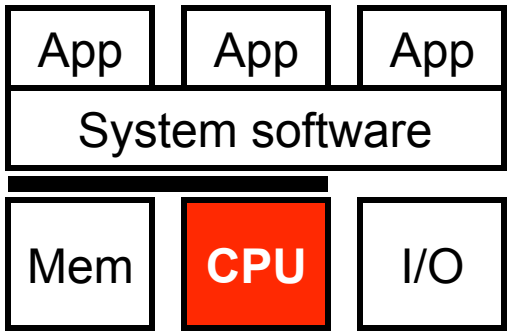
# CIS 501
# Computer Architecture

## Unit 8: Static and Dynamic Scheduling

Slides originally developed by Drew Hilton, Amir Roth
and Milo Martin at University of Pennsylvania

# This Unit: Static & Dynamic Scheduling




- Code scheduling
  - To reduce pipeline stalls
  - To increase ILP (insn level parallelism)
- Static scheduling by the compiler
  - Approach & limitations
- Dynamic scheduling in hardware
  - Register renaming
  - Instruction selection
  - Handling memory operations

# Readings

- Textbook (MA:FSPTCM)
  - Sections 3.3.1 – 3.3.4 (but not "Sidebar:")
  - Sections 5.0-5.2, 5.3.3, 5.4, 5.5

- Paper to read for in-class discussion:
  - "The MIPS R10000 Superscalar Microprocessor" by Kenneth Yeager

- Paper for **group** discussion and questions:
  - "Memory Dependence Prediction using Store Sets" by Chrysos & Emer

# Code Scheduling & Limitations

# Code Scheduling

- Scheduling: act of finding independent instructions
  - "Static" done at compile time by the compiler (software)
  - "Dynamic" done at runtime by the processor (hardware)

- Why schedule code?
  - Scalar pipelines: fill in load-to-use delay slots to improve CPI
  - Superscalar: place independent instructions together
    - As above, load-to-use delay slots
    - Allow multiple-issue decode logic to let them execute at the same time

# Compiler Scheduling

- Compiler can schedule (move) instructions to reduce stalls
  - **Basic pipeline scheduling**: eliminate back-to-back load-use pairs
  - Example code sequence: `a = b + c;  d = f - e;`
    - `sp` stack pointer, `sp+0` is "a", `sp+4` is "b", etc…

Before

```
ld r2,4(sp)
ld r3,8(sp)
add r3,r2,r1   //stall
st r1,0(sp)
ld r5,16(sp)
ld r6,20(sp)
sub r5,r6,r4   //stall
st r4,12(sp)
```

After

```
ld r2,4(sp)
ld r3,8(sp)
ld r5,16(sp)
add r3,r2,r1   //no stall
ld r6,20(sp)
st r1,0(sp)
sub r5,r6,r4   //no stall
st r4,12(sp)
```

# Compiler Scheduling Requires

- **Large scheduling scope**
  - Independent instruction to put between load-use pairs
  - \+ Original example: large scope, two independent computations
  - – This example: small scope, one computation

Before

```
ld r2,4(sp)
ld r3,8(sp)
add r3,r2,r1   //stall
st r1,0(sp)
```

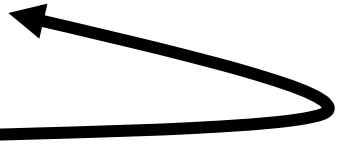
After

```
ld r2,4(sp)
ld r3,8(sp)
add r3,r2,r1   //stall
st r1,0(sp)
```

- One way to create larger scheduling scopes?
  - Loop unrolling

# Scheduling Scope Limited by Branches

```
r1 and r2 are inputs
loop:
  jz r1, not_found
  ld [r1+0] -> r3
  sub r2, r3 -> r4
  jz r4, found
  ld [r1+4] -> r1
  jmp loop
```



Aside: what does this code do?

Legal to move load up past branch?

# Compiler Scheduling Requires

- **Enough registers**
  - To hold additional "live" values
  - Example code contains 7 different values (including **sp**)
  - Before: max 3 values live at any time → 3 registers enough
  - After: max 4 values live → 3 registers not enough

Original

```
ld r2,4(sp)
ld r1,8(sp)
add r1,r2,r1   //stall
st r1,0(sp)
ld r2,16(sp)
ld r1,20(sp)
sub r2,r1,r1   //stall
st r1,12(sp)
```

Wrong!

```
ld r2,4(sp)
ld r1,8(sp)
ld r2,16(sp)
add r1,r2,r1   // wrong r2
ld r1,20(sp)
st r1,0(sp)    // wrong r1
sub r2,r1,r1
st r1,12(sp)
```

# Compiler Scheduling Requires

- **Alias analysis**
  - Ability to tell whether load/store reference same memory locations
    - Effectively, whether load/store can be rearranged
  - Example code: easy, all loads/stores use same base register (**sp**)
  - New example: can compiler tell that **r8** != **sp**?
  - Must be **conservative**

Before

```
ld r2,4(sp)
ld r3,8(sp)
add r3,r2,r1  //stall
st r1,0(sp)
ld r5,0(r8)
ld r6,4(r8)
sub r5,r6,r4  //stall
st r4,8(r8)
```

Wrong(?)

```
ld r2,4(sp)
ld r3,8(sp)
ld r5,0(r8)  //does r8==sp?
add r3,r2,r1
ld r6,4(r8)  //does r8+4==sp?
st r1,0(sp)
sub r5,r6,r4
st r4,8(r8)
```

# Code Scheduling Example

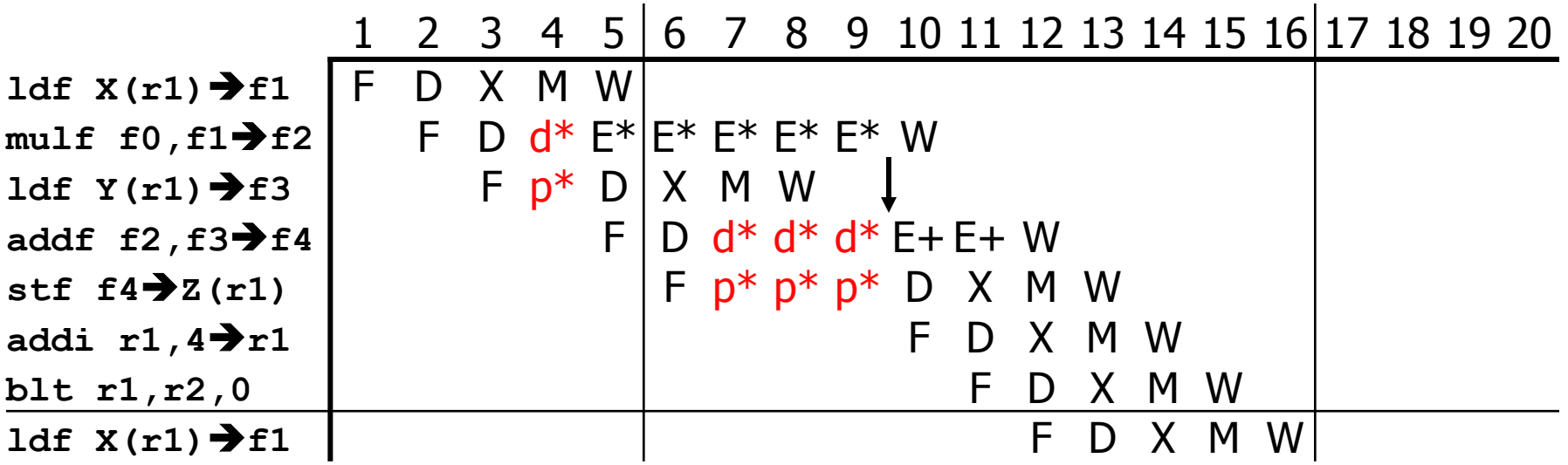
# Code Example: SAXPY

- **SAXPY** (Single-precision A X Plus Y)
  - Linear algebra routine (used in solving systems of equations)
  - Part of early "Livermore Loops" benchmark suite
  - Uses floating point values in "F" registers
  - Uses floating point version of instructions (ldf, addf, mulf, stf, etc.)

```
for (i=0;i<N;i++)
  Z[i]=(A*X[i])+Y[i];

0: ldf X(r1)➔f1       // loop
1: mulf f0,f1➔f2      // A in f0
2: ldf Y(r1)➔f3       // X,Y,Z are constant addresses
3: addf f2,f3➔f4
4: stf f4➔Z(r1)
5: addi r1,4➔r1       // i in r1
6: blt r1,r2,0        // N*4 in r2
```

# SAXPY Performance and Utilization




- Scalar pipeline
  - Full bypassing, 5-cycle E*, 2-cycle E+, branches predicted taken
  - Single iteration (7 insns) latency: 16–5 = 11 cycles
  - **Performance**: 7 insns / 11 cycles = 0.64 IPC
  - **Utilization**: 0.64 actual IPC / 1 peak IPC = 64%

# Static (Compiler) Instruction Scheduling

- Idea: place independent insns between slow ops and uses
  - Otherwise, pipeline stalls while waiting for RAW hazards to resolve
  - Have already seen pipeline scheduling

- To schedule well you need … **independent insns**
- **Scheduling scope**: code region we are scheduling
  - The bigger the better (more independent insns to choose from)
  - Once scope is defined, schedule is pretty obvious
  - Trick is creating a large scope (must schedule across branches)

- Scope enlarging techniques
  - Loop unrolling
  - Others: "superblocks", "hyperblocks", "trace scheduling", etc.

# Loop Unrolling SAXPY

- Goal: separate dependent insns from one another
- SAXPY problem: not enough flexibility within one iteration
  - Longest chain of insns is 9 cycles
    - Load (1)
    - Forward to multiply (5)
    - Forward to add (2)
    - Forward to store (1)
  - Can't hide a 9-cycle chain using only 7 insns
  - But how about two 9-cycle chains using 14 insns?
- **Loop unrolling**: schedule two or more iterations together
  - Fuse iterations
  - Schedule to reduce stalls
  - Schedule introduces ordering problems, rename registers to fix

# Unrolling SAXPY I: Fuse Iterations

- Combine two (in general K) iterations of loop
  - Fuse loop control: induction variable (`i`) increment + branch
  - Adjust (implicit) induction uses: constants → constants + 4

```
ldf X(r1),f1
mulf f0,f1,f2
ldf Y(r1),f3
addf f2,f3,f4
stf f4,Z(r1)
addi r1,4,r1
blt r1,r2,0
```

```
ldf X(r1),f1
mulf f0,f1,f2
ldf Y(r1),f3
addf f2,f3,f4
stf f4,Z(r1)
ldf X+4(r1),f1
mulf f0,f1,f2
ldf Y+4(r1),f3
addf f2,f3,f4
stf f4,Z+4(r1)
addi r1,8,r1
blt r1,r2,0
```

# Unrolling SAXPY II: Pipeline Schedule

- Pipeline schedule to reduce stalls
  - Have already seen this: pipeline scheduling

```
ldf X(r1),f1
mulf f0,f1,f2
ldf Y(r1),f3
addf f2,f3,f4
stf f4,Z(r1)
ldf X+4(r1),f1
mulf f0,f1,f2
ldf Y+4(r1),f3
addf f2,f3,f4
stf f4,Z+4(r1)
addi r1,8,r1
blt r1,r2,0
```

```
ldf X(r1),f1
ldf X+4(r1),f1
mulf f0,f1,f2
ldf Y(r1),f3
ldf Y+4(r1),f3
addf f2,f3,f4
addf f2,f3,f4
stf f4,Z(r1)
stf f4,Z+4(r1)
addi r1,8,r1
blt r1,r2,0
```

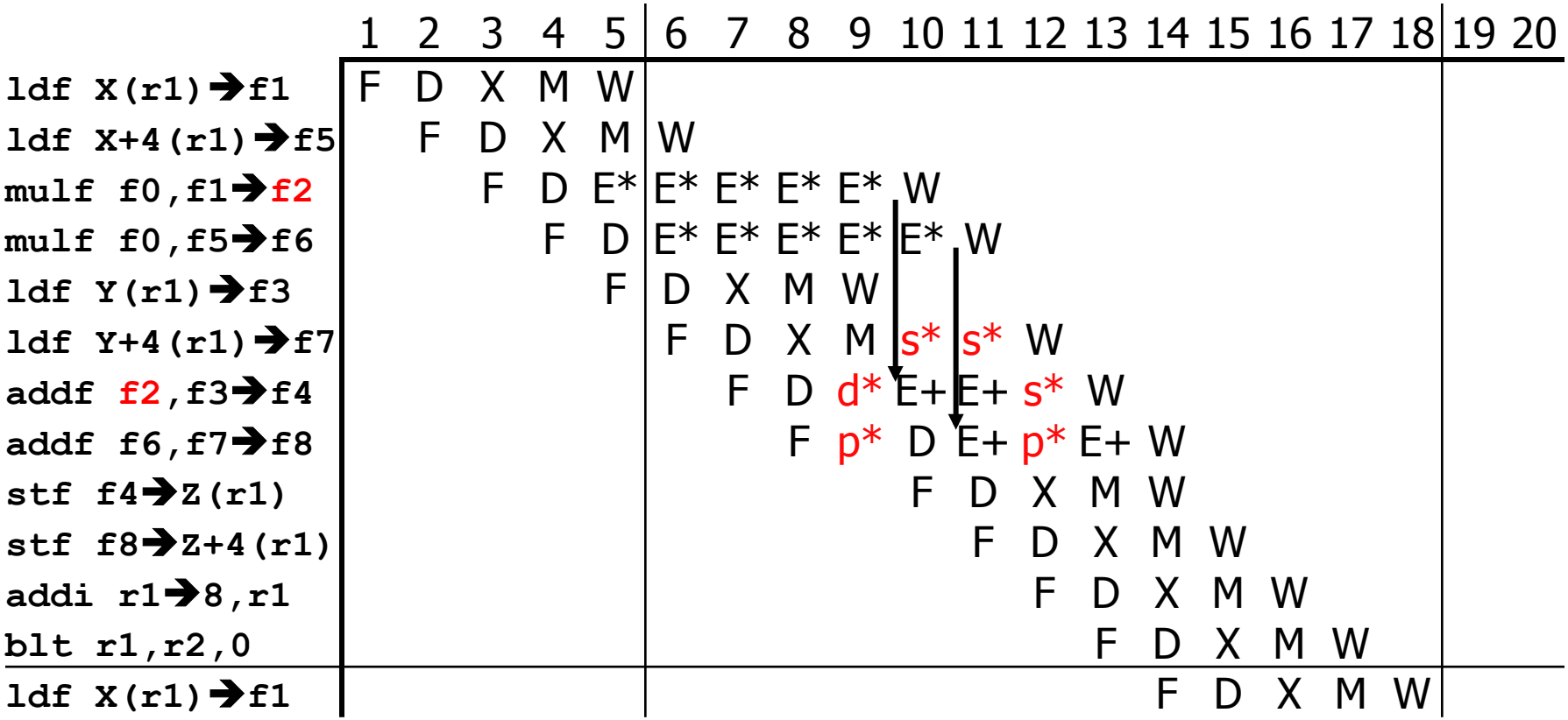
# Unrolling SAXPY III: "Rename" Registers

- Pipeline scheduling causes reordering violations
  - Use different register names to fix problem

```
ldf X(r1),f1
ldf X+4(r1),f1
mulf f0,f1,f2
mulf f0,f1,f2
ldf Y(r1),f3
ldf Y+4(r1),f3
addf f2,f3,f4
addf f2,f3,f4
stf f4,Z(r1)
stf f4,Z+4(r1)
addi r1,8,r1
blt r1,r2,0
```

```
ldf X(r1),f1
ldf X+4(r1),f5
mulf f0,f1,f2
mulf f0,f5,f6
ldf Y(r1),f3
ldf Y+4(r1),f7
addf f2,f3,f4
addf f6,f7,f8
stf f4,Z(r1)
stf f8,Z+4(r1)
addi r1,8,r1
blt r1,r2,0
```

# Unrolled SAXPY Performance/Utilization




+ Performance: 12 insn / 13 cycles = 0.92 IPC
+ Utilization: 0.92 actual IPC / 1 peak IPC = 92%
+ **Speedup**: (2 * 11 cycles) / 13 cycles = 1.69

# Loop Unrolling Shortcomings

- Static code growth → more instruction cache misses (limits degree of unrolling)
- Needs more registers to hold values (ISA limits this)
- Doesn't handle non-loops
- Doesn't handle inter-iteration dependences

```
for (i=0;i<N;i++)
  X[i]=A*X[i-1];
```

```
ldf X-4(r1),f1
mulf f0,f1,f2
stf f2,X(r1)
addi r1,4,r1
blt r1,r2,0
ldf X-4(r1),f1
mulf f0,f1,f2
stf f2,X(r1)
addi r1,4,r1
blt r1,r2,0
```

```
ldf X-4(r1),f1
mulf f0,f1,f2
stf f2,X(r1)
mulf f0,f2,f3
stf f3,X+4(r1)
addi r1,8,r1
blt r1,r2,0
```

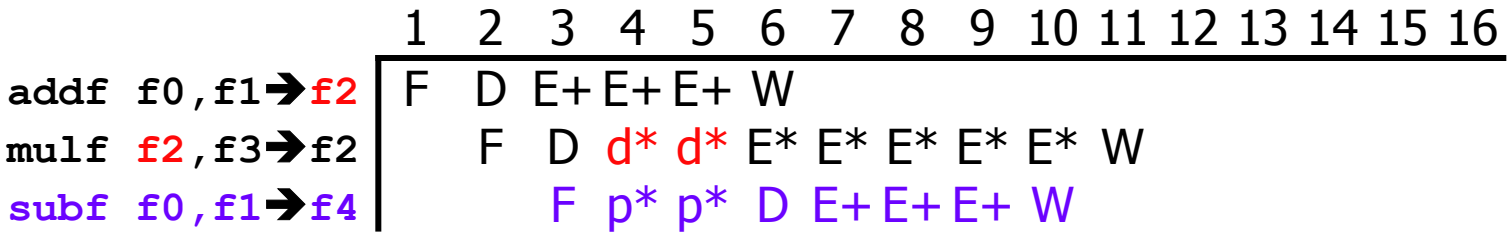
- Two `mulf`'s are not parallel
- Other (more advanced) techniques help

# Static Scheduling Limitations (Summary)

- Limited number of registers (set by ISA)
- Scheduling scope
  - Example: can't generally move memory operations past branches
- Inexact memory aliasing information
  - Often prevents reordering of loads above stores
- Caches misses (or any runtime event) confound scheduling
  - How can the compiler know which loads will miss vs hit?
  - Can impact the compiler's scheduling decisions

# Dynamic (Hardware) Scheduling

# The Problem With In-Order Pipelines




- What's happening in cycle 4?
  - **`mulf`** stalls due to **data dependence**
    - OK, this is a fundamental problem
  - **`subf`** stalls due to **pipeline (propagation) hazard**
    - Why? **`subf`** can't proceed into D because **`mulf`** is there
    - That is the only reason, and it isn't a fundamental one
  - Maintaining in-order writes to register file

- Why can't **`subf`** go into D in cycle 4 and E+ in cycle 5?

# Can Hardware Overcome These Limits?

- **Dynamically-scheduled processors**
  - Also called "out-of-order" processors
  - Hardware re-schedules insns…
  - …within a sliding window of VonNeumann insns
  - As with pipelining and superscalar, ISA unchanged
    - Same hardware/software interface, appearance of in-order
- Increases scheduling scope
  - Does loop unrolling transparently
  - Uses branch prediction to "unroll" branches
- Examples:
  - Pentium Pro/II/III (3-wide), Core 2 (4-wide),
    Alpha 21264 (4-wide), MIPS R10000 (4-wide), Power5 (5-wide)
- Basic overview of approach

# Example: In-Order Limitations #1

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ld [r1] -> r2 | F | D | X | $M_1$ | $M_2$ | W | | | | | | | |
| add r2 + r3 -> r4 | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | | |
| xor r4 ^ r5 -> r6 | | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | |
| ld [r7] -> r4 | | F | D | **p*** | **p*** | **p*** | X | $M_1$ | $M_2$ | W | | | |

- In-order pipeline, two-cycle load-use penalty
  - 2-wide
- Why not:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ld [r1] -> r2 | F | D | X | $M_1$ | $M_2$ | W | | | | | | | |
| add r2 + r3 -> **r4** | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | | |
| xor **r4** ^ r5 -> r6 | | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | |
| ld [r7] -> **r4** | | F | D | **X** | **$M_1$** | **$M_2$** | **W** | | | | | | |

# Example: In-Order Limitations #2

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ld [p1] -> p2 | F | D | X | $M_1$ | $M_2$ | W | | | | | | | |
| add p2 + p3 -> p4 | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | | |
| xor p4 ^ p5 -> p6 | | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | |
| ld [p7] -> p8 | | F | D | **p*** | **p*** | **p*** | X | $M_1$ | $M_2$ | W | | | |

- In-order pipeline, two-cycle load-use penalty
  - 2-wide
- Why not:

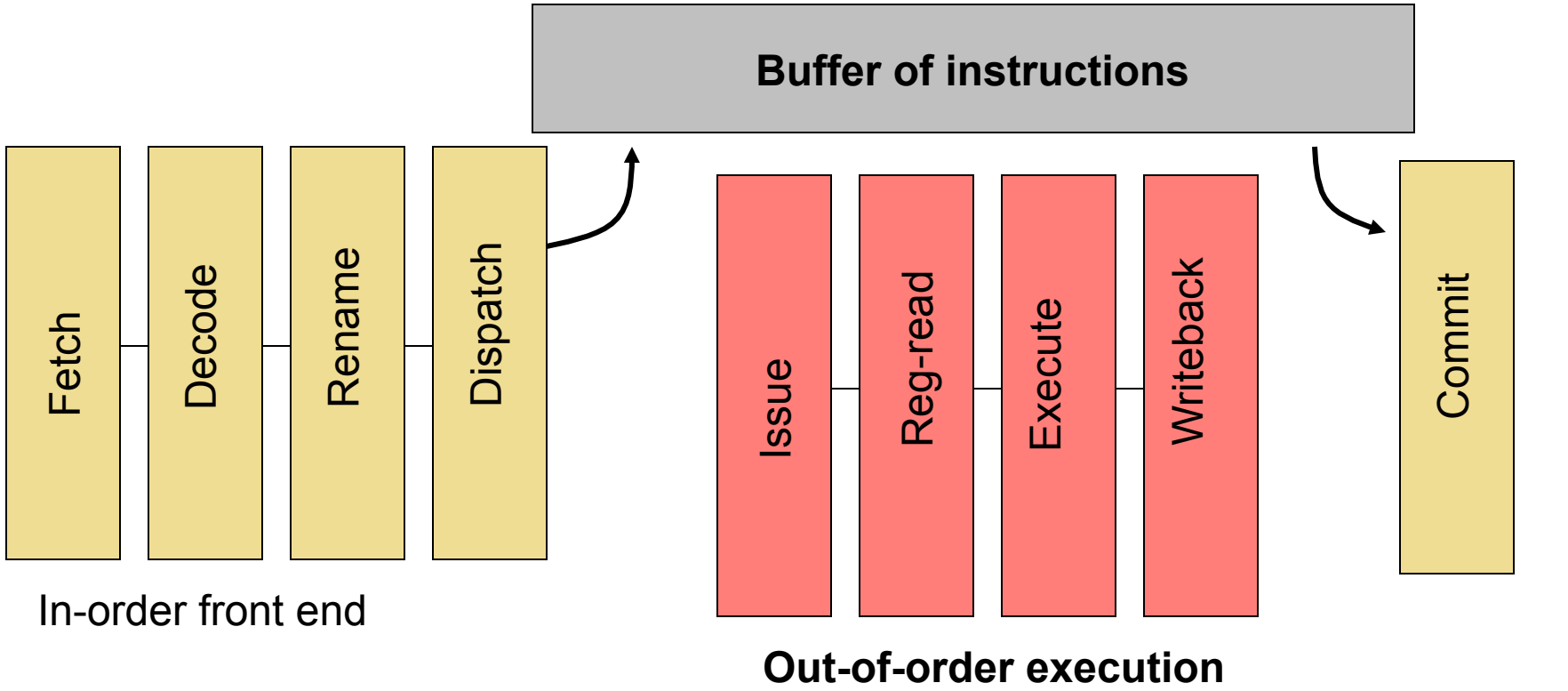
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ld [p1] -> p2 | F | D | X | $M_1$ | $M_2$ | W | | | | | | | |
| add p2 + p3 -> **p4** | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | | |
| xor **p4** ^ p5 -> p6 | | F | D | **d*** | **d*** | **d*** | X | $M_1$ | $M_2$ | W | | | |
| ld [p7] -> **p8** | | F | D | **X** | **$M_1$** | **$M_2$** | **W** | | | | | | |

# Out-of-Order to the Rescue

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Ld [p1] -> p2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add p2 + p3 -> p4 | F | Di | | | | I | RR | X | W | C | | | |
| xor p4 ^ p5 -> p6 | | F | Di | | | | I | RR | X | W | C | | |
| ld [p7] -> p8 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | C | | |

- "Dynamic scheduling" done by the hardware
- Still 2-wide superscalar, but now out-of-order, too
  - Allows instructions to issues when dependences are ready
- Longer pipeline
  - In-order front end: Fetch, "**Dispatch**"
  - Out-of-order execution core:
    - "**Issue**", "**RegisterRead**", Execute, Memory, Writeback
  - In-order retirement: "**Commit**"

# Out-of-order Pipeline



# Code Example

- Code:

Raw insns

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

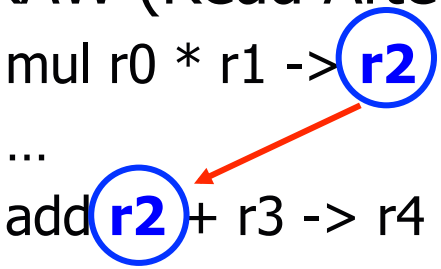
"Renamed" insns

```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```

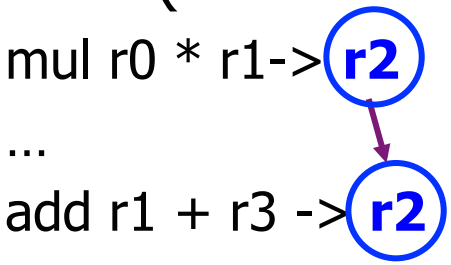
- Difficult to reorder above code, names get in the way
- Divide insn independent of subtract and multiply insns
  - Should be able to execute in parallel with subtract
- Many registers re-used
  - Just as in static scheduling, the register names get in the way
  - How does the hardware get around this?
- Approach: (step #1) rename registers, (step #2) schedule

# Dependence types

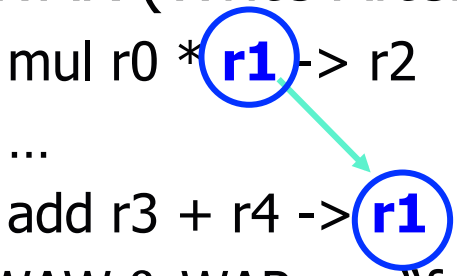
- RAW (Read After Write) = "true dependence"




- WAW (Write After Write) = "output dependence"




- WAR (Write After Read) = "anti-dependence"




- WAW & WAR are "false", Can be **totally eliminated** by "renaming"

# Step #1: Register Renaming

- To eliminate register conflicts/hazards
- "Architected" vs "Physical" registers – level of indirection
  - Names: `r1,r2,r3`
  - Locations: `p1,p2,p3,p4,p5,p6,p7`
  - Original mapping: `r1`→`p1`, `r2`→`p2`, `r3`→`p3`, `p4`–`p7` are "available"

MapTable

| r1 | r2 | r3 |
|---|---|---|
| p1 | p2 | p3 |
| p4 | p2 | p3 |
| p4 | p2 | p5 |
| p4 | p2 | p6 |

FreeList

| |
|---|
| p4,p5,p6,p7 |
| p5,p6,p7 |
| p6,p7 |
| p7 |

Original insns

```
add r2,r3,r1
sub r2,r1,r3
mul r2,r3,r3
div r1,4,r1
```

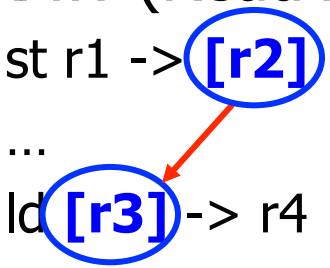
Renamed insns

```
add p2,p3,p4
sub p2,p4,p5
mul p2,p5,p6
div p4,4,p7
```

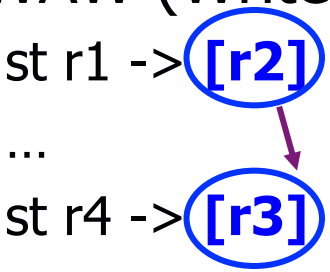
- Renaming – conceptually write each register once
  - + Removes **false** dependences
  - + Leaves **true** dependences intact!
- When to reuse a physical register? After overwriting insn done

# Memory dependences

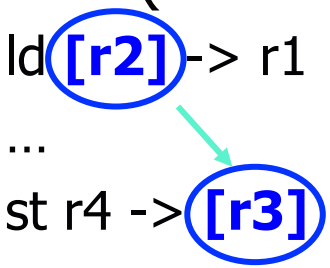
- **If value in "r2" and "r3" is the same…**
- RAW (Read After Write) – True dependency




- WAW (Write After Write)

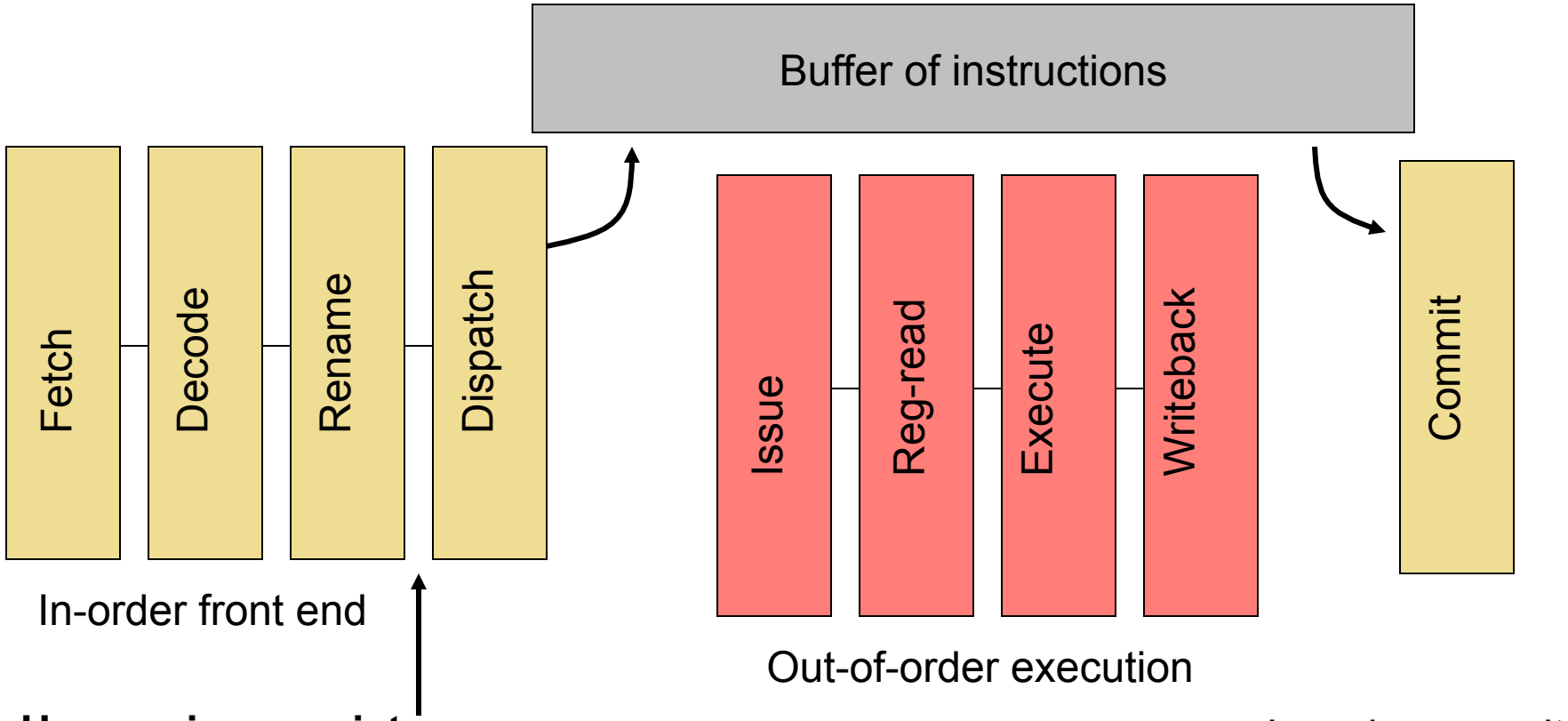



- WAR (Write After Read)



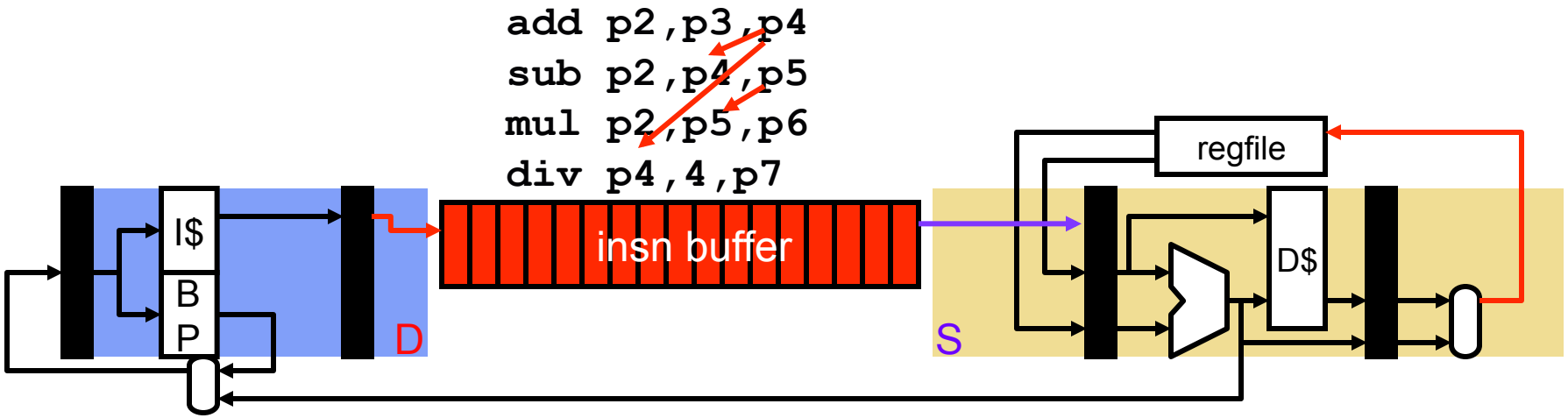

WAR/WAW are "false dependencies"
- Can't rename memory in same way as registers
  - Why?
- Need to use other tricks

# Out-of-order Pipeline




**Have unique register names**
**Now put into out-of-order execution structures**

# Step #2: Dynamic Scheduling




Ready Table

Time

| P2 | P3 | P4 | P5 | P6 | P7 |
|---|---|---|---|---|---|
| Yes | Yes | | | | |
| Yes | Yes | Yes | | | |
| Yes | Yes | Yes | Yes | | Yes |
| Yes | Yes | Yes | Yes | Yes | Yes |

```
add p2,p3,p4
sub p2,p4,p5  and div p4,4,p7
mul p2,p5,p6
```

- Instructions fetch/decoded/renamed into *Instruction Buffer*
  - Also called "instruction window" or "instruction scheduler"
- Instructions (conceptually) check ready bits every cycle
  - Execute when ready

# Dynamic Scheduling/Issue Algorithm

- Data structures:
  - Ready table[phys_reg] ➔ yes/no (part of "issue queue")

- Algorithm at "schedule" stage (prior to read registers):

```
foreach instruction:
   if table[insn.phys_input1] == ready &&
       table[insn.phys_input2] == ready then
           insn is "ready"
select the oldest "ready" instruction
   table[insn.phys_output] = ready
```

# REGISTER RENAMING

# Register Renaming Algorithm

- Data structures:
  - maptable[architectural_reg] ➔ physical_reg
  - Free list: get/put free register (implemented as a queue)
- Algorithm: at decode for each instruction:

```
insn.phys_input1 = maptable[insn.arch_input1]
insn.phys_input2 = maptable[insn.arch_input2]
insn.phys_to_free = maptable[arch_output]
new_reg = get_free_phys_reg()
maptable[arch_output] = new_reg
insn.phys_output = new_reg
```

- At "commit"
  - Once all older instructions have committed, free register `put_free_phys_reg(insn.phys_to_free)`

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| r3 | p3 |
| r4 | p4 |
| r5 | p5 |

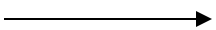
Map table

| |
|---|
| p6 |
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

xor **r1** ^ **r2** -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1



xor **p1** ^ **p2** ->

| **r1** | **p1** |
|---|---|
| **r2** | **p2** |
| r3 | p3 |
| r4 | p4 |
| r5 | p5 |

Map table

| p6 |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p3 |
| r4 | p4 |
| r5 | p5 |

Map table

| **p6** |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| **r3** | **p6** |
| r4 | p4 |
| r5 | p5 |

Map table

| |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 ->
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| **r3** | **p6** |
| **r4** | **p4** |
| r5 | p5 |

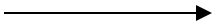
Map table

| |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
```

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| r3 | p6 |
| r4 | p4 |
| r5 | p5 |

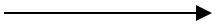
Map table

| |
|---|
| **p7** |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p6 |
| **r4** | **p7** |
| r5 | p5 |

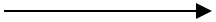
Map table

| |
|---|
| |
| |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 ->
```

| | |
|---|---|
| r1 | p1 |
| **r2** | **p2** |
| r3 | p6 |
| r4 | p7 |
| **r5** | **p5** |

Map table

| |
|---|
| |
| |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p6 |
| r4 | p7 |
| r5 | p5 |

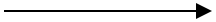
Map table

| |
|---|
| **p8** |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
```

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| **r3** | **p8** |
| r4 | p7 |
| r5 | p5 |

Map table

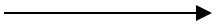
| |
|---|
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 ->
```



| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| **r3** | **p8** |
| r4 | p7 |
| r5 | p5 |

Map table

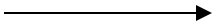
| |
|---|
| |
| |
| |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```



| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| |
| |
| |
| **p9** |
| p10 |

Free-list

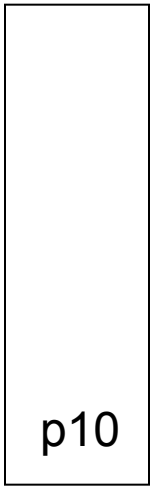
# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

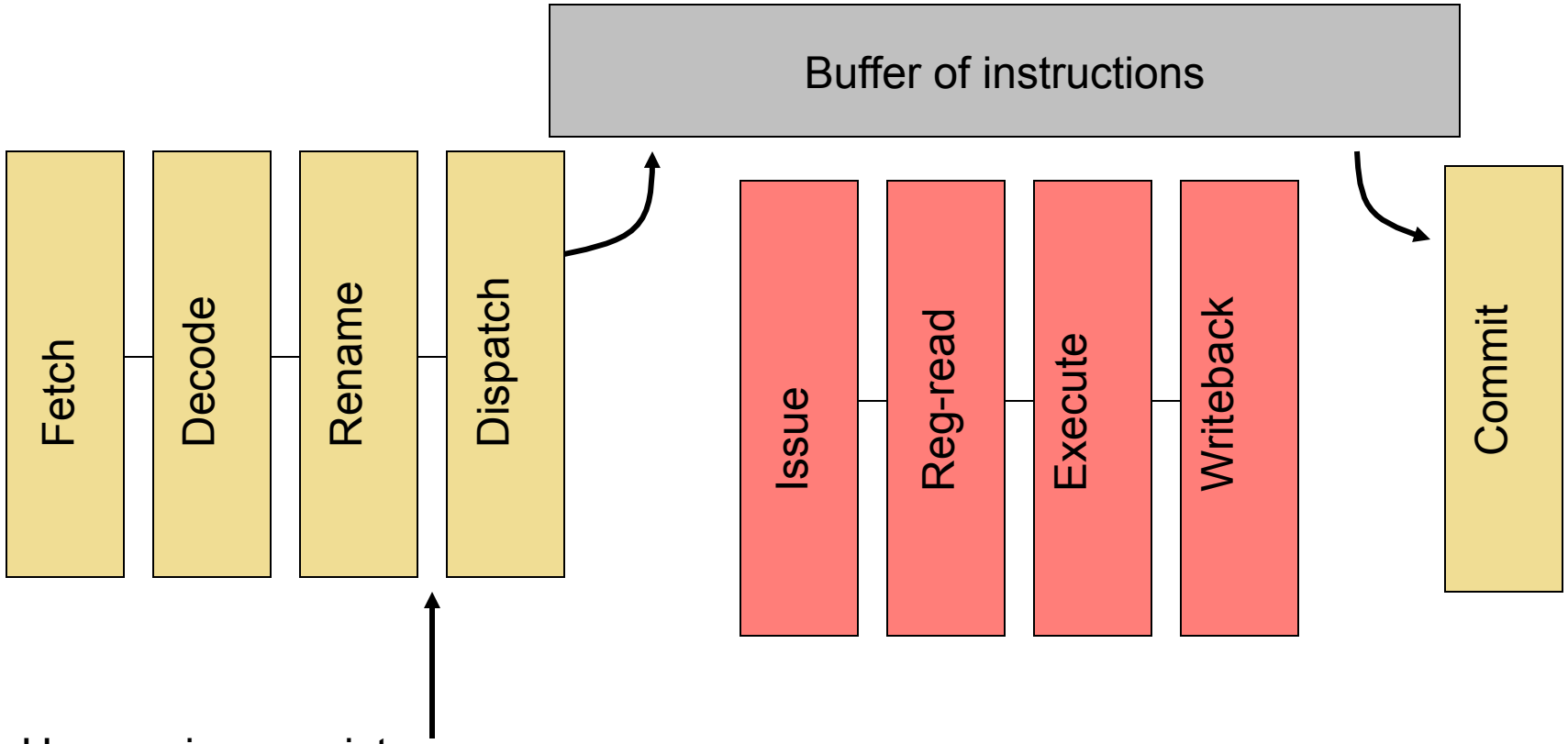
| **r1** | **p9** |
|---|---|
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table




Free-list

# Out-of-order Pipeline




Have unique register names
Now put into out-of-order execution structures

# DYNAMIC SCHEDULING MECHANISMS

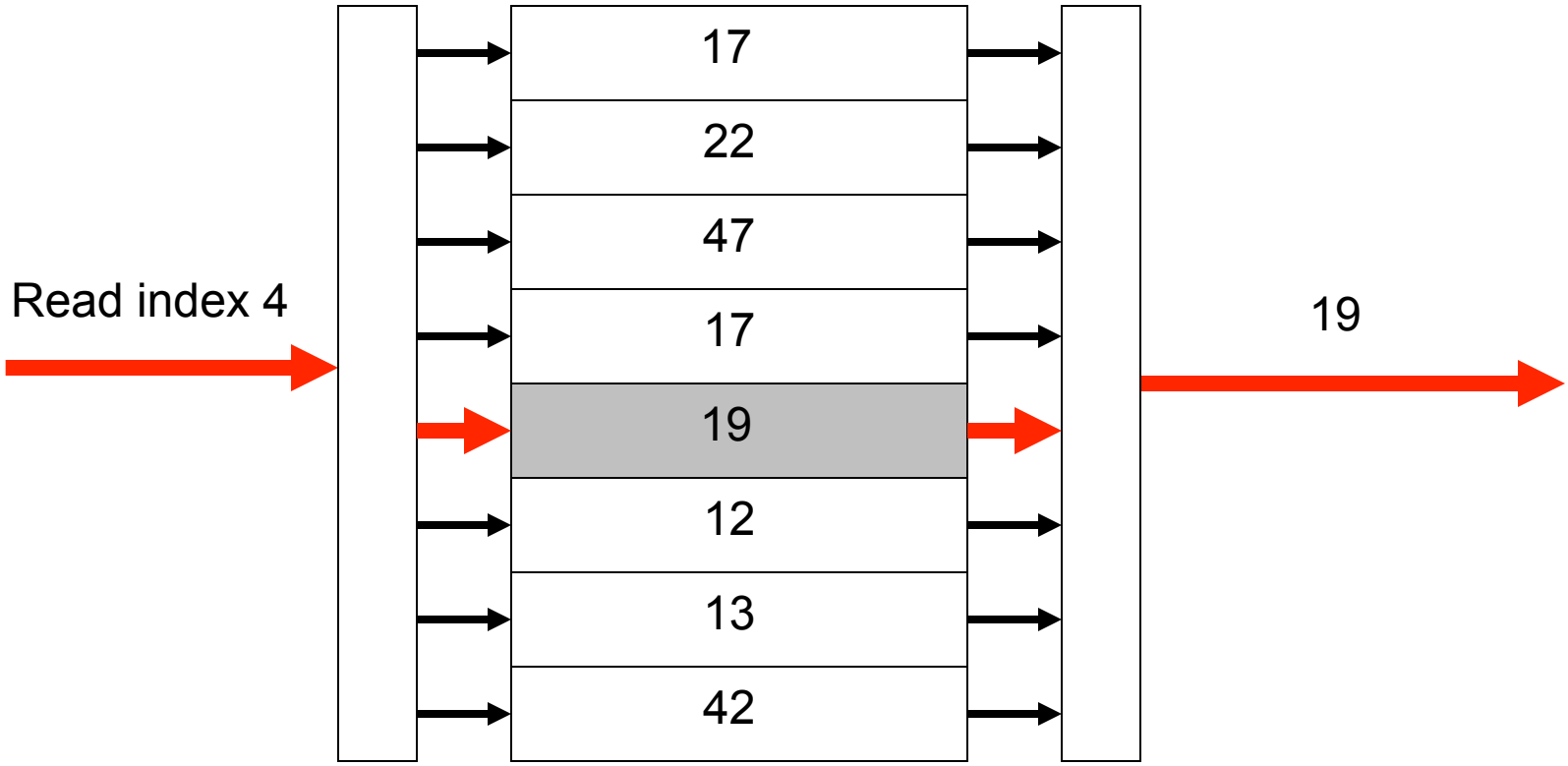
# Dispatch

- Renamed instructions into out-of-order structures
  - Re-order buffer (ROB)
    - All instruction until commit
  - Issue Queue
    - Un-executed instructions
    - Central piece of scheduling logic
    - Content Addressable Memory (CAM)

# RAM vs CAM

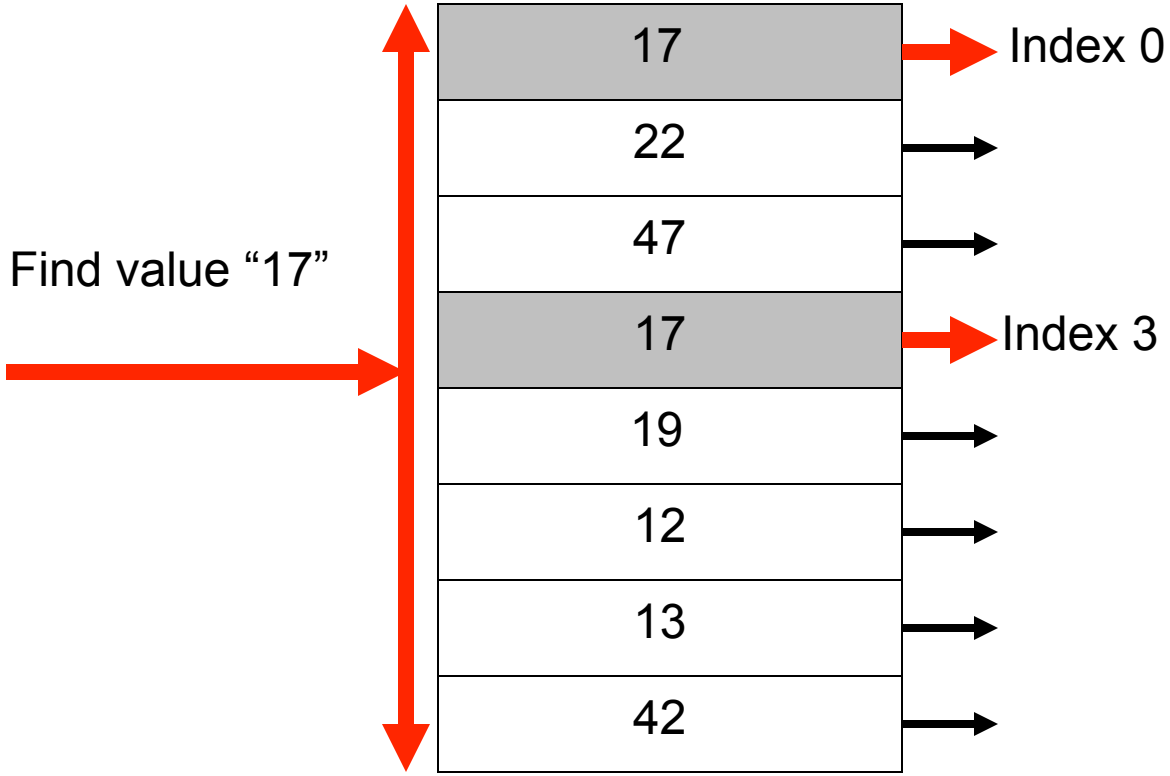
- Random Access Memory
  - Read/write specific index
  - Get/set value there
- Content Addressable Memory
  - Search for a value (send value to all entries)
  - Find matching indices (use comparator at each entry)
  - Output: one bit per entry (multiple match)
- One structure can have ports of both types

# RAM vs CAM: RAM




RAM: read/write specific index

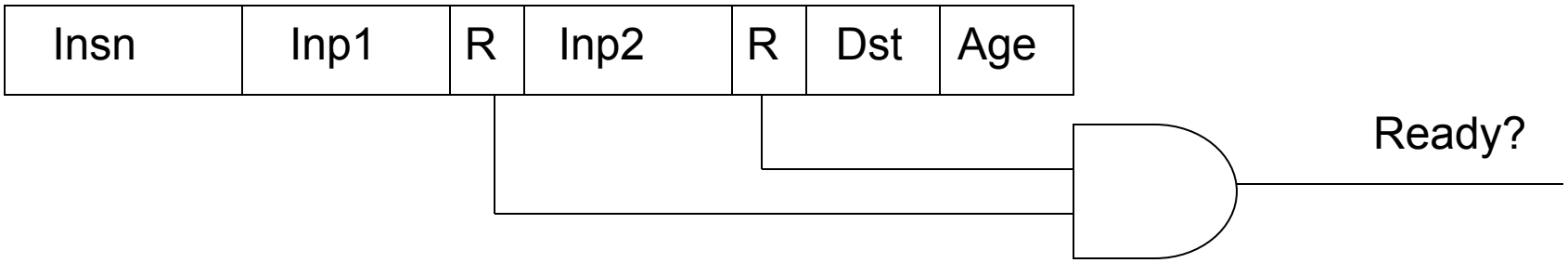
# RAM vs CAM: CAM




CAM: search for value

# Issue Queue

- Holds un-executed instructions
- Tracks ready inputs
  - Physical register names + ready bit
  - "AND" the bits to tell if ready



# Dispatch Steps

- Allocate IQ slot
  - Full?  Stall
- Read **ready bits** of inputs
  - Table 1-bit per physical reg
- Clear **ready bit** of output in table
  - Instruction has not produced value yet
- Write data in IQ slot

# Dispatch Example

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| p6 | y |
| p7 | y |
| p8 | y |
| p9 | y |

**Issue Queue**

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

# Dispatch Example

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

**Issue Queue**

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| xor | p1 | y | p2 | y | p6 | 0 |
| | | | | | | |
| | | | | | | |
| | | | | | | |

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| **p6** | **n** |
| p7 | y |
| p8 | y |
| p9 | y |

# Dispatch Example

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

**Issue Queue**

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| xor | p1 | y | p2 | y | p6 | 0 |
| add | p6 | n | p4 | y | p7 | 1 |
| | | | | | | |
| | | | | | | |

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| p6 | n |
| **p7** | **n** |
| p8 | y |
| p9 | y |

# Dispatch Example

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

**Issue Queue**

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| xor | p1 | y | p2 | y | p6 | 0 |
| add | p6 | n | p4 | y | p7 | 1 |
| sub | p5 | y | p2 | y | p8 | 2 |
| | | | | | | |

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| p6 | n |
| p7 | n |
| **p8** | **n** |
| p9 | y |

# Dispatch Example

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

**Issue Queue**

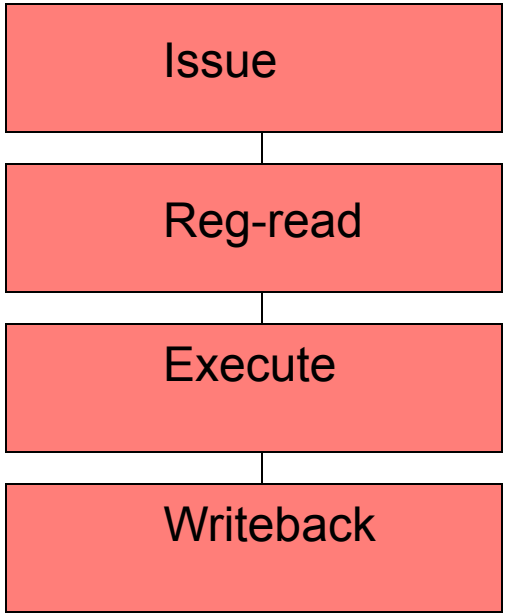
| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| xor | p1 | y | p2 | y | p6 | 0 |
| add | p6 | n | p4 | y | p7 | 1 |
| sub | p5 | y | p2 | y | p8 | 2 |
| addi | p8 | n | --- | y | p9 | 3 |

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| p6 | n |
| p7 | n |
| p8 | n |
| **p9** | **n** |

# Out-of-order pipeline

- Execution (out-of-order) stages
- **Select** ready instructions
  - Send for execution
- **Wakeup** dependents



# Dynamic Scheduling/Issue Algorithm

- Data structures:
  - Ready table[phys_reg] ➔ yes/no (part of issue queue)

- Algorithm at "schedule" stage (prior to read registers):

```
foreach instruction:
   if table[insn.phys_input1] == ready &&
      table[insn.phys_input2] == ready then
         insn is "ready"
select the oldest "ready" instruction
   table[insn.phys_output] = ready
```

# Issue = Select + Wakeup

- **Select** N oldest, ready instructions
  - ➢ "xor" is the oldest ready instruction below
  - ➢ "xor" and "sub" are the two oldest ready instructions below
  - Note: may have resource constraints: i.e. load/store/fp

| Insn | Inp1 | R | Inp2 | R | Dst | Age | |
|---|---|---|---|---|---|---|---|
| xor | p1 | **y** | p2 | **y** | p6 | 0 | **Ready!** |
| add | p6 | n | p4 | y | p7 | 1 | |
| sub | p5 | **y** | p2 | **y** | p8 | 2 | **Ready!** |
| addi | p8 | n | --- | y | p9 | 3 | |

# Issue = Select + Wakeup

- **Wakeup** dependent instructions
  - CAM search for destination (Dst) in inputs
  - Set "ready" bit
  - Also update ready-bit table for future instructions

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| xor | p1 | y | p2 | y | **p6** | 0 |
| add | **p6** | **y** | p4 | y | p7 | 1 |
| sub | p5 | y | p2 | y | **p8** | 2 |
| addi | **p8** | **y** | --- | y | p9 | 3 |

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| **p6** | **y** |
| p7 | n |
| **p8** | **y** |
| p9 | n |

# Issue

- **Select/Wakeup** one cycle
- Dependents go back to back
  - Next cycle: add/addi are ready:

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| | | | | | | |
| add | p6 | **y** | p4 | y | p7 | 1 |
| | | | | | | |
| addi | p8 | **y** | --- | y | p9 | 3 |

# When Does Register Read Occur?

- Current approach: after select, right before execute
  - **Not during in-order part of pipeline, in out-of-order part**
  - Read **physical** register (renamed)
  - Or get value via bypassing (based on physical register name)
  - This is Pentium 4, MIPS R10k, Alpha 21264, IBM Power4, Intel's "Sandy Bridge" (2011)
  - Physical register file may be large
    - Multi-cycle read

- Older approach:
  - Read as part of "issue" stage, keep values in Issue Queue
  - Pentium Pro, Core 2, Core i7
  - Simpler, but less energy efficient (move more values around)

# Renaming review

Everyone rename this instruction:

mul r4 * r5 -> r1

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| r3 | p3 |
| r4 | p4 |
| r5 | p5 |

Map table

| |
|---|
| p6 |
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Dispatch Review

Everyone dispatch this instruction:

div p7 / p6 -> p1

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| | | | | | | |

**Ready bits**

| | |
|---|---|
| p1 | y |
| p2 | y |
| p3 | y |
| p4 | y |
| p5 | y |
| p6 | n |
| p7 | y |
| p8 | y |
| p9 | y |

# Select Review

| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| add | p3 | y | p1 | y | p2 | 0 |
| mul | p2 | n | p4 | y | p5 | 1 |
| div | p1 | y | p5 | n | p6 | 2 |
| xor | p4 | y | p1 | y | p9 | 3 |

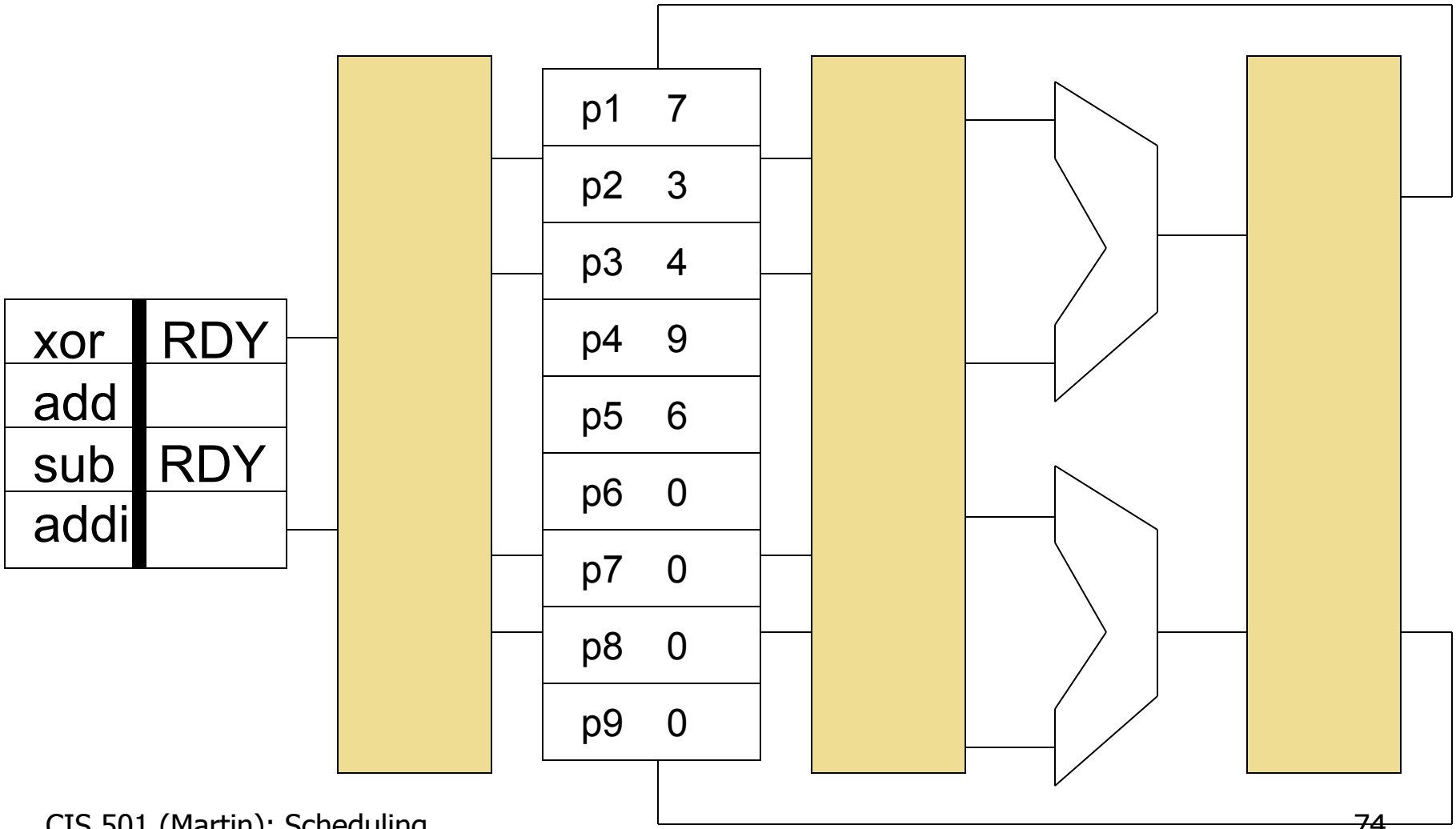
Determine which instructions are ready.
Which will be issued on a 1-wide machine?
Which will be issued on a 2-wide machine?

# Wakeup Review

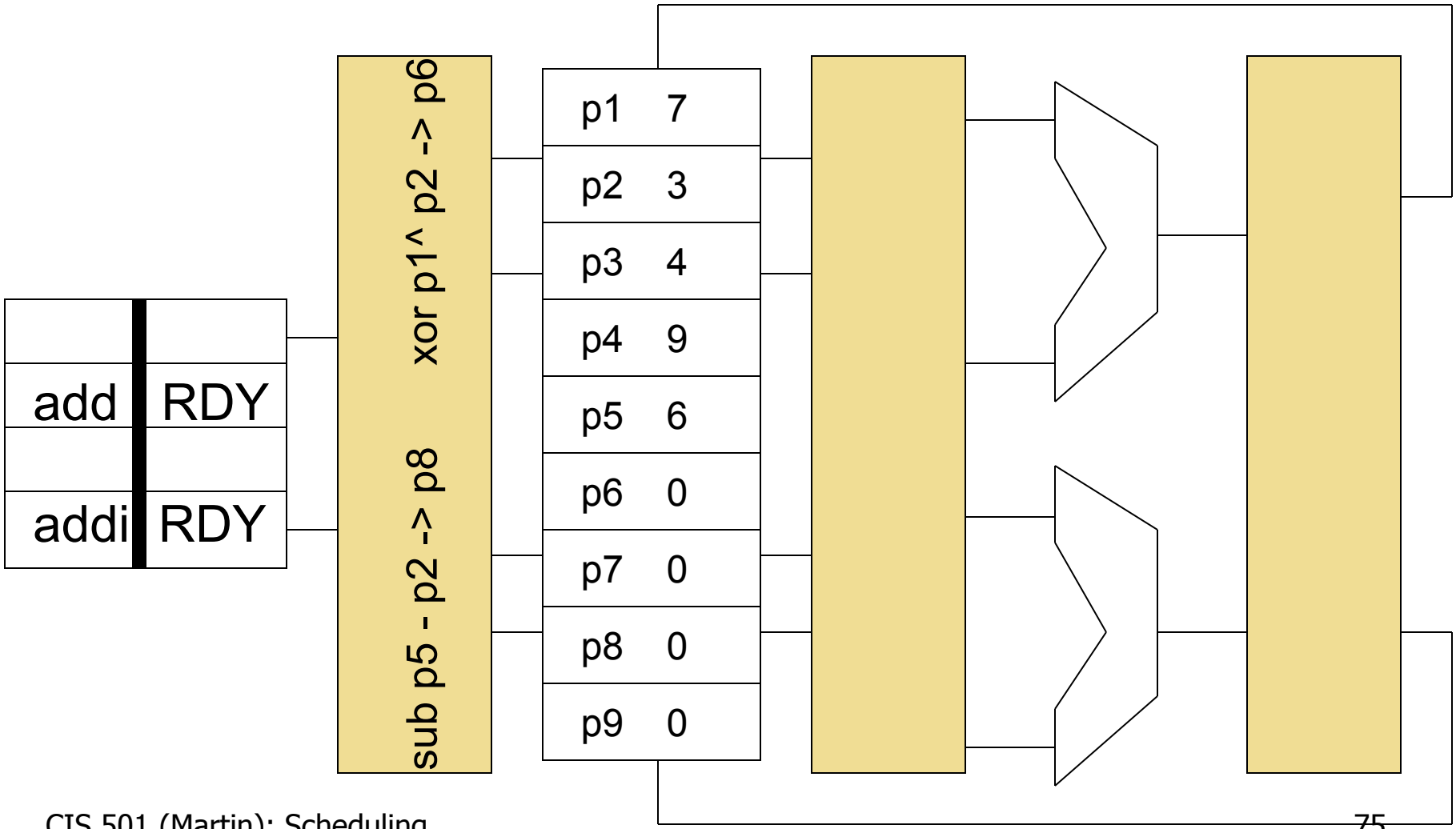
| Insn | Inp1 | R | Inp2 | R | Dst | Age |
|---|---|---|---|---|---|---|
| add | p3 | y | p1 | y | p2 | 0 |
| mul | p2 | n | p4 | y | p5 | 1 |
| div | p1 | y | p5 | n | p6 | 2 |
| xor | p4 | y | p1 | y | p9 | 3 |

What information will change if we issue the add?

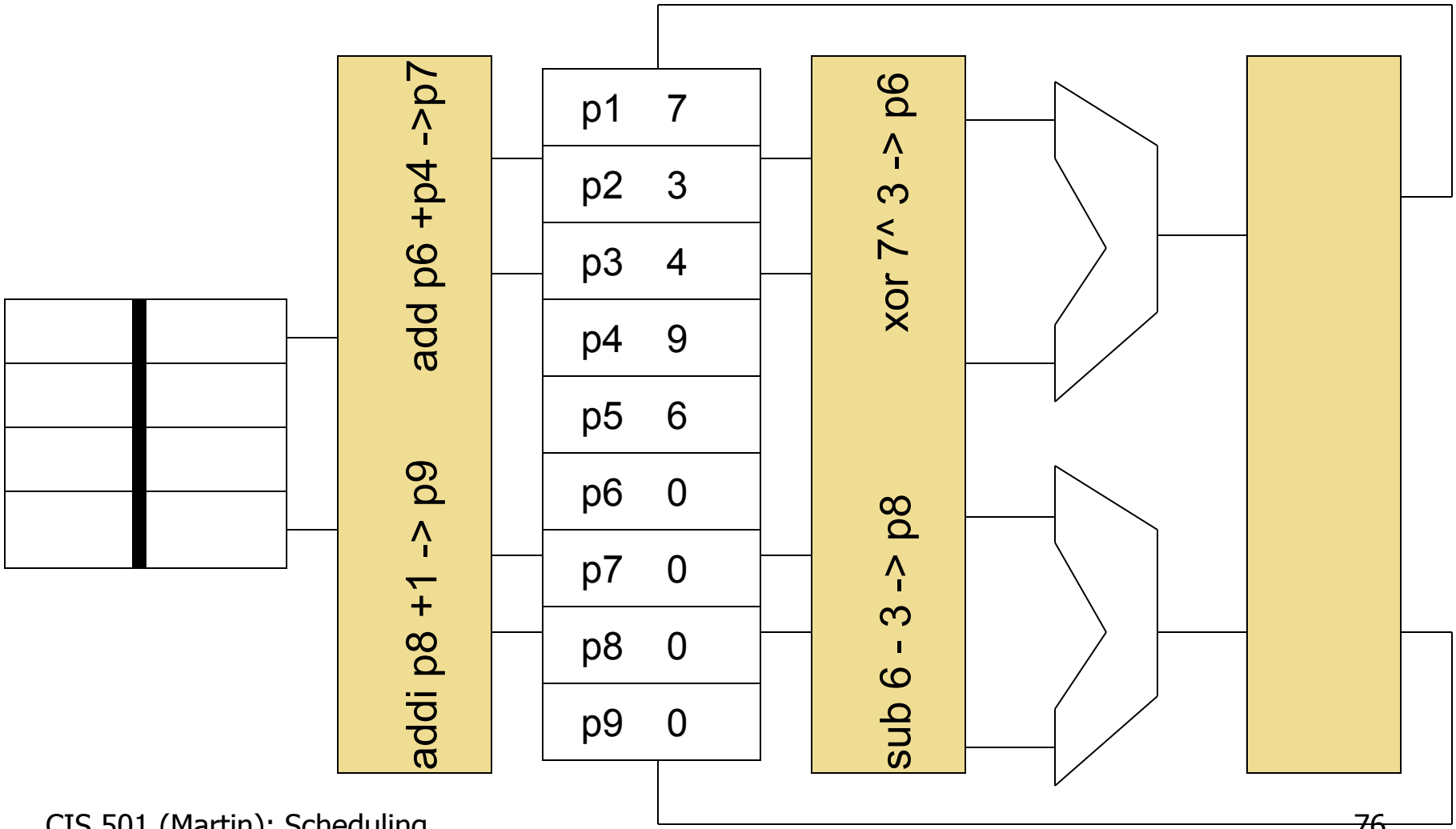
# OOO execution (2-wide)



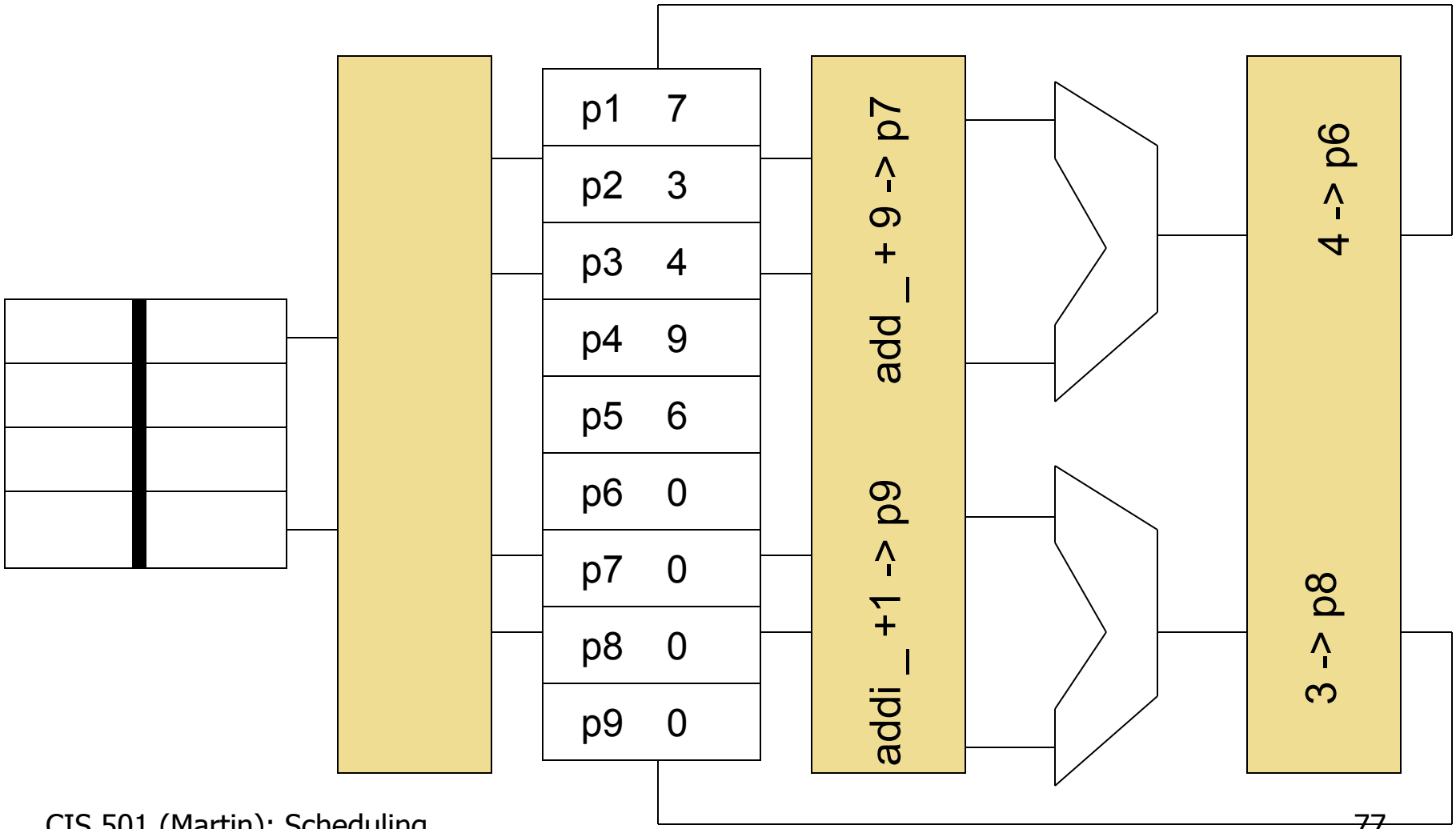
# OOO execution (2-wide)



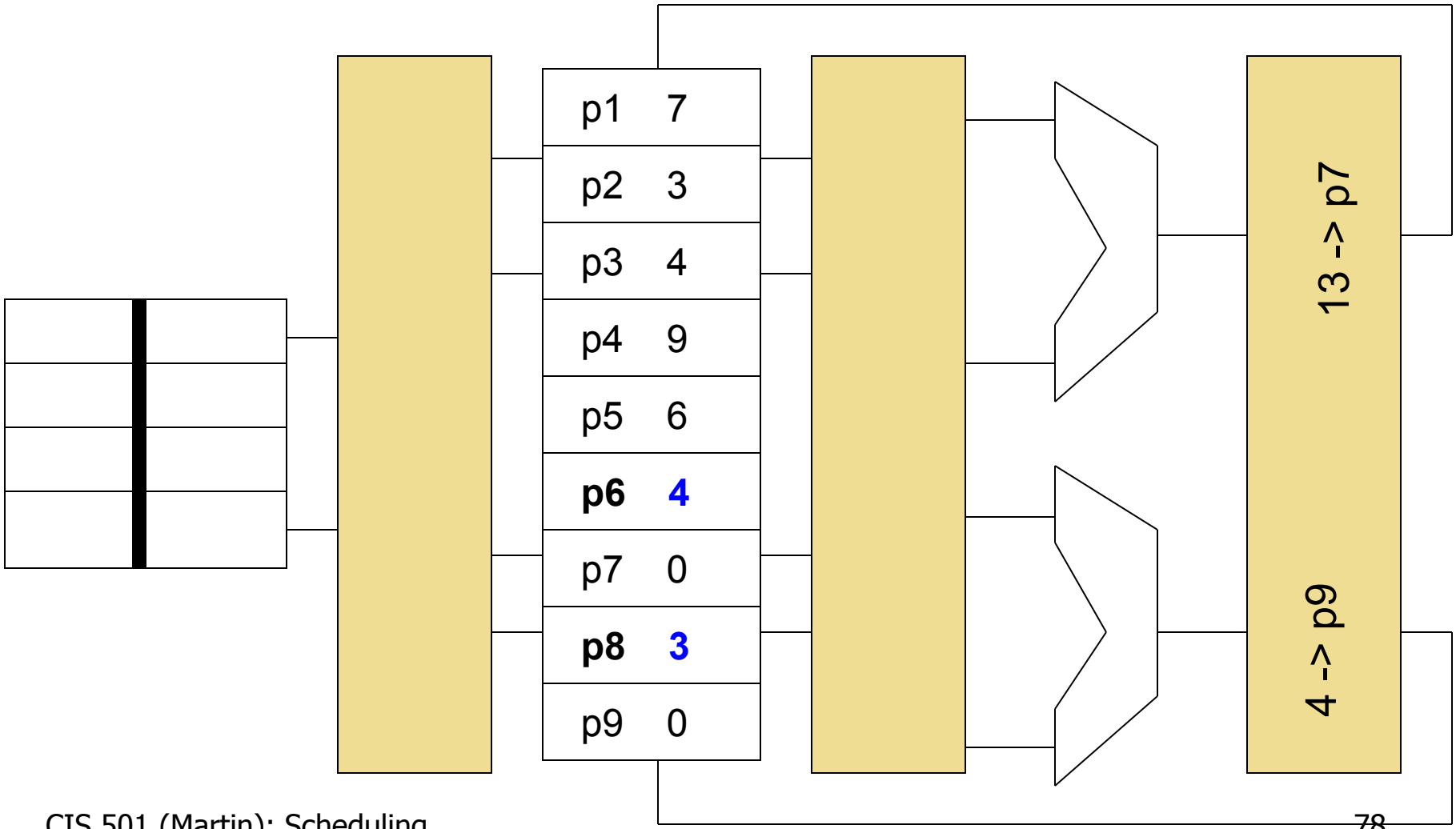
# OOO execution (2-wide)



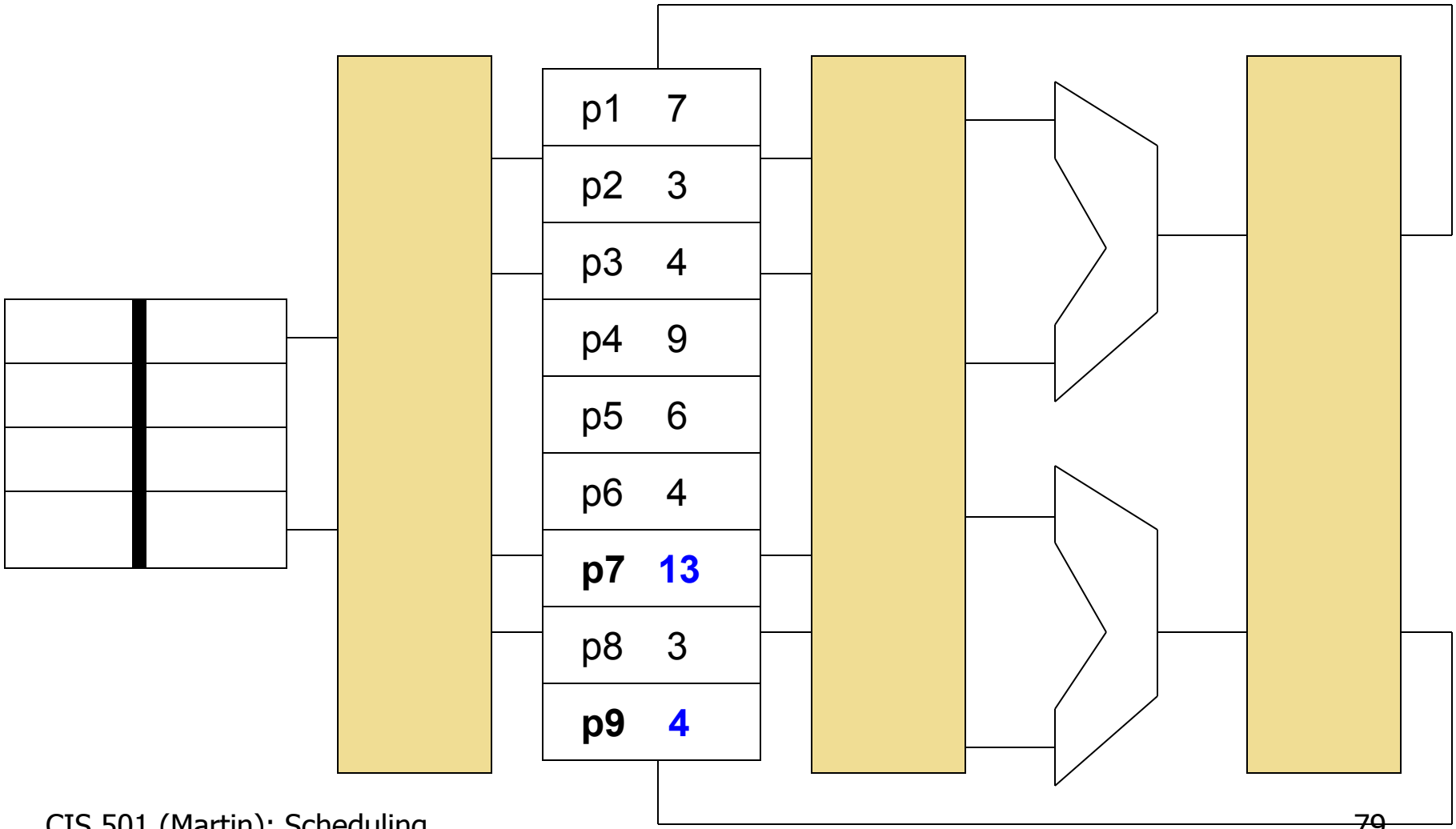
# OOO execution (2-wide)



# OOO execution (2-wide)

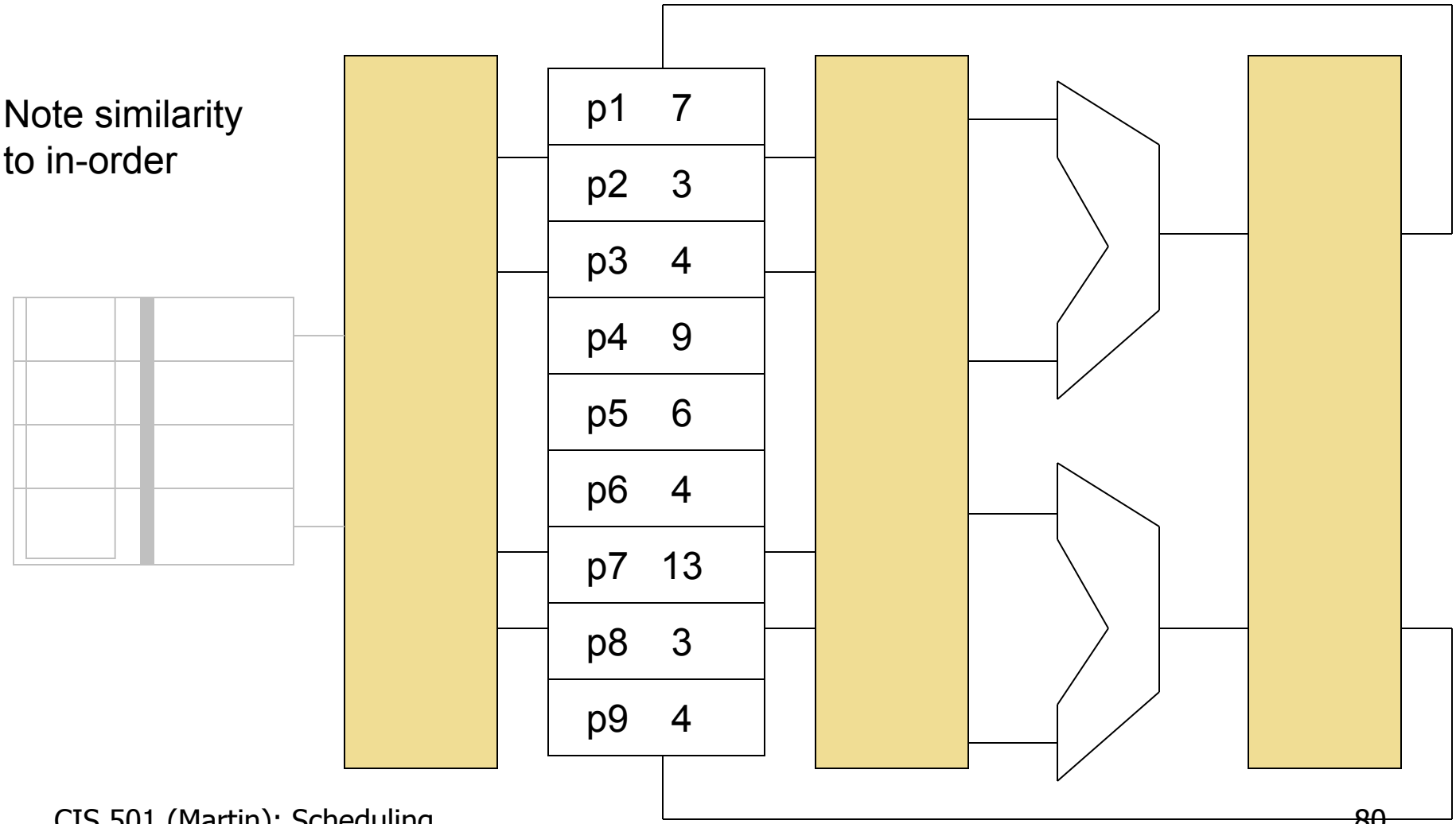


# OOO execution (2-wide)



# OOO execution (2-wide)

Note similarity
to in-order



# Multi-cycle operations

- Multi-cycle ops (load, fp, multiply, etc)
  - Wakeup deferred a few cycles
  - Include checks to avoid structural hazards

- Cache misses?
  - Speculative wake-up (assume hit)
  - Cancel exec of dependents
  - Re-issue later
  - Details: complicated, not important

# RENAMING REVISITED

# Re-order Buffer (ROB)

- ROB entry holds all info for recover/commit
  - All instructions in order
  - Logical register names, physical register names, instruction types
- Purpose: tracking for in-order commit
  - Maintain appearance of in-order execution
  - Used to support:
    - **Misprediction recovery**
    - **Freeing of physical registers**

- Dispatch: insert at tail
  - Full? Stall
- Commit: remove from head
  - Not completed? Stall

# Renaming revisited

- Track (or "log") the "overwritten register"
  - Freed this register at commit
  - Also used to restore in map table on recovery
    - Branch mis-prediction recovery

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| r3 | p3 |
| r4 | p4 |
| r5 | p5 |

Map table

| |
|---|
| p6 |
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 ->                    [ p3 ]
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | **p3** |
| r4 | p4 |
| r5 | p5 |

Map table

| p6 |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6                [ p3 ]
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | **p6** |
| r4 | p4 |
| r5 | p5 |

Map table

| |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 ->
```

[ p3 ]
[ p4 ]

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| r3 | p6 |
| r4 | **p4** |
| r5 | p5 |

Map table

| |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
```

```
[ p3 ]
[ p4 ]
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p6 |
| r4 | **p7** |
| r5 | p5 |

Map table

| |
|---|
| |
| |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```



```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 ->
```

[ p3 ]
[ p4 ]
[ p6 ]

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | **p6** |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| |
| |
| p8 |
| p9 |
| p10 |

Free-list

# Renaming example

```
xor r1 ^ r2 -> r3          xor  p1 ^ p2 -> p6          [ p3 ]
add r3 + r4 -> r4          add p6 + p4 -> p7           [ p4 ]
sub r5 - r2 -> r3          sub p5 - p2 -> p8           [ p6 ]
addi r3 + 1 -> r1
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | **p8** |
| r4 | p7 |
| r5 | p5 |

Map table

p9

p10

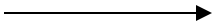
Free-list

# Renaming example

```
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 ->
```

```
[ p3 ]
[ p4 ]
[ p6 ]
[ p1 ]
```



| | |
|---|---|
| r1 | **p1** |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| p9 |
| p10 |

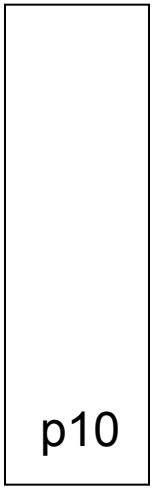
Free-list

# Renaming example

| | | |
|---|---|---|
| xor r1 ^ r2 -> r3 | xor p1 ^ p2 -> p6 | [ p3 ] |
| add r3 + r4 -> r4 | add p6 + p4 -> p7 | [ p4 ] |
| sub r5 - r2 -> r3 | sub p5 - p2 -> p8 | [ p6 ] |
| addi r3 + 1 -> r1 | addi p8 + 1 -> p9 | [ p1 ] |

| | |
|---|---|
| r1 | **p9** |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table




Free-list

# Recovery

- Completely remove wrong path instructions
  - Flush from IQ
  - Remove from ROB
  - Restore map table to before misprediction
  - Free destination registers

# Recovery example

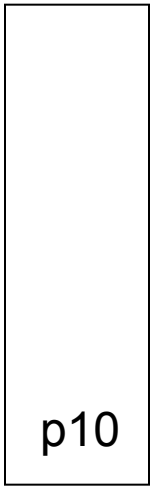
```
bnz r1 loop
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
bnz p1, loop
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

```
[     ]
[ p3 ]
[ p4 ]
[ p6 ]
[ p1 ]
```

| | |
|---|---|
| r1 | p9 |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table




Free-list

# Recovery example

```
bnz r1 loop
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
addi r3 + 1 -> r1
```

```
bnz p1, loop
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
addi p8 + 1 -> p9
```

```
[     ]
[ p3 ]
[ p4 ]
[ p6 ]
[ p1 ]
```

| r1 | **p1** |
|---|---|
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| |
| |
| |
| p9 |
| p10 |

Free-list

# Recovery example

```
bnz r1 loop
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
sub r5 - r2 -> r3
```

```
bnz p1, loop
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
sub p5 - p2 -> p8
```

```
[     ]
[ p3 ]
[ p4 ]
[ p6 ]
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | **p6** |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| |
| |
| p8 |
| p9 |
| p10 |

Free-list

# Recovery example

```
bnz r1 loop
xor r1 ^ r2 -> r3
add r3 + r4 -> r4
```

```
bnz p1, loop
xor  p1 ^ p2 -> p6
add p6 + p4 -> p7
```

```
[     ]
[ p3 ]
[ p4 ]
```

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p6 |
| r4 | **p4** |
| r5 | p5 |

Map table

| |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Recovery example

bnz r1 loop
xor r1 ^ r2 -> r3

bnz p1, loop
xor p1 ^ p2 -> p6

[ ]
[ p3 ]

| | |
|---|---|
| r1 | p1 |
| r2 | p2 |
| r3 | **p3** |
| r4 | p4 |
| r5 | p5 |

Map table

| |
|---|
| p6 |
| p7 |
| p8 |
| p9 |
| p10 |

Free-list

# Recovery example

bnz r1 loop

bnz p1, loop

[ ]

| r1 | p1 |
|---|---|
| r2 | p2 |
| r3 | p3 |
| r4 | p4 |
| r5 | p5 |

Map table

| p6 |
|---|
| p7 |
| p8 |
| p9 |
| p10 |

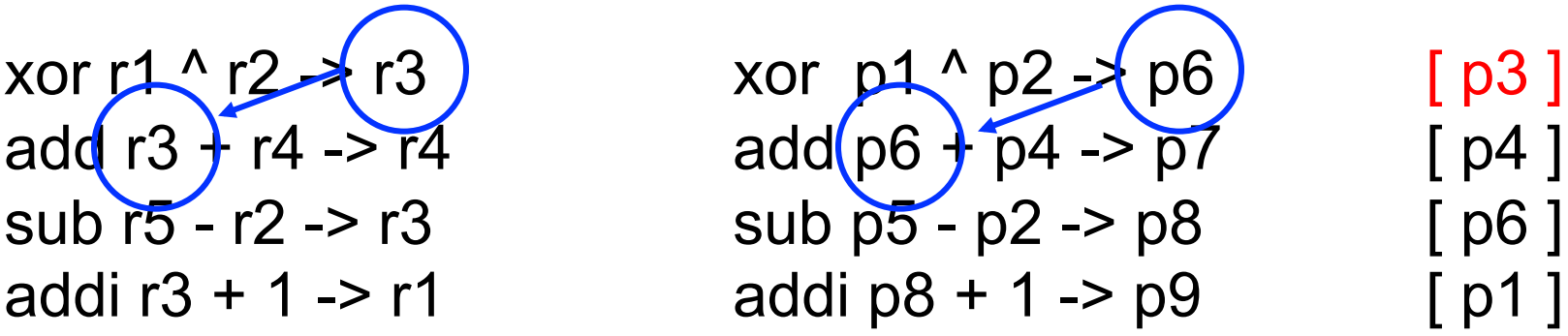
Free-list

# Commit

| | | |
|---|---|---|
| xor r1 ^ r2 -> r3 | xor p1 ^ p2 -> p6 | [ p3 ] |
| add r3 + r4 -> r4 | add p6 + p4 -> p7 | [ p4 ] |
| sub r5 - r2 -> r3 | sub p5 - p2 -> p8 | [ p6 ] |
| addi r3 + 1 -> r1 | addi p8 + 1 -> p9 | [ p1 ] |

- Commit: instruction becomes **architected state**
  - In-order, only when instructions are finished
  - Free overwritten register (why?)

# Freeing over-written register




- P3 was r3 **before** xor
- P6 is r3 **after** xor
  - Anything older than xor should read p3
  - Anything younger than xor should p6 (until next r3 writing instruction
- At commit of xor, no older instructions exist

# Commit Example

| | | |
|---|---|---|
| xor r1 ^ r2 -> r3 | xor p1 ^ p2 -> p6 | [ p3 ] |
| add r3 + r4 -> r4 | add p6 + p4 -> p7 | [ p4 ] |
| sub r5 - r2 -> r3 | sub p5 - p2 -> p8 | [ p6 ] |
| addi r3 + 1 -> r1 | addi p8 + 1 -> p9 | [ p1 ] |

| | |
|---|---|
| r1 | p9 |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| p10 |

Free-list

# Commit Example

| | | |
|---|---|---|
| xor r1 ^ r2 -> r3 | xor p1 ^ p2 -> p6 | [ p3 ] |
| add r3 + r4 -> r4 | add p6 + p4 -> p7 | [ p4 ] |
| sub r5 - r2 -> r3 | sub p5 - p2 -> p8 | [ p6 ] |
| addi r3 + 1 -> r1 | addi p8 + 1 -> p9 | [ p1 ] |

| | |
|---|---|
| r1 | p9 |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| p10 |
| p3 |

Free-list

# Commit Example

| | | |
|---|---|---|
| add r3 + r4 -> r4 | add p6 + p4 -> p7 | [ p4 ] |
| sub r5 - r2 -> r3 | sub p5 - p2 -> p8 | [ p6 ] |
| addi r3 + 1 -> r1 | addi p8 + 1 -> p9 | [ p1 ] |

| | |
|---|---|
| r1 | p9 |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| p10 |
| p3 |
| p4 |

Free-list

# Commit Example

| | | |
|---|---|---|
| sub r5 - r2 -> r3 | sub p5 - p2 -> p8 | [ p6 ] |
| addi r3 + 1 -> r1 | addi p8 + 1 -> p9 | [ p1 ] |

| | |
|---|---|
| r1 | p9 |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| p10 |
| p3 |
| p4 |
| p6 |
| |

Free-list

# Commit Example

addi r3 + 1 -> r1 addi p8 + 1 -> p9 [ p1 ]

| | |
|---|---|
| r1 | p9 |
| r2 | p2 |
| r3 | p8 |
| r4 | p7 |
| r5 | p5 |

Map table

| |
|---|
| p10 |
| p3 |
| p4 |
| p6 |
| p1 |

Free-list

# OUT-OF-ORDER PIPELINE EXAMPLE

# Recall: Motivating Example

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [p1] -> p2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add p2 + p3 -> p4 | F | Di | | | | I | RR | X | W | C | | | |
| xor p4 ^ p5 -> p6 | | F | Di | | | | I | RR | X | W | C | | |
| ld [p7] -> p8 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | C | | |

- How would this execution occur cycle-by-cycle?

# Out-of-Order Pipeline – Cycle 0

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | | | | | | | | | | | | |
| add r2 + r3 -> r4 | F | | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | | | | | | | | | | | | |
| ld [r7] -> r4 | | | | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p7 |
| r3 | p6 |
| r4 | p5 |
| r5 | p4 |
| r6 | p3 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | --- |
| p10 | --- |
| p11 | --- |
| p12 | --- |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | | no |
| add | | no |
| | | |
| | | |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | 110 |

# Out-of-Order Pipeline – Cycle 1a

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | | | | | | | | | | | |
| add r2 + r3 -> r4 | F | | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | | | | | | | | | | | | |
| ld [r7] -> r4 | | | | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p5 |
| r5 | p4 |
| r6 | p3 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | --- |
| p11 | --- |
| p12 | --- |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | | no |
| | | |
| | | |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ld | p8 | yes | --- | yes | p9 | 0 |
| | | | | | | |
| | | | | | | |
| | | | | | | 111 |

# Out-of-Order Pipeline – Cycle 1b

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | | | | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | | | | | | | | | | | | |
| ld [r7] -> r4 | | | | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p10 |
| r5 | p4 |
| r6 | p3 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | no |
| p11 | --- |
| p12 | --- |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| | | |
| | | |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ld | p8 | yes | --- | yes | p9 | 0 |
| add | p9 | no | p6 | yes | p10 | 1 |
| | | | | | | |
| | | | | | | |

# Out-of-Order Pipeline – Cycle 1c

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | | | | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | | | | | | | | | | | |
| ld [r7] -> r4 | | F | | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p10 |
| r5 | p4 |
| r6 | p3 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | no |
| p11 | --- |
| p12 | --- |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | | no |
| ld | | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ld | p8 | yes | --- | yes | p9 | 0 |
| add | p9 | no | p6 | yes | p10 | 1 |
| | | | | | | |
| | | | | | | |

# Out-of-Order Pipeline – Cycle 2a

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | | | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | | | | | | | | | | | |
| ld [r7] -> r4 | | F | | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p10 |
| r5 | p4 |
| r6 | p3 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | no |
| p11 | --- |
| p12 | --- |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | | no |
| ld | | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ld | p8 | yes | --- | yes | p9 | 0 |
| add | p9 | no | p6 | yes | p10 | 1 |
| | | | | | | |
| | | | | | | |

# Out-of-Order Pipeline – Cycle 2b

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | | | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | | | | | | | |
| ld [r7] -> r4 | | F | | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p10 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | no |
| p11 | no |
| p12 | --- |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| add | p9 | no | p6 | yes | p10 | 1 |
| xor | p10 | no | p4 | yes | p11 | 2 |
| | | | | | | |

# Out-of-Order Pipeline – Cycle 2c

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | | | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | | | | | | | |
| ld [r7] -> r4 | | F | Di | | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | no |
| p11 | no |
| p12 | no |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| add | p9 | no | p6 | yes | p10 | 1 |
| xor | p10 | no | p4 | yes | p11 | 2 |
| ld | p2 | yes | --- | yes | p12 | 3 |

# Out-of-Order Pipeline – Cycle 3

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | | | | | | | |
| ld [r7] -> r4 | | F | Di | I | | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | no |
| p10 | no |
| p11 | no |
| p12 | no |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| add | p9 | no | p6 | yes | p10 | 1 |
| xor | p10 | no | p4 | yes | p11 | 2 |
| ld | p2 | yes | --- | yes | p12 | 3 |

# Out-of-Order Pipeline – Cycle 4

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | | | | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | yes |
| p10 | no |
| p11 | no |
| p12 | no |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| add | p9 | yes | p6 | yes | p10 | 1 |
| xor | p10 | no | p4 | yes | p11 | 2 |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 5a

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | | | | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | no |
| p12 | no |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| add | p9 | yes | p6 | yes | p10 | 1 |
| xor | p10 | yes | p4 | yes | p11 | 2 |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 5b

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | | | | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | no |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| xor | p10 | yes | p4 | yes | p11 | 2 |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 6

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | | | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | no |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| xor | p10 | yes | p4 | yes | p11 | 2 |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 7

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | yes |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ld | p7 | yes |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | — | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 8a

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | --- |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ~~ld~~ | ~~p7~~ | ~~yes~~ |
| add | p5 | no |
| xor | p3 | no |
| ld | p10 | no |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | --- | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 8b

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | W | | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | X | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | yes |
| p6 | yes |
| p7 | --- |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ~~ld~~ | ~~p7~~ | ~~yes~~ |
| add | p5 | yes |
| xor | p3 | no |
| ld | p10 | yes |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | --- | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 9a

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | W | C | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | X | | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | --- |
| p6 | yes |
| p7 | --- |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ~~ld~~ | ~~p7~~ | ~~yes~~ |
| ~~add~~ | ~~p5~~ | ~~yes~~ |
| xor | p3 | no |
| ld | p10 | yes |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | --- | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 9b

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | W | C | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | X | W | | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | yes |
| p4 | yes |
| p5 | --- |
| p6 | yes |
| p7 | --- |
| p8 | yes |
| p9 | yes |
| p10 | yes |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ~~ld~~ | ~~p7~~ | ~~yes~~ |
| ~~add~~ | ~~p5~~ | ~~yes~~ |
| xor | p3 | yes |
| ld | p10 | yes |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | --- | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Cycle 10

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | W | C | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | X | W | C | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | C | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | --- |
| p4 | yes |
| p5 | --- |
| p6 | yes |
| p7 | --- |
| p8 | yes |
| p9 | yes |
| p10 | --- |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ~~ld~~ | ~~p7~~ | ~~yes~~ |
| ~~add~~ | ~~p5~~ | ~~yes~~ |
| ~~xor~~ | ~~p3~~ | ~~yes~~ |
| ~~ld~~ | ~~p10~~ | ~~yes~~ |

Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | --- | ~~yes~~ | ~~p12~~ | ~~3~~ |

# Out-of-Order Pipeline – Done!

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [r1] -> r2 | F | Di | I | RR | X | $M_1$ | $M_2$ | W | C | | | | |
| add r2 + r3 -> r4 | F | Di | | | | I | RR | X | W | C | | | |
| xor r4 ^ r5 -> r6 | | F | Di | | | | I | RR | X | W | C | | |
| ld [r7] -> r4 | | F | Di | I | RR | X | $M_1$ | $M_2$ | W | | C | | |

Map Table

| | |
|---|---|
| r1 | p8 |
| r2 | p9 |
| r3 | p6 |
| r4 | p12 |
| r5 | p4 |
| r6 | p11 |
| r7 | p2 |
| r8 | p1 |

Ready Table

| | |
|---|---|
| p1 | yes |
| p2 | yes |
| p3 | --- |
| p4 | yes |
| p5 | --- |
| p6 | yes |
| p7 | --- |
| p8 | yes |
| p9 | yes |
| p10 | --- |
| p11 | yes |
| p12 | yes |

Reorder Buffer

| Insn | To Free | Done? |
|---|---|---|
| ~~ld~~ | ~~p7~~ | ~~yes~~ |
| ~~add~~ | ~~p5~~ | ~~yes~~ |
| ~~xor~~ | ~~p3~~ | ~~yes~~ |
| ~~ld~~ | ~~p10~~ | ~~yes~~ |

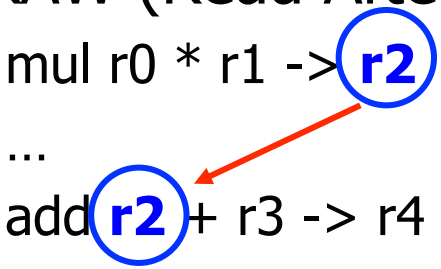
Issue Queue

| Insn | Src1 | R? | Src2 | R? | Dest | Age |
|---|---|---|---|---|---|---|
| ~~ld~~ | ~~p8~~ | ~~yes~~ | | ~~yes~~ | ~~p9~~ | ~~0~~ |
| ~~add~~ | ~~p9~~ | ~~yes~~ | ~~p6~~ | ~~yes~~ | ~~p10~~ | ~~1~~ |
| ~~xor~~ | ~~p10~~ | ~~yes~~ | ~~p4~~ | ~~yes~~ | ~~p11~~ | ~~2~~ |
| ~~ld~~ | ~~p2~~ | ~~yes~~ | --- | ~~yes~~ | ~~p12~~ | ~~3~~ |

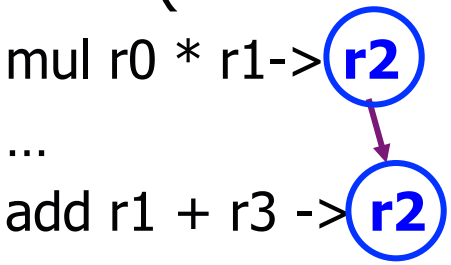
# HANDLING MEMORY OPERATIONS

# Recall: Types of Dependencies

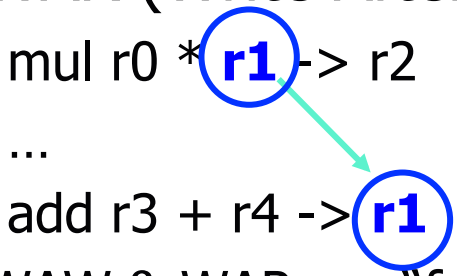
- RAW (Read After Write) = "true dependence"




- WAW (Write After Write) = "output dependence"



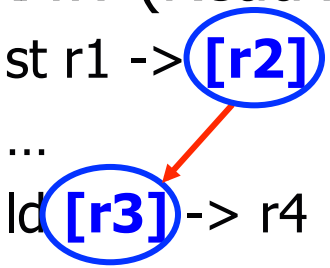

- WAR (Write After Read) = "anti-dependence"



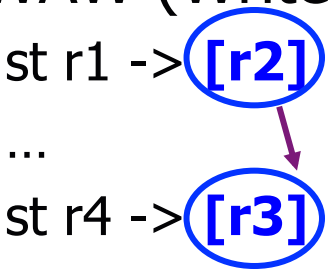

- WAW & WAR are "false", Can be **totally eliminated** by "renaming"

# Also Have Dependencies via Memory

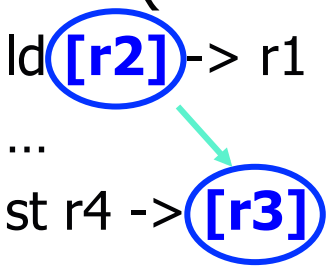
- **If value in "r2" and "r3" is the same…**
- RAW (Read After Write) – True dependency




- WAW (Write After Write)




- WAR (Write After Read)




WAR/WAW are "false dependencies"

- Can't rename memory in same way as registers
  - **Why?**
- Need to use other tricks

# Let's Start with Just Stores

- Stores: Write data cache, not registers
  - Can we rename memory?
  - Recover in the cache?
- ➢ No (at least not easily)
  - Cache writes unrecoverable
- Solution: write stores into cache only when certain

# Handling Stores

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mul p1 * p2 -> p3 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | W | C | | | |
| jump-not-zero p3 | F | Di | | | | | I | RR | X | W | C | | |
| st p5 -> [p3+4] | | F | Di | | | | I | RR | X | W | C | | |
| st p4 -> [p6+8] | | F | Di | I? | | | | | | | | | |

- Can "st p4 -> [p6+8]" issue and begin execution?
  - Its registers inputs are ready…
  - Why or why not?

# Problem #1: Out-of-Order Stores

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mul p1 * p2 -> p3 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | W | C | | | |
| jump-not-zero p3 | F | Di | | | | | I | RR | X | W | C | | |
| st p5 -> [p3+4] | | F | Di | | | | I | RR | X | **M** | W | C | |
| st p4 -> [p6+8] | | F | Di | **I?** | **RR** | **X** | **M** | **W** | | | | C | |

- Can "st p4 -> [p6+8]" write the cache in cycle 6?
  - "st p5 -> [p3+4]" has not yet executed
- What if "p3+4 == p6+8"
  - The two stores write the same address! WAW dependency!
  - Not known until their "X" stages (cycle 5 & 8)
- Unappealing solution: all stores execute in-order
- We can do better…

# Problem #2: Speculative Stores

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mul p1 * p2 -> p3 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | W | C | | | |
| jump-not-zero p3 | F | Di | | | | | I | RR | **X** | W | C | | |
| st p5 -> [p3+4] | | F | Di | | | | I | RR | X | M | W | C | |
| st p4 -> [p6+8] | | F | Di | **I?** | **RR** | **X** | **M** | **W** | | | | **C** | |

- Can "st p4 -> [p6+8]" write the cache in cycle 6?
  - Store is still "speculative" at this point
- What if "jump-not-zero" is mis-predicted?
  - Not known until its "X" stage (cycle 8)
- How does it "undo" the store once it hits the cache?
  - Answer: it can't; stores write the cache only at **commit**
  - Guaranteed to be non-speculative at that point

# Store Queue (SQ)

- Two problems
  - Speculative stores
  - Out-of-order stores
- Solution: Store Queue (SQ)
  - **At dispatch, each store is given a slot in the Store Queue**
  - First-in-first-out (FIFO) queue
  - Each entry contains: "address", "value", and "age"
- Operation:
  - Dispatch (in-order): allocate entry in SQ (stall if full)
  - Execute (out-of-order): write store value into store queue
  - Commit (in-order): read value from SQ and write into data cache
  - Branch recovery: remove entries from the store queue
- Address the above two problems, plus more…

# Loads and Stores

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fdiv p1 / p2 -> p9 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | W | C | |
| st p4 -> [p5+4] | F | Di | I | RR | X | W | | | | | | C | |
| st p3 -> [p6+8] | | F | Di | I | RR | X | W | | | | | | C |
| ld [p7] -> p8 | | F | Di | I? | RR | X | $M_1$ | $M_2$ | W | | | | C |

- Can "ld [p7] -> p8" issue and begin execution?
  - Why or why not?

# Loads and Stores

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fdiv p1 / p2 -> p9 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | W | C | |
| st p4 -> [p5+4] | F | Di | I | RR | X | SQ | | | | | | C | |
| st p3 -> [p6+8] | | F | Di | I | RR | X | SQ | | | | | | C |
| ld [p7] -> p8 | | F | Di | I? | RR | X | $M_1$ | $M_2$ | W | | | | C |

- Can "ld [p7] -> p8" issue and begin execution?
  - Why or why not?
- If the load reads from either of the store's addresses…
  - The load must get the value, but it isn't written to the cache until commit…

# Loads and Stores

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| fdiv p1 / p2 -> p9 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | W | C | |
| st p4 -> [p5+4] | F | Di | I | RR | X | SQ | | | | | | C | |
| st p3 -> [p6+8] | | F | Di | I | RR | X | SQ | | | | | | C |
| ld [p7] -> p8 | | F | Di | I? | RR | X | $M_1$ | $M_2$ | W | | | | C |

- Can "ld [p7] -> p8" issue and begin execution?
  - Why or why not?
- If the load reads from either of the store's addresses…
  - The load must get the value, but it isn't written to the cache until commit…
- Solution: "memory forwarding"
  - Loads also read from the Store Queue (in parallel with the cache)

# Memory Forwarding

- Stores write cache at commit
  - Why? Allows stores to be "undone" on branch mis-predictions, etc.
  - Commit is in-order, delayed until all prior instructions are done
- Loads read cache
  - Early execution of loads is critical
- Forwarding
  - Allow store to load communication before store commit
  - Conceptually like register bypassing, but different implementation
    - Why? Addresses unknown until execute

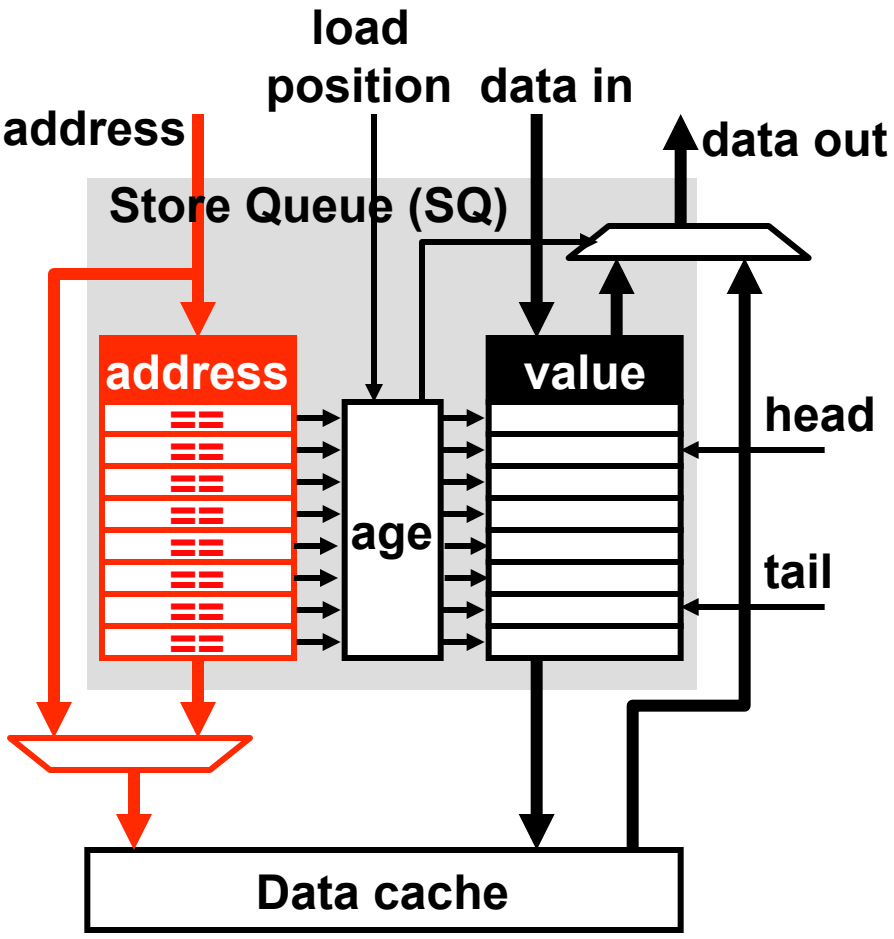
# Problem #3: WAR Hazards

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mul p1 * p2 -> p3 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | W | C | | | |
| jump-not-zero p3 | F | Di | | | | | I | RR | X | W | C | | |
| ld [p3+4] -> p5 | | F | Di | | | | I | RR | X | $M_1$ | $M_2$ | W | C |
| st p4 -> [p6+8] | | F | Di | I | RR | X | SQ | | | | | | C |

- What if "p3+4 == p6 + 8"?
  - Then load and store access same memory location
- Need to make sure that load doesn't read store's result
  - Need to get values based on "program order" not "execution order"
- Bad solution: require all stores/loads to execute in-order
- Good solution: add "age" fields to store queue (SQ)
  - Loads read matching address that is "earlier" (or "older") than it
  - Another reason the SQ is a FIFO queue

# Memory Forwarding via Store Queue

- Store Queue (SQ)
  - Holds all in-flight stores
  - CAM: searchable by address
  - Age logic: determine youngest matching store older than load
- Store rename/dispatch
  - Allocate entry in SQ
- Store execution
  - Update SQ
    - Address + Data
- Load execution
  - Search SQ identify youngest older matching store
    - Match? Read SQ
    - No Match? Read cache



# Store Queue (SQ)

- On load execution, select the store that is:
  - To same address as load
  - Older than the load (before the load in program order)
- Of these, select the youngest store
  - The store to the same address that immediately precedes the load

# When Can Loads Execute?

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mul p1 * p2 -> p3 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | W | C | | | |
| jump-not-zero p3 | F | Di | | | | | I | RR | **X** | W | C | | |
| st p5 -> [p3+4] | | F | Di | | | | I | RR | X | SQ | C | | |
| ld [p6+8] -> p7 | | F | Di | **I?** | **RR** | **X** | **$M_1$** | **$M_2$** | **W** | | | C | |

- Can "ld [p6+8] -> p7" issue in cycle 3
  - Why or why not?

# When Can Loads Execute?

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| mul p1 * p2 -> p3 | F | Di | I | RR | $X_1$ | $X_2$ | $X_3$ | $X_4$ | W | C | | | |
| jump-not-zero p3 | F | Di | | | | | I | RR | **X** | W | C | | |
| st p5 -> [p3+4] | | F | Di | | | | I | RR | X | SQ | C | | |
| ld [p6+8] -> p7 | | F | Di | **I?** | **RR** | **X** | **$M_1$** | **$M_2$** | **W** | | | C | |

- Aliasing! Does p3+4 == p6+8?
  - If no, load should get value from memory
    - **Can it start to execute?**
  - If yes, load should get value from store
    - By reading the store queue?
    - **But the value isn't put into the store queue until cycle 9**
- **Key challenge**: don't know addresses until execution!
  - One solution: require all loads to wait for all earlier (prior) stores

# Dynamically Scheduling Memory Ops

- Compilers must schedule memory ops conservatively
- Options for hardware:
  - Don't execute any load until all prior stores execute (conservative)
  - Execute loads as soon as possible, detect violations (optimistic)
    - When a store executes, it checks if any later loads executed too early (to same address). If so, flush pipeline
  - Learn violations over time, selectively reorder (predictive)

Before

```
ld r2,4(sp)
ld r3,8(sp)
add r3,r2,r1  //stall
st r1,0(sp)
ld r5,0(r8)
ld r6,4(r8)
sub r5,r6,r4  //stall
st r4,8(r8)
```

Wrong(?)

```
ld r2,4(sp)
ld r3,8(sp)
ld r5,0(r8) //does r8==sp?
add r3,r2,r1
ld r6,4(r8) //does r8+4==sp?
st r1,0(sp)
sub r5,r6,r4
st r4,8(r8)
```

# Conservative Load Scheduling

- Conservative load scheduling:
  - All older stores have executed
    - Some architectures: split store address / store data
      - Only requires knowing addresses (not the store values)
  - Advantage: always safe
  - Disadvantage: performance (limits out-of-orderness)

# Conservative Load Scheduling

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [p1] -> p4 | F | Di | I | Rr | X | $M_1$ | $M_2$ | W | C | | | | | | | |
| ld [p2] -> p5 | F | Di | I | Rr | X | $M_1$ | $M_2$ | W | C | | | | | | | |
| add p4, p5 -> p6 | | F | Di | | | I | Rr | X | W | C | | | | | | |
| st p6 -> [p3] | | F | Di | | | | I | Rr | X | SQ | C | | | | | |
| ld [p1+4] -> p7 | | | F | Di | | | | I | Rr | X | $M_1$ | $M_2$ | W | C | | |
| ld [p2+4] -> p8 | | | F | Di | | | | I | Rr | X | $M_1$ | $M_2$ | W | C | | |
| add p7, p8 -> p9 | | | | F | Di | | | | | | I | Rr | X | W | C | |
| st p9 -> [p3+4] | | | | F | Di | | | | | | | I | Rr | X | SQ | C |

**Conservative load scheduling: can't issue ld [p1+4] until cycle 7!**
**Might as well be an in-order machine on this example**
**Can we do better? How?**

# Optimistic Load Scheduling

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ld [p1] -> p4 | F | Di | I | Rr | X | $M_1$ | $M_2$ | W | C | | | | | | | |
| ld [p2] -> p5 | F | Di | I | Rr | X | $M_1$ | $M_2$ | W | C | | | | | | | |
| add p4, p5 -> p6 | | F | Di | | | I | Rr | X | W | C | | | | | | |
| st p6 -> [p3] | | F | Di | | | | I | Rr | X | SQ | C | | | | | |
| ld [p1+4] -> p7 | | | F | Di | I | Rr | X | $M_1$ | $M_2$ | W | C | | | | | |
| ld [p2+4] -> p8 | | | F | Di | I | Rr | X | $M_1$ | $M_2$ | W | | C | | | | |
| add p7, p8 -> p9 | | | | F | Di | | | I | Rr | X | W | C | | | | |
| st p9 -> [p3+4] | | | | F | Di | | | | I | Rr | X | SQ | C | | | |

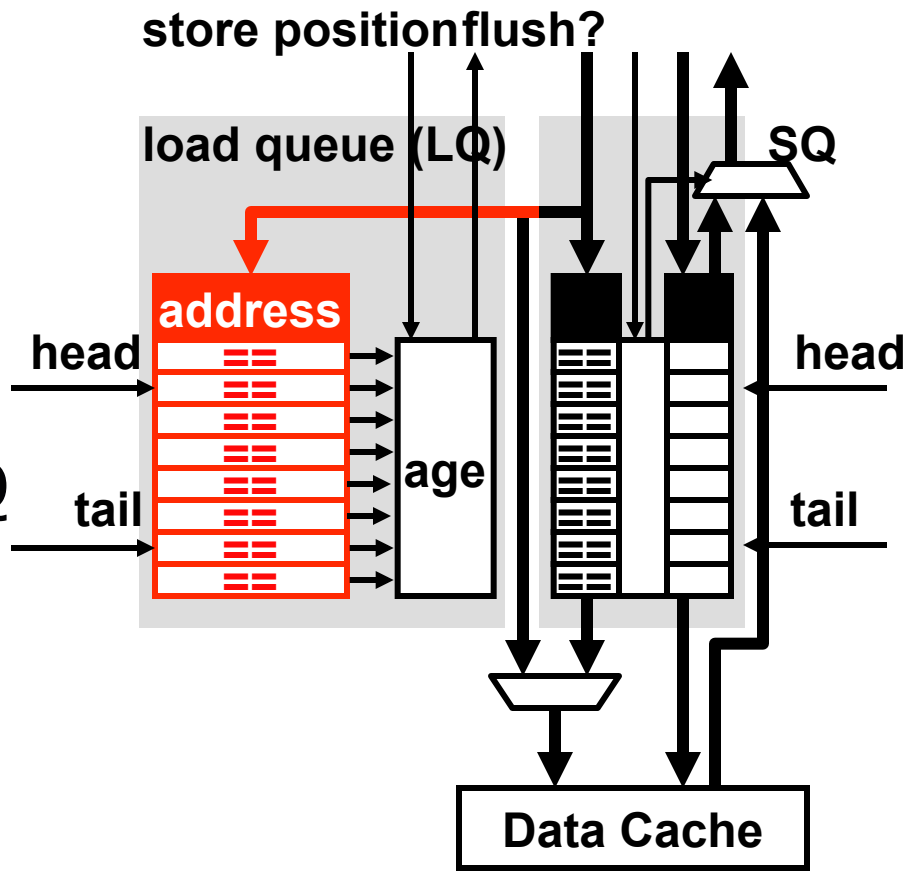
**Optimistic load scheduling: can actually benefit from out-of-order!**
**But how do we know when out speculation (optimism) fails?**

# Load Speculation

- Speculation requires two things…..
  - 1. Detection of mis-speculations
    - How can we do this?
  - 2. Recovery from mis-speculations
    - Squash from offending load
    - Saw how to squash from branches: same method

# Load Queue

- Detects load ordering violations
- Load execution: Write address into LQ
  - Also note any store forwarded from
- Store execution: Search LQ
  - Younger load with same addr?
  - Didn't forward from younger store? (optimization for full renaming)



# Store Queue + Load Queue

- Store Queue: handles forwarding
  - Entry per store (allocated @ dispatch, deallocated @ commit)
  - Written by stores (@ execute)
  - Searched by loads (@ execute)
  - Read from to write data cache (@ commit)
- Load Queue: detects ordering violations
  - Entry per load (allocated @ dispatch, deallocated @ commit)
  - Written by loads (@ execute)
  - Searched by stores (@ execute)
- Both together
  - Allows aggressive load scheduling
  - Stores don't constrain load execution

# Optimistic Load Scheduling Problem

- Allows loads to issue before older stores
  - Increases out-of-orderness
  + Good: When no conflict, increases performance
  - Bad: Conflict => squash => worse performance than waiting

- Can we have our cake AND eat it too?

# Predictive Load Scheduling

- Predict which loads must wait for stores

- Fool me once, shame on you-- fool me twice?
  - Loads default to aggressive
  - Keep table of load PCs that have been caused squashes
    - Schedule these conservatively
  + Simple predictor
  - Makes "bad" loads wait for all older stores is not so great

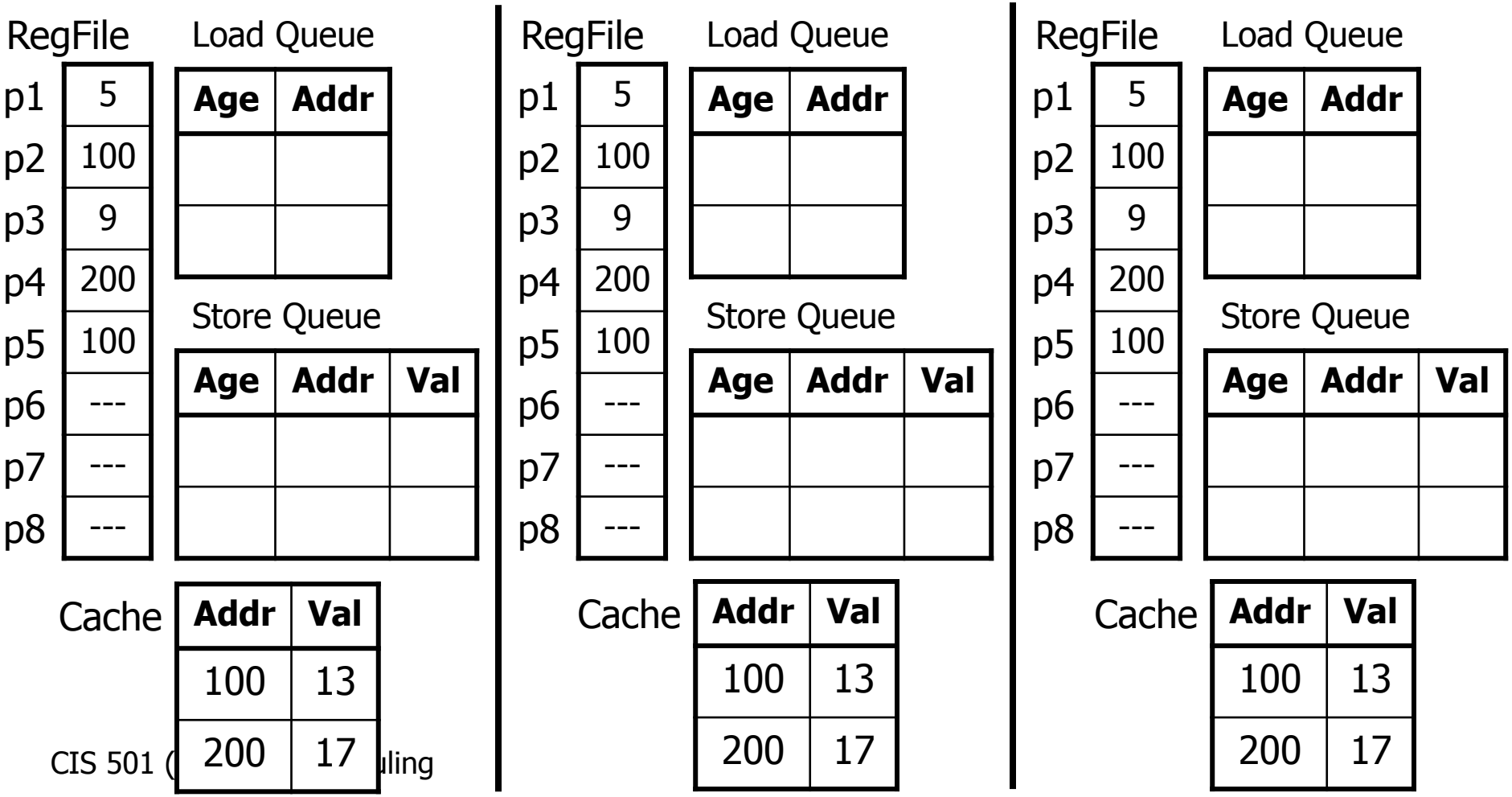
- More complex predictors used in practice
  - Predict which stores loads should wait for
  - "Store Sets" paper for next time

# LOAD/STORE QUEUE EXAMPLES

# Initial State

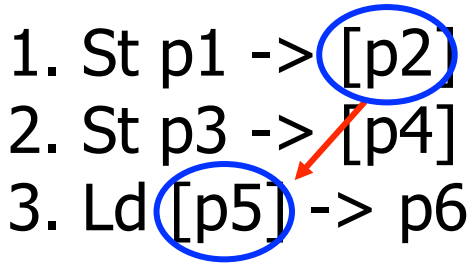
(Stores to different addresses)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6



# Good Interleaving

(Shows importance of address check)




## 1. St p1 -> [p2]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 200 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

## 2. St p3 -> [p4]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 200 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 200 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

## 3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 200 |
| p5 | 100 |
| p6 | **5** |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 200 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# Different Initial State

(All to same address)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |



RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# Good Interleaving #1

(Program Order)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6

### 1. St p1 -> [p2]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

### 2. St p3 -> [p4]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

### 3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | **9** |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# Good Interleaving #2

(Stores reordered)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6

## 2. St p3 -> [p4]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

## 1. St p1 -> [p2]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

## 3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | **9** |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# Bad Interleaving #1

(Load reads the cache)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6

### 3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | 13 |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

### 2. St p3 -> [p4]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | 13 |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# Bad Interleaving #2

(Load gets value from wrong store)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6

### 1. St p1 -> [p2]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

### 3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | **5** |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| | | |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

### 2. St p3 -> [p4]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | 5 |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# Bad/Good Interleaving

(Load gets value from correct store, but does it work?)

1. St p1 -> [p2]
2. St p3 -> [p4]
3. Ld [p5] -> p6

## 2. St p3 -> [p4]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | --- |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| | |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

## 3. Ld [p5] -> p6

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | **9** |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

Store Queue

| Age | Addr | Val |
|---|---|---|
| | | |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

## 1. St p1 -> [p2]

RegFile

| | |
|---|---|
| p1 | 5 |
| p2 | 100 |
| p3 | 9 |
| p4 | 100 |
| p5 | 100 |
| p6 | 9 |
| p7 | --- |
| p8 | --- |

Load Queue

| Age | Addr |
|---|---|
| 3 | 100 |
| | |

?

Store Queue

| Age | Addr | Val |
|---|---|---|
| 1 | 100 | 5 |
| 2 | 100 | 9 |

Cache

| Addr | Val |
|---|---|
| 100 | 13 |
| 200 | 17 |

# OUT-OF-ORDER: BENEFITS & CHALLENGES

# Out of Order: Benefits

- Allows speculative re-ordering
  - Loads / stores
  - Branch prediction to look past branches
- Done by hardware
  - Compiler may want different schedule for different hw configs
  - Hardware has only its own configuration to deal with
- Schedule can change due to cache misses
  - Different schedule optimal from on cache hit
- **Memory-level parallelism**
  - Executes "around" cache misses to find independent instructions
  - Finds and initiates independent misses, reducing memory latency
    - Especially good at hiding L2 hits (~12 cycles in Core i7)

# Challenges for Out-of-Order Cores

- Design complexity
  - More complicated than in-order? Certainly!
  - But, we have managed to overcome the design complexity
- Clock frequency
  - Can we build a "high ILP" machine at high clock frequency?
  - Yep, with some additional pipe stages, clever design
- Limits to (efficiently) scaling the window and ILP
  - Large physical register file
  - Fast register renaming/wakeup/select
  - Branch & memory depend. prediction (limits effective window size)
  - Plus all the issues of build "wide" in-order superscalar
- Power efficiency
  - Today, mobile phone chips are still in-order cores

# Out of Order: Window Size

- Scheduling scope = out-of-order window size
  - Larger = better
  - Constrained by physical registers (#preg)
    - Window limited by  #preg = ROB size + #logical registers
    - Big register file = hard/slow
  - Constrained by issue queue
    - Limits number of un-executed instructions
    - CAM = can't make big (power + area)
  - Constrained by load + store queues
    - Limit number of loads/stores
    - CAMs
- Active area of research: scaling window sizes
- Usefulness of large window: limited by branch prediction
  - 95% branch mis-prediction rate: 1 in 20 branches, or 1 in 100 insn.

# Reprise: Static vs Dynamic Scheduling

- If we can do this in software…
- …why build complex (slow-clock, high-power) hardware?
  - + Performance portability
    - Don't want to recompile for new machines
  - + More information available
    - Memory addresses, branch directions, cache misses
  - + More registers available
    - Compiler may not have enough to schedule well
  - + Speculative memory operation re-ordering
    - Compiler must be conservative, hardware can speculate
  - – But compiler has a larger scope
    - **Compiler does as much as it can (not much)**
    - **Hardware does the rest**

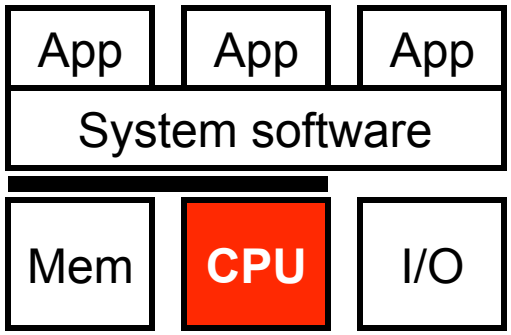
# Recap: Dynamic Scheduling

- Dynamic scheduling
  - Totally in the hardware
  - Also called "out-of-order execution" (OoO)
- Fetch many instructions into instruction window
  - Use branch prediction to speculate past (multiple) branches
  - Flush pipeline on branch misprediction
- Rename to avoid false dependencies
- Execute instructions as soon as possible
  - Register dependencies are known
  - Handling memory dependencies more tricky
- "Commit" instructions in order
  - Anything strange happens before commit, just flush the pipeline
- Current machines: 100+ instruction scheduling window

# Out of Order: Top 5 Things to Know

- Register renaming
  - How to perform is and how to recover it
- Commit
  - Precise state (ROB)
  - How/when registers are freed
- Issue/Select
  - Wakeup
  - Choose N oldest ready instructions
- Stores
  - Write at commit
  - Forward to loads via Load Queue
- Loads
  - Conservative/optimistic/predictive scheduling
  - Violation detection

# Summary: Scheduling




- Pipelining and superscalar review
- Code scheduling
  - To reduce pipeline stalls
  - To increase ILP (insn level parallelism)
- Two approaches
  - Static scheduling by the compiler
  - Dynamic scheduling by the hardware
- Up next: multicore