Chapter 8 - Survey of Superscalar Processors

Last updated: May 6, 2004

1. *Although logic design techniques and microarchitectural tradeoffs can be treated as independent design decisions, explain the typical pairing of synthesized logic and a brainiac design style versus full custom logic and a speed-demon design style.*

   Performance argues for both (a) doing a lot of work per cycle (brainiac) and (b) short cycle times (speed demon). Managing complexity and reducing time to market argues for doing at most one of these.

   Examples:

   - IBM POWER designs illustrate choice (a). They traditionally followed the brainiac approach and used short pipelines and standard-cell circuit designs. The result was relatively high IPC but low clock frequency.
   - Alpha designs, on the other hand, traditionally followed (b), the speed demon path. The first designs had relatively simple but deep pipelines and simple superscalar issue logic. The designers instead made a large investment in full-custom circuit design and layout. Even the more complex 21264 out-of-order design had several simplifying design approaches to reduce its microarchitectural complexity (and verification effort).

2. *In the late 1950s, the Stretch designers placed a limit of 23 gate levels on any logic path. As recently as 1995, the UltraSPARC-I was designed with 20 gate levels per pipe stage. Yet many designers have tried to drastically reduce this number. For example, the ACS had a target of five gate levels of logic per stage, and the UltraSPARC-III uses the equivalent of eight gate levels per stage. Explain the rationale for desiring low gate-level counts. (You may also want to examine the lower level-count trend in recent Intel processors, as discussed in Hinton, et al. [2001].)*

   The number of gate levels per stage affects the clock frequency. Fewer gate levels means a higher clock.

   ACS chose five gate levels to try to reach a 100 MHz clock rate goal (10 nsec cycle time) in the mid-1960s, and the UltraSPARC-III clock target was to break the 1 GHz barrier for SPARC implementations some thirty-five years later.

   Interested students might want to follow up on this topic by reading some of the research papers on optimal pipeline depth. For example, the first three papers of ISCA 2002 were devoted to this topic.

   - A. Hartstein and T.R. Puzak, "The optimum pipeline depth for a microprocessor," pp. 7-13.
   - M.S. Hrishikesh, et al., "The optimal logic depth per pipeline stage is 6 to 8 FO4

inverter delays," pp. 14-24.
- E. Sprangle and D. Carmean, "Increasing processor performance by implementing deeper pipelines," pp. 25-34.

3. *Prepare a table comparing the approaches to floating-point arithmetic exception handling found on these IBM designs: Stretch, ACS, RIOS, PowerPC 601, PowerPC 620, POWER4.*

   - Stretch: precise exceptions using the lookahead as a history buffer
   - ACS: mode bit (slower execution with precise exceptions vs. faster execution with imprecise exceptions)
   - RIOS: mode bit
   - PPC 601: precise exceptions using instruction tagging
   - PPC 620: precise exceptions using the completion buffer
   - POWER4: precise exceptions using the global completion table

   [Note: The PPC architecture specifies a mode bit in the MSR, but the 601/620/POWER4 implementations are not affected by the mode setting.]

4. *Consider Table 8-2. Can you identify any trends? If so, suggest a rationale for each trend you identify.*

   Some trends include

   - no longer holding operand and/or result copies in the buffers - using a physical register file is simpler
   - increase in number of reorder buffer entries - deeper pipelines and wider issue increase the number of instructions in flight
   - instruction fusion (multiple instructions per reorder buffer entry) - using instruction groups allow easier management of a large number of instructions in flight

5. *Explain the market forces that led to the demise of the Compaq/DEC Alpha. Are there any known blemishes in the Alpha instruction set that make high-performance implementations difficult or inefficient? Is the Alpha tradition of full custom logic too difficult or resource intensive?*

   The design and fab cost of any leading-edge microprocessor is enormous. DEC to an extent and then later especially Compaq could not keep up with the investments in microprocessor design teams and fab facilities being made by competitors like IBM and Intel.

   DEC also failed to place Alpha in the mid-range processor market as opposed to only the high-performance processor market, even though there were attempts to port Windows NT to Alpha and later highly-integrated designs, such as the 21264PC.

   After buying DEC, Compaq decided to stretch out design cycles for Alpha and to lower performance targets. Finally, Compaq became convinced that the Intel Itanium would

dominate the high-end market and decided to cancel Alpha development in the summer of 2001.

There were no blemishes on the instruction set apart from the later addition of byte load/stores in the 21164A, which was done to help placement in the mid-range market rather than high-performance market.

Alpha designers argue that full-custom design is neither as difficult or as time-consuming as is typically thought (see the references in section 8.2.2). However, the Alpha design style typically resulted in high-wattage power dissipation and thus required extra box-level design attention to cooling. Some designers also caution that full-custom is not as robust as synthesized logic in the face of process shrinks.

6. *Compare how the Compaq/DEC Alpha and IBM RIOS eliminated the type of complex instruction pairing rules that are found in the Intel i960 CA.*

   The Alpha 21064 included a swap stage and the 21164 included a slotting stage immediately after instruction fetch in order to route instructions to certain dispatch slots. The 21264 was an out-of-order design, but it retained a degree of hardware preslotting logic.

   The IBM RIOS used wider dispatch paths to send instruction pairs to both the FPU and FXU. Fixed-point-only instructions were discarded by the FPU, and floating-point-only instructions were discarded by the FXU. A predecode tag attached to the instructions made the discarding choice easy.

7. *Explain the importance of caches in HP processor designs. How was the assist cache used in the HP 7200 both a surprise and a natural development in the HP design philosophy?*

   HP designers have always paid attention to commercial workloads (e.g., database and transaction processing). Because of the large working sets and thus relatively poor locality found in these types of applications, the 7xxx series designs typically started with a simple five-stage pipeline that was designed to run as fast as contemporary off-chip SRAMs. These off-chips caches were typically large and direct-mapped.

   The assist cache in the HP 7200 was surprising in that (1) it was on-chip, and (2) it was fully-associative.

   The assist cache in the HP 7200 was also a natural development since it reduced the pollution of the off-chip cache by (1) holding the data that exhibited spatial locality but not temporal locality, and (2) providing a buffer for data prefetching.

8. *Find a description of load/store locality hints in the Itanium Processor Family. Compare the Itanium approach with the approaches used in the HP 7200 and MIPS R8000.*

   - Itanium load/store/prefetch hint specifiers
     - (none) - temporal locality in all levels is assumed as default
     - .nt1 - non-temporal level 1

- .nt2 - non-temporal level 2
- .nta - non-temporal all levels

- HP 7200 - load/store hint for spatial locality only

- MIPS R8000 - floating-point data not held in on-chip cache

The IPF approach is an extension of the HP 7200 approach in which the temporal vs. spatial locality information is conveyed by the specific load/store instructions.

The MIPS R8000 approach associates poor temporal locality / good spatial locality with a data type. This is too rigid a distinction, and it also leads to coherency problems between the on-chip and off-chip caches that must be solved by additional hardware mechanisms (extra valid bits).

9. *Consider the IBM RIOS.*

   a. *The integer unit pipeline design is the same as the pipeline design used in the IBM 801. Explain the benefit of routing the cache bypass path directly from the ALU stage to the writeback stage as opposed to this bypass being contained within the cache stage (and thus having ALU results required to flow through the ALU/cache and cache/writeback latches as done in the simple five- and six-stage scalar pipeline designs of Chapter 2). What is the cost of this approach in terms of the integer register file design?*

      The advantage is that loads and stores that cause a cache miss can stall in the cache stage and subsequent, independent instructions can still continue to execute and writeback. Of course, the first subsequent instruction that needs the loaded value will stall the complete pipeline (i.e., a data hazard), or the next subsequent load/store will stall the complete pipeline (i.e., a structural hazard). The design is a simple form of non-blocking.

      The cost is an extra write port on the integer register file.

   b. *New physical registers are assigned only for floating-point loads. For what types of code segments is this sufficient?*

      The RIOS was designed with double-precision floating-point inner loops in mind, such as matrix inner product. The renaming of destination registers for floating-point loads allows the hardware to overlap the iterations of these types of loops. If these registers were not renamed, high performance would require loop unrolling or software pipelining to be done by the compiler. On the other hand, renaming all registers is more general and is not required for these types of loops.

      If a student is interested in more details, here is an illustration of a loop that performs x[i] = x[i] + c*y[i] (similar to examples from the Weiss and Smith book)

```
loop: lfd   fp0 = x(r1)         // load x[i]
      lfdu  fp2 = y(r2,8)       // load y[i], update ptr
      fma   fp0 = fp2,fp0,fp4   // x[i] = x[i] + c*y[i]
      stfdu x(r2,8) = fp0       // store x[i], update ptr
      bc    loop,ctr!=0

stages:
```

```
  F - fetch
  S - dispatch
  P - partial decode
  R - rename
  M - multiply
  A - add
  W - writeback
  D - decode
  E - execute
  C - cache

Note that loads and stores go to both pipelines (FXU and FPU).

      1  2  3  4  5  6  7  8  9  10 11 12 13
      -  -  -  -  -  -  -  -  -  -  -  -  -

lfd   F  S  P  R  .  W                      \
                  D  E  C                   |
lfdu  F  .  S  P  R  .  W                   |
                  D  E  C                   | iteration 1
fma   F  .  S  P  R  D  M  A  W             |
stfdu F  .  .  S  P  R  .  D                |
                  D  E  .  .  .  C          |
bc       F  .  S                            /

lfd               F  S  P  R  .  W          \
                              D  E  C       |
lfdu              F  .  S  P  R  .  W       |
                     D  E  C                | iteration 2
fma               F  .  S  P  R  D  M  A  W  |
stfdu             F  .  .  S  P  R  .  D    |
                                 D  E  .  .  .  C  |
bc                   F  .  S                /
```

If the floating-point registers were not renamed, then the first load of the second iteration could not start until cycle 11. Smith and Weiss call the overlap allowed by renaming the ability to "telescope" the pipeline flow.

c. *Draw a pipeline timing diagram showing that a floating-point load and a dependent floating-point instructions can be fetched, dispatched, and issued together without any stalls resulting.*

The pipelines support zero-cycle load/use delay for floating-point loads. The extra stages at the front of the FPU pipeline overlap the load and allow execution to begin just at the point the loaded data is returned.

```
see the pipeline stage names above

lfd  F  S  P  R  .  W
           D  E  C
                 \
fma  F  S  P  R  D  M  A  W
```

10. *Why did the IBM RIOS provide three separate logic units, each with a separate register set? This legacy has been carried into the PowerPC instruction set. Is this legacy a help, hindrance, or inconsequential to high-issue-rate PowerPC implementations?*

The original design used three separate units to simplify dependency checking. That is,

instructions for different units were by definition independent. The legacy is helpful for high-issue-rate implementations since it reduces the need for a high number of read ports on a combined register file.

11. *Identify market and/or design factors that have led to the long life span of the Intel i960 CA.*

    The i960 CA was designed for the embedded market where its performance level has been adequate for many years.

12. *Is the Intel P6 a speed demon, a brainiac, or both? Explain your answer.*

    It is both. The P6 had an unusually deep pipeline for its time, and it also went to great effort in decoding x86 instructions into uops and dynamically scheduling the uops so that a lot of work could be done each cycle.

13. *Consider the completion/retirement logic of the PowerPC designs. How are the 601, 620, and POWER4 related?*

    The common thread is that groups of multiple instructions are formed in the front-end of the pipeline and the instructions in these groups must retire together.

    - PPC 601 - Instruction groups of up to three instructions are formed by linking up to two other instructions with an anchoring integer instruction. This is done by special tagging logic and ensures that these instructions retire together.
    - PPC 620 - The 620 alters the typical completion/reorder buffer logic of the 604 by allocating and releasing completion buffer entries in pairs; thus, this forms instruction groups of two instructions each.
    - POWER4 - Instruction groups of up to five instructions are formed relative to a branch instruction. Each group is allocated a single entry in the global completion table.

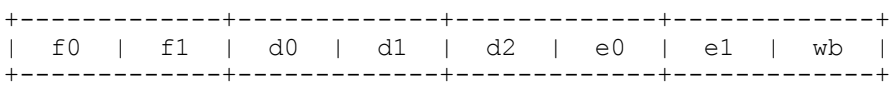
    [If a student is interested in more details on the PPC 601, refer them to the 1994 IEEE Micro paper by Potter, et al.]

14. *Draw a pipeline timing diagram illustrating how the SuperSPARC processor deals with a delayed branch instruction and its delay slot instruction.*

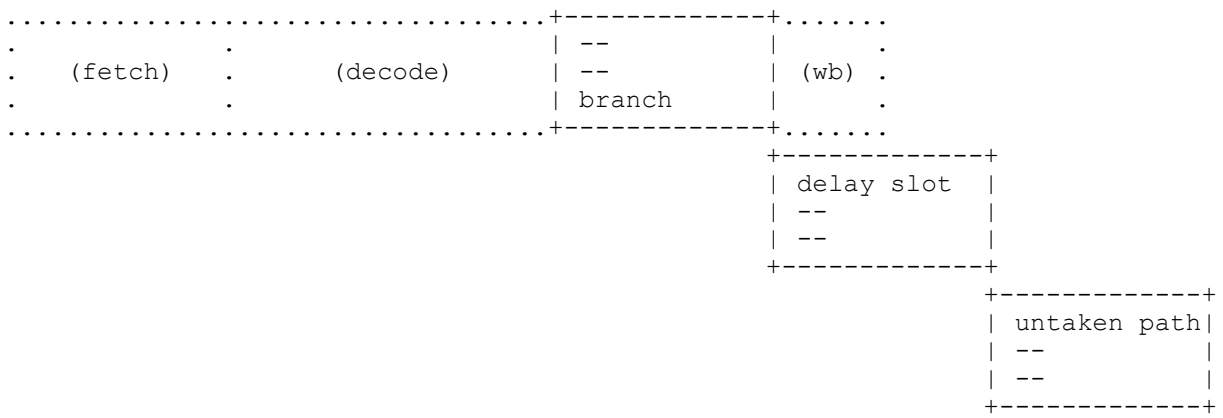
    The key idea is that instructions flow through the execute stage with the delay slot instruction placed in a separate instruction group than the branch instruction.

    The SuperSPARC pipeline is four stages, with stage alignment on some half-cycle boundaries.

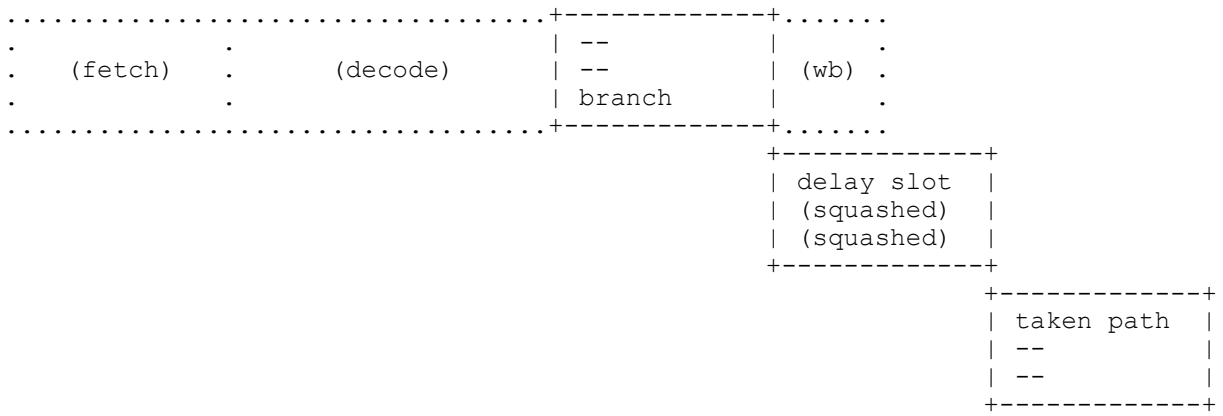
```
+-------------+-------------+-------------+-------------+
|   cycle 0   |   cycle 1   |   cycle 2   |   cycle 3   |
+-------------+-------------+-------------+-------------+
|    fetch    |       decode       |   execute   |  wb  |
```



untaken branches (boxes are instruction groups)



taken branches (boxes are instruction groups)



15. *The UltraSPARC-I provides in-order completion at the cost of empty stages and additional forwarding paths.*

    a. *Give a list of the pipe stage destinations for the forwarding paths that must accompany an empty integer pipeline stage.*

        1. integer execute stage (integer instructions)
        2. ld/st address stage (address calculations)
        3. ld/st data stage (store instructions)

    b. *List the possible sources of inputs to the multiplexer that fronts one leg of the ALU in the integer execution pipeline stage. (Note: This is more than the empty integer stages.)*

        1. int execute / int wait 1 latch
        2. int wait 1 / int wait 2 latch
        3. int wait 2 / int wait 3 latch

4. int wait 3 / exception check latch
5. exception check / writeback latch
6. ld/st data / ld/st miss latch

c. *Describe how the number of forwarding paths was reduced in the UltraSPARC-III, which had even more pipeline stages.*

The UltraSPARC-III uses a variant of the future file. Results are written into the working register file as soon as they are available, and instructions are dispatched when needed operands can be read from the working register file. Reading from the working register file at the time of dispatch eliminates WAR hazards. Age bits are added to the register identifiers to eliminate WAW hazards.