

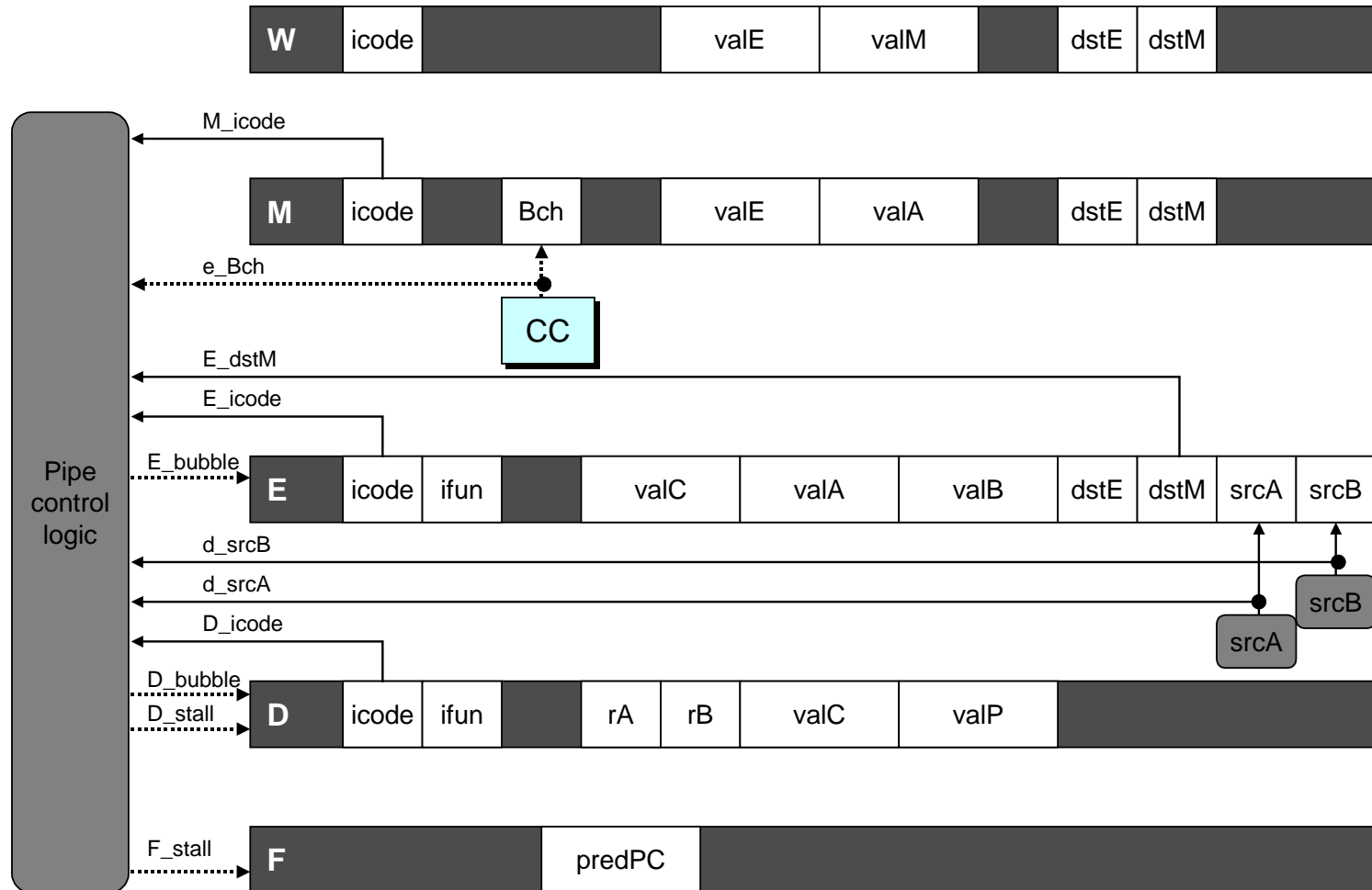
Systems I

Pipelining IV

Topics

- Implementing pipeline control
- Pipelining and performance analysis

Implementing Pipeline Control



- **Combinational logic generates pipeline control signals**
- **Action occurs at start of following cycle**

Initial Version of Pipeline Control

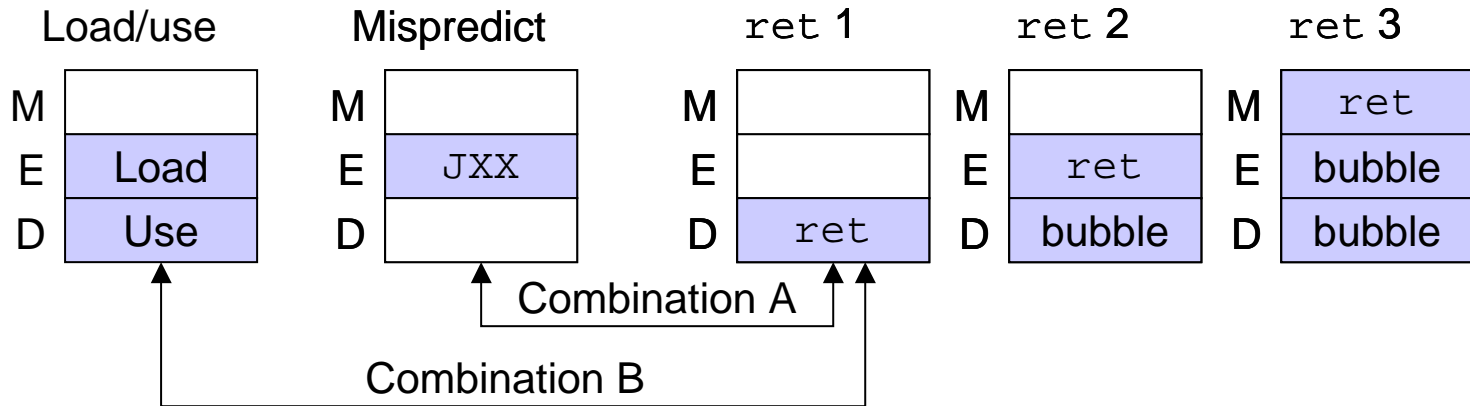
```
bool F_stall =
    # Conditions for a load/use hazard
    E_icode in { IMRMOVL, IPOPL } && E_dstM in { d_srcA, d_srcB } ||
    # Stalling at fetch while ret passes through pipeline
    IRET in { D_icode, E_icode, M_icode };

bool D_stall =
    # Conditions for a load/use hazard
    E_icode in { IMRMOVL, IPOPL } && E_dstM in { d_srcA, d_srcB };

bool D_bubble =
    # Mispredicted branch
    (E_icode == IJXX && !e_Bch) ||
    # Bubble for ret
    IRET in { D_icode, E_icode, M_icode };

bool E_bubble =
    # Mispredicted branch
    (E_icode == IJXX && !e_Bch) ||
    # Load/use hazard
    E_icode in { IMRMOVL, IPOPL } && E_dstM in { d_srcA, d_srcB};
```

Control Combinations



- Special cases that can arise on same clock cycle

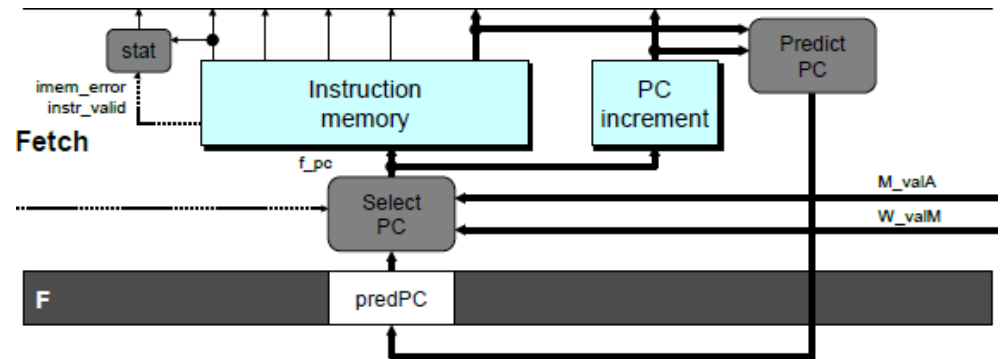
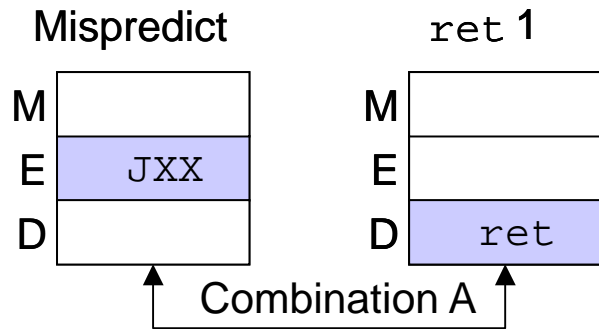
Combination A

- Not-taken branch
- `ret` instruction at branch target

Combination B

- Instruction that reads from memory to `%esp`
- Followed by `ret` instruction

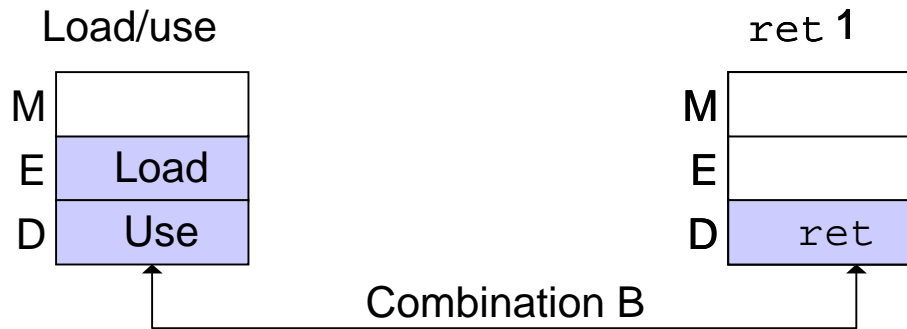
Control Combination A



Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal
Mispredicted Branch	normal	bubble	bubble	normal	normal
<i>Combination</i>	<i>stall</i>	<i>bubble</i>	<i>bubble</i>	<i>normal</i>	<i>normal</i>

- Should handle as mispredicted branch
- Stalls F pipeline register
- But PC selection logic will be using M_valM anyhow

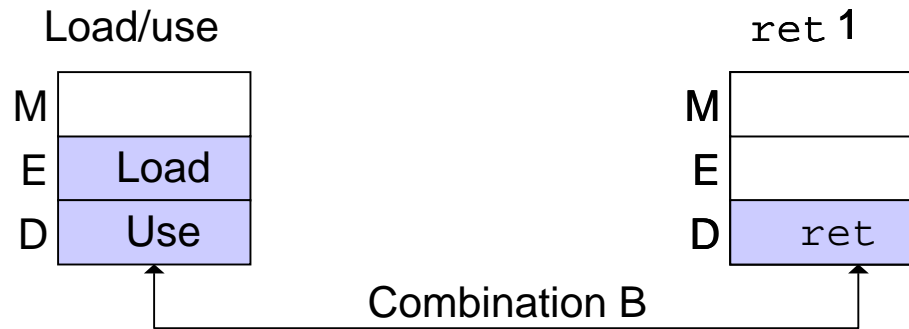
Control Combination B



Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal
Load/Use Hazard	stall	stall	bubble	normal	normal
<i>Combination</i>	<i>stall</i>	<i>bubble + stall</i>	<i>bubble</i>	<i>normal</i>	<i>normal</i>

- Would attempt to bubble *and* stall pipeline register D
- Signaled by processor as pipeline error

Handling Control Combination B



Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal
Load/Use Hazard	stall	stall	bubble	normal	normal
<i>Combination</i>	<i>stall</i>	<i>stall</i>	<i>bubble</i>	<i>normal</i>	<i>normal</i>

- Load/use hazard should get priority
- `ret` instruction should be held in decode stage for additional cycle

Corrected Pipeline Control Logic

```
bool D_bubble =  
    # Mispredicted branch  
    (E_icode == IJXX && !e_Bch) ||  
    # Stalling at fetch while ret passes through pipeline  
    IRET in { D_icode, E_icode, M_icode }  
    # but not condition for a load/use hazard  
    && !(E_icode in { IMRMOVL, IPOPL }  
        && E_dstM in { d_srcA, d_srcB }));
```

Condition	F	D	E	M	W
Processing ret	stall	bubble	normal	normal	normal
Load/Use Hazard	stall	stall	bubble	normal	normal
<i>Combination</i>	<i>stall</i>	<i>stall</i>	<i>bubble</i>	<i>normal</i>	<i>normal</i>

- Load/use hazard should get priority
- `ret` instruction should be held in decode stage for additional cycle

Pipeline Summary

Data Hazards

- Most handled by forwarding
 - No performance penalty
- Load/use hazard requires one cycle stall

Control Hazards

- Cancel instructions when detect mispredicted branch
 - Two clock cycles wasted
- Stall fetch stage while `ret` passes through pipeline
 - Three clock cycles wasted

Control Combinations

- Must analyze carefully
- First version had subtle bug
 - Only arises with unusual instruction combination

Performance Analysis with Pipelining

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Seconds}}{\text{Cycle}}$$

Ideal pipelined machine: CPI = 1

- One instruction completed per cycle
- But much faster cycle time than unpipelined machine

However - hazards are working against the ideal

- Hazards resolved using forwarding are fine
- Stalling degrades performance and instruction completion rate is interrupted

CPI is measure of “architectural efficiency” of design

CPI for PIPE

CPI \approx 1.0

- Fetch instruction each clock cycle
- Effectively process new instruction almost every cycle
 - Although each individual instruction has latency of 5 cycles

CPI $>$ 1.0

- Sometimes must stall or cancel branches

Computing CPI

- C clock cycles
- I instructions executed to completion
- B bubbles injected ($C = I + B$)

$$\text{CPI} = C/I = (I+B)/I = 1.0 + B/I$$

- Factor B/I represents average penalty due to bubbles

Computing CPI

CPI

- Function of useful instruction and bubbles

$$CPI = \frac{C_i + C_b}{C_i} = 1.0 + \frac{C_b}{C_i}$$

- C_b/C_i represents the pipeline penalty due to stalls

Can reformulate to account for

- load penalties (lp)
- branch misprediction penalties (mp)
- return penalties (rp)

$$CPI = 1.0 + lp + mp + rp$$

Computing CPI - II

So how do we determine the penalties?

- Depends on how often each situation occurs on average
- How often does a load occur and how often does that load cause a stall?
- How often does a branch occur and how often is it mispredicted
- How often does a return occur?

We can measure these

- simulator
- hardware performance counters

We can estimate through historical averages

- Then use to make early design tradeoffs for architecture

Computing CPI - III

Cause	Name	Instruction Frequency	Condition Frequency	Stalls	Product
Load/Use	lp	0.30	0.3	1	0.09
Mispredict	mp	0.20	0.4	2	0.16
Return	rp	0.02	1.0	3	0.06
Total penalty					0.31

$\text{CPI} = 1 + 0.31 = 1.31 \Rightarrow 31\% \text{ worse than ideal}$

This gets worse when:

- Account for non-ideal memory access latency
- Deeper pipelines (where stalls per hazard increase)

CPI for PIPE (Cont.)

$$B/I = LP + MP + RP$$

- | | Typical Values |
|---|----------------|
| ■ LP: Penalty due to load/use hazard stalling | |
| ● Fraction of instructions that are loads | 0.25 |
| ● Fraction of load instructions requiring stall | 0.20 |
| ● Number of bubbles injected each time | 1 |
| ⇒ $LP = 0.25 * 0.20 * 1 = 0.05$ | |
| ■ MP: Penalty due to mispredicted branches | |
| ● Fraction of instructions that are cond. jumps | 0.20 |
| ● Fraction of cond. jumps mispredicted | 0.40 |
| ● Number of bubbles injected each time | 2 |
| ⇒ $MP = 0.20 * 0.40 * 2 = 0.16$ | |
| ■ RP: Penalty due to <code>ret</code> instructions | |
| ● Fraction of instructions that are returns | 0.02 |
| ● Number of bubbles injected each time | 3 |
| ⇒ $RP = 0.02 * 3 = 0.06$ | |
| ■ Net effect of penalties $0.05 + 0.16 + 0.06 = 0.27$ | |
| ⇒ $CPI = 1.27$ (Not bad!) | |

Summary

Today

- Pipeline control logic
- Effect on CPI and performance

Next Time

- Further mitigation of branch mispredictions
- State machine design