CSE 490/590 Computer Architecture

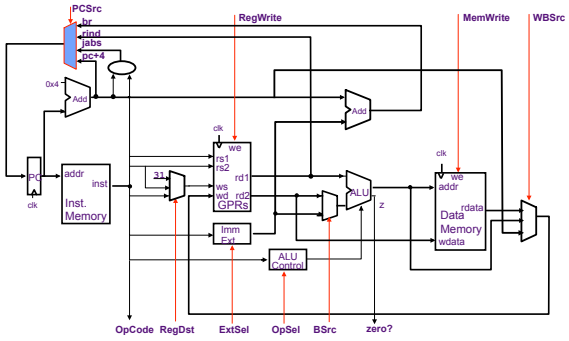
Pipelining II

Steve Ko
Computer Sciences and Engineering
University at Buffalo

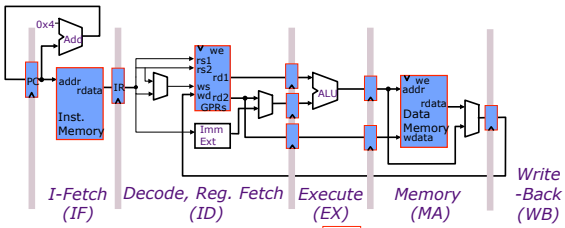
Last Time…

- MIPS instructions
  - Reg-Reg: ADD R1,R2,R3
  - Reg-Imm: LD R1,30(R2), BEQZ R4, name
  - Jump/Call: J name, JAL name
- MIPS single cycle implementation
  - Fetch → Decode & Reg fetch → execute → mem → WB
  - All in one cycle
- MIPS pipelining
  - 5 stages
  - Iron law: performance benefit analysis
    - » Pipelining reduces time/cycle
    - » Also try to maintain CPI == 1

Harvard-Style Datapath for MIPS



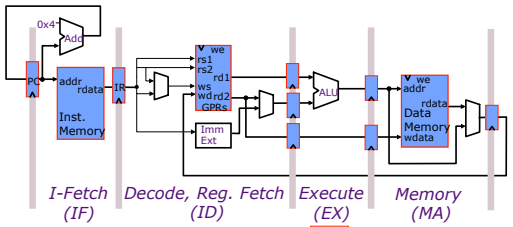
5-Stage Pipelined Execution



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

5-Stage Pipelined Execution

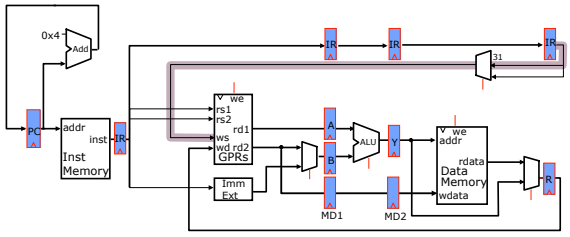
Resource Usage Diagram



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

Pipelined Execution:

ALU Instructions

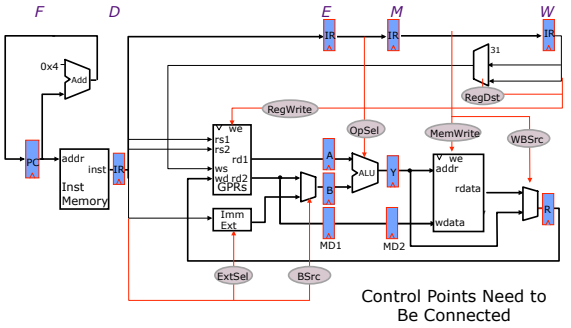


Not quite correct!

We need an Instruction Reg (IR) for each stage

Pipelined MIPS Datapath

without jumps



Control Points Need to Be Connected

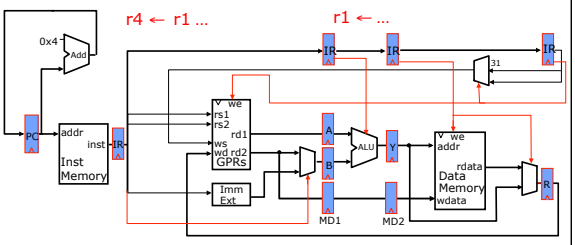
Instructions interact with each other in pipeline

- An instruction in the pipeline may need a resource being used by another instruction in the pipeline → *structural hazard*
- An instruction may depend on something produced by an earlier instruction
  - Dependence may be for a data value
    → data hazard
  - Dependence may be for the next instruction's address
    → control hazard (branches, exceptions)

Resolving Structural Hazards

- Structural hazards occurs when two instructions need same hardware resource at same time
  - Can resolve in hardware by stalling newer instruction till older instruction finished with resource
- A structural hazard can always be avoided by adding more hardware to design
  - E.g., if two instructions both need a port to memory at same time, could avoid hazard by adding second port to memory
- Our 5-stage pipe has no structural hazards by design
  - Thanks to MIPS ISA, which was designed for pipelining

Data Hazards




```
...
r1 ← r0 + 10
r4 ← r1 + 17
...
```

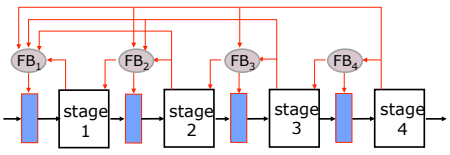
r1 is stale. Oops!

Resolving Data Hazards (1)

Strategy 1:

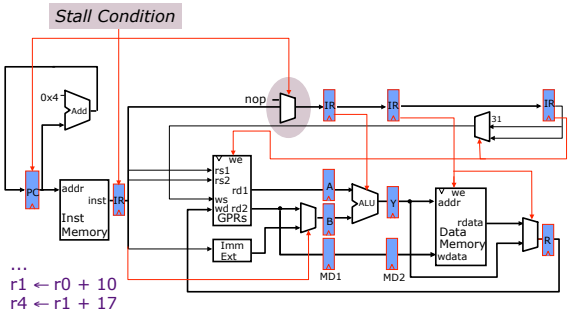
Wait for the result to be available by freezing earlier pipeline stages → *interlocks*

Feedback to Resolve Hazards



- Later stages provide dependence information to earlier stages which can *stall (or kill) instructions*

Interlocks to resolve Data Hazards



...
r1 ← r0 + 10
r4 ← r1 + 17
...

Stalled Stages and Pipeline Bubbles

	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 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

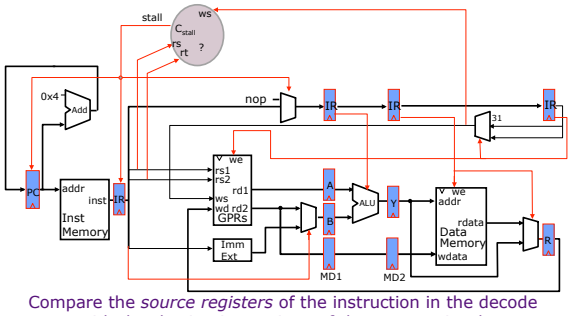
Resource Usage	time t0	t1	t2	t3	t4	t5	t6	t7	. . . .
IF	I_1	I_2	I_3	I_3	I_3	I_3	I_4	I_5	
ID		I_1	I_2	I_2	I_2	I_2	I_3	I_4	I_5
EX			I_1	nop	nop	nop	I_2	I_3	I_4 I_5
MA				I_1	nop	nop	nop	I_2	I_3 I_4 I_5
WB					I_1	nop	nop	nop	I_2 I_3 I_4 I_5

nop ⇒ *pipeline bubble*

CSE 490/590 Administrivia

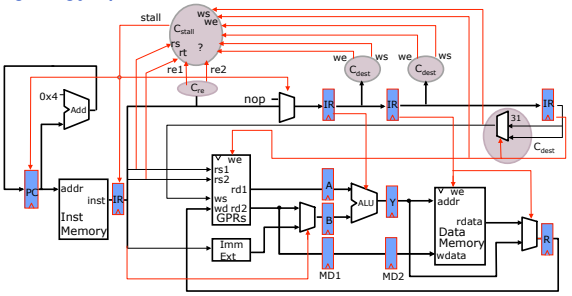
- Please purchase a BASYS2 board (100K) as soon as possible.
  - Projects should be done individually.
- Quiz 1
  - Fri, 2/4
  - Closed book, in-class
- Lecture notes
  - (Hopefully) will be up by a day before.
  - But will probably do some editing before and after each class.
  - Also available in pptx now

Interlock Control Logic



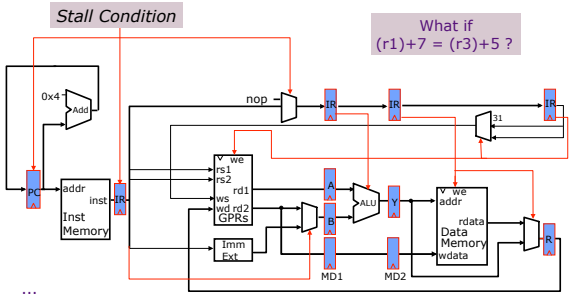
Compare the *source registers* of the instruction in the decode stage with the *destination register* of the *uncommitted* instructions.

Interlock Control Logic
ignoring jumps & branches



Should we always stall if the rs field matches some rd?
not every instruction writes a register ⇒ we
not every instruction reads a register ⇒ re

Hazards due to Loads & Stores



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

Is there any possible data hazard in this instruction sequence?

Load & Store Hazards

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

(r1)+7 = (r3)+5 ⇒ *data hazard*

However, the hazard is avoided because *our memory system completes writes in one cycle !*

Load/Store hazards are sometimes resolved in the pipeline and sometimes in the memory system itself.

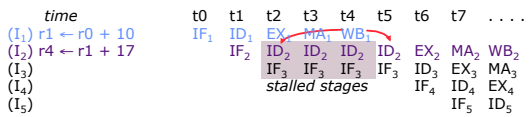
More on this later in the course.

Resolving Data Hazards (2)

Strategy 2:

Route data as soon as possible after it is calculated to the earlier pipeline stage → *bypass*

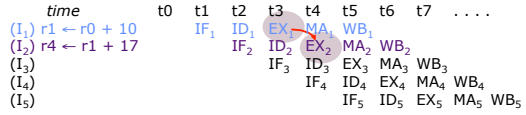
Bypassing



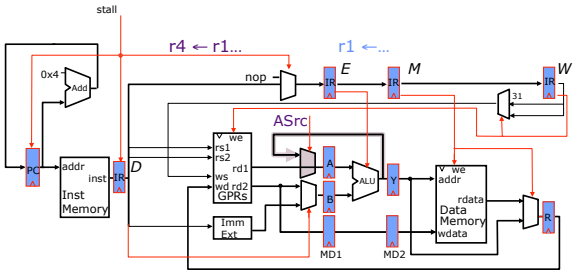
Each *stall or kill* introduces a bubble in the pipeline

⇒ *CPI > 1*

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



Adding a Bypass



When does this bypass help?

```
...
(I1)  r1 ← r0 + 10        r1 ← M[r0 + 10]      JAL 500
(I2)  r4 ← r1 + 17        r4 ← r1 + 17         r4 ← r31 + 17
                  yes                  no                   no
```


Acknowledgements

- These slides heavily contain material developed and copyright by
  - Krste Asanovic (MIT/UCB)
  - David Patterson (UCB)
- And also by:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
- MIT material derived from course 6.823
- UCB material derived from course CS252