548

Lecture 5

Terminology

- FLR = Register
- CDB = Result bus
- *sink* = destination register
- FLB/SDB = store-queue
- FLOS = issue window / decode buffer

3 requirements...?

- Must exploit parallelism
  - High utilization
  - Pipelining
  - Recognize independence
- Must be correct (preserve dependence)

What is a “reservation station”?

- register for functional units that maintains multiple active instructions
  - operands and their tags
  - busy / occupied bit
  - destination tag (register name)
  - operation (or opcode)
- The collection of them is the issue-buffer

Two techniques

- One way: Rename, Reg-Read, Issue-Buffer, Ex, Completion (back to IB)
- Another way: Rename, Issue-Window, Reg-Read, Ex, Completion Buffer

Mem->Decode

- WaitFor
  - Must have bits! (the instruction itself)
  - Need space in the FLOS
- Do
  - Put it in the FLOS

Issue Window -> Issue

- WaitFor
  - Space in the appropriate reservation station
- Do
  - Read operands that you can
  - for busy operands send the tag
  - set the destination busy and with the reservation station tag

Issue ->Execute

- WaitFor
    - All input operands
    - For functional unit to be free
- Do
    - Execute it!

Execute -> Complete

- WaitFor
  - Our time on the CDB (prefetch request 2 cycles ahead)
  - (finish executing... less)
- Do
  - Write results + tag
  - If you write to the register, clear busy bit

Advantages?

- Remove WAW and WAR false (output name) dependencies
- Exposes ILP instruction level parallelism
- Intellectually simpler

Disadvantages?

- Distributed reservation stations are inefficient
- No precise exceptions
- Don’t really solve TSL problem (total-store-load)

TSL - Total Store-Load order

- This order must be maintained:
  - e.g. SLLLSLLSSLSSSLLL
- Why?
  - Imperative languages

Why not ~ inf registers?

- $/ns
- loops
- procedure calls

HPSm - innovations?

- dependent instruction bundles
  - e.g. ADD r5, r6 -> r12; STORE r12, @r3
  - e.g. ADD r5, r6 -> r3; STORE r12, @r3
- Precise exceptions
  - Commit architectural state in order at the end
- Handling mispredictions in OOO

What is up with r12?

How to handle precise exceptions?

How to handle memory aliasing?