18-747 Lecture 7: More Modern Micro-Dataflow

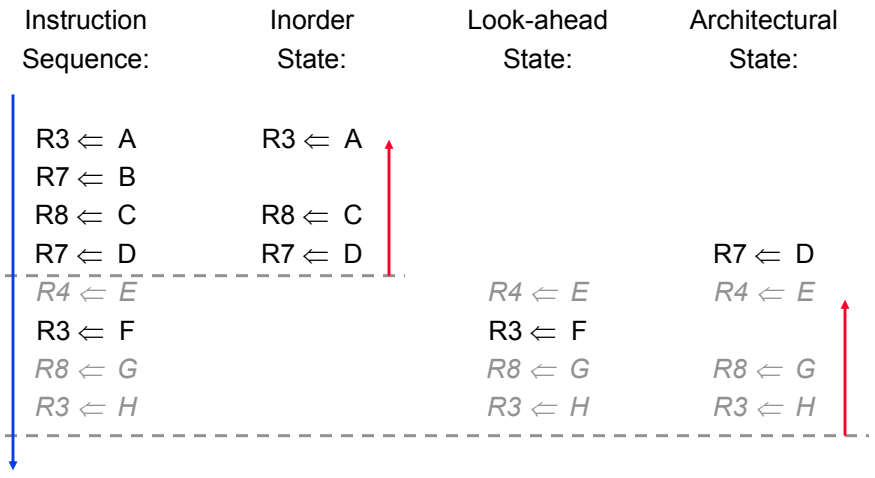
James C. Hoe
Dept of ECE, CMU
September 19, 2001

Reading Assignments: MJ Ch7, Monday's handout
"Microarchitecture of Superscalar Processors"
by Smith and Sohi

Announcements: PS1 and Project 0 due 2:30 Friday

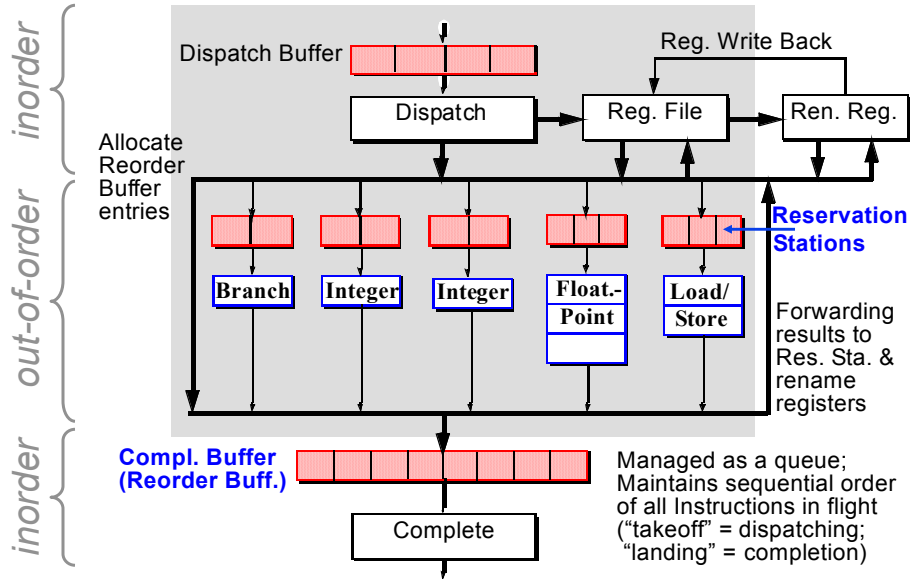
Handouts:

Out-of-Order Machine State

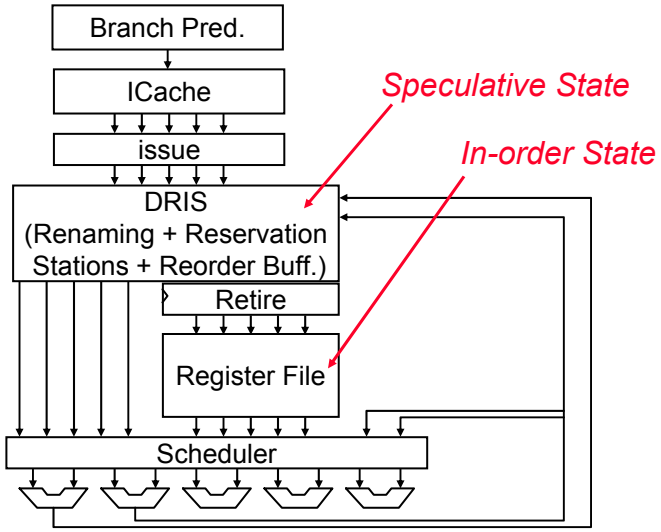


gray=dispatched but not yet executed instructions

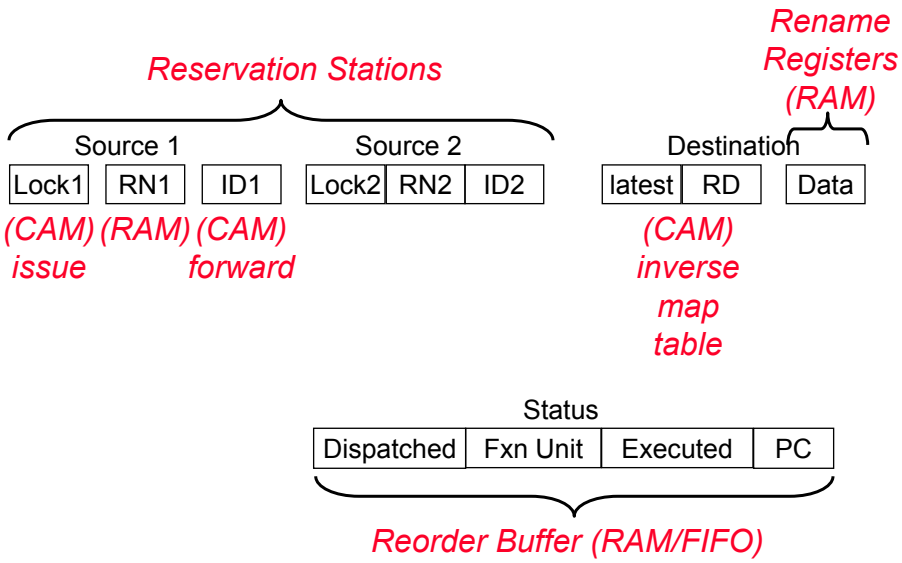
Elements of Modern Micro-dataflow



Metaflow Datapath



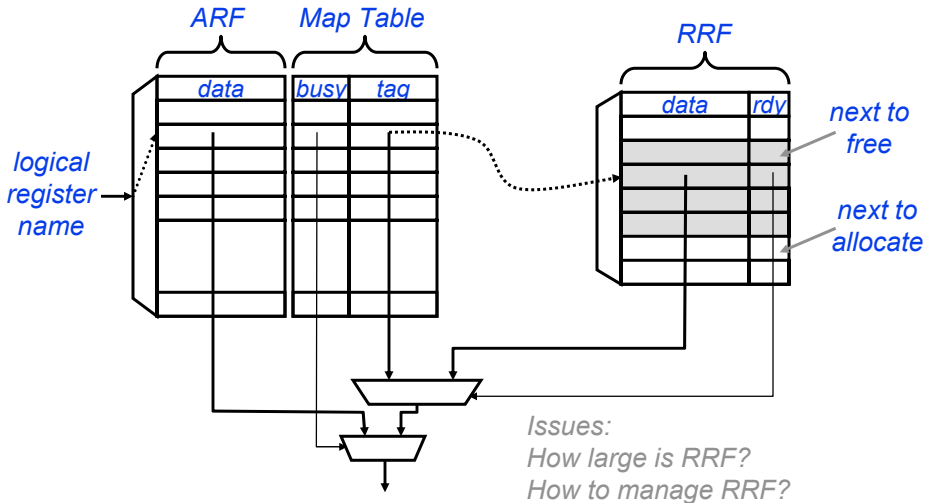
Logical vs. Physical Organization:
Metaflow Example



Elements of Register Renaming

- A pool of extra registers
  - managed as temporary, single-assignment registers
    (eliminates WAW and WAR)
  - logically separate from architectural registers named in ISA
  - many physical organizations are possible
- An allocation and mapping mechanism
  - given a logical/architectural register name, where is its current definition *(value, location, ready or not)*
  - given a logical/architectural destination register name,
    1. how to find an unused rename register
    2. how to establish a new mapping for later references
  - when to reclaim a mapping? when to reclaim a register?

Register Map Table
with Separate Architectural and Rename Registers



Issues:
How large is RRF?
How to manage RRF?
How to recover the Map Table on br. mispredict & exceptions?

Pentium-Pro

How to recover the Rename Map

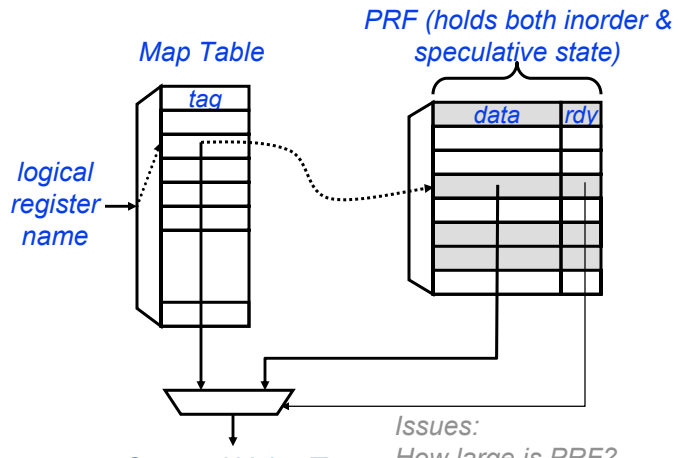
- Metaflow DRIS: *remembers everything about all outstanding instructions*
  - associative lookup always gives the right answer from the perspective of the youngest instruction
- Map Table: *rename history of a logical register is overwritten after each WAW*

  Solutions:

  1. Discard all speculative state and revert to default ARF mapping
  2. Record the rename history
     - add another field to RRF to remember the last rename register mapped by the same logical register
     - map table can be rebuilt by tracing backwards sequentially in the active RRF region *(recall RRF is "program-ordered")*
  3. Take snapshots of the entire rename table at strategic times, i.e. when predicting branches

What about exceptions?

Register Map Table
with unified Physical Registers



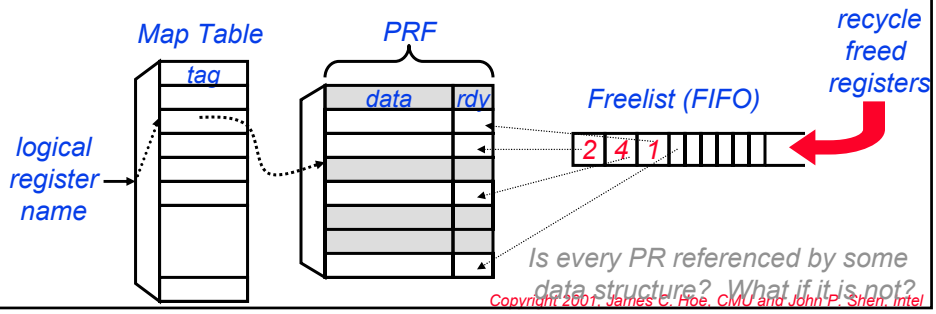
Issues:
How large is PRF?
How to manage PRF?
How to recover the Map Table on br. mispredict. & exceptions?

R10000, P4

Allocation and De-allocation of PRF

- When speculative register state is committed, no copying is required *(in fact, not much happens)*
- PRF registers are not de/allocated in order *Why?*
- Freelist - stores a pool of unused registers
- When to recycle a physical register?

  i.e. when do we know a register is no longer referred to (by logical name or by physical name)



Is every PR referenced by some data structure? What if it is not?

How to recover the Rename Map

- Map Table: *rename history of a logical register is overwritten after each WAW*

  Solutions:

  1. ~~Discard all speculative state and revert to default ARF mapping~~
  2. Record the rename history
     - ~~add another field to RRF to remember the last rename register mapped to the same logical register~~
     - map table can be rebuilt by tracing backwards, one at a time
     - Rename history must be stored with instruction entries in ROB

     Also useful for reclaiming physical registers for reuse!
  3. Take snapshots of the entire rename table at strategic times, i.e. when predicting branches

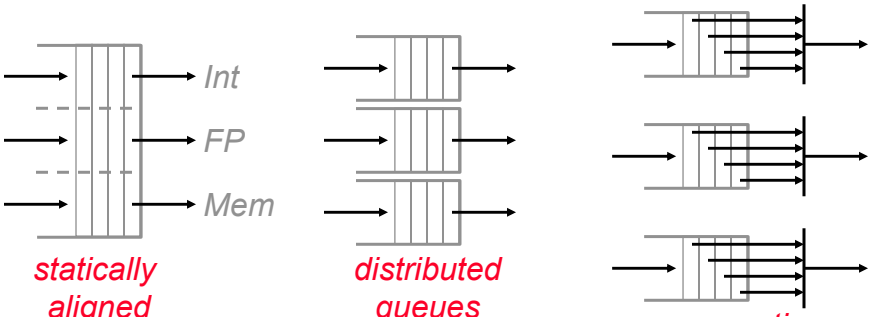
     What about exceptions?

Reservation Stations / Instruction Queues

- A place where instructions can wait for dataflow resolutions
  - unresolved operand values are represented by a unique tag *(usually the name of the "renamed" operand register or the instruction to produce the operand value, sometimes synonymous)*
  - a completing instruction can forward is result to all reservations holding the right tags
- Management policies
  - single vs. multi buffers
  - random access or FIFO
  - age-prioritized
  - value vs. tag only
  - when to deallocate

  Reservation stations for Ld/St units are more complicated

Queue Organizations



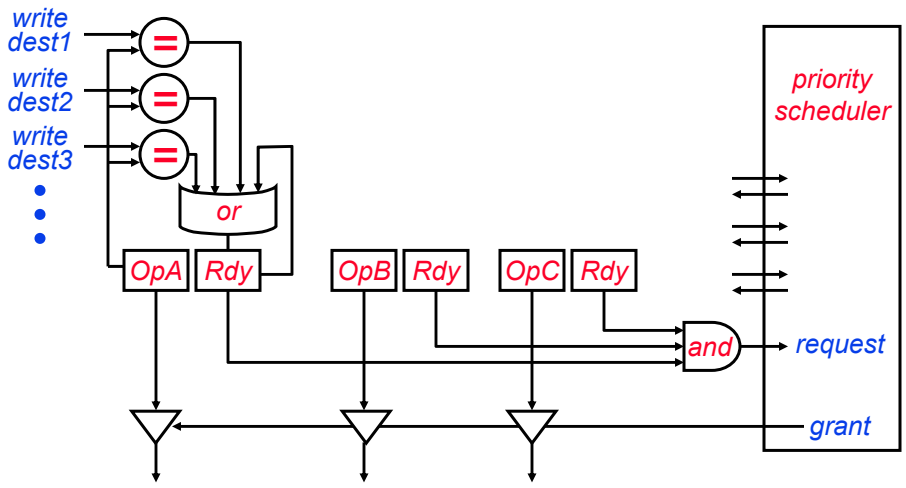
statically aligned (inorder)

distributed queues

reservation stations

[Smith and Sohi]

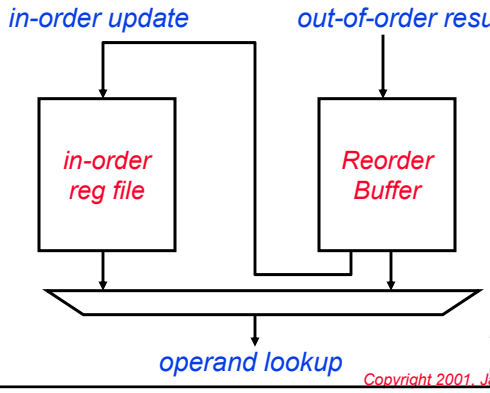
Dataflow Dependence & Scheduling



To PRF read ports

Reorder Buffer

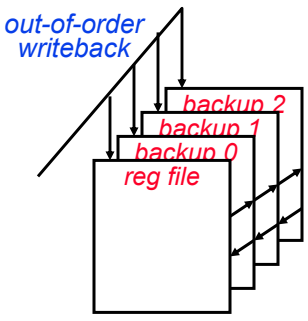
- An program-ordered record of all outstanding (out-of-order) instructions
  - delay update of in-order program state by instructions completing out-of-order
  - allow inorder commit of in-order program state



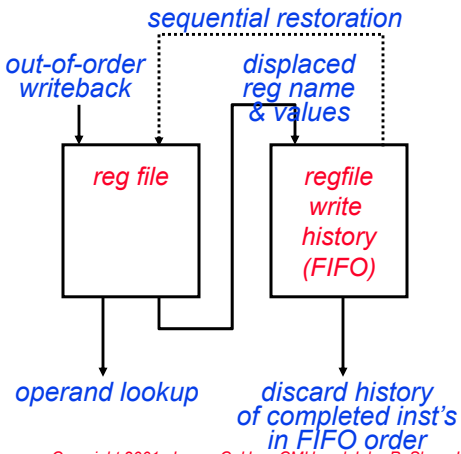
Don't forget about PC and memory writes!!

Tracking the Three States: *Other Options*

- Keeping history to rewind and restart execution
  - rewinds to special points (i.e. branch instructions) in the program by checkpointing
  - rewinds to arbitrary points (incremental history and recovery)



Checkpoint Stack



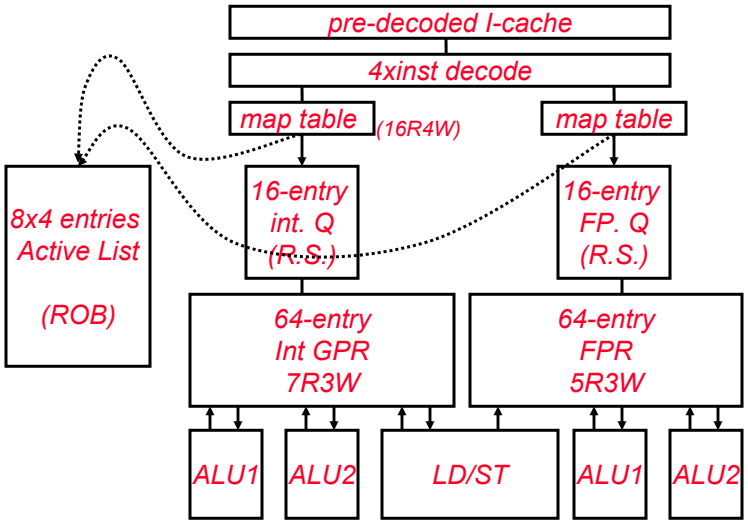
MIPS R10000 *circa 1996*

- 4-way superscalar
  (fetch, decode, dispatch and complete)
- 5 execution pipelines *(2 Int, FP Add, FP Mult, Ld/St)*
- Micro-dataflow instruction scheduling
  (16+16 instruction window)
- Register renaming + memory renaming
  64 physical integer registers to hold 33 logical registers + renamed registers
- Speculative execution pass 4 unresolved branches
- Precise Interrupts

Design Choices

- Register Renaming
  - map table lookup + dependency check on simultaneous dispatches
  - unified physical register file
  - 4-deep branch stack to backup the map table on branch predictions
  - sequential (4-at-a-time) back-tracking to recover from exceptions
- Instruction Queues
  - separate 16-entry floating point and integer instruction queues
  - prioritized, dataflow-ordered scheduling
- Reorder Buffer
  - one per outstanding instruction, FIFO ordered
  - stores PC, logical destination number, old physical destination number *Why not current physical destination number?*

MIPS R10000



Flow Path Model of Superscalars