

Dynamic Scheduling

Why go out of style?

- expensive hardware for the time (actually, still is, relatively)
- register files grew so less register pressure
- early RISCs had lower CPIs

Dynamic Scheduling

Why come back?

- higher chip densities
- out-of-order hardware design was generalized
 - greater need to hide other kinds of latencies as:
 - discrepancy between CPU & memory speeds increased
 - branch misprediction penalty increased from superpipelining
 - used a more general register renaming mechanism that included integers
- out-of-order hardware design was updated
 - need to preserve precise interrupts
 - therefore commit instructions in-order
- more need to exploit ILP
 - processors now issue multiple instructions at the same time

2 styles: large physical register file & reorder buffer
(R10000-style) (PentiumPro-style)

Register Renaming with A Physical Register File

Register renaming provides a **mapping** between 2 register sets

- **architectural registers** defined by the ISA
- **physical registers** implemented in the CPU
 - hold results of the instructions committed so far
 - hold results of subsequent instructions that have executed but have not yet committed
 - more of them than architectural registers
 - \sim issue width * # pipeline stages between register renaming & commit

Register Renaming with A Physical Register File

How does it work?:

- An architectural register is mapped to a physical register during a register renaming stage in the pipeline
 - destination registers create mappings
 - source registers in subsequent instructions use mappings
- After renaming, operands are called by their physical register number
 - values accessed using physical register numbers
 - hazards determined by comparing physical register numbers
 - results are written using physical register numbers

A Register Renaming Example

Code Segment	Register Mapping	Comments
<code>ld r7, 0(r6)</code> ...	<code>r7 -> p1</code>	<code>p1</code> is allocated
<code>add r8, r9, r7</code> ...	<code>r8 -> p2</code>	use <code>p1</code> , not <code>r7</code>
<code>sub r7, r2, r3</code>	<code>r7 -> p3</code>	<code>p3</code> is allocated <code>p1</code> is deallocated when <code>sub</code> commits

Register Renaming with A Physical Register File

Effects:

- reduces WAW and WAR hazards (*name dependences*)
- increases ILP

An Implementation (R10000)

Modular design with regular hardware data structures

Structures for register renaming

- 64 **physical registers** (each, for integer & FP)
- **map tables** for the **current** architectural-to-physical register mapping (separate, for integer & FP)
 - current means latest defined destination register
 - accessed with the architectural register number of a source operand
 - produces a physical register number for that operand
- a destination register is assigned a new physical register number from a **free register list** (separate, for integer & FP)

An Implementation (R10000)

Instruction “queues” (integer, FP & data transfer)

- contains decoded & mapped instructions with the current physical register mappings
 - instructions entered into free locations in the IQ
 - sit there until they are dispatched to functional units
 - somewhat analogous to Tomasulo reservation stations but no value fields
- used to determine when operands are available
 - compare physical register numbers of each source operand for instructions already in the IQ to physical register numbers of destination values just computed
- determines when an appropriate functional unit is available
- dispatches instructions to functional units

An Implementation (R10000)

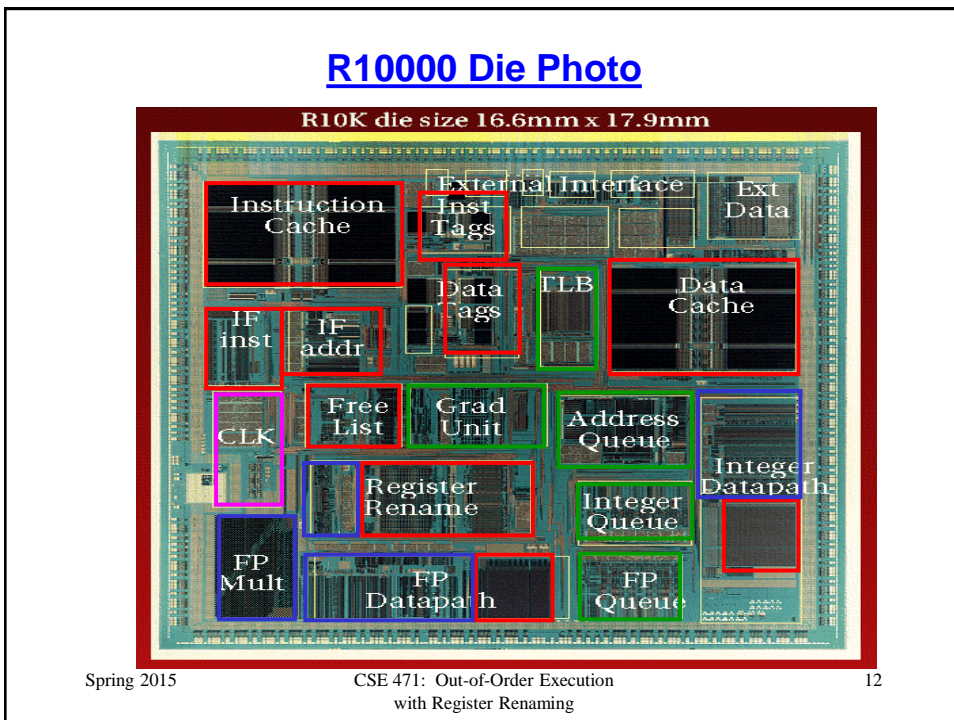
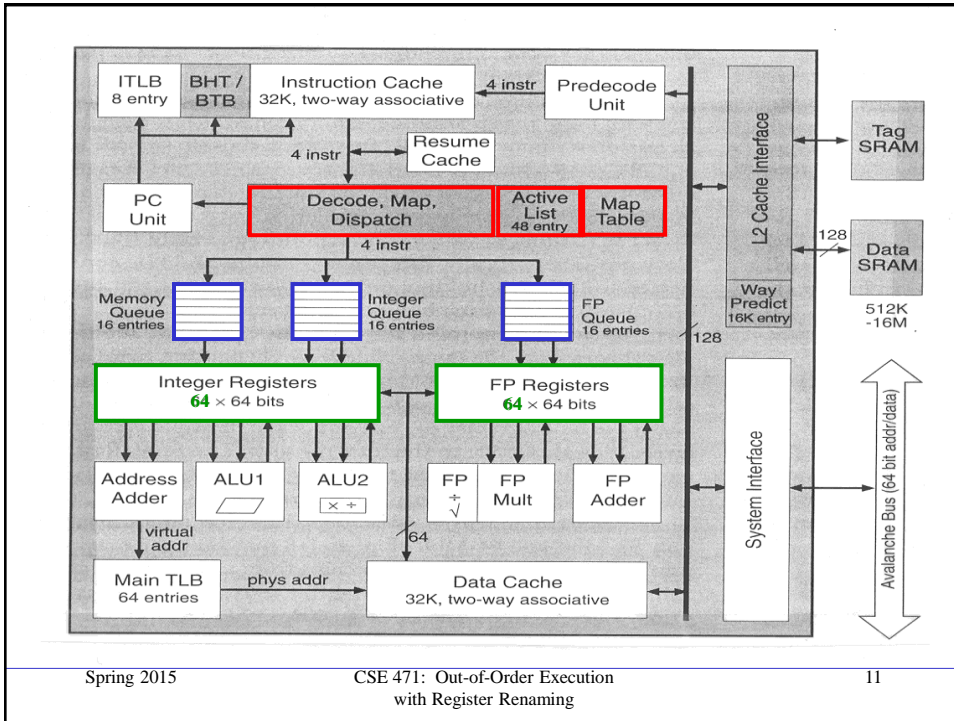
active list for all uncommitted instructions

- the mechanism for maintaining precise interrupts
 - instructions entered in program-generated order (in-order)
 - allows instructions to complete in program-generated order
- instructions are removed from the active list:
 - when they are committed - an instruction commits if:
 - the instruction has completed execution
 - all instructions ahead of it have **committed**
 - branch is mispredicted
 - an exception occurs
- contains the **previous** architectural-to-physical destination register mapping
 - used to recreate the map table for instruction restart after an exception
- instructions in the other hardware structures & the functional units are identified by their active list location

An Implementation (R10000)

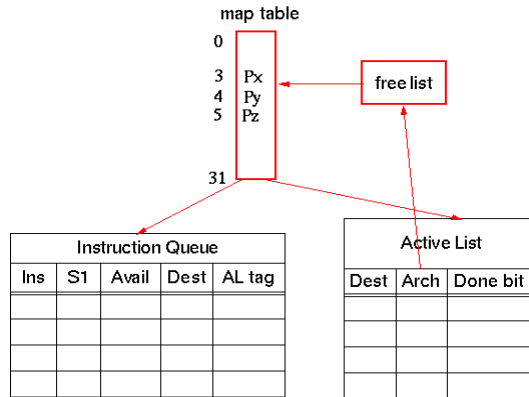
busy-register table (integer & FP):

- indicates whether a physical register contains a value
- somewhat analogous to Tomasulo's register status
- used to determine operand availability
 - bit is set when a register is mapped & leaves the free list (not available yet)
 - cleared when a FU writes the register (now there's a value)



The R10000 in Action 1

```
ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
add   A4, A3, reg    arch register A3 used
sub   A3, reg, reg   arch register A3 redefined
                        name dependence
or    A5, A3, reg    arch register A3 used
```



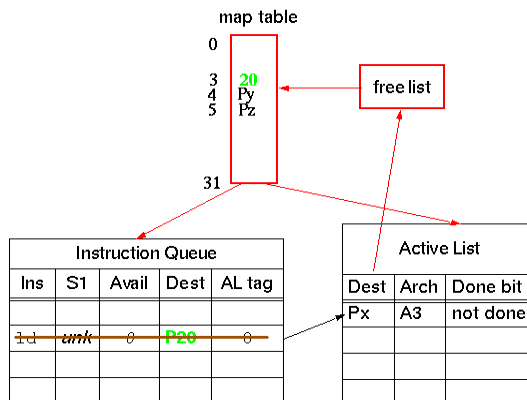
Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

13

The R10000 in Action 2

```
→ ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
add   A4, A3, reg
sub   A3, reg, reg
or    A5, A3, reg
```



Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

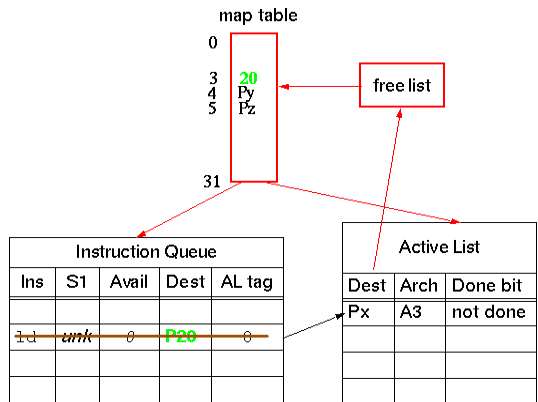
14

The R10000 in Action 3

```

ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
→ add  A4, A3, reg
sub   A3, reg, reg

or    A5, A3, reg
    
```



Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

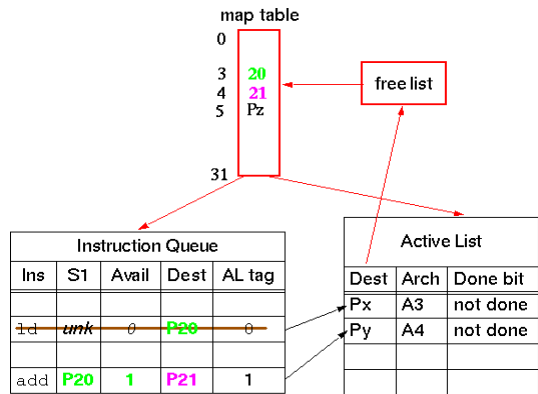
15

The R10000 in Action 4

```

ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
→ add  A4, A3, reg
sub   A3, reg, reg
                        arch register A3 used

or    A5, A3, reg
    
```



Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

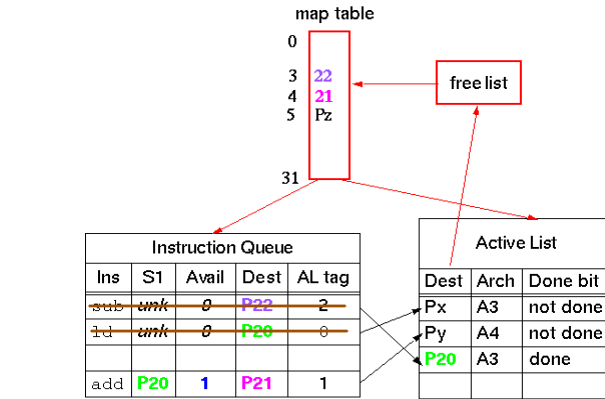
16

The R10000 in Action 5

```

ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
add   A4, A3, reg    arch register A3 used
sub   A3, reg, reg    arch register A3 redefined
                        name dependence
→ or  A5, A3, reg

```



Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

17

An Interrupt

Ld has a page fault

- Sweep backwards through the active list to the faulting instruction, one instruction at a time
- Unmap the registers, so no results are written
- The old mapping is used to reexecute the code sequence once the interrupt is serviced

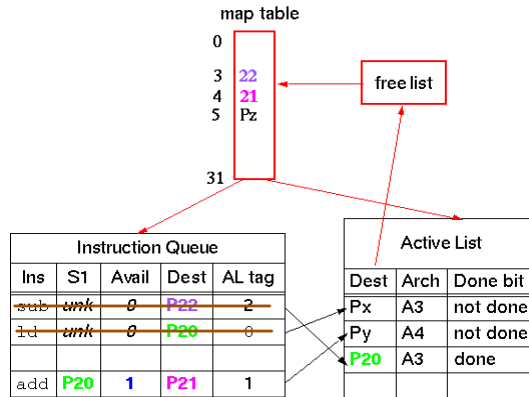
Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

18

The R10000 in Action 5 : Interrupts 1

```
ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
add   A4, A3, reg    arch register A3 used
sub   A3, reg, reg   arch register A3 redefined
                        name dependence
→ or  A5, A3, reg
```



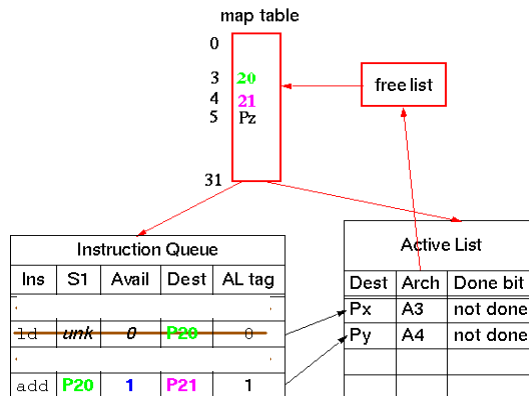
Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

19

The R10000 in Action: Interrupts 2

```
ld    A3, #(reg)    arch register A3 defined
                        potential multi-cycle
→ add  A4, A3, reg    arch register A3 used
sub   A3, reg, reg   arch register A3 redefined
                        name dependence
or    A5, A3, reg    arch register A3 used
```



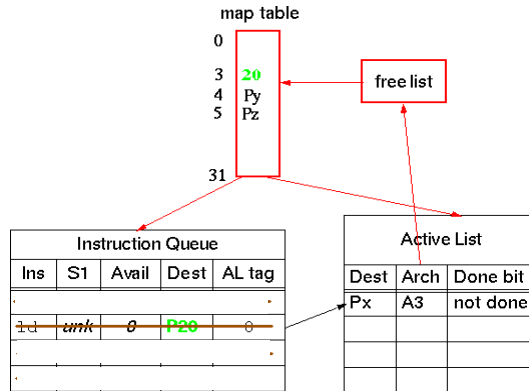
Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

20

The R10000 in Action: Interrupts 3

→ ld A3, #(reg) arch register **A3 defined**
 potential multi-cycle
 add A4, A3, reg arch register **A3 used**
 sub A3, reg, reg arch register **A3 redefined**
 name dependence
 or A5, A3, reg arch register **A3 used**



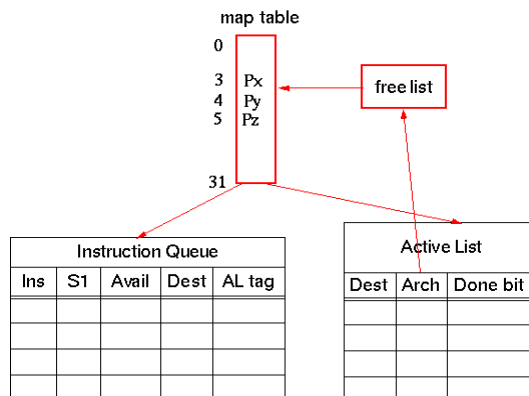
Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

21

The R10000 in Action: Interrupts 4

→ ld A3, #(reg) arch register **A3 defined**
 potential multi-cycle
 add A4, A3, reg arch register **A3 used**
 sub A3, reg, reg arch register **A3 redefined**
 name dependence
 or A5, A3, reg arch register **A3 used**



Spring 2015

CSE 471: Out-of-Order Execution
with Register Renaming

22

R10000 Execution

In-order issue (assume have already fetched instructions)

- rename architectural registers to physical registers via a map table
- detect structural hazards for instruction queues (integer, memory & FP) & active list
- issue up to 4 instructions to the instruction queues

Out-of-order execution (to increase ILP)

- instruction queues that detect when an operand has been calculated
- detect functional unit structural & RAW hazards
- dispatch instructions to functional units & execute them
- set busy-register table entry for the destination register

In-order commit (to preserve precise interrupts)

- this & previous program-generated instructions have completed
- physical register in previous mapping returned to free list
- rollback on interrupts

Limits

Limits on out-of-order execution

- amount of ILP in the code
- scheduling window size (instruction queues)
 - need to do associative searches & its effect on cycle time
 - relatively few instructions in window
- number & types of functional units
- number of locations for values
- number of ports to memory
- issue width