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
- greater need to hide latencies as:
  - discrepancy between CPU & memory speeds increases
  - branch misprediction penalty increases from superpipelining
- dynamic scheduling was generalized to cover more than floating point operations
  - handles branches & hides branch latencies
  - hides cache misses
  - can be implemented with a more general register renaming mechanism
- commits instructions in-order to preserve precise interrupts
- processors now issue multiple instructions at the same time
  - more need to exploit ILP

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, independent instructions that have not yet committed
  - more of them than architectural registers
    - ~ 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
- operands thereafter are called by their physical register number
  - hazards determined by comparing physical register numbers, not architectural register numbers

A Register Renaming Example

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

Register Renaming with A Physical Register File

Effects:

- eliminates WAW and WAR hazards (*false 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)
  - accessed with an architectural register number
  - produces a physical register number
- a destination register is assigned a new physical register number from a **free register list** (separate, for integer & FP)
  - source operands refer to the latest defined destination register, i.e., the current mappings

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 without value fields or valid bits
- used to determine when operands are available
  - compare each source operand of instructions in the IQ to destination values just computed
- determines when an appropriate functional unit is available
- dispatches instructions to functional units

An Implementation (R10000)

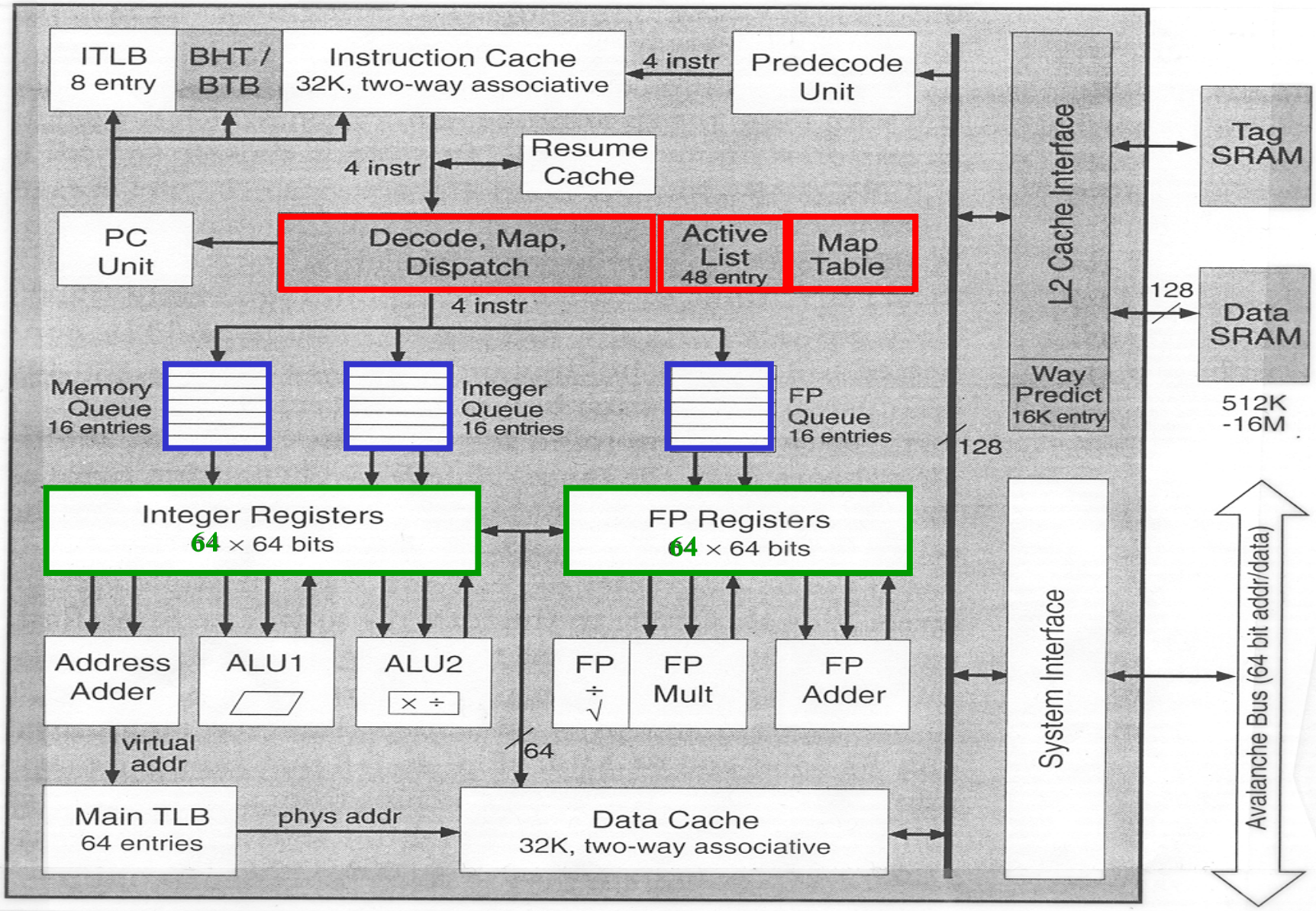
active list for all uncommitted instructions

- the mechanism for maintaining precise interrupts
  - instructions entered in program-generated order
  - allows instructions to complete in program-generated order
- instructions removed from the active list when:
  - an instruction commits:
    - the instruction has completed execution
    - all instructions ahead of it have also completed
  - 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

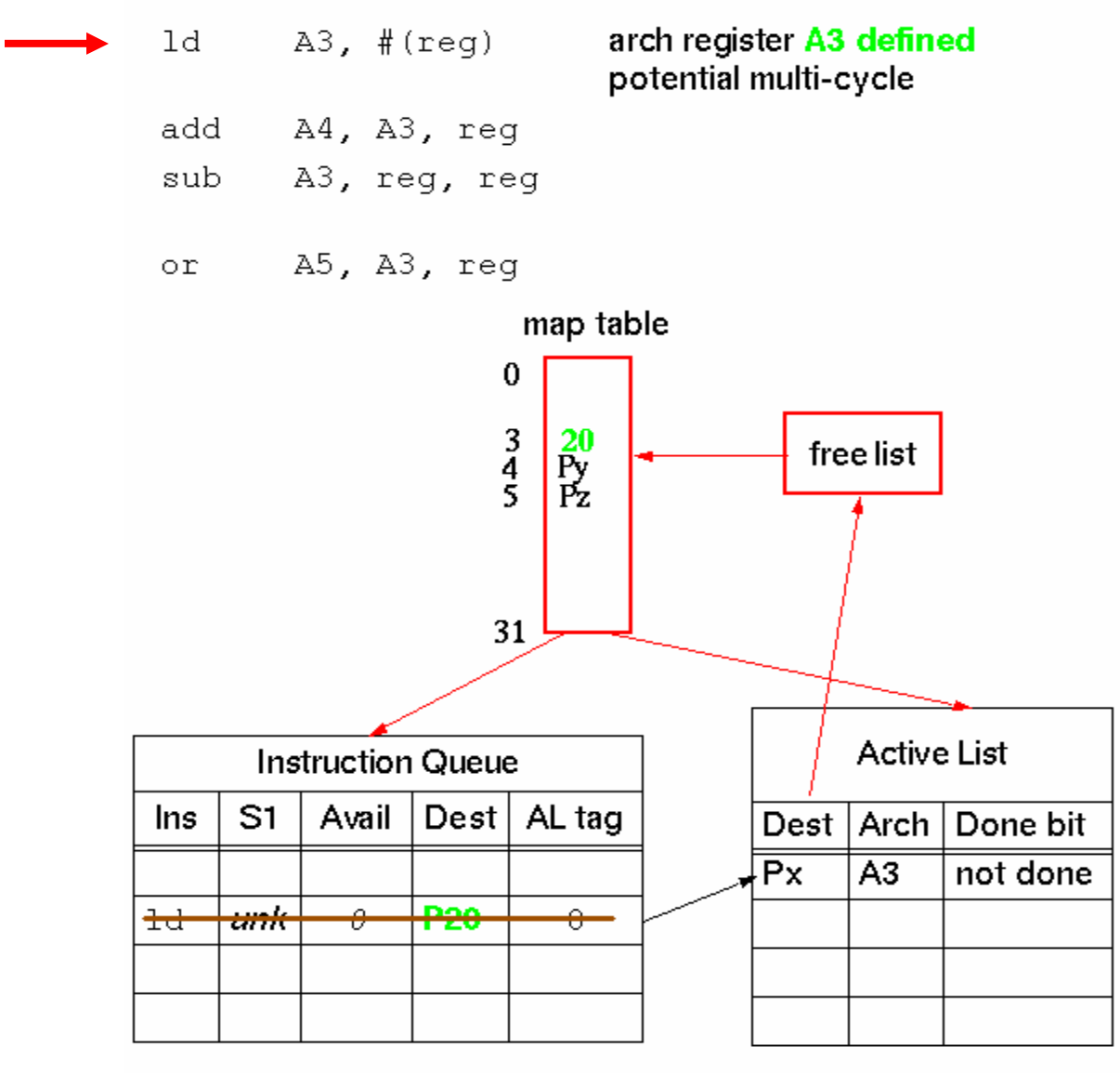
An Implementation (R10000)

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)


ITLB
8 entry
BHT / BTB
Instruction Cache
32K, two-way associative
4 instr
Predecode Unit
Tag SRAM
4 instr
Resume Cache
PC Unit
Decode, Map, Dispatch
Active List
48 entry
Map Table
L2 Cache Interface
128
Data SRAM
4 instr
Memory Queue
16 entries
Integer Queue
16 entries
FP Queue
16 entries
Way Predict
16K entry
512K
-16M
128
Integer Registers
64 × 64 bits
FP Registers
64 × 64 bits
Address Adder
ALU1
ALU2
FP
FP Mult
FP Adder
System Interface
virtual addr
64
Main TLB
64 entries
phys addr
Data Cache
32K, two-way associative
Avalanche Bus (64 bit addr/data)

The R10000 in Action 1

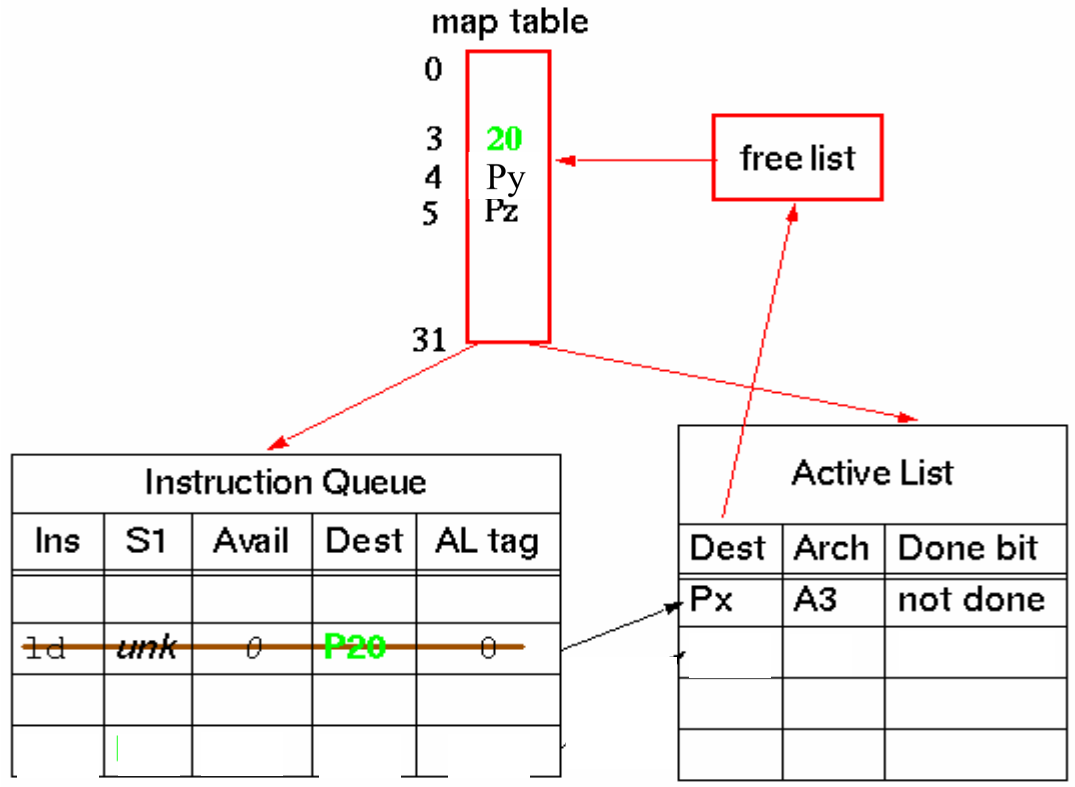


The R10000 in Action 2

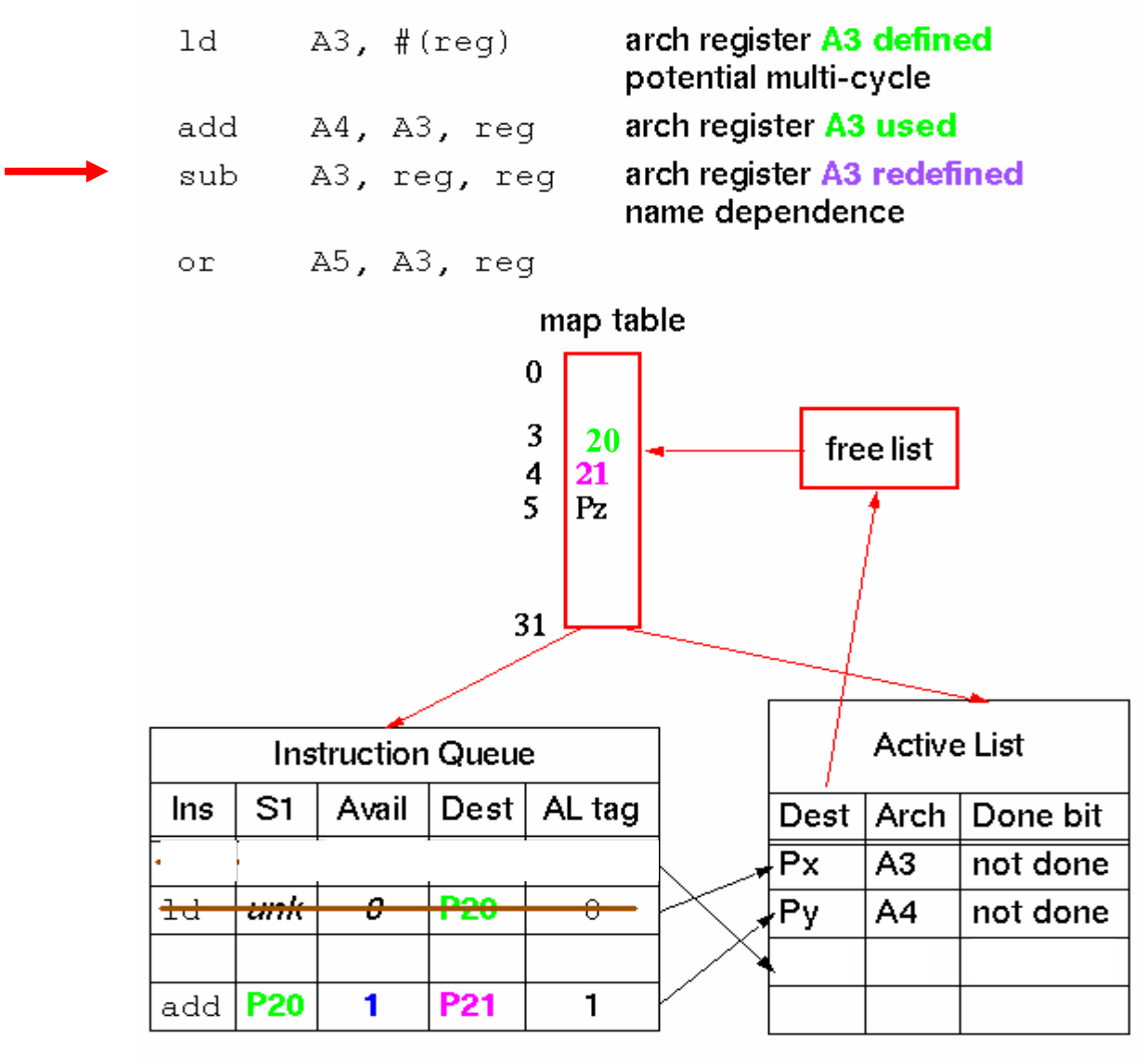
```
ld      A3, #(reg)
add     A4, A3, reg
sub     A3, reg, reg

or      A5, A3, reg
```

arch register **A3 defined**
potential multi-cycle

arch register **A3 used**

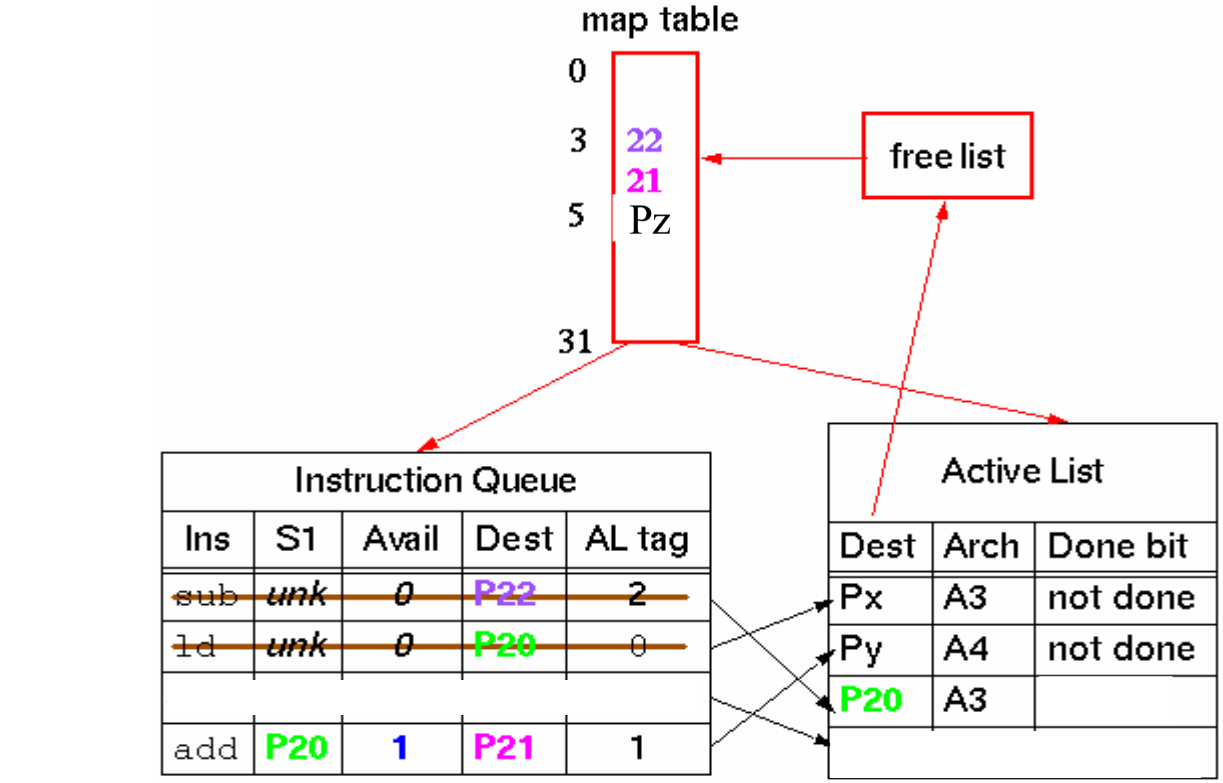


The R10000 in Action 3



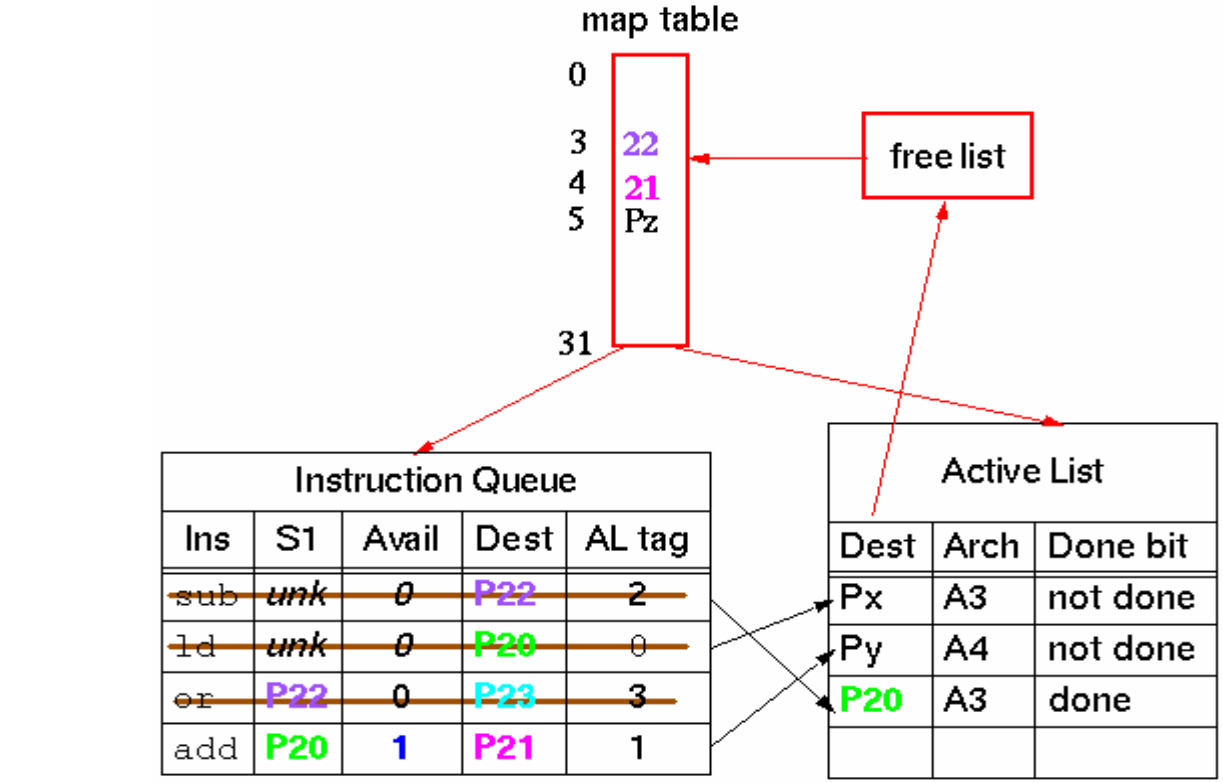
The R10000 in Action 4

```
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
```

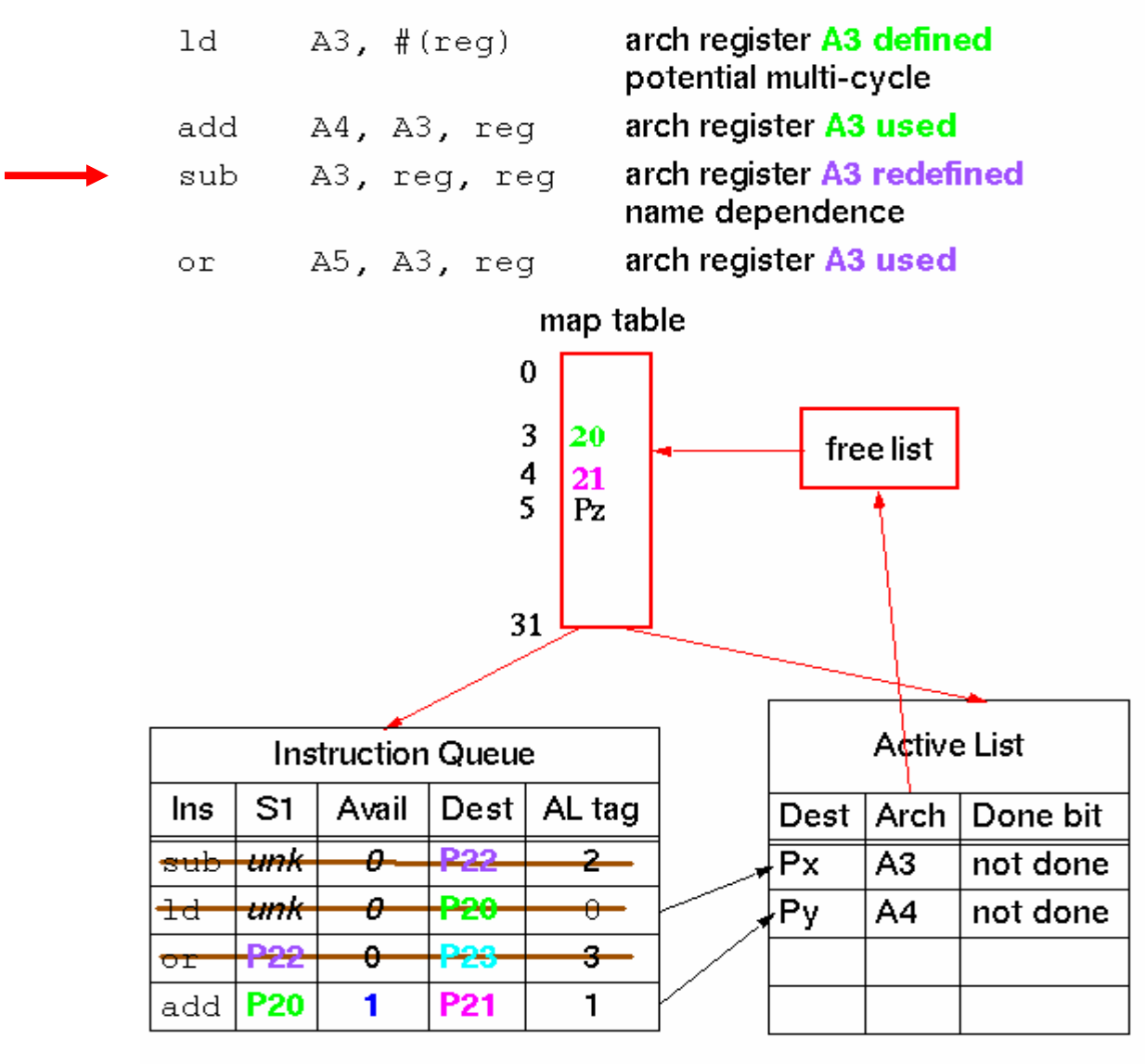


The R10000 in Action: Interrupts 1

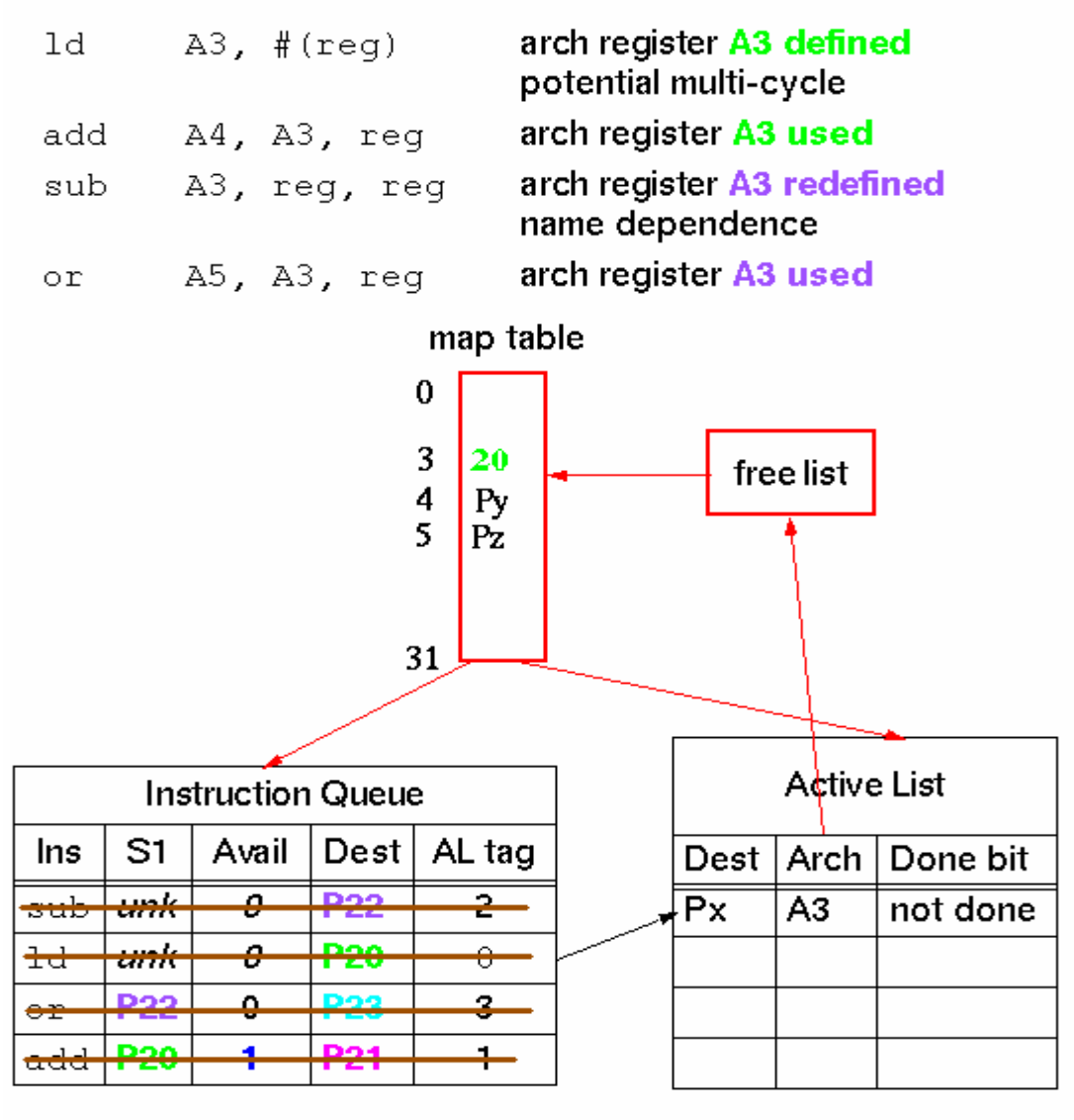
```
ld      A3, #(reg)      arch register A3 defined
                        potential multi-cycle
add     A4, A3, reg     arch register A3 used
sub     A3, reg, reg    arch register A3 redefined
                        name dependence
or      A5, A3, reg     arch register A3 used
```



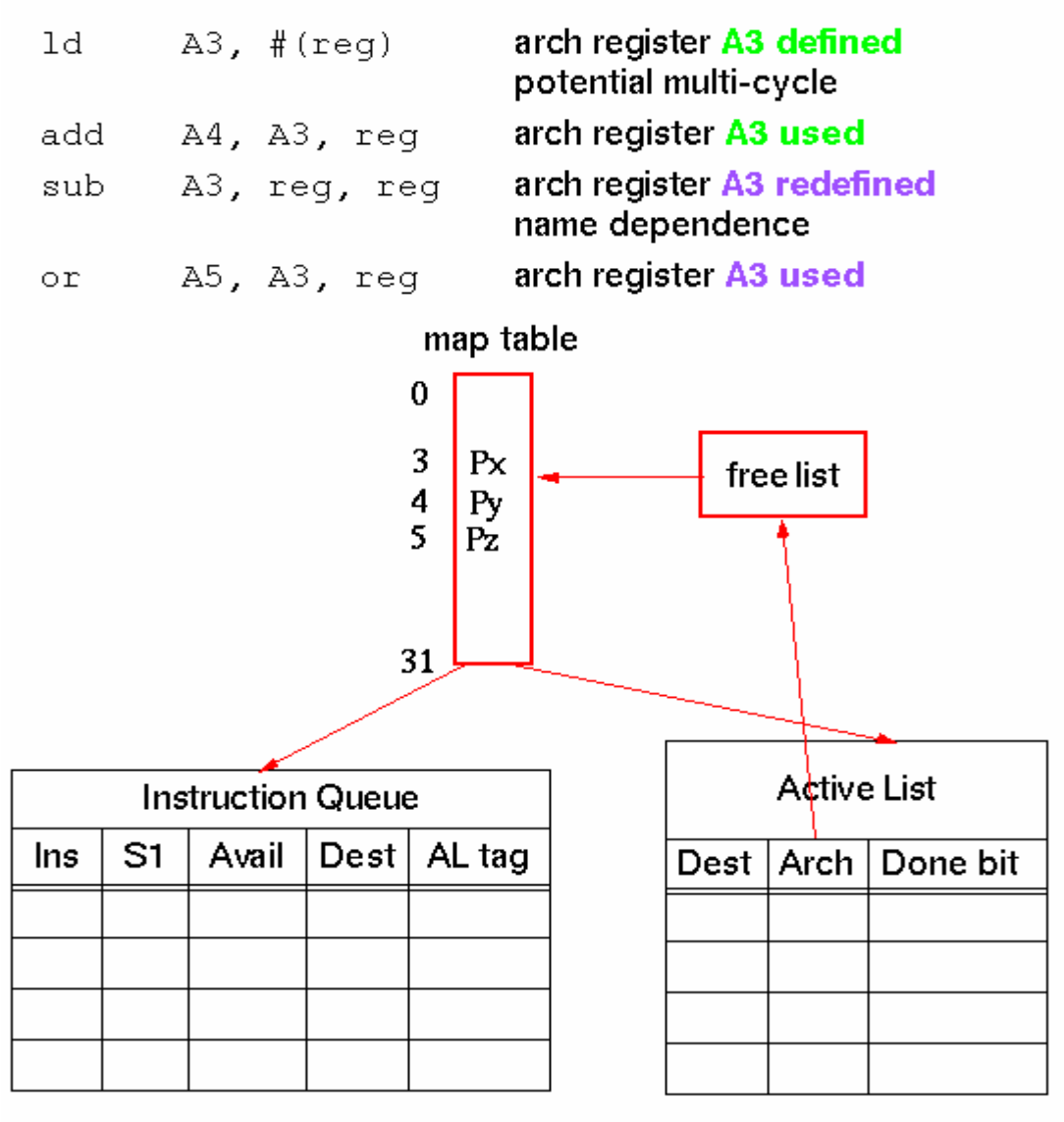
The R10000 in Action: Interrupts 2



The R10000 in Action: Interrupts 3



The R10000 in Action: Interrupts 4



R10000 Execution

In-order issue (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)

- reservation-station-like instruction queues that indicate when an operand has been calculated
  - each instruction monitors the setting of the busy-register table
- set busy-register table entry for the destination register
- detect functional unit structural & RAW hazards
- dispatch instructions to functional units

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