18-747 Lecture 6: Modern Micro-Dataflow

James C. Hoe
Dept of ECE, CMU
September 17, 2001

Reading Assignments: MJ Ch7, Last Wednesday's and today's handout

Announcements: HW1 and Project 0 due 2:30 Friday

Handouts: *"The MIPS R10000 Superscalar Microprocessor"*

In-order State and Precise Interrupt

- If an IBM 360/91 instruction causes an exception, can we stop the processor in a precise state?

i: R4 ← R0 x R8
j: R2 ← R0 + R4 Exception!!
k: R4 ← R0 + R8
l: R8 ← R4 x R8

- By the time *j* executes, *k* has already updated *R4? How do you rewind the register file to the state just after i?*

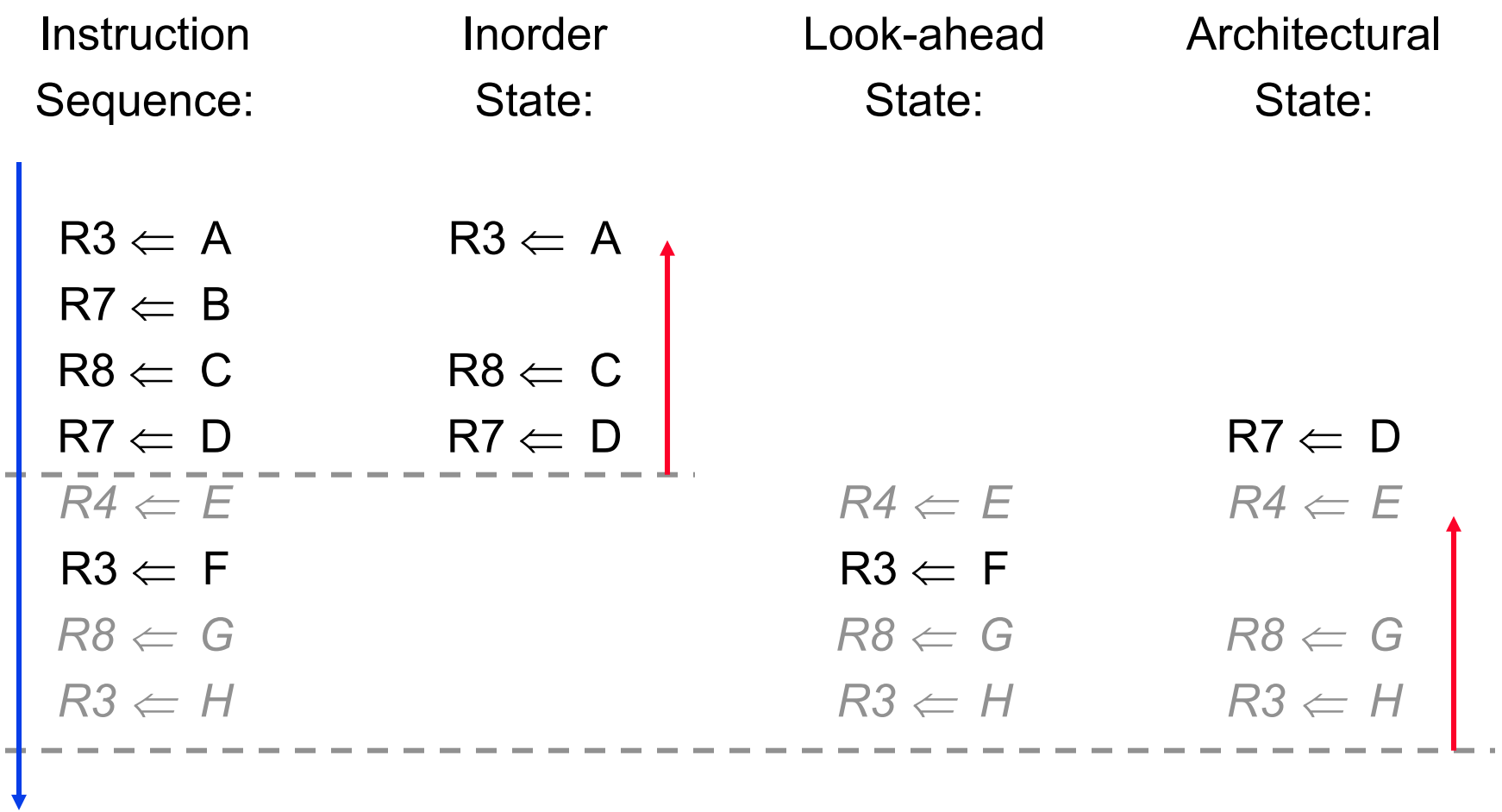
Recall, i never even got to update R4!!

- Next time, how to maintain an "in-order" state of the machine. (In-order state = the machine state as viewed by the first not-yet-completed instruction.)

Modern Enhancements to Tomasulo's Algorithm

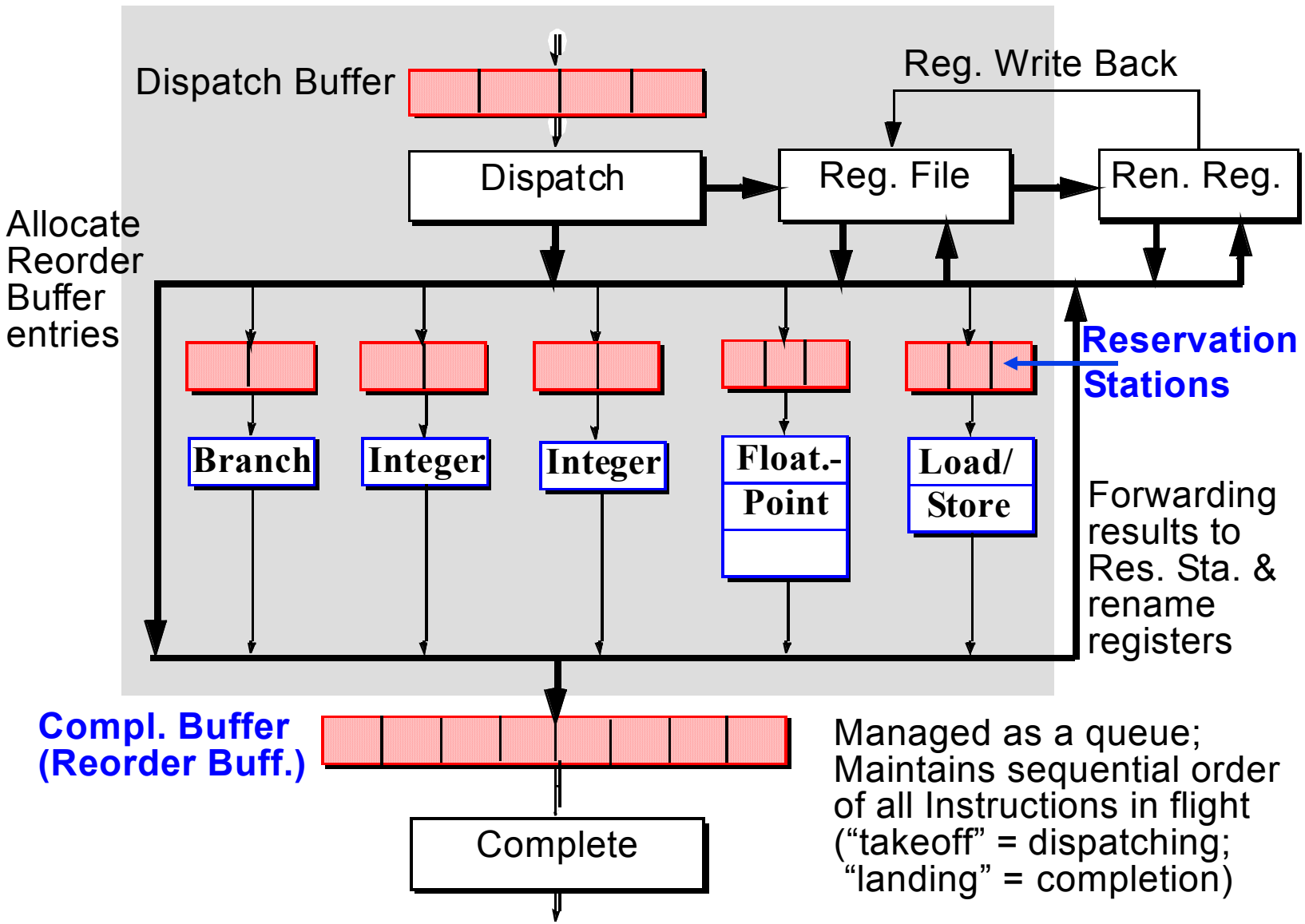
	Tomasulo	Modern
Machine Width -	Peak IPC = 1	"Peak" IPC = 8
(Structural Dep.)	2 F.P. functional. units	6-10 functional units
	Single CDB	Many forwarding buses
Anti-Dep. -	Operand copying	Renamed register
Output Dep. -	Reserv. Station Tag	Renamed register
True Data Dep. -	Tag-based forwarding	Tag-based forwarding
Exceptions -	Imprecise	Precise (Require ROB)

Out-of-Order Machine State



gray=dispatched but not yet executed instructions

Elements of Modern Micro-dataflow



Steps during Dynamic Execution

- **DISPATCH:**
  - Read operands from Register File (ARF) and/or Rename Register File (RRF) *(RRF may return value or Tag)*
  - Allocate new RRF entry and rename destination register to it
  - Allocate Reorder Buffer (ROB) entry
  - Advance instruction to appropriate Reservation Station (RS)
- **EXECUTE:**
  - RS entry monitors result bus for rename register Tag(s) to latch in pending operand(s)
  - When all operands ready, issue instruction into Functional Unit (FU) and deallocate RS entry (no further stalling in execution pipe)
  - When execution finishes, broadcast result to waiting RS entries and RRF entry
- **COMPLETE:**
  - When ready to commit result into “in-order” state:
    1. Update architectural register from RRF entry, deallocate RRF entry, and if it is a store instruction, advance it to Store Buffer
    2. Deallocate ROB entry and instruction is considered architecturally completed

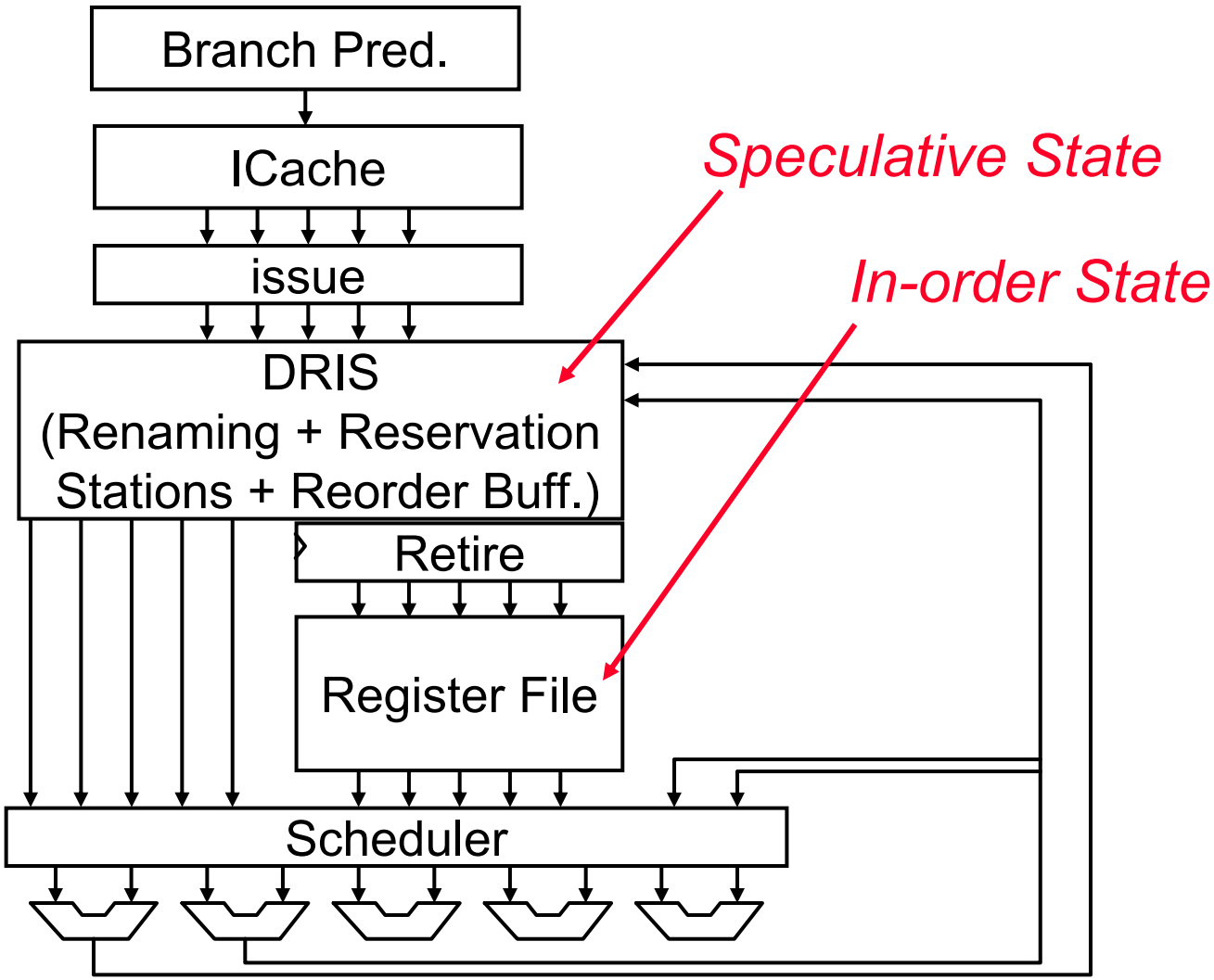
Metaflow Lightning SPARC Processor

- Superscalar fetch, issue, and execution
- Micro-dataflow instruction scheduling
- register renaming + memory renaming
- Speculative execution with rapid rewinding
- Precise Interrupts

circa 1991

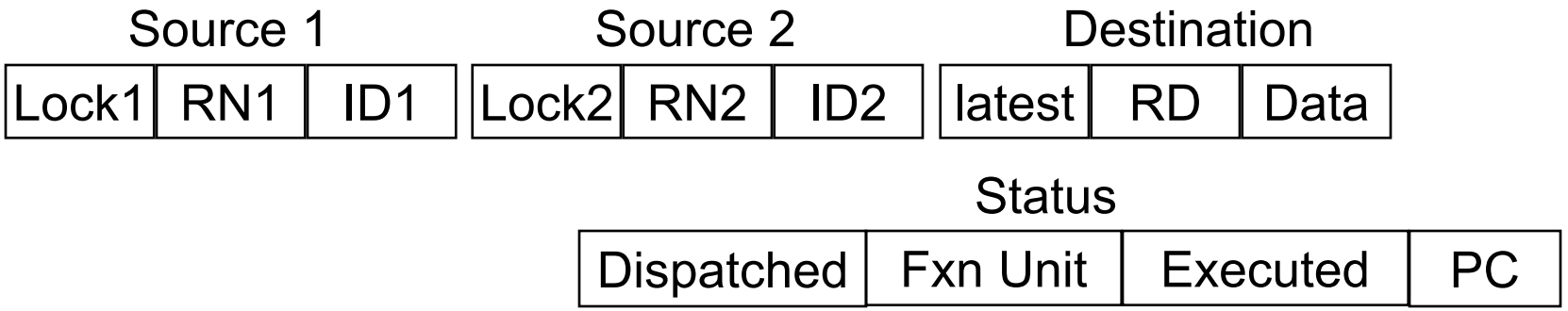
Claim: "Factor of 2-3 performance advantage from architecture"

Metaflow Datapath



Metaflow DRIS

- Deferred-scheduling Register-renaming Instruction Shelf (i.e. ROB + Rename Table + Reservation Stations)
- A storage array with multiported RAMs and CAMs (a.k.a. a very-very complicated register-file like thing)
- A DRIS entry is maintained for every instruction in flight.

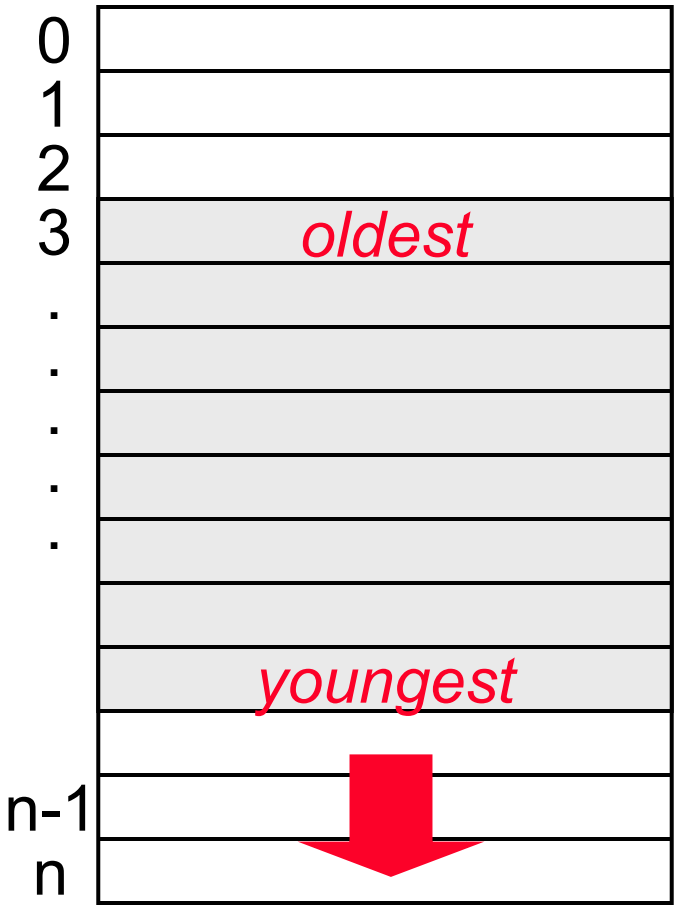


DRIS

- Circular Queue Structure

  Instructions stored in original program order

- New entries are allocated at the head of the queue as new instructions are issued
- Entries are committed in-order from the tail of the queue to the register file and memory



Issue*: (Rename+Decode)

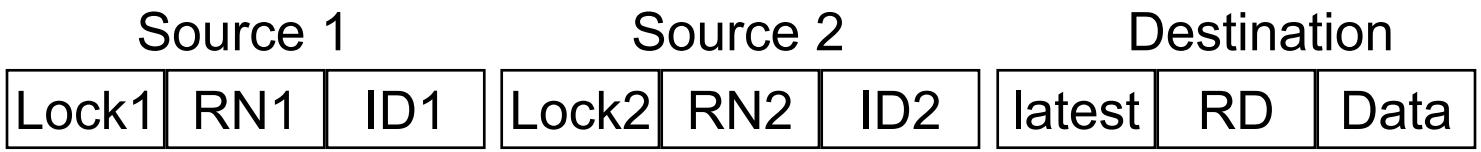
- A new ID (aka Tag) is allocated to each instruction when issued into DRIS

  ID is the index of the allocated DRIS entry location

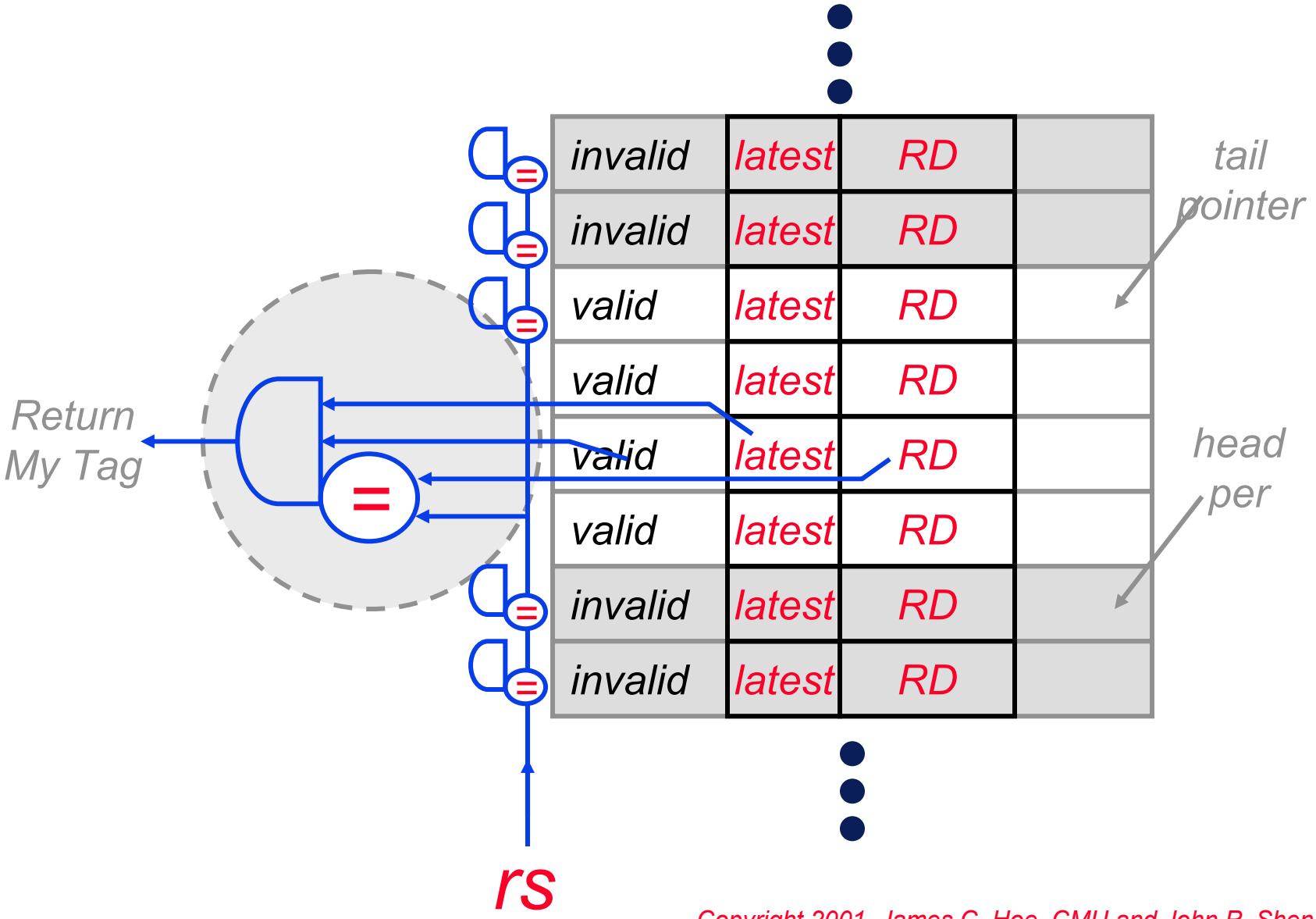
- The ID is used to refer to the result of that instruction

- Register operand lookup, add rd, rs, rt

1. Search DRIS to see if an older instruction has rs or rt as its destination. If so, rename the sources by setting the ID field.
2. If renamed, check to see if data are ready. If not, set the locked bit.



Associative Lookup



Micro-Dataflow Scheduling

- The scheduler dispatches according to
  - availability of pending instructions' operands
  - availability of the functional units
  - chronological order of the instructions

- Find the "oldest" *N* instructions such that

  !locked1[id] && !locked2[id] &&

  Dispatched[id]=*false* && Executed[id]=*false* &&

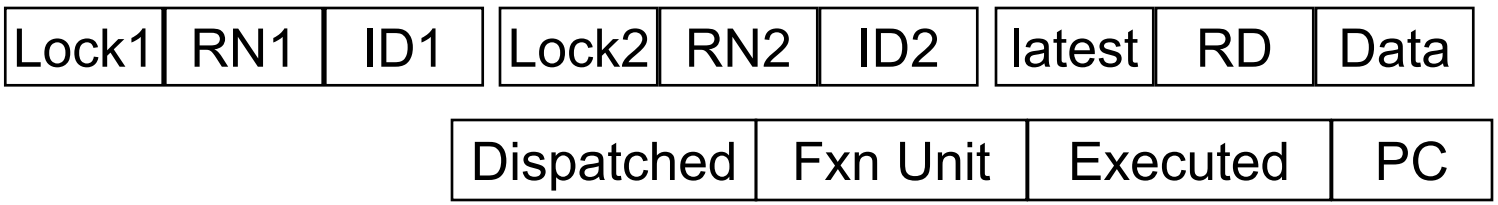
  notBusy(fxnUnit[id])

  Is "oldest-firsrt" always the best strategy?

- Dispatch and set Dispatch[id]=*true*

Dispatching*: add rd,rs,rt

- A dispatched instruction is sent to the functional unit with its operands and its *ID*
- Operands could come from:
  - DRIS: Data[IDx[*ID*]] *speculative state*
    when DRIS[IDx[*ID*]] is active
  - Register File: RF[RNx[*ID*]] *in-order state*
    when DRIS[IDx[*ID*]] is invalid or retired



Scheduling Memory Operations

- Memory data dependence *(RAW, WAR, WAW)*
- When to start a load instruction (on a uniprocessor)?
  - no more older store instructions in DRIS *or*
  - must know the addresses of all older stores in DRIS *or*
  - load speculatively and just *reload* if RAW hazard
- Storing to memory irrevocably changes the in-order machine state, therefore, a store instruction can only be executed when
  - it is the oldest instruction in DRIS *or*
  - all instructions before the store have completed and thus can no longer cause exceptions

(no unresolved/predicted branches)

Update

- A Fxn unit returns both the result and the associated *ID*
- The DRIS entry is updated

  Data[*ID*]=result ;

  Executed[*ID*]=*true* ;

- Enable other instructions that uses this result

  if (ID1[id]==*ID*) Locked1[id]=*false*;

  if (ID2[id]== *ID*) Locked2[id]= *false*;

Lock1	RN1	ID1	Lock2	RN2	ID2	latest	RD	Data

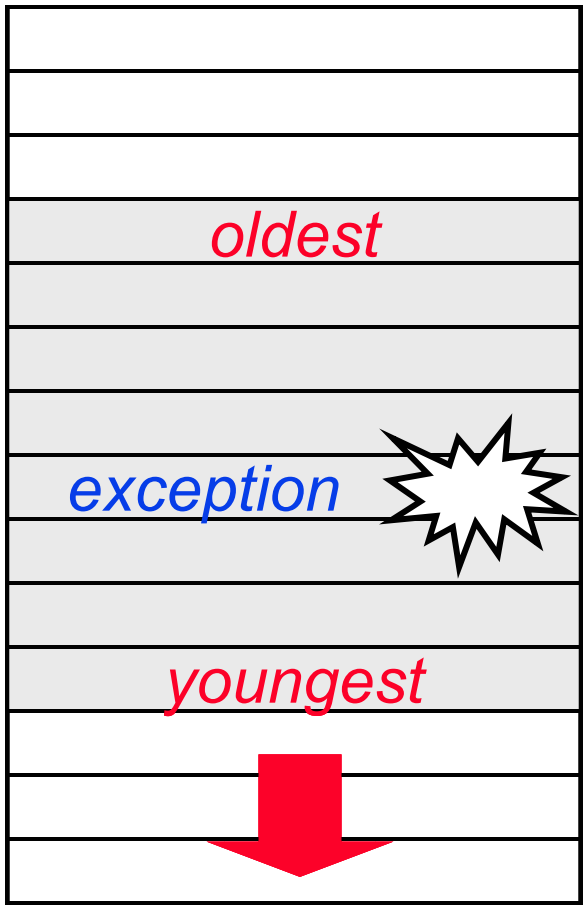
Retire

- Instructions retires strictly in-order from the oldest entry of the DRIS
- Data[*retiree*] is written (aka. committed) to the register file

  (speculative $\Rightarrow$ in-order state)
- Store instructions are only executed when retiring from DRIS

Precise Exceptions

- Discard all DRIS entries younger than the offending instruction
- How about older instruction that hasn't finished yet?
- When to start executing the interrupt handler?
  - Performance
  - Protection
  - An earlier exception?

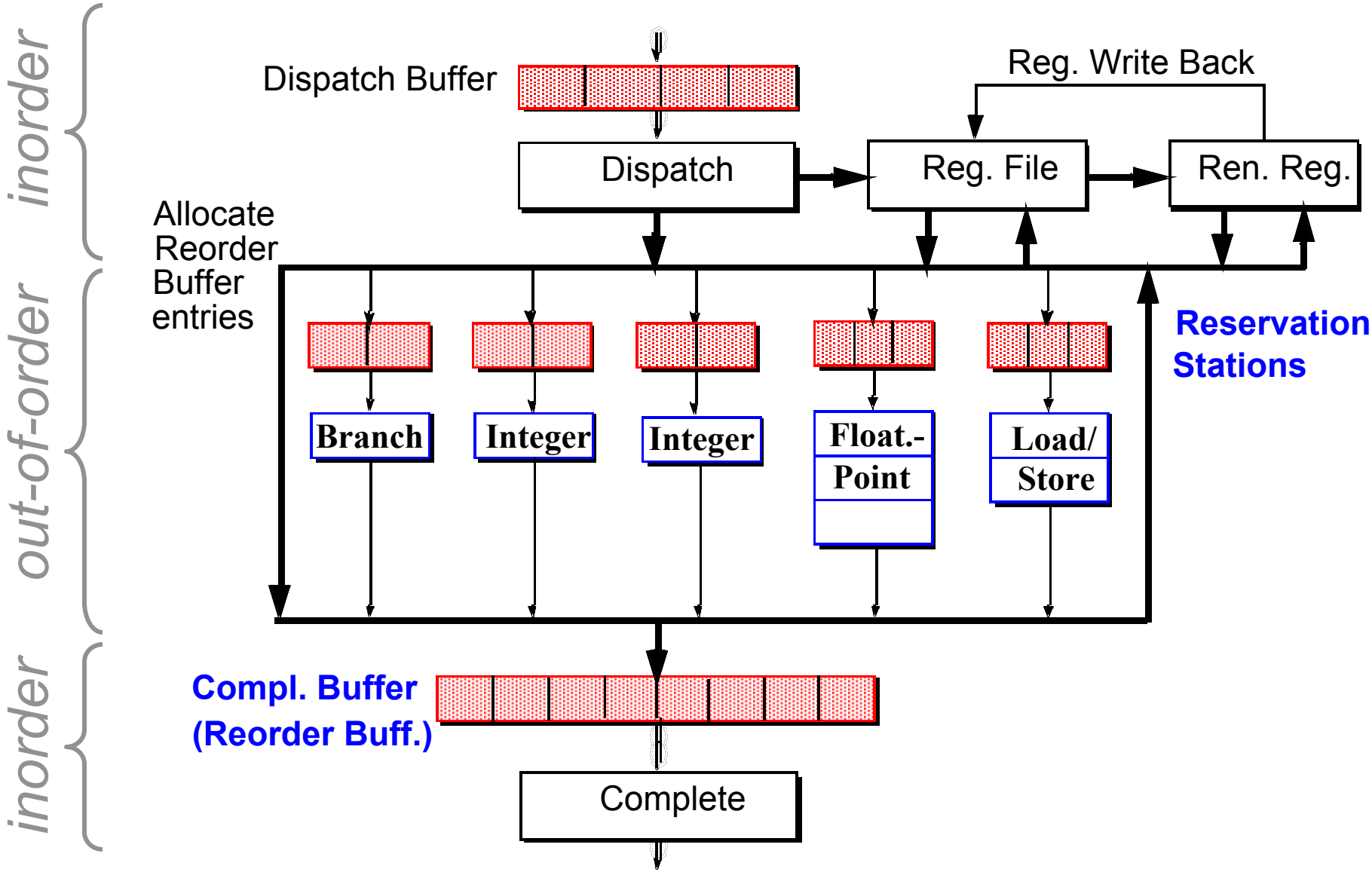


This works on branch misprediction too!!

The Cost of Implementing DRIS

- To support *N-way* issue into DRIS per cycle
  - *N*x3 simultaneous 5-bit associative lookups
- To support *N-way* dispatch per cycle
  - 1 prioritized associative lookup of *N* entries
  - *N*x2 indexed lookup in DRIS
  - *N*x2 indexed lookup in the GPR
- To support *N-way update per cycle*
  - *N* indexed write to DRIS
  - *N*x2 associative lookup and write in DRIS
- To support *N-way* retire per cycle
  - *N* indexed lookup in DRIS
  - *N* indexed write to GPR

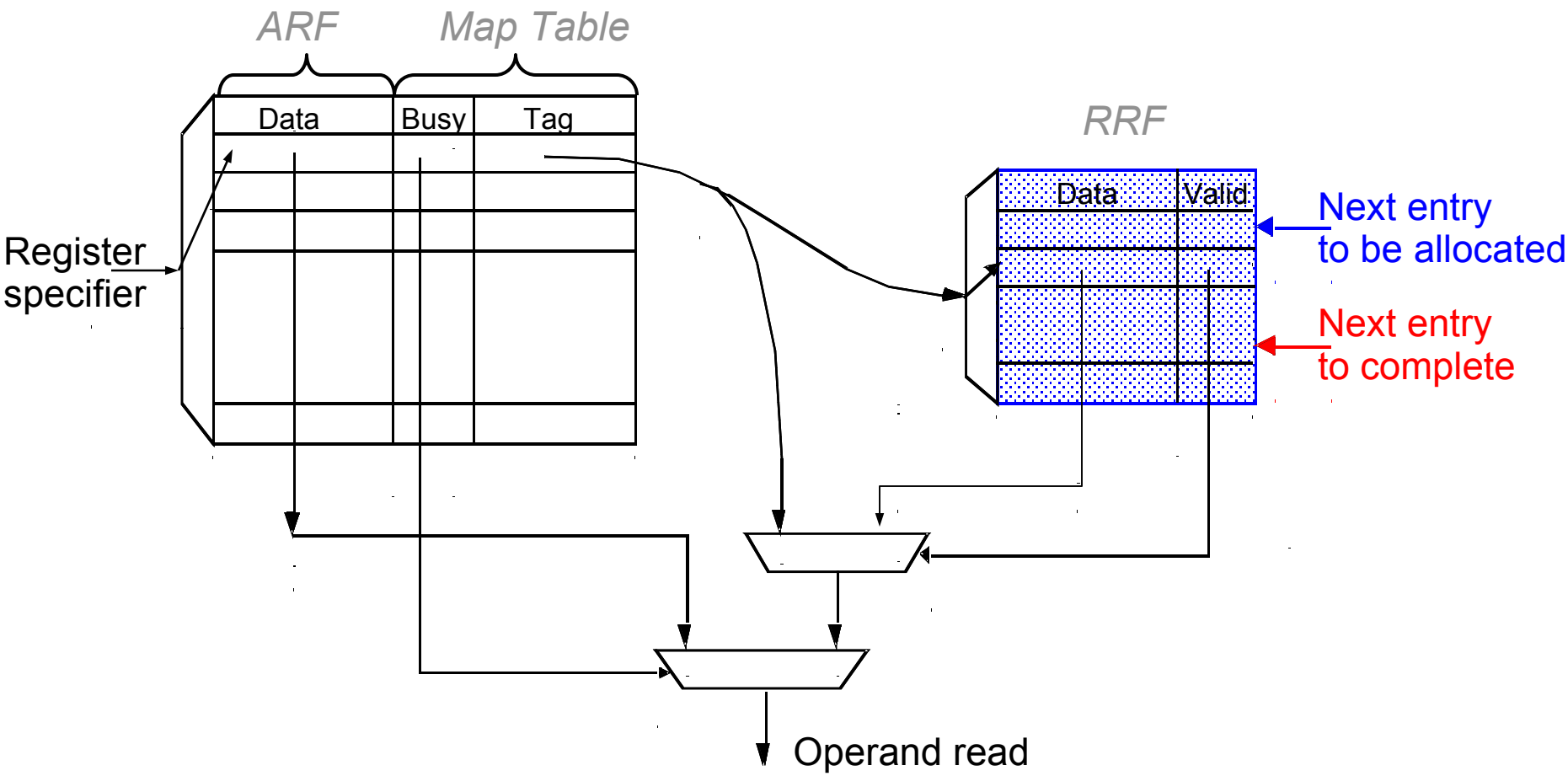
Decentralized Reordering Structure



Register Renaming Alternatives

- Number of rename registers
- Organization of rename registers
  - Separate rename register file
  - Pooled architectural/rename register file
- Allocation of rename registers
  - Fixed for each architectural register
  - Shared by all architectural registers
- Physical Location of rename registers
  - Attached to the architectural register file
  - Attached to the reorder buffer
- Methods for rename lookup

Register Renaming Mechanisms



What happens when you get an exception?

Register Renaming in the RS/6000

Incoming FPU instructions pass through a renaming table prior to decode

Physical register names only within the FPU!!

32 architectural registers ⇒ 40 physical registers

Complex control logic maintains active register mapping

FPU Register Renaming