# ECE 4750 Computer Architecture Topic 3: Pipelining Structural & Data Hazards

Christopher Batten
School of Electrical and Computer Engineering
Cornell University

`http://www.csl.cornell.edu/courses/ece4750`

slide revision: 2012-09-05-17-41

## Iron Law of Processor Performance

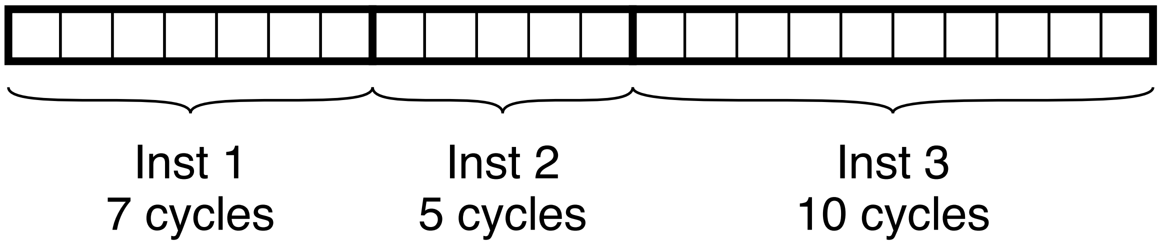
$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycles}}$$

| | **Microarchitecture** | **CPI** | **Cycle Time** |
|---|---|---|---|
| topic 2 $\rightarrow$ | Microcoded | $>1$ | short |
| topic 3 $\rightarrow$ | Single-Cycle Unpipelined | 1 | long |
| topic 3 $\rightarrow$ | Multi-Cycle Unpipelined | $>1$ | short |
| topics 4–6 $\rightarrow$ | Pipelined | $\approx 1$ | short |

## CPI Of Various Microarchitectures

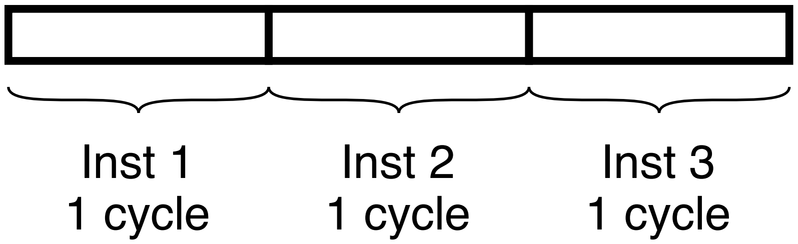
Microcoded

**CPI = 7.33**



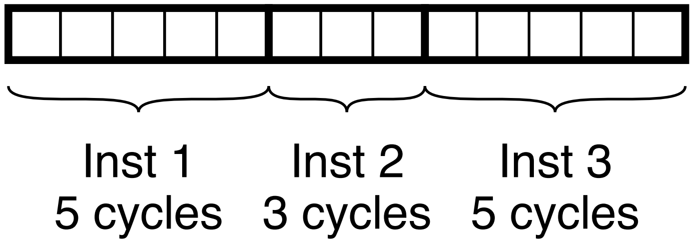
Single-Cycle Unpipelined

**CPI = 1**



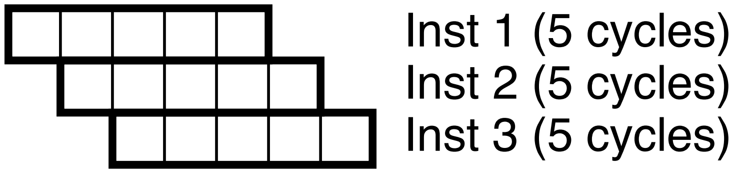
Multi-Cycle Unpipelined

**CPI = 4.33**



Pipelined

**CPI = 1**



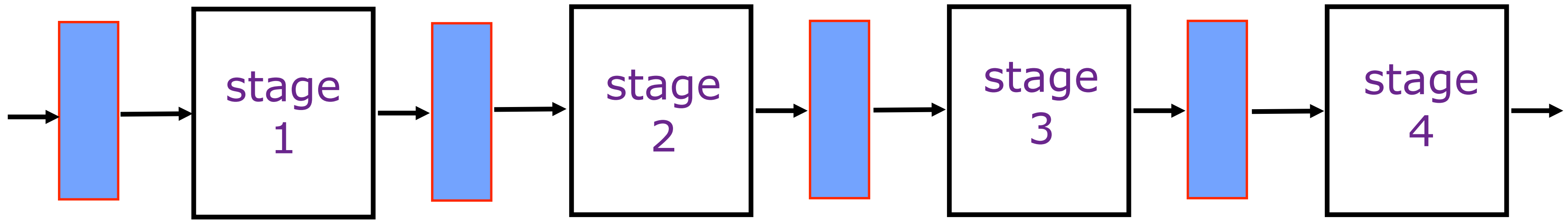
## Agenda

Pipelining Basics

Structural Hazards

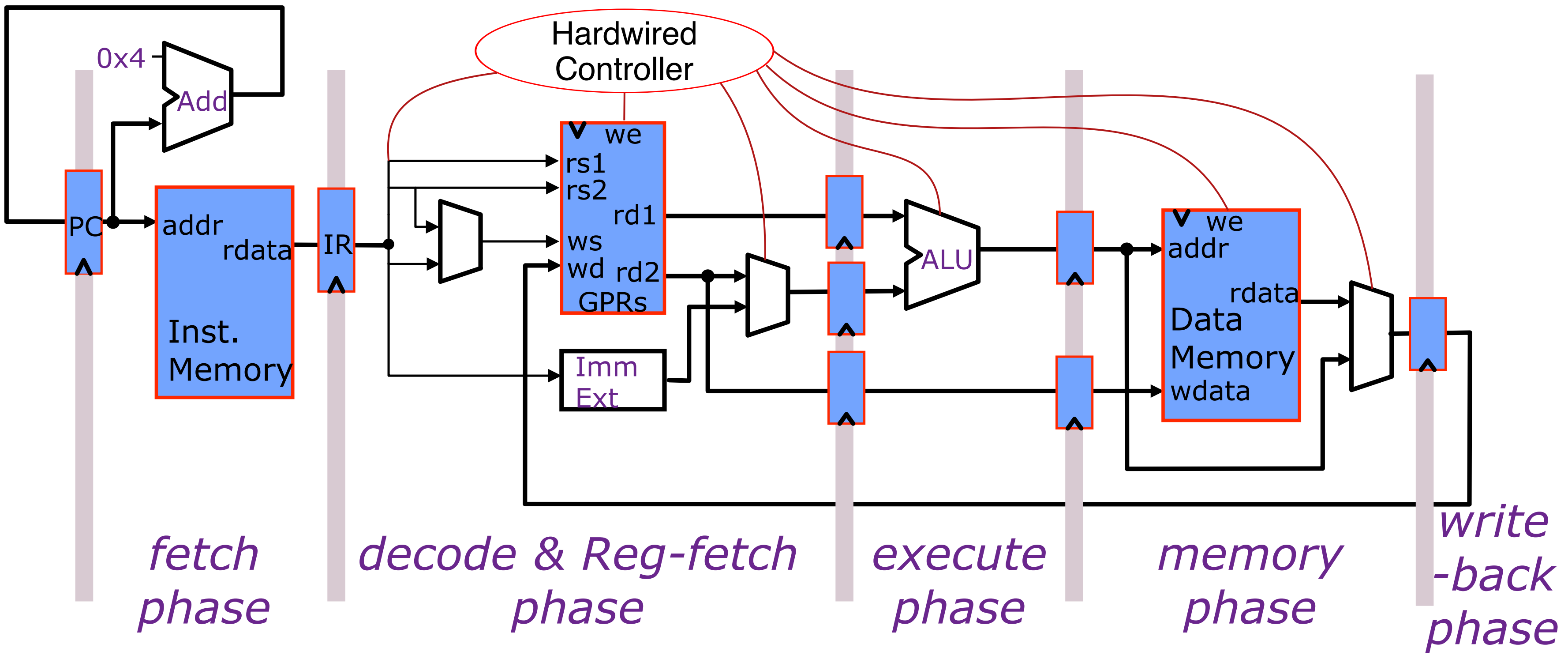
Data Hazards

## An Ideal Pipeline



- All objects go through the same stages
- No sharing of resources between any two stages
- Propagation delay through all pipeline stages is equal
- Scheduling of a transaction entering pipeline is not affected by transactions in other stages
- These conditions generally hold for industry assembly lines, but instructions depend on each other causing various hazards

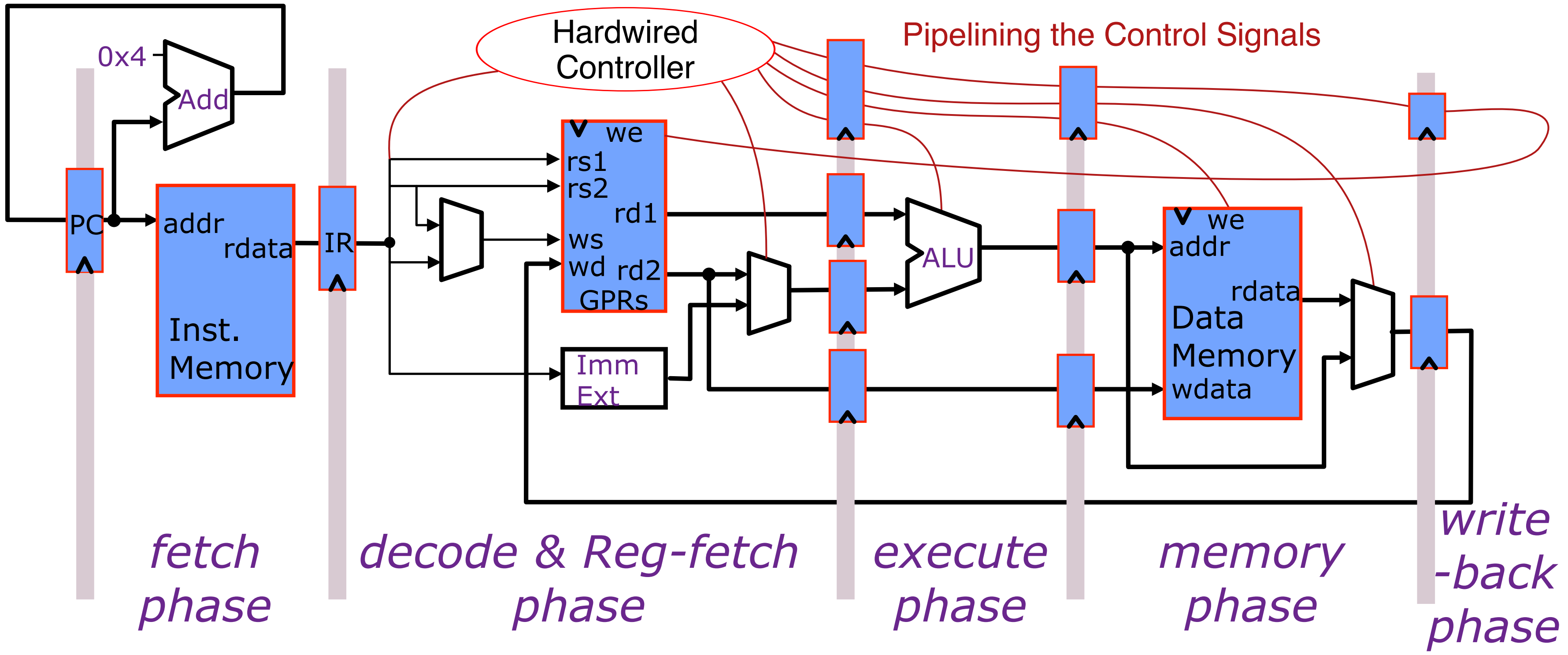
# Pipelined MIPS Processor: Start With Multi-Cycle Unpipelined Processor



Clock period is reduced by dividing the execution of an instruction into multiple cycles

$$t_c < max(t_{ifetch}, t_{rf}, t_{ALU}, t_{dmem}, t_{rfwr})$$

# Pipelined MIPS Processor: Add Pipeline Registers to Control Unit



As instruction goes down the pipeline, fewer control bits are needed

## Pipelining Technology Assumptions

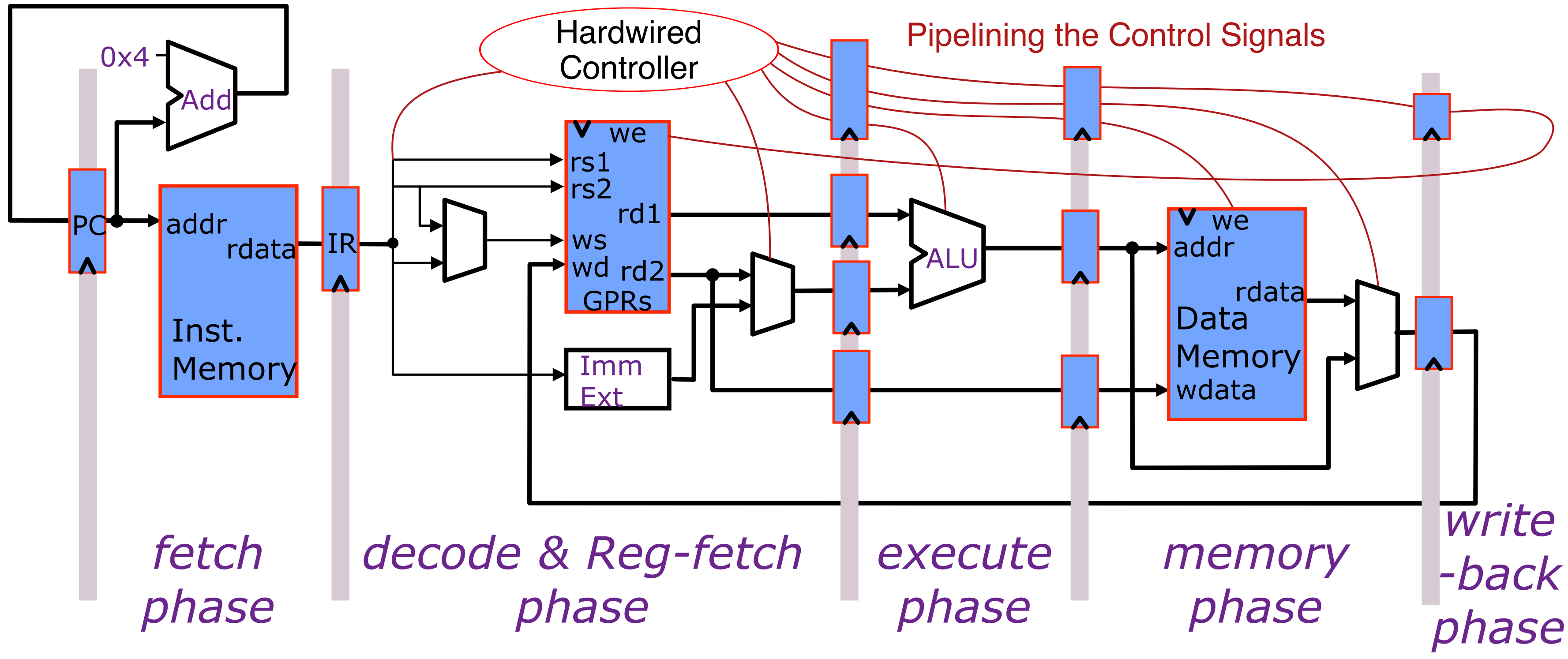
- Small amount of very fast memory (caches), backed by large, slower memory
- Fast ALU (at least for integers)
- Multiported register files (slower!)

Thus, the following timing assumption is reasonable

$$t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{RW}$$

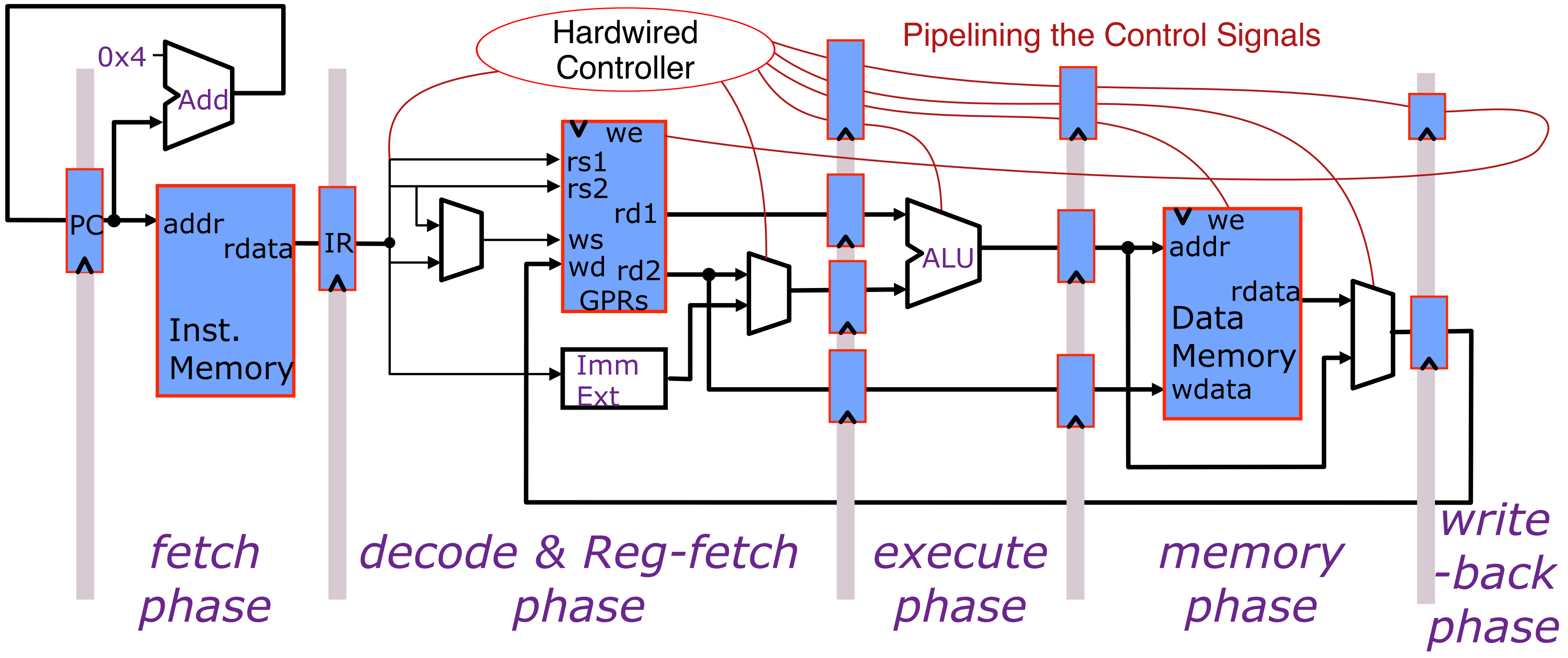
A 5-stage pipeline will be the focus of our detailed design, although real commercial designs usually have many more stages

## Pipeline Diagrams: Block Diagram



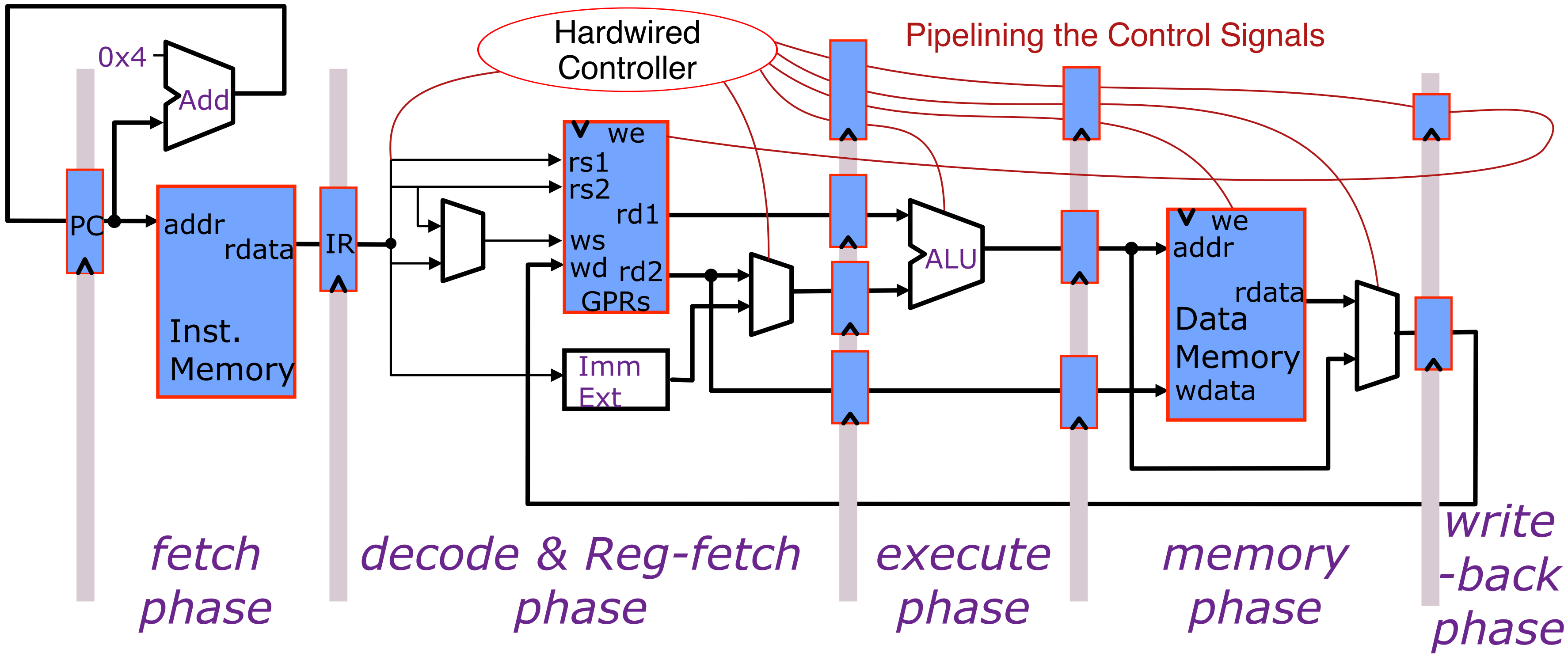
We need to be able to show multiple simultaneous transactions in both space and time

## Pipeline Diagrams: Transactions vs. Time



| *time* | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | . . . . |
|---|---|---|---|---|---|---|---|---|---|
| instruction1 | $IF_1$ | $ID_1$ | $EX_1$ | $MA_1$ | $WB_1$ | | | | |
| instruction2 | | $IF_2$ | $ID_2$ | $EX_2$ | $MA_2$ | $WB_2$ | | | |
| instruction3 | | | $IF_3$ | $ID_3$ | $EX_3$ | $MA_3$ | $WB_3$ | | |
| instruction4 | | | | $IF_4$ | $ID_4$ | $EX_4$ | $MA_4$ | $WB_4$ | |
| instruction5 | | | | | $IF_5$ | $ID_5$ | $EX_5$ | $MA_5$ | $WB_5$ |

## Pipeline Diagrams: Space vs. Time



| Resources | time | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | . . . . |
|---|---|---|---|---|---|---|---|---|---|---|
| | IF | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | | | | |
| | ID | | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | | | |
| | EX | | | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | | |
| | MA | | | | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | |
| | WB | | | | | $I_1$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ |

## Instructions Interact With Each Other in Pipeline

- **Structural Hazard –** An instruction in the pipeline needs a resource being used by another instruction in the pipeline
- **Data Hazard –** An instruction depends on a data value produced by an earlier instruction
- **Control Hazard –** Whether or not an instruction should be executed depends on a control decision made by an earlier instruction

## Agenda

Pipelining Basics

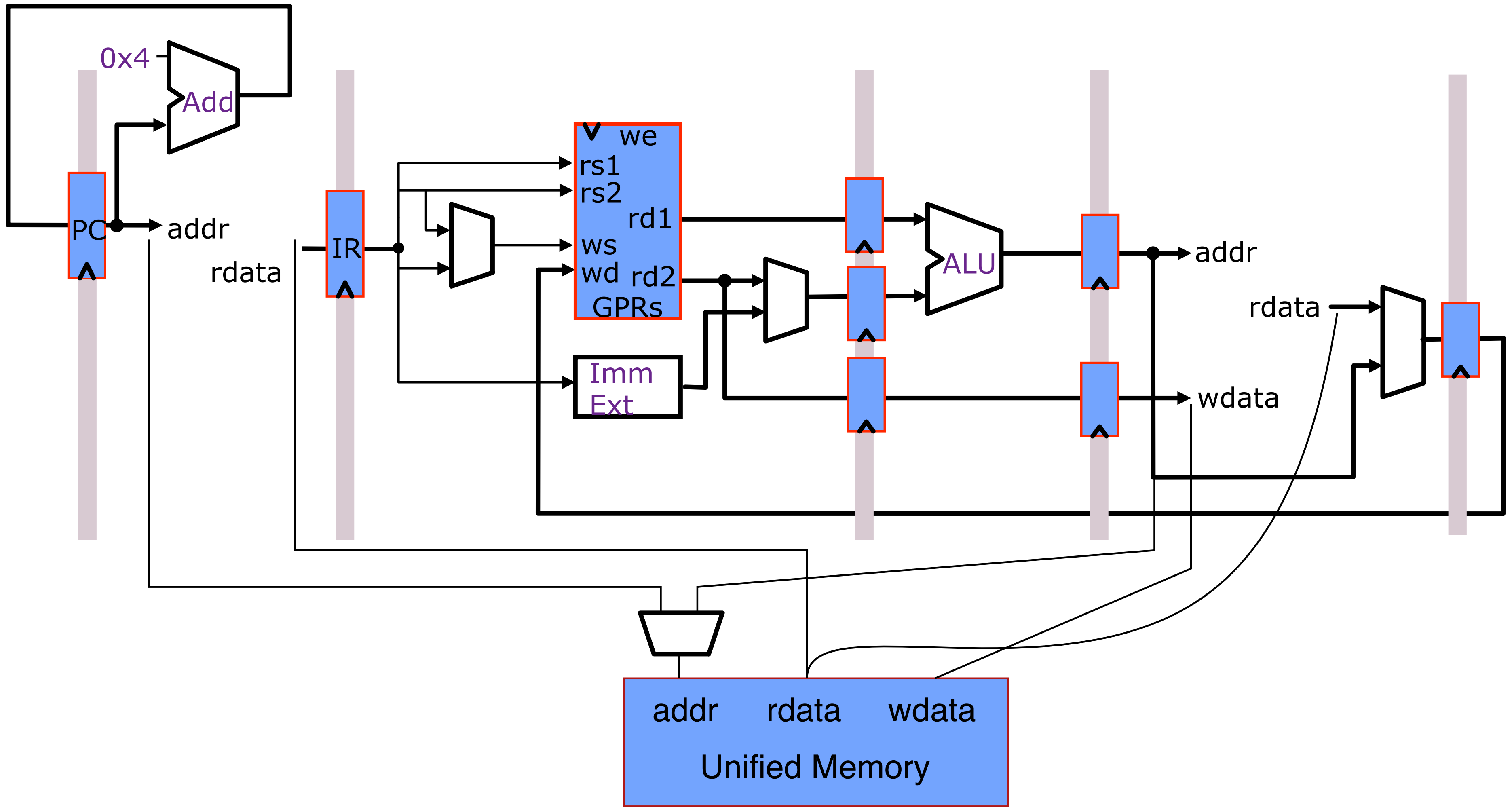
Structural Hazards

Data Hazards

## Overview Structural Hazards

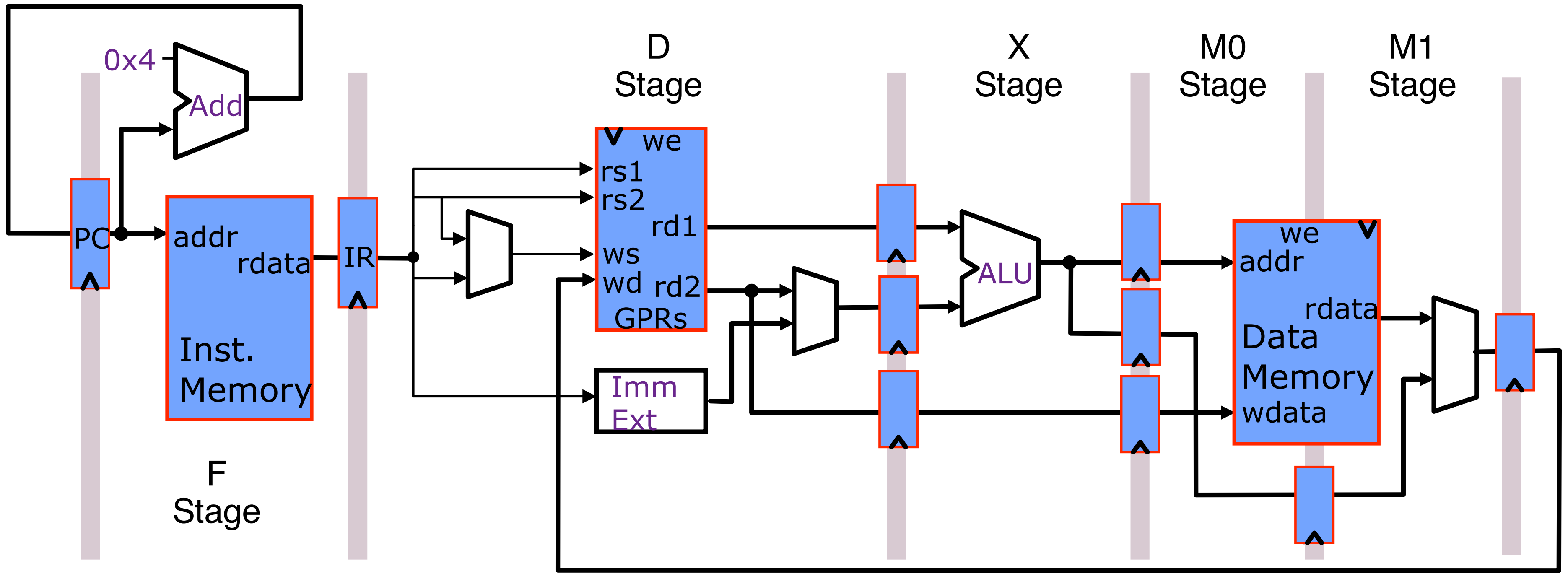
- Structural hazards occur when two instructions need the same hardware resource at the same time
- Approaches to resolving structural hazards
  - **Schedule –** Programmer explicitly avoids scheduling instructions that would create structural hazards
  - **Stall –** Hardware includes control logic that stalls until earlier instruction is no longer using contended resource
  - **Duplicate –** Add more hardware to design so that each instruction can access to independent resources at the same time
- Simple 5-stage MIPS pipeline has no structural hazards specifically because ISA was designed that way

## Example Structural Hazard: Unified Memory



Pipeline diagram on board

# Example Structural Hazard: 2-Cycle Data Memory



Pipeline diagram on board

## Agenda

Pipelining Basics

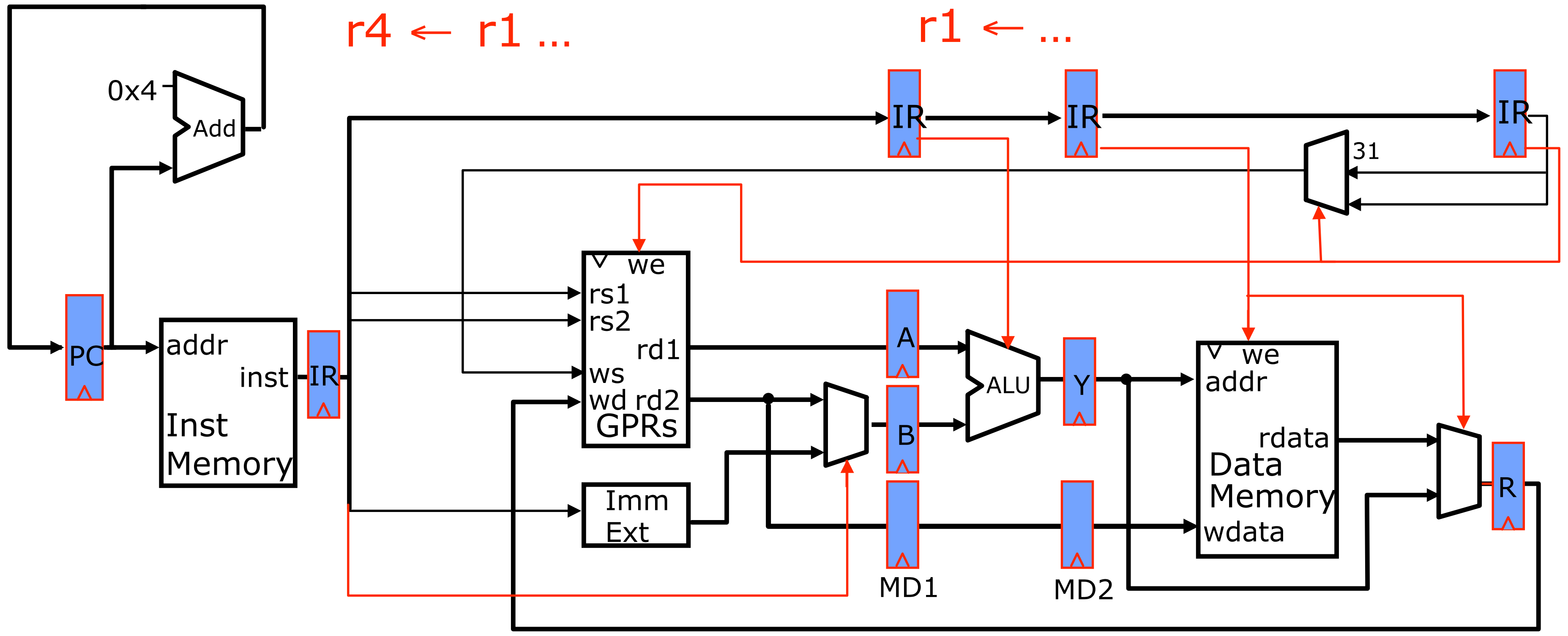
Structural Hazards

Data Hazards

## Overview of Data Hazards

- Data hazards occur when one instruction depends on a data value produced by an preceding instruction still in the pipeline
- Approaches to resolving data hazards
  - **Schedule –** Programmer explicitly avoids scheduling instructions that would create data hazards
  - **Stall –** Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
  - **Bypass –** Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
  - **Speculate –** Guess that there is not a problem, if incorrect kill speculative instruction and restart

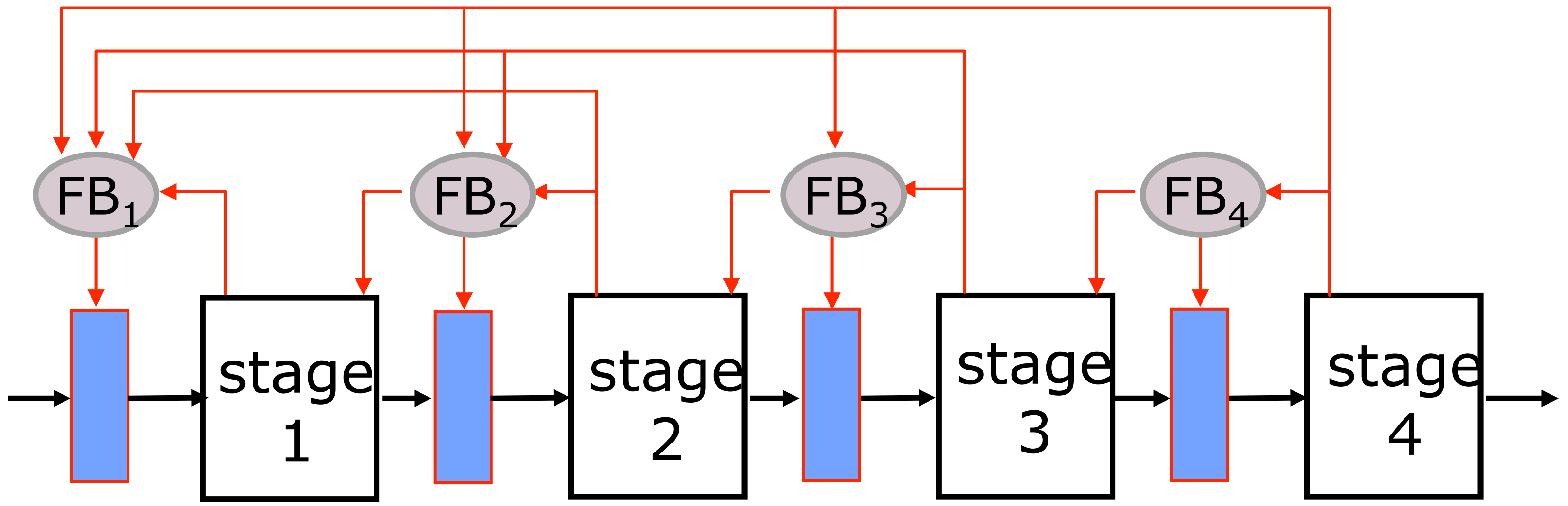
# Example Data Hazard



...
r1 ← r0 + 10
r4 ← r1 + 17
...

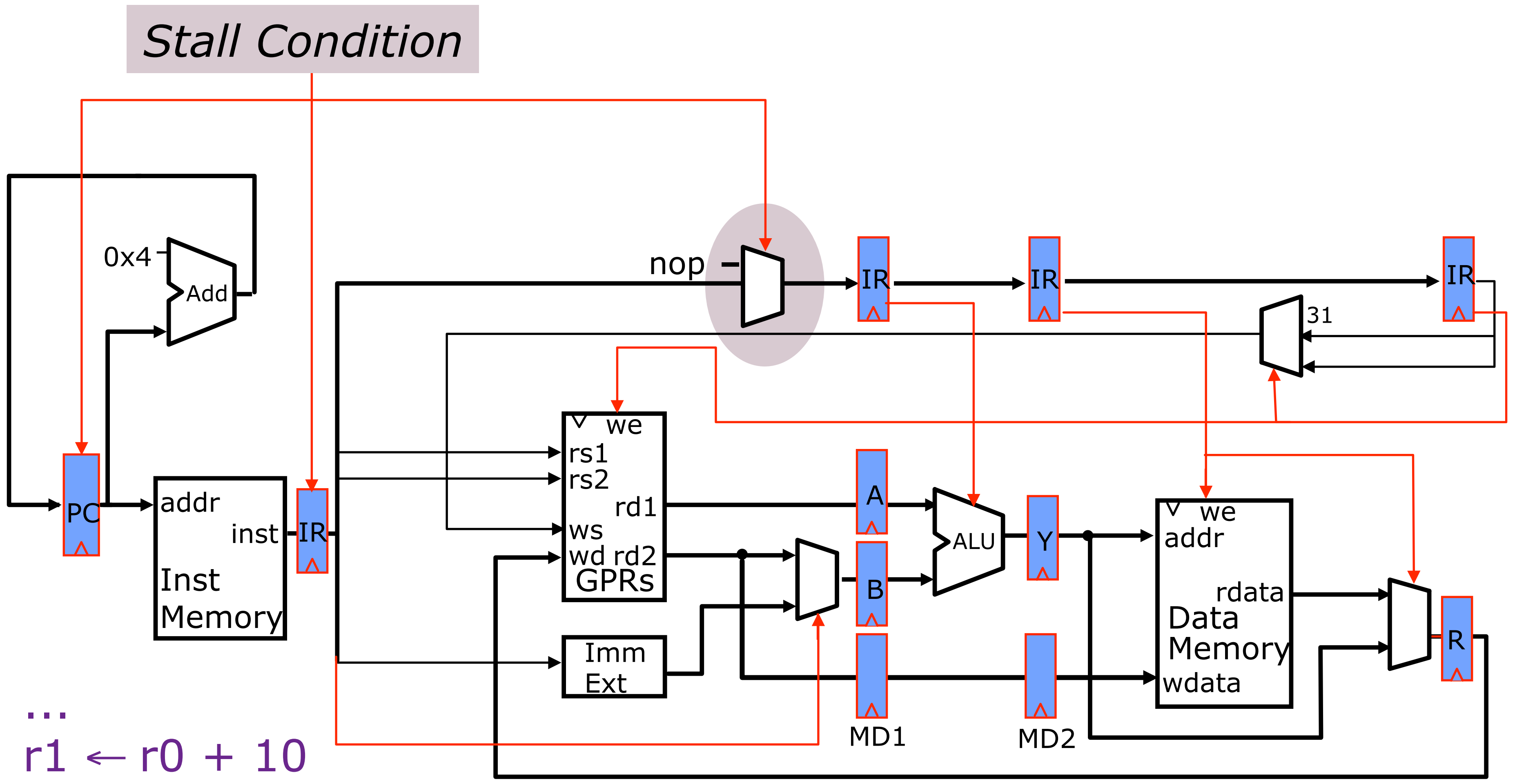
Pipeline diagram on board

# Feedback to Resolve Hazards



- Later stages provide dependence information to earlier stages which can stall (or kill) following instructions
- Controlling a pipeline in this manner works provided the instruction at stage i+1 can complete without any interaction from instructions in stages 1 to i (otherwise deadlock)

## Resolving Data Hazards with Stalls



...
$r1 \leftarrow r0 + 10$
$r4 \leftarrow r1 + 17$
...

Pipeline diagram on board

# Stalled Stages and Pipeline Bubbles

*time*

| | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | . . . . | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $(I_1)$ $r1 \leftarrow (r0) + 10$ | $IF_1$ | $ID_1$ | $EX_1$ | $MA_1$ | $WB_1$ | | | | | | | |
| $(I_2)$ $r4 \leftarrow (r1) + 17$ | | $IF_2$ | $ID_2$ | $ID_2$ | $ID_2$ | $ID_2$ | $EX_2$ | $MA_2$ | $WB_2$ | | | |
| $(I_3)$ | | | $IF_3$ | $IF_3$ | $IF_3$ | $IF_3$ | $ID_3$ | $EX_3$ | $MA_3$ | $WB_3$ | | |
| $(I_4)$ | | | | | | | $IF_4$ | $ID_4$ | $EX_4$ | $MA_4$ | $WB_4$ | |
| $(I_5)$ | | | | | | | | $IF_5$ | $ID_5$ | $EX_5$ | $MA_5$ | $WB_5$ |

*stalled stages*

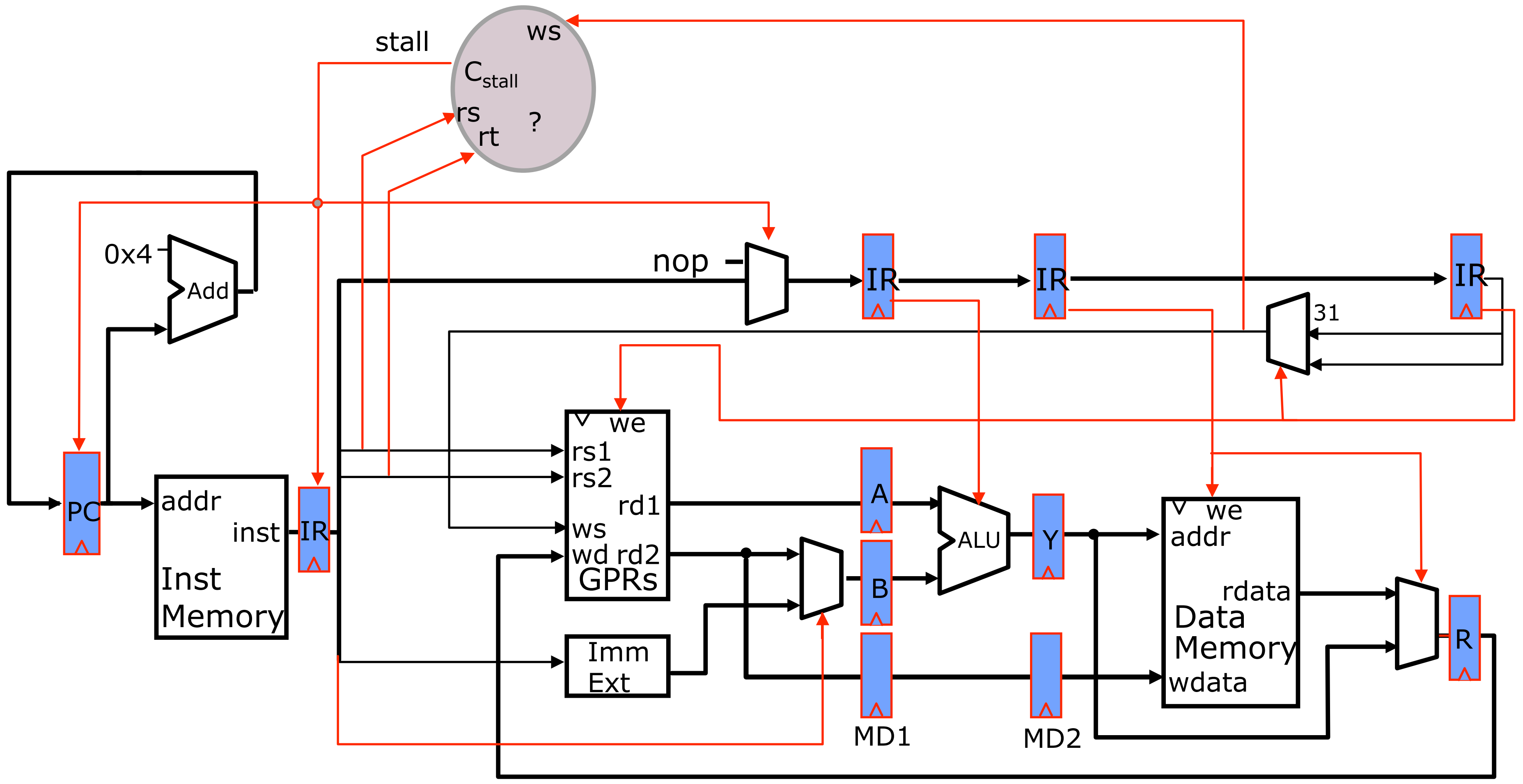
*Resource Usage*

*time*

| | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | . . . . | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IF | $I_1$ | $I_2$ | $I_3$ | $I_3$ | $I_3$ | $I_3$ | $I_4$ | $I_5$ | | | | |
| ID | | $I_1$ | $I_2$ | $I_2$ | $I_2$ | $I_2$ | $I_3$ | $I_4$ | $I_5$ | | | |
| EX | | | $I_1$ | nop | nop | nop | $I_2$ | $I_3$ | $I_4$ | $I_5$ | | |
| MA | | | | $I_1$ | nop | nop | nop | $I_2$ | $I_3$ | $I_4$ | $I_5$ | |
| WB | | | | | $I_1$ | nop | nop | nop | $I_2$ | $I_3$ | $I_4$ | $I_5$ |

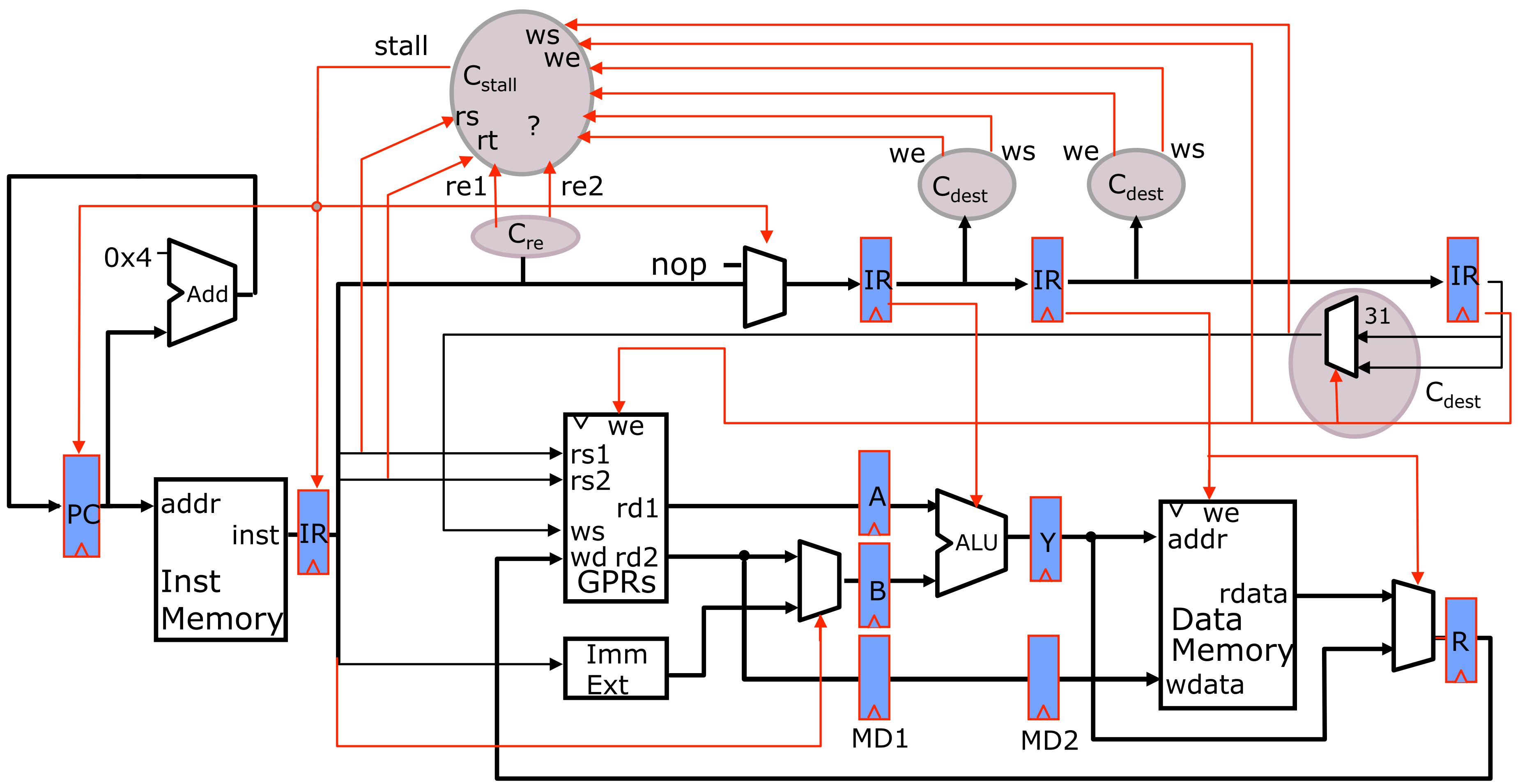
*nop* $\Rightarrow$ *pipeline bubble*

## Stall Control logic



Compare the source registers of the instruction in the decode stage with the destination register of the uncommitted instructions

## Stall Control logic (ignoring jumps/branches)



Always stall if rs field matches some rd?

Not every instruction writes or reads a register!

## Source and Destination Registers

| | | srcs | dest |
|---|---|---|---|
| ALU | R[rd] $\leftarrow$ R[rs] func R[rt] | rs, rt | rd |
| ALUI | R[rt] $\leftarrow$ R[rs] op immediate | rs | rt |
| LW | R[rt] $\leftarrow$ M[ R[rs] + offset ] | rs | rt |
| SW | M[ R[rs] + offset ] $\leftarrow$ R[rt] | rs, rt | |
| BEQZ | if ( R[rs] == 0 ) PC $\leftarrow$PC+4 + offset*4 | rs | |
| J | PC $\leftarrow$ jtarg(PC,imm) | | |
| JAL | R[31] $\leftarrow$ PC; PC $\leftarrow$ jtarg(PC,imm) | | 31 |
| JR | PC $\leftarrow$ R[rs] | rs | |
| JALR | R[31] $\leftarrow$ PC, PC $\leftarrow$R[rs] | rs | 31 |

## Deriving the Stall Signal

$C_{dest}$

ws = *Case* opcode

| | |
|---|---|
| ALU | ⇒ rd |
| ALUi, LW | ⇒ rt |
| JAL, JALR | ⇒ R31 |

we = *Case* opcode

| | |
|---|---|
| ALU, ALUi, LW | ⇒ (ws ≠ 0) |
| JAL, JALR | ⇒ on |
| ... | ⇒ off |

$C_{re}$

re1 = *Case* opcode

| | |
|---|---|
| ALU, ALUi, LW, SW, BZ, JR, JALR | ⇒ on |
| J, JAL | ⇒ off |

re2 = *Case* opcode

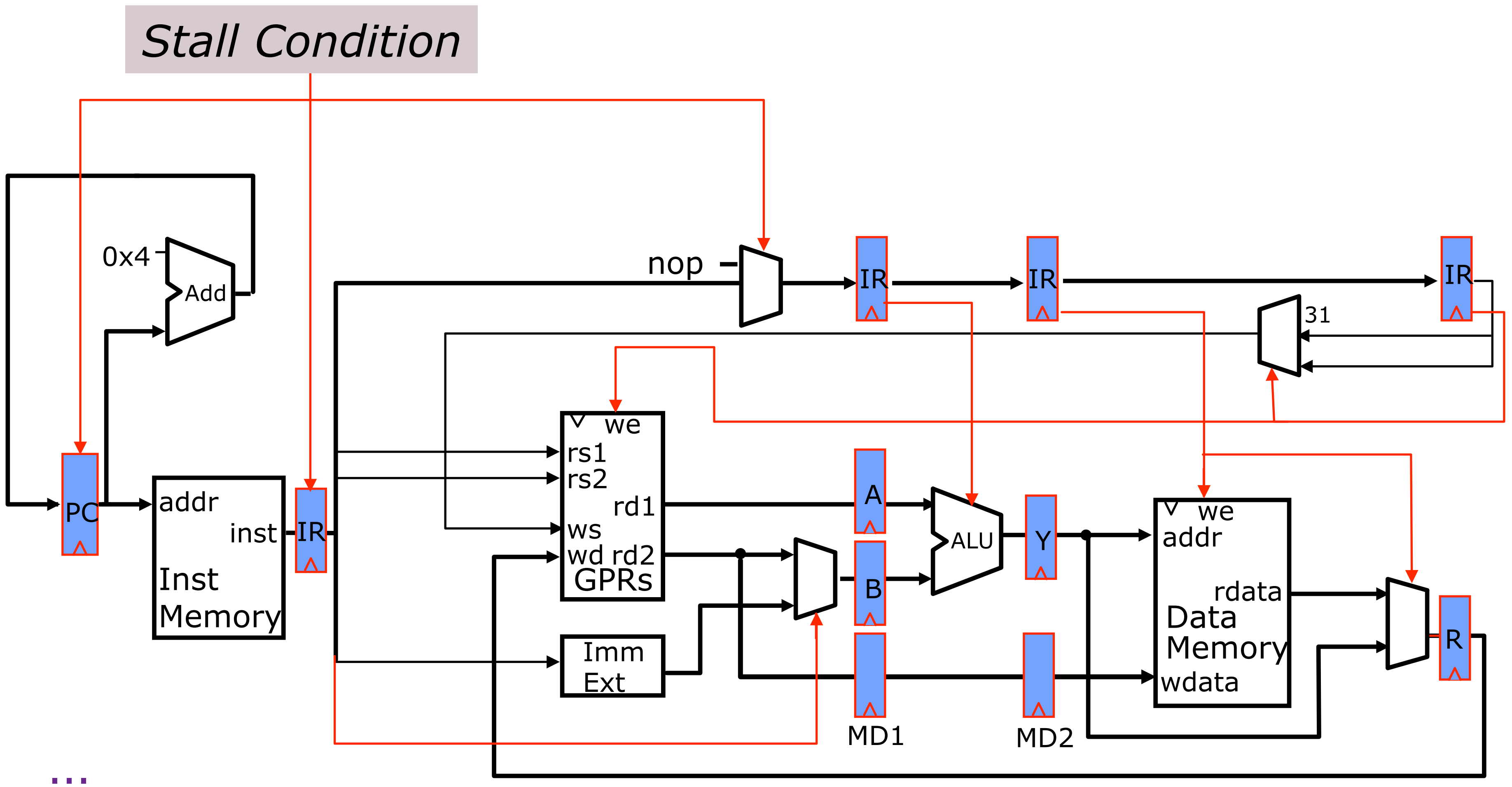
| | |
|---|---|
| ALU, SW | ⇒ on |
| ... | ⇒ off |

$C_{stall}$

$$\begin{aligned}stall = &((rs_D = ws_E).we_E + \\ &(rs_D = ws_M).we_M + \\ &(rs_D = ws_W).we_W) \,.\, re1_D + \\ &((rt_D = ws_E).we_E + \\ &(rt_D = ws_M).we_M + \\ &(rt_D = ws_W).we_W) \,.\, re2_D\end{aligned}$$

This is not the full story !

# Data Hazards Due to Loads and Stores



...
M[(r1)+7] ← (r2)
r4 ← M[(r3)+5]
...

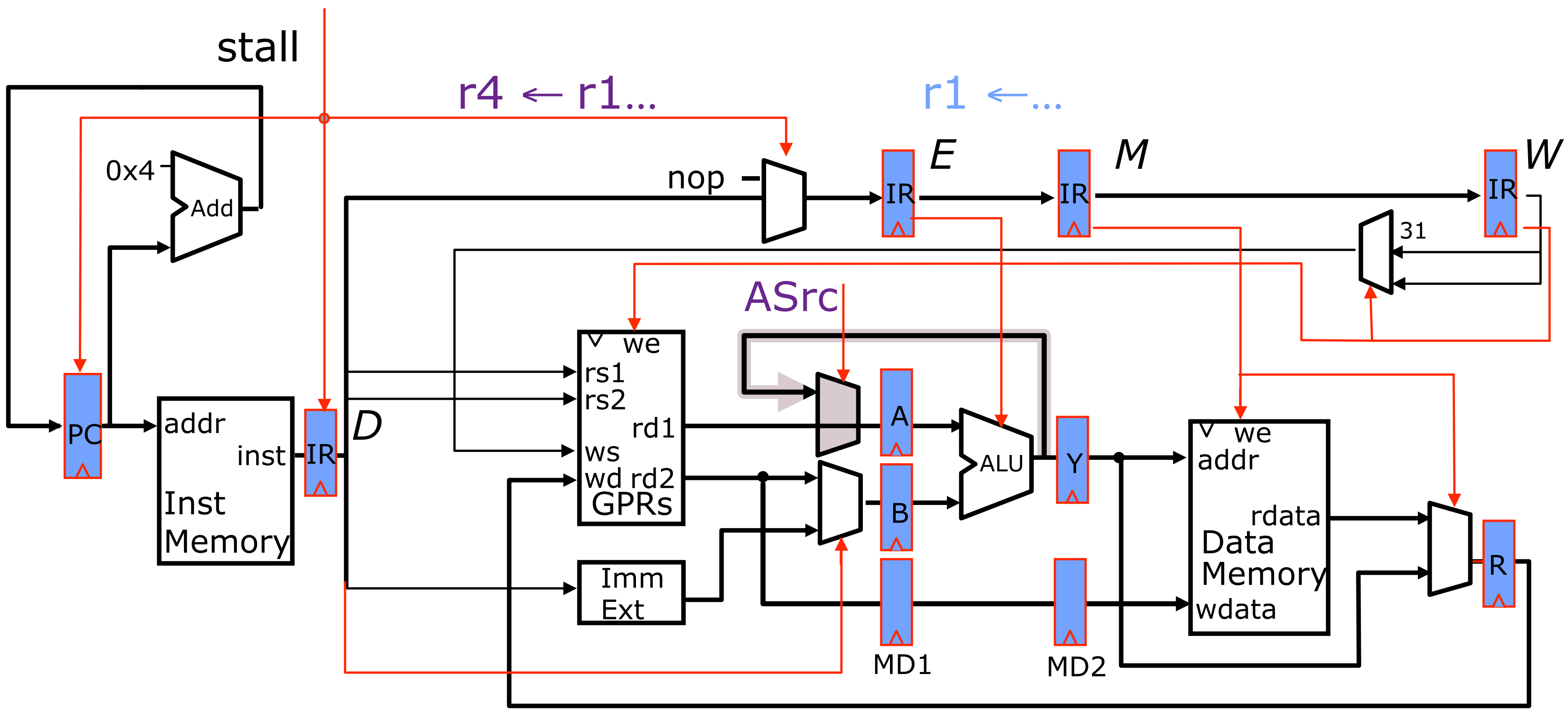
*Is there any possible data hazard in this instruction sequence?*

## Data Hazards Due to Loads and Stores

- Example instruction sequence
  - M[ R[r1] + 7 ] $\leftarrow$ R[r2]
  - R[r4] $\leftarrow$ M[ R[r3] + 5 ]
- What if R[r1]+7 == R[r3]+5 ?
  - Writing and reading from the same address
  - Hazard is avoided because our memory system completes writes in a single cycle
  - More realistic memory system will require more careful handling of data hazards due to loads and stores

Pipeline diagram on board

## Adding a Bypass to the Datapath



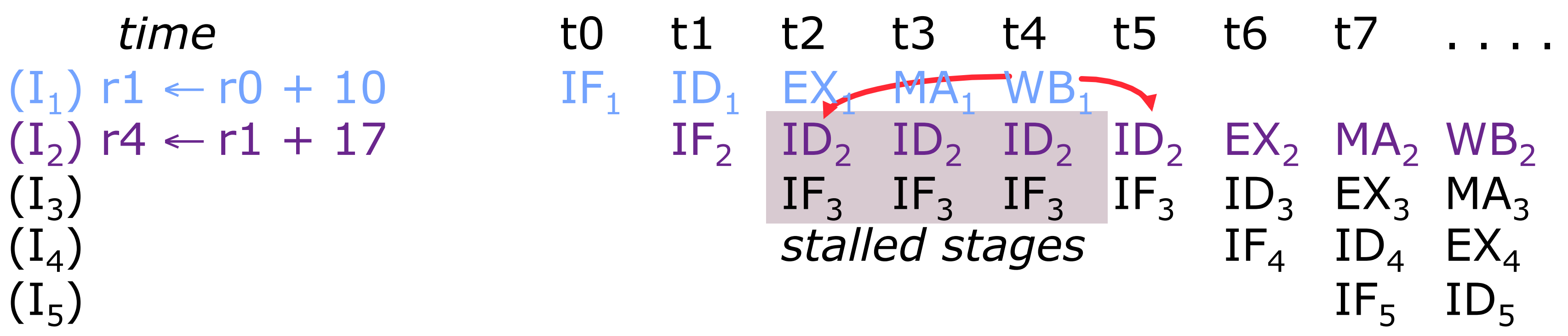
*When does this bypass help?*

...

| | | | |
|---|---|---|---|
| $(I_1)$ | r1 ← r0 + 10 | r1 ← M[r0 + 10] | JAL 500 |
| $(I_2)$ | r4 ← r1 + 17 | r4 ← r1 + 17 | r4 ← r31 + 17 |

Pipeline diagram on board

## Deriving the Bypass Signal

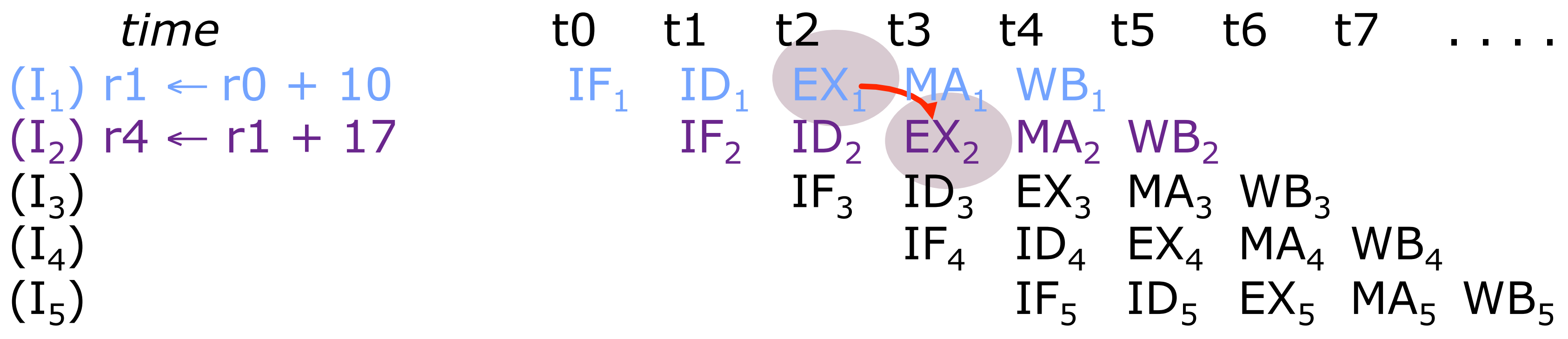


| *time* | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | . . . . |
|---|---|---|---|---|---|---|---|---|---|
| ($I_1$) r1 ← r0 + 10 | $IF_1$ | $ID_1$ | $EX_1$ | $MA_1$ | $WB_1$ | | | | |
| ($I_2$) r4 ← r1 + 17 | | $IF_2$ | $ID_2$ | $ID_2$ | $ID_2$ | $ID_2$ | $EX_2$ | $MA_2$ | $WB_2$ |
| ($I_3$) | | | $IF_3$ | $IF_3$ | $IF_3$ | $IF_3$ | $ID_3$ | $EX_3$ | $MA_3$ |
| ($I_4$) | | | *stalled stages* | | | | $IF_4$ | $ID_4$ | $EX_4$ |
| ($I_5$) | | | | | | | | $IF_5$ | $ID_5$ |

Each *stall or kill* introduces a bubble in the pipeline

⇒ *CPI > 1*

A new datapath, i.e., *a bypass*, can get the data from the output of the ALU to its input



| *time* | t0 | t1 | t2 | t3 | t4 | t5 | t6 | t7 | . . . . |
|---|---|---|---|---|---|---|---|---|---|
| ($I_1$) r1 ← r0 + 10 | $IF_1$ | $ID_1$ | $EX_1$ | $MA_1$ | $WB_1$ | | | | |
| ($I_2$) r4 ← r1 + 17 | | $IF_2$ | $ID_2$ | $EX_2$ | $MA_2$ | $WB_2$ | | | |
| ($I_3$) | | | $IF_3$ | $ID_3$ | $EX_3$ | $MA_3$ | $WB_3$ | | |
| ($I_4$) | | | | $IF_4$ | $ID_4$ | $EX_4$ | $MA_4$ | $WB_4$ | |
| ($I_5$) | | | | | $IF_5$ | $ID_5$ | $EX_5$ | $MA_5$ | $WB_5$ |

## Deriving the Bypass Signal

stall = ( ~~$((rs_D = ws_E).we_E$~~ $+ (rs_D = ws_M).we_M + (rs_D = ws_W).we_W).re1_D$
$+((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W).re2_D$ )

ws = *Case* opcode

| | |
|---|---|
| ALU | ⇒ rd |
| ALUi, LW | ⇒ rt |
| JAL, JALR | ⇒ R31 |

we = *Case* opcode

| | |
|---|---|
| ALU, ALUi, LW | ⇒(ws ≠ 0) |
| JAL, JALR | ⇒ on |
| ... | ⇒ off |

$ASrc = (rs_D = ws_E).we_E.re1_D$

Is this correct?

No, because only ALU and ALUI instruction can benefit from this bypass

Split $we_E$ into two components: we-bypass and we-stall

## Bypass and Stall Signals

Split $we_E$ into two components: we-bypass and we-stall

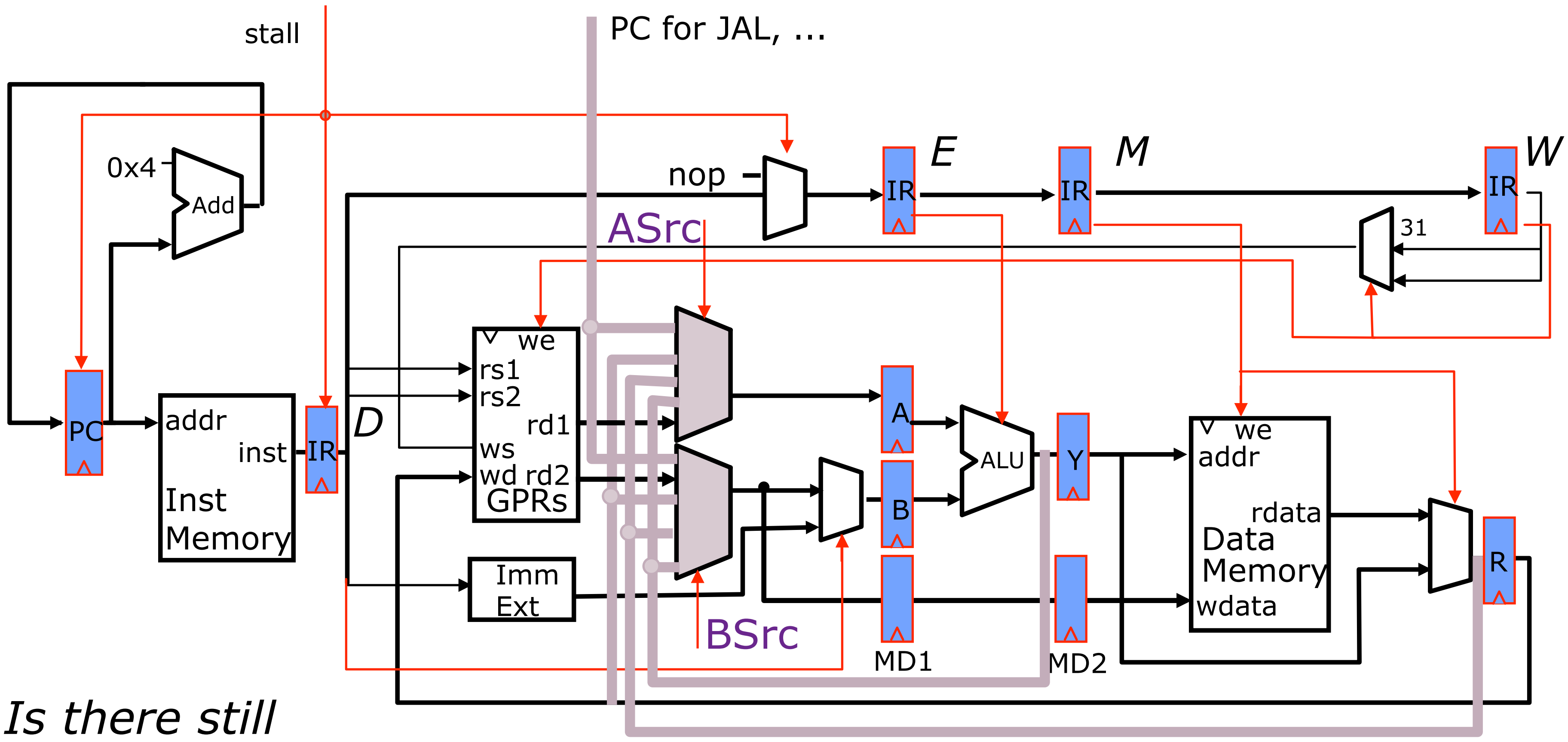
$\text{we-bypass}_E = \textit{Case}\ \text{opcode}_E$
ALU, ALUi $\Rightarrow (ws \neq 0)$
... $\Rightarrow$ off

$\text{we-stall}_E = \textit{Case}\ \text{opcode}_E$
LW $\Rightarrow (ws \neq 0)$
JAL, JALR $\Rightarrow$ on
... $\Rightarrow$ off

$$\text{ASrc} = (rs_D = ws_E).\text{we-bypass}_E\ .\ re1_D$$

$$\text{stall} = ((rs_D = ws_E).\text{we-stall}_E + (rs_D = ws_M).we_M + (rs_D = ws_W).we_W).\ re1_D + ((rt_D = ws_E).we_E + (rt_D = ws_M).we_M + (rt_D = ws_W).we_W).\ re2_D$$

# Fully Bypassed Datapath



*Is there still a need for the stall signal ?*

$$\boxed{\begin{aligned}\text{stall} = \;& (rs_D{=}ws_E).\,(opcode_E{=}LW_E).(ws_E{\neq}0\,).re1_D \\ &+ (rt_D{=}ws_E).\,(opcode_E{=}LW_E).(ws_E{\neq}0\,).re2_D\end{aligned}}$$

## Summary

- **Structural Hazard –** An instruction in the pipeline needs a resource being used by another instruction in the pipeline (this topic)
- **Data Hazard –** An instruction depends on a data value produced by an earlier instruction (this topic)
- **Control Hazard –** Whether or not an instruction should be executed depends on a control decision made by an earlier instruction (next topic)

## Acknowledgements

Some of these slides contain material developed and copyrighted by:

Arvind (MIT), Krste Asanović (MIT/UCB), Joel Emer (Intel/MIT)
James Hoe (CMU), John Kubiatowicz (UCB), David Patterson (UCB)

MIT material derived from course 6.823
UCB material derived from courses CS152 and CS252