequence of hexadecimal digits. (We can represent other lengths as well, but this is not our subject here).

2) Consider an 8-bit register that encodes two 4-bit quantities, 'a' and 'b'. Then one digit of the register representation corresponds to each quantity. But suppose the 8-bit register encodes one 3-bit quantity and one 5-bit quantity. We can still give a hexadecimal representation of the register, but digits no longer correspond to quantities.

Lecture 8 remarks

```
1) Most cache parameters are powers of 2.  A cache line is 2^a bytes.  A
cache array is 2^b elements.  Note that an indexable cache element may be
either a cache frame or a set of cache frames.

2) This gives a a standard way to place, or to lookup, a particular cache
line in a cache.

3) Starting from the byte address, the cache line number is obtained by
discarding the low-order 'a' bits (this corresponds to integer division by
2^a).  The line number is turned into an index by extracting the remaining
low-order 'b' bits (this corresponds to computing the residue modulo 2^b).

4) Any bits left over form the tag field.

5) To lookup, we repeat this process to get the index.  Then we compare
the tag field of the line we are looking up to one or more tag fields
stored in the cache.  If we find a match, then we have a cache hit.

Lecture 3 remarks

Learn the whole lecture thoroughly.
```

- Previous message: [comp228-w16] extra POD sessions are available (even for people who work!)
- **Messages sorted by:** [date] [thread] [subject] [author]

More information about the comp228-w16 mailing list