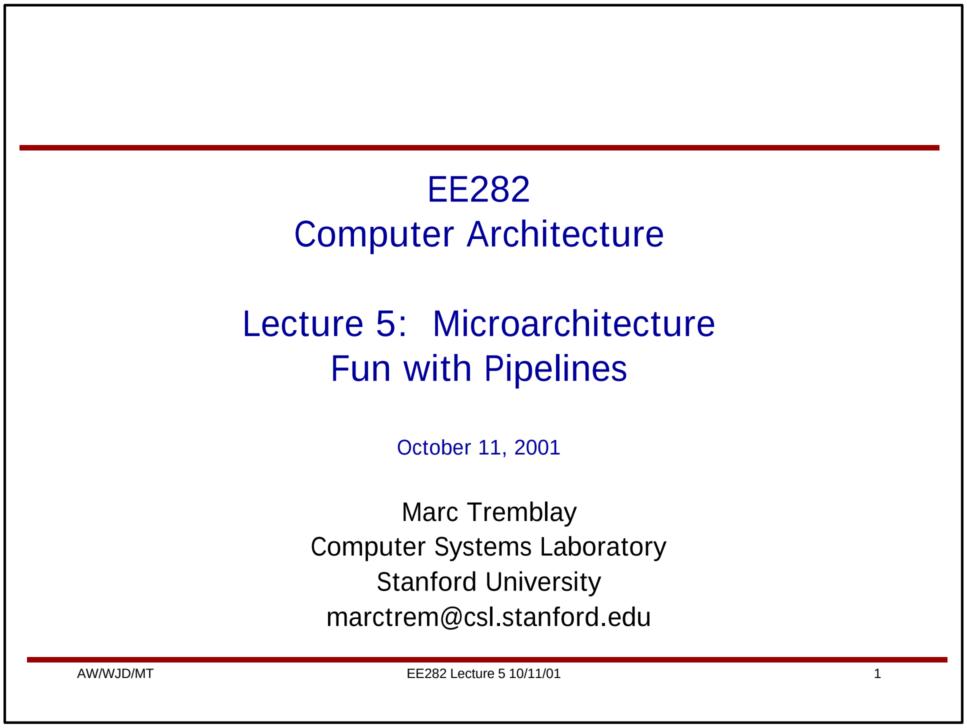
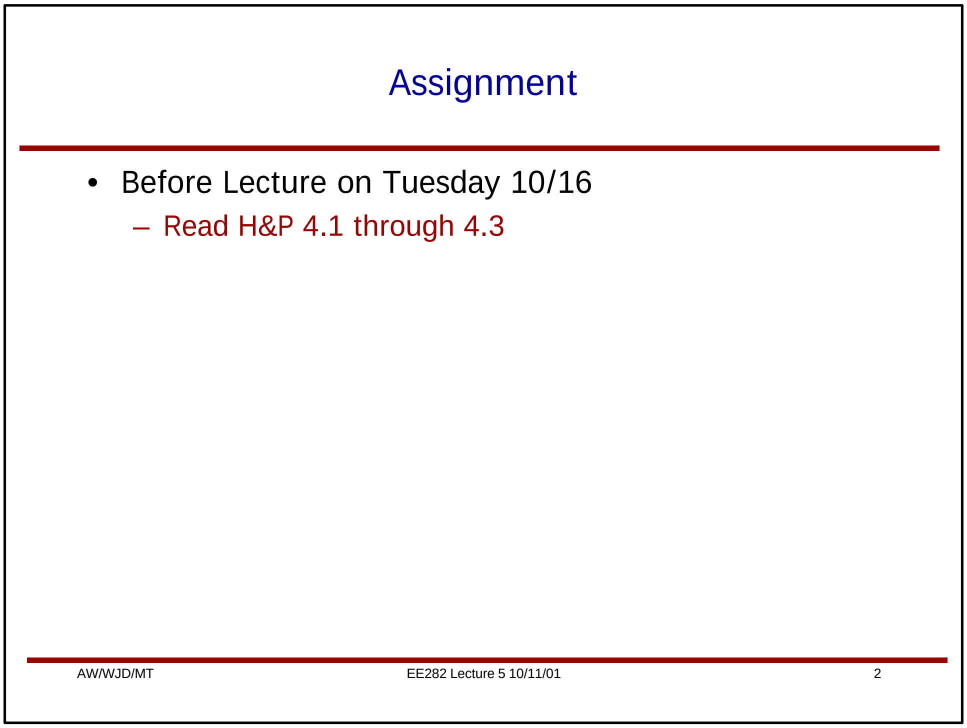


EE282
Computer Architecture

Lecture 5: Microarchitecture
Fun with Pipelines

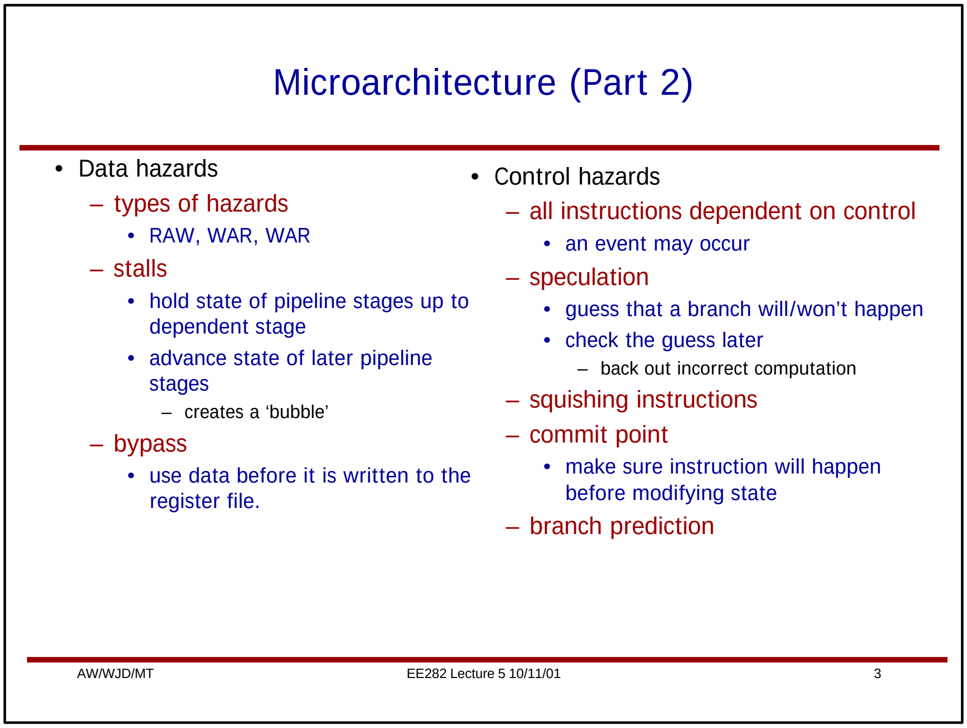
October 11, 2001

Marc Tremblay
Computer Systems Laboratory
Stanford University
marctrem@csl.stanford.edu



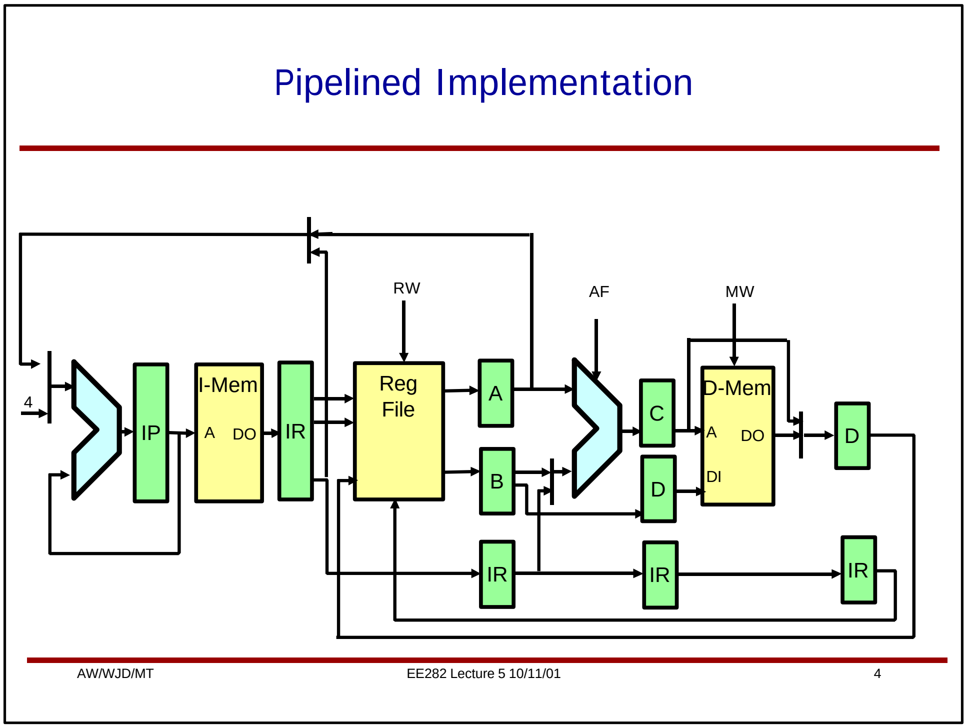
Assignment

- Before Lecture on Tuesday 10/16
  - Read H&P 4.1 through 4.3

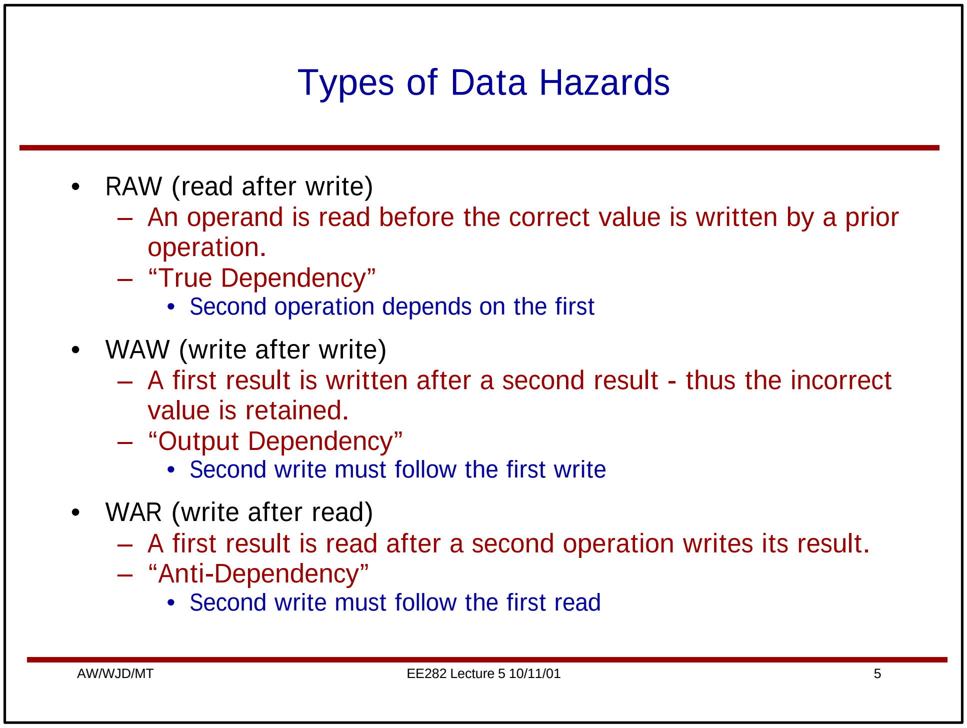


Microarchitecture (Part 2)

- Data hazards
  - – types of hazards
    - RAW, WAR, WAR
  - – stalls
    - hold state of pipeline stages up to dependent stage
    - advance state of later pipeline stages
      - – creates a 'bubble'
  - – bypass
    - use data before it is written to the register file.
- Control hazards
  - – all instructions dependent on control
    - an event may occur
  - – speculation
    - guess that a branch will/won't happen
    - check the guess later
      - – back out incorrect computation
  - – squishing instructions
  - – commit point
    - make sure instruction will happen before modifying state
  - – branch prediction

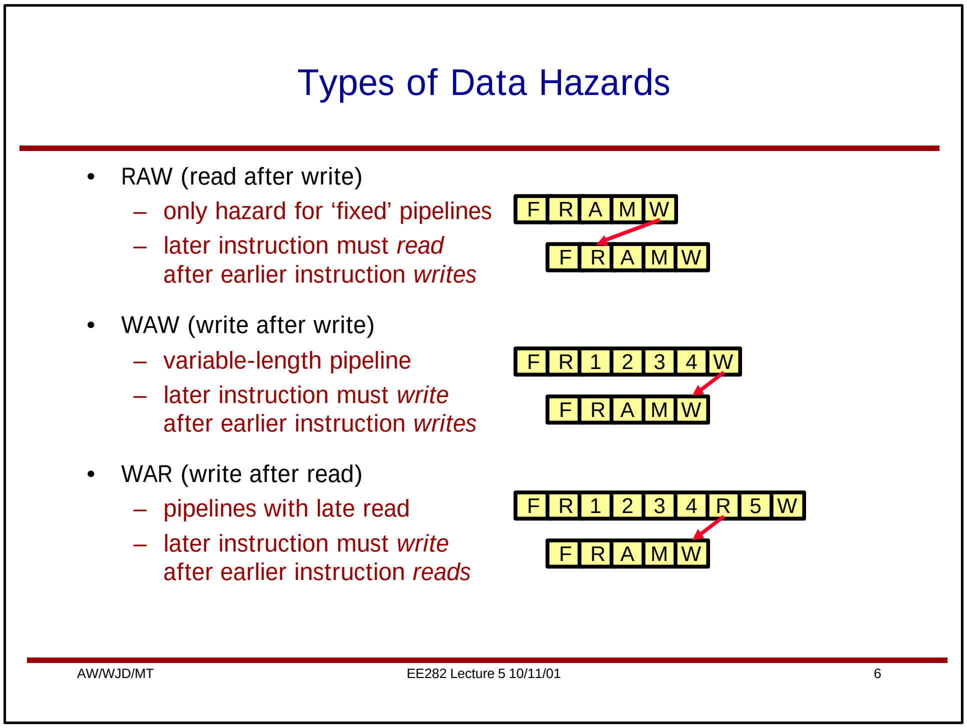


Pipelined Implementation



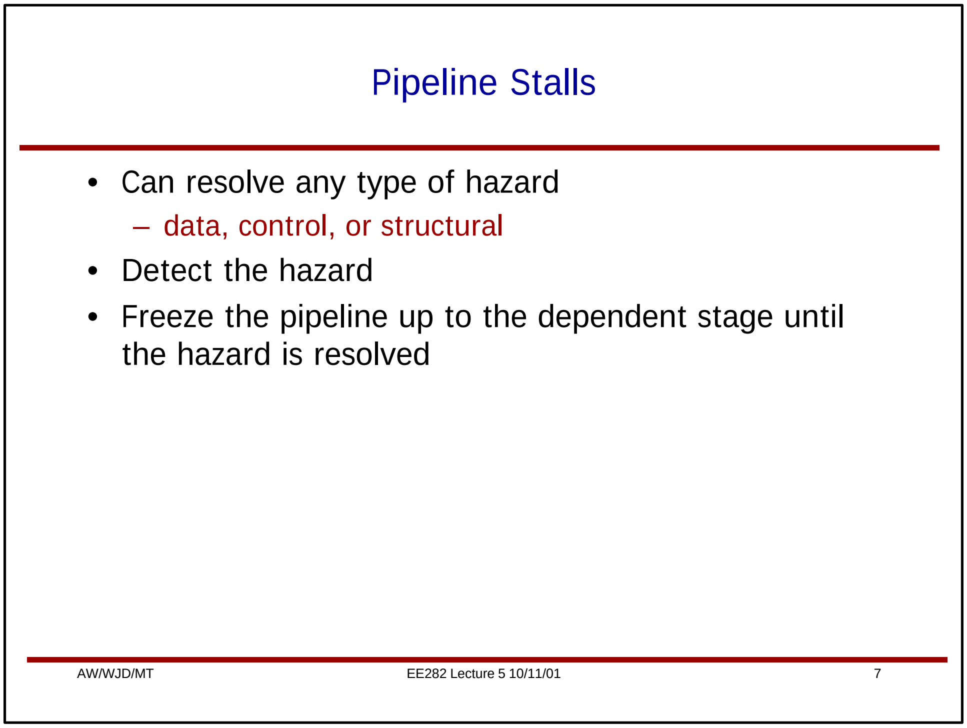
Types of Data Hazards

- RAW (read after write)
  - An operand is read before the correct value is written by a prior operation.
  - "True Dependency"
    - Second operation depends on the first
- WAW (write after write)
  - A first result is written after a second result - thus the incorrect value is retained.
  - "Output Dependency"
    - Second write must follow the first write
- WAR (write after read)
  - A first result is read after a second operation writes its result.
  - "Anti-Dependency"
    - Second write must follow the first read



Types of Data Hazards

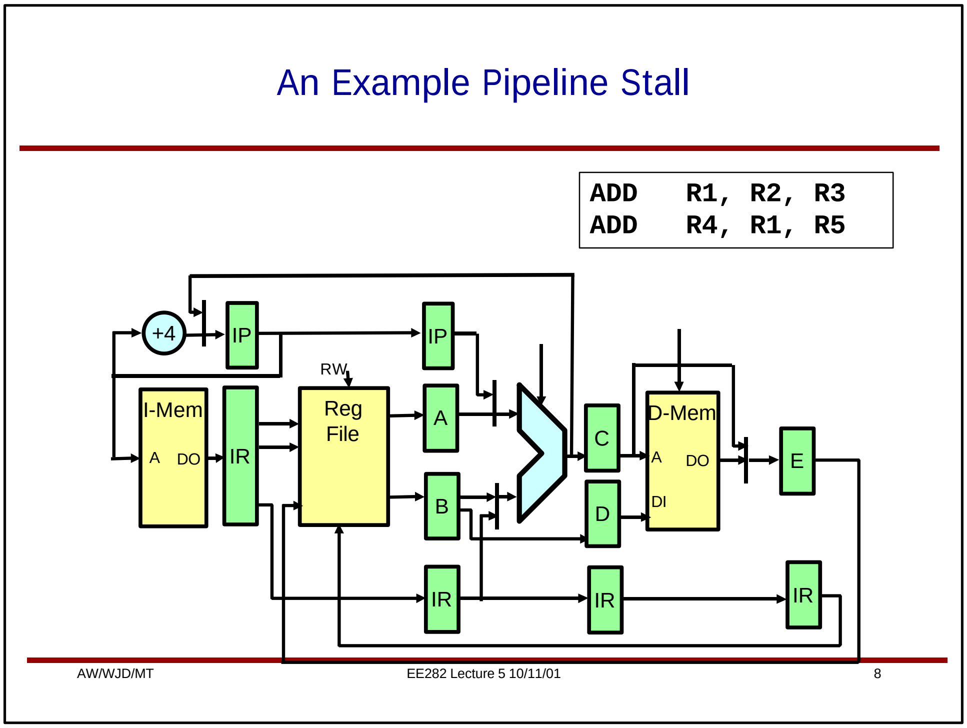
- RAW (read after write)
  - only hazard for 'fixed' pipelines
  - later instruction must *read* after earlier instruction *writes*
- WAW (write after write)
  - variable-length pipeline
  - later instruction must *write* after earlier instruction *writes*
- WAR (write after read)
  - pipelines with late read
  - later instruction must *write* after earlier instruction *reads*



Pipeline Stalls

- Can resolve any type of hazard
  - data, control, or structural
- Detect the hazard
- Freeze the pipeline up to the dependent stage until the hazard is resolved

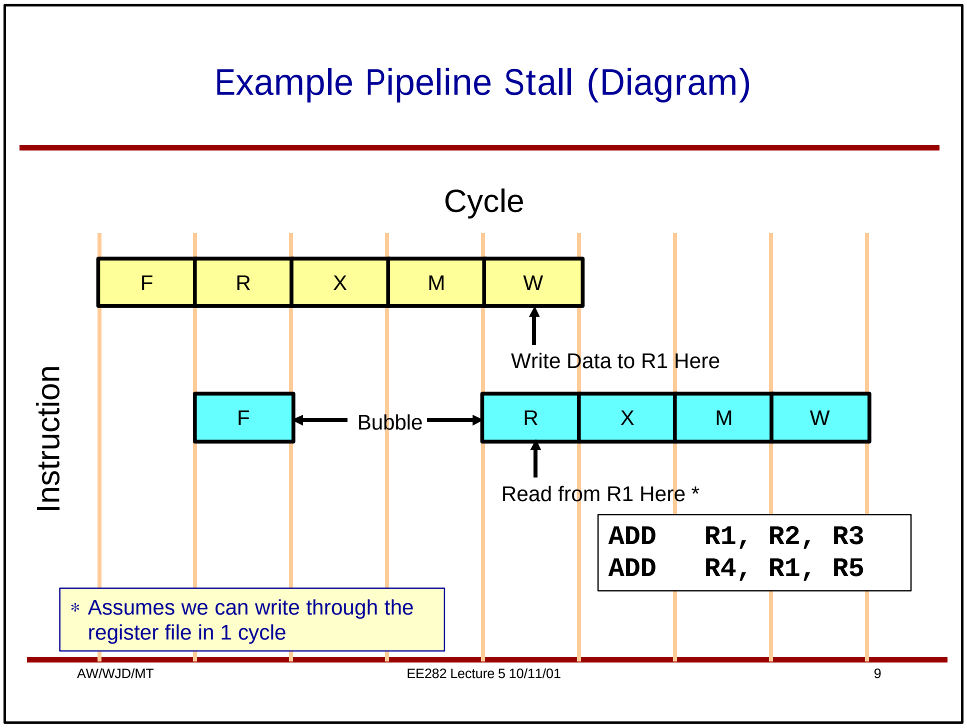
AW/WJD/MT EE282 Lecture 5 10/11/01 7



An Example Pipeline Stall

```
ADD   R1, R2, R3
ADD   R4, R1, R5
```

AW/WJD/MT EE282 Lecture 5 10/11/01 8

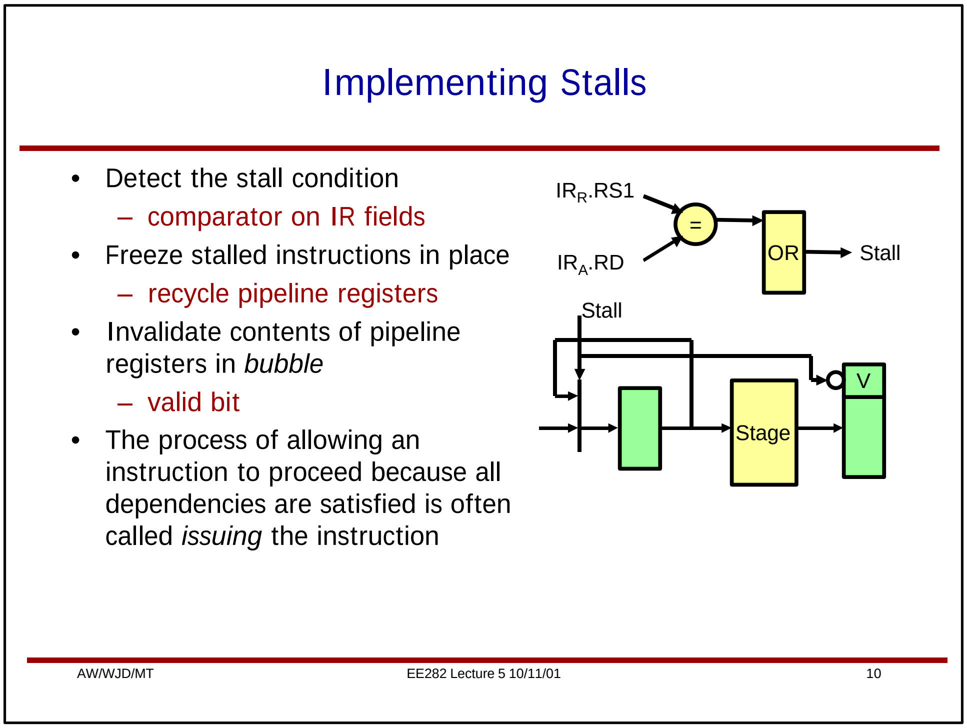


Example Pipeline Stall (Diagram)

```
ADD   R1, R2, R3
ADD   R4, R1, R5
```

* Assumes we can write through the register file in 1 cycle

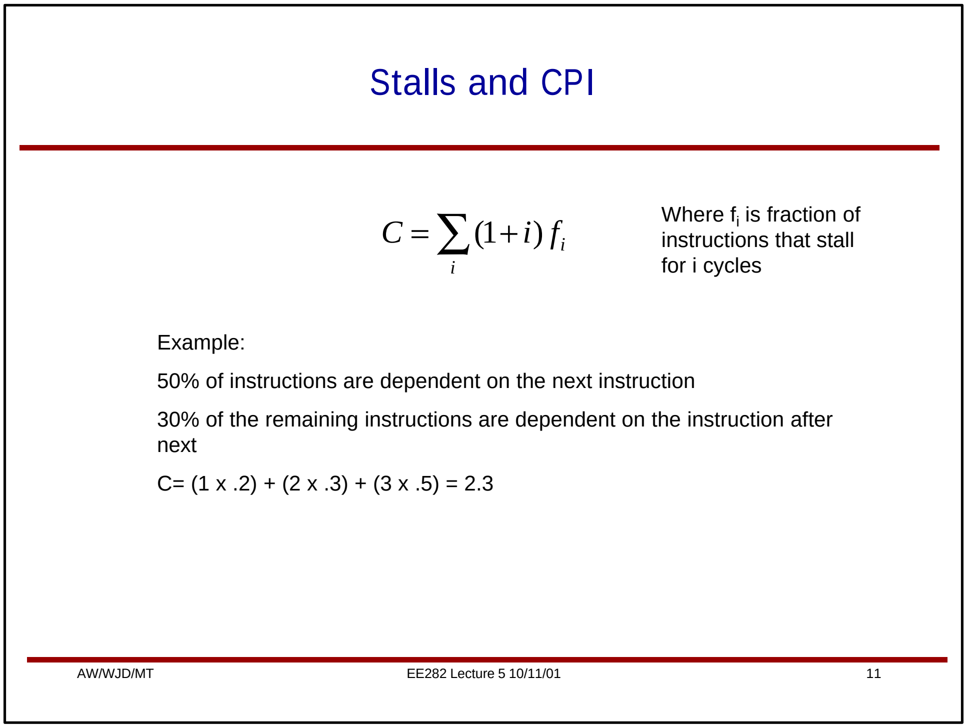
AW/WJD/MT EE282 Lecture 5 10/11/01 9



Implementing Stalls

- Detect the stall condition
  - comparator on IR fields
- Freeze stalled instructions in place
  - recycle pipeline registers
- Invalidate contents of pipeline registers in *bubble*
  - valid bit
- The process of allowing an instruction to proceed because all dependencies are satisfied is often called *issuing* the instruction

AW/WJD/MT EE282 Lecture 5 10/11/01 10



Stalls and CPI

$$C = \sum_i (1+i)\, f_i$$

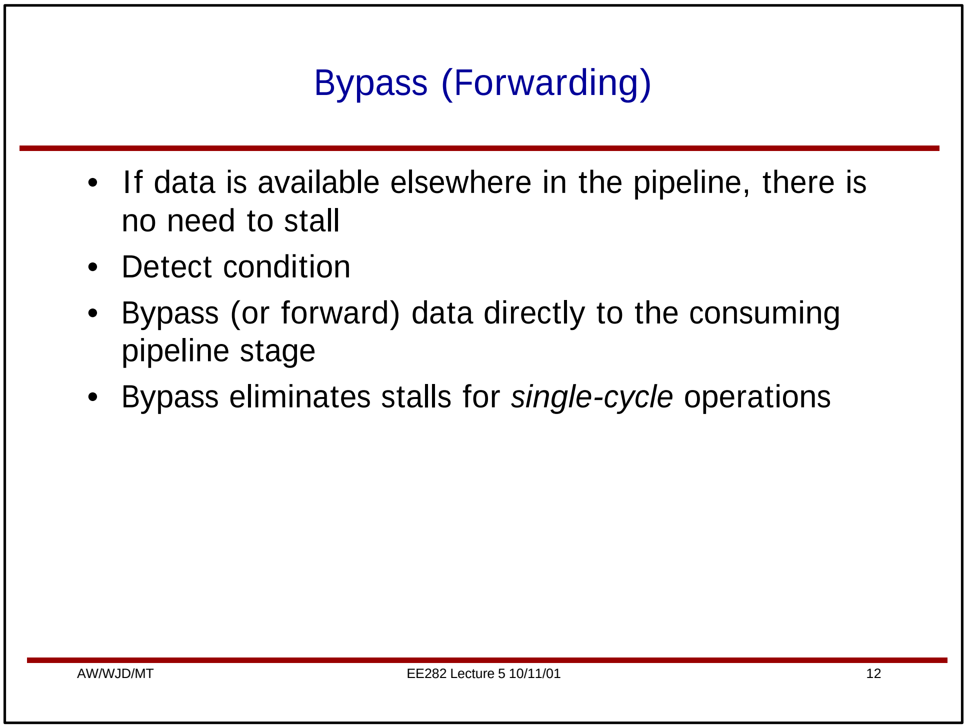
Where f_i is fraction of instructions that stall for i cycles

Example:

50% of instructions are dependent on the next instruction

30% of the remaining instructions are dependent on the instruction after next

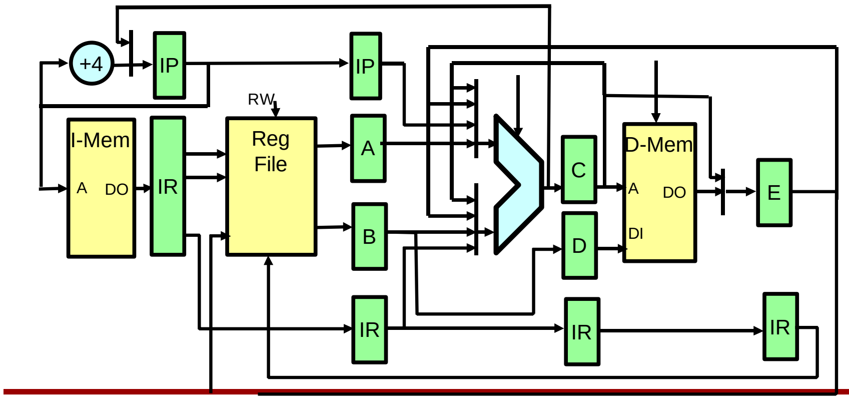
C= (1 x .2) + (2 x .3) + (3 x .5) = 2.3



Bypass (Forwarding)

- If data is available elsewhere in the pipeline, there is no need to stall
- Detect condition
- Bypass (or forward) data directly to the consuming pipeline stage
- Bypass eliminates stalls for *single-cycle* operations

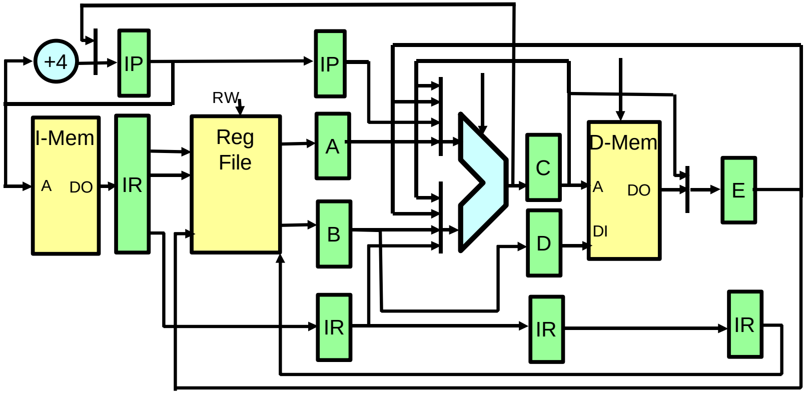
Simple Pipeline with Bypass Multiplexers



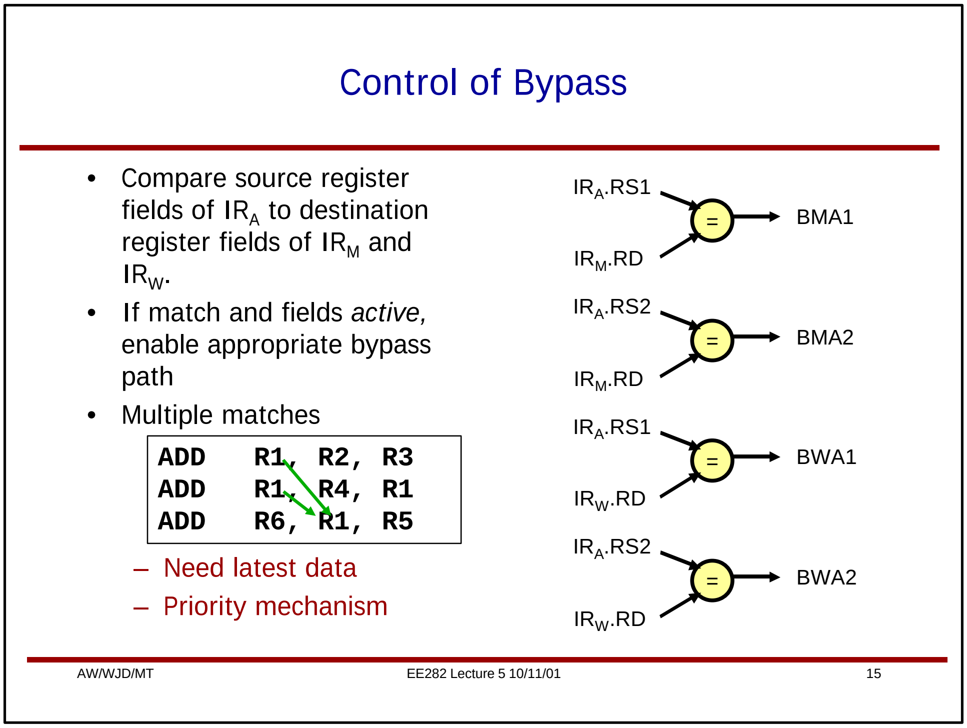
AW/WJD/MT EE282 Lecture 5 10/11/01 13

Example Execution with Bypass

```
ADD   R1, R2, R3
ADD   R4, R1, R5
```



AW/WJD/MT EE282 Lecture 5 10/11/01 14



Control of Bypass

- Compare source register fields of IR_A to destination register fields of IR_M and IR_W.
- If match and fields *active*, enable appropriate bypass path
- Multiple matches

```
ADD   R1, R2, R3
ADD   R1, R4, R1
ADD   R6, R1, R5
```

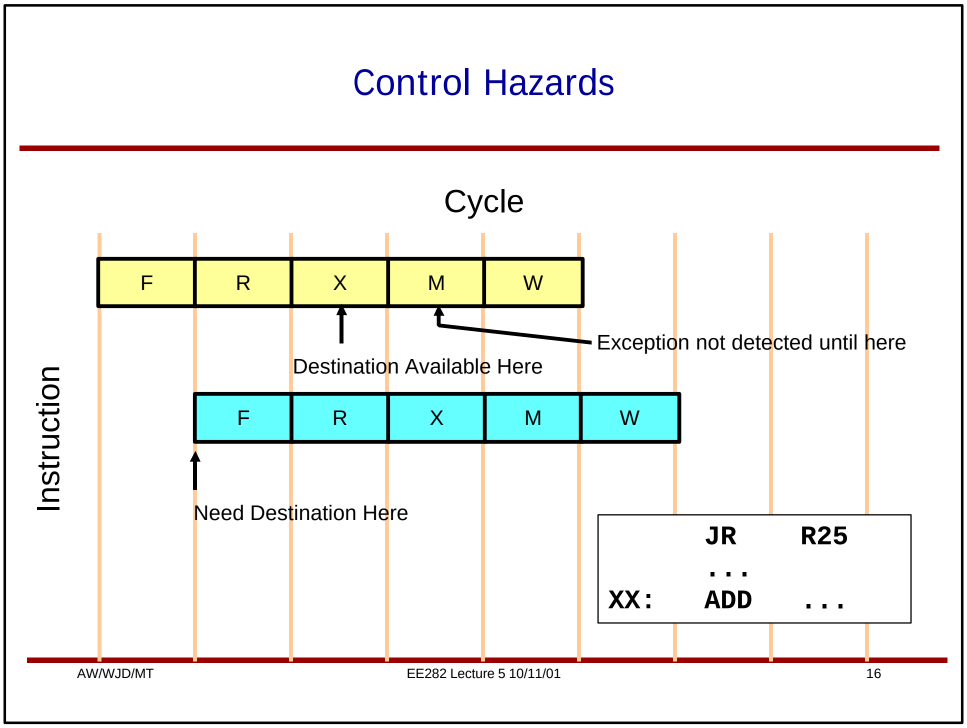
- Need latest data
- Priority mechanism

IR_A.RS1, IR_M.RD = BMA1

IR_A.RS2, IR_M.RD = BMA2

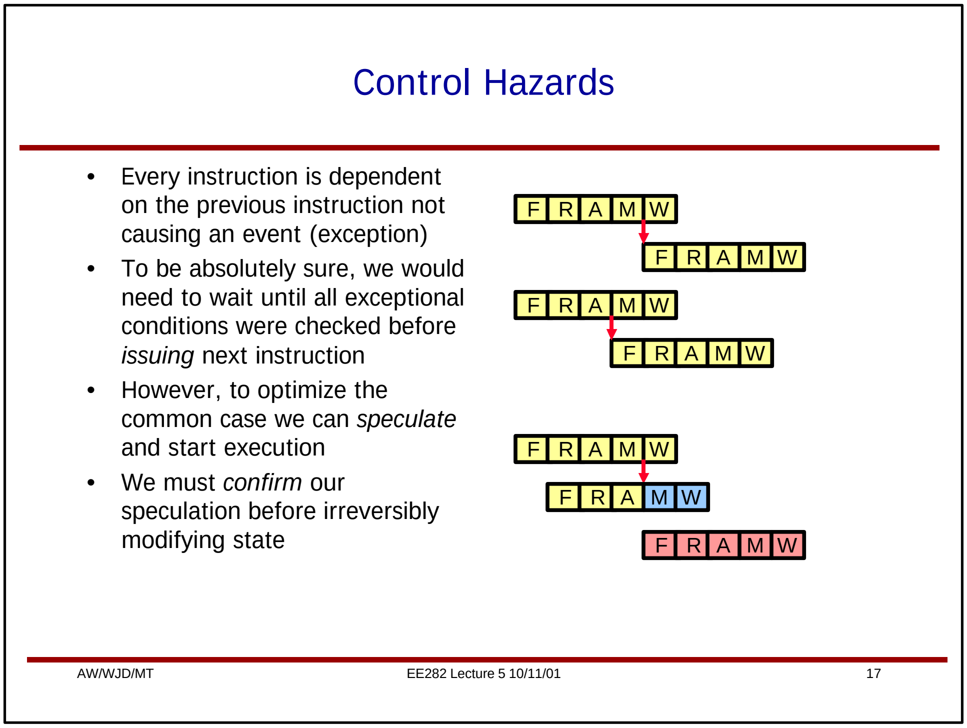
IR_A.RS1, IR_W.RD = BWA1

IR_A.RS2, IR_W.RD = BWA2



Control Hazards

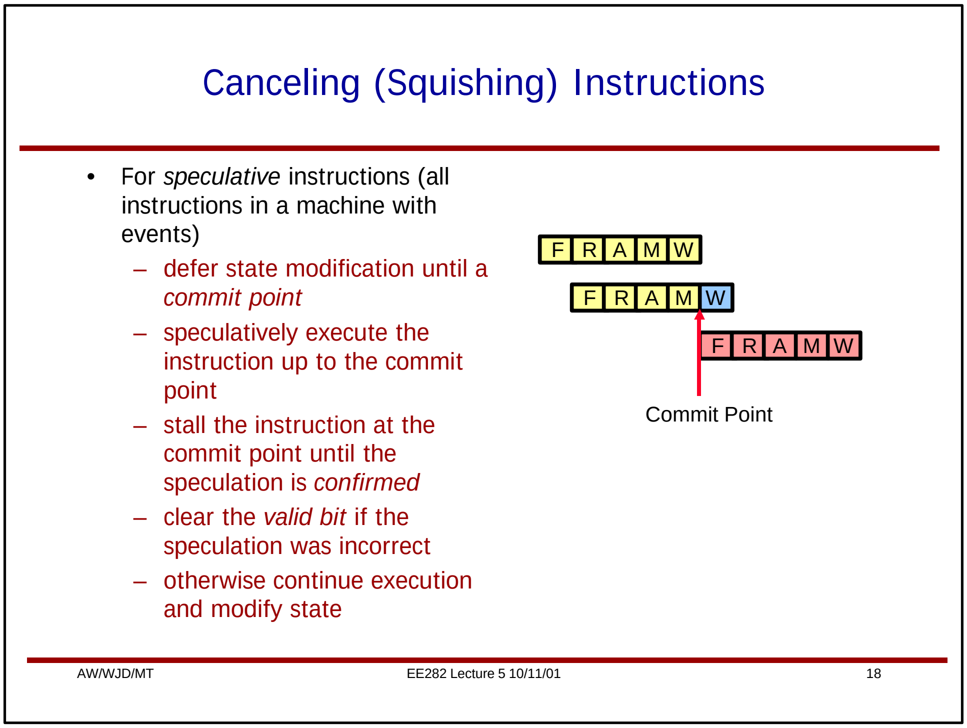
```
       JR     R25
       ...
XX:    ADD    ...
```



Control Hazards

- Every instruction is dependent on the previous instruction not causing an event (exception)
- To be absolutely sure, we would need to wait until all exceptional conditions were checked before *issuing* next instruction
- However, to optimize the common case we can *speculate* and start execution
- We must *confirm* our speculation before irreversibly modifying state

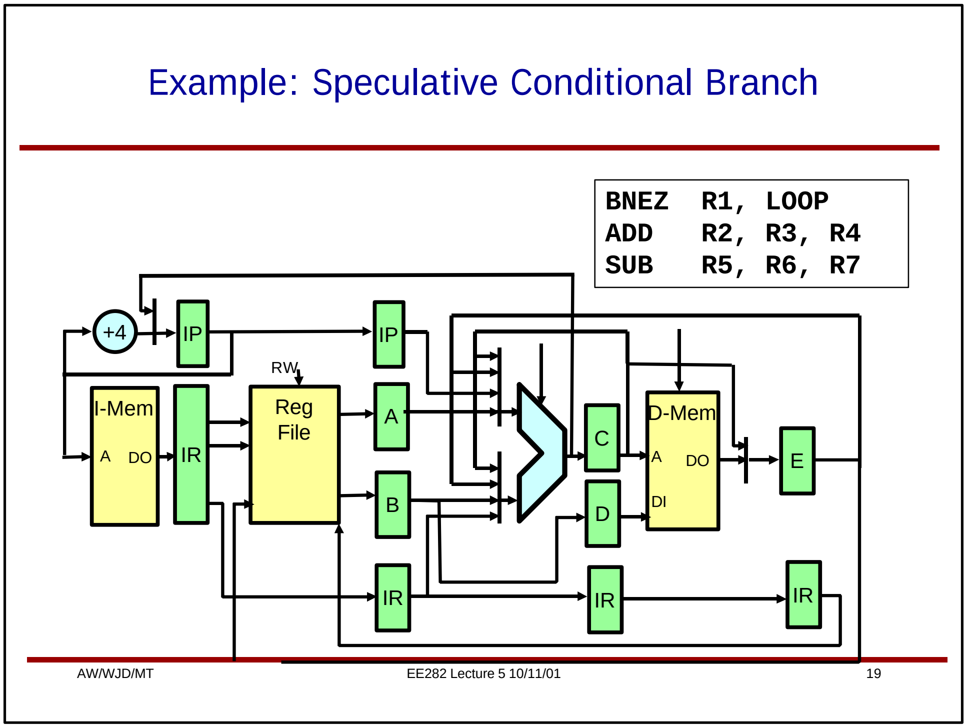
AW/WJD/MT EE282 Lecture 5 10/11/01 17



Canceling (Squishing) Instructions

- For *speculative* instructions (all instructions in a machine with events)
  - defer state modification until a *commit point*
  - speculatively execute the instruction up to the commit point
  - stall the instruction at the commit point until the speculation is *confirmed*
  - clear the *valid bit* if the speculation was incorrect
  - otherwise continue execution and modify state

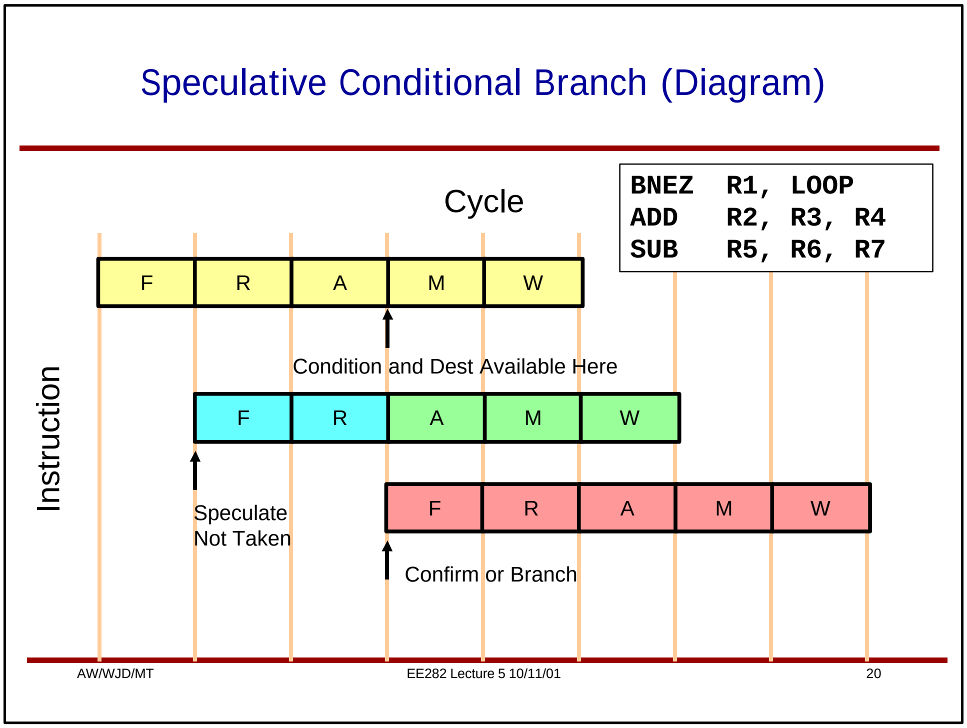
AW/WJD/MT EE282 Lecture 5 10/11/01 18



Example: Speculative Conditional Branch

```
BNEZ   R1, LOOP
ADD    R2, R3, R4
SUB    R5, R6, R7
```

AW/WJD/MT EE282 Lecture 5 10/11/01 19



Speculative Conditional Branch (Diagram)

```
BNEZ   R1, LOOP
ADD    R2, R3, R4
SUB    R5, R6, R7
```

Cycle

Instruction

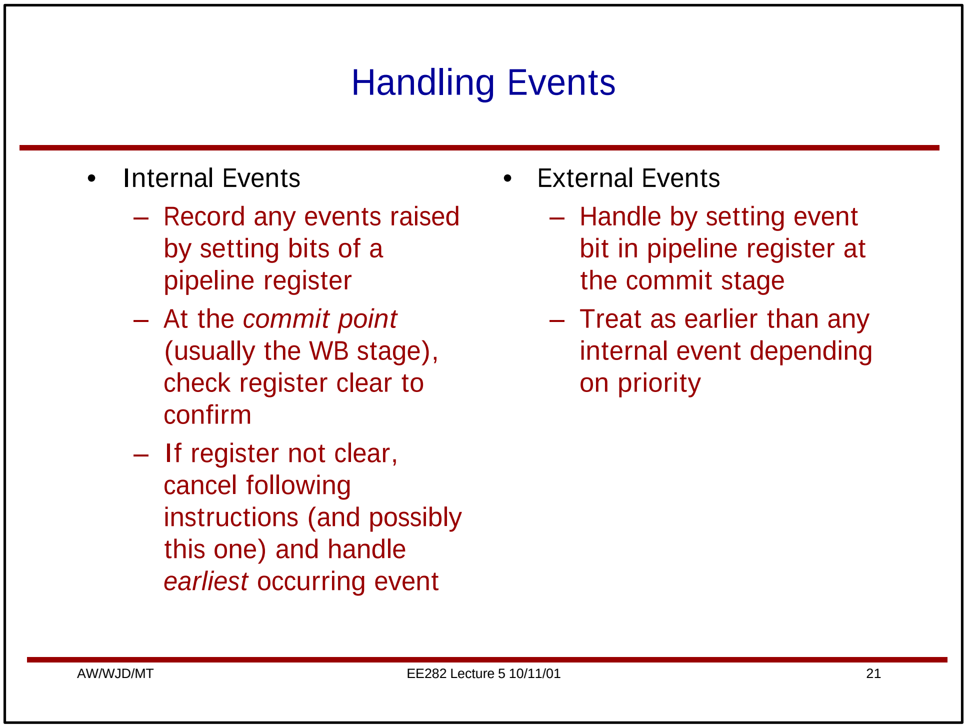
F	R	A	M	W			
	F	R	A	M	W		
			F	R	A	M	W

Condition and Dest Available Here

Speculate Not Taken

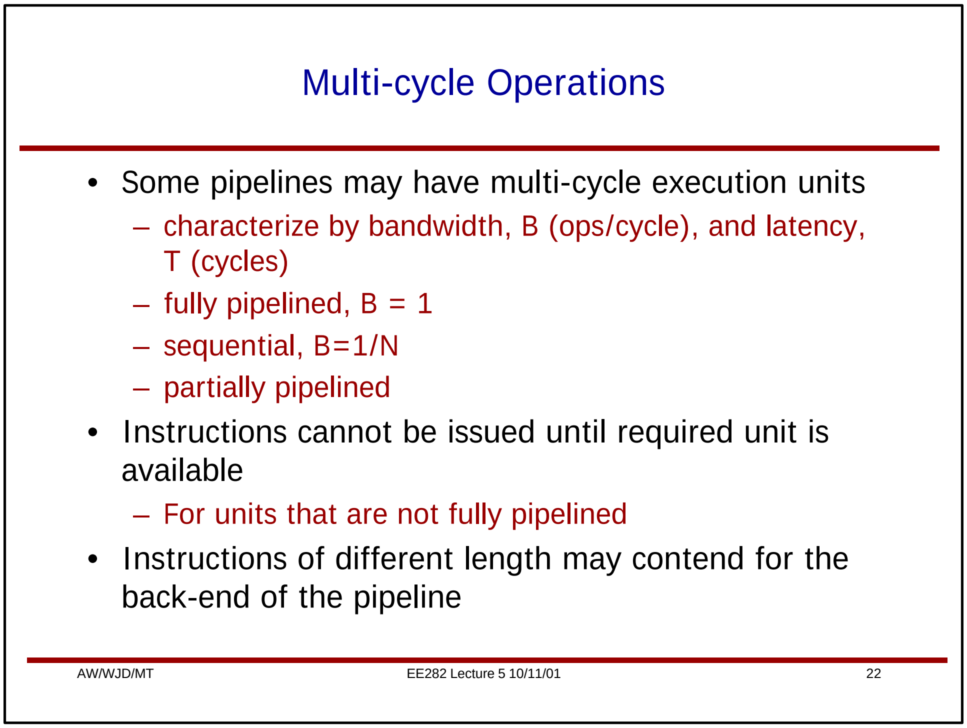
Confirm or Branch

AW/WJD/MT EE282 Lecture 5 10/11/01 20



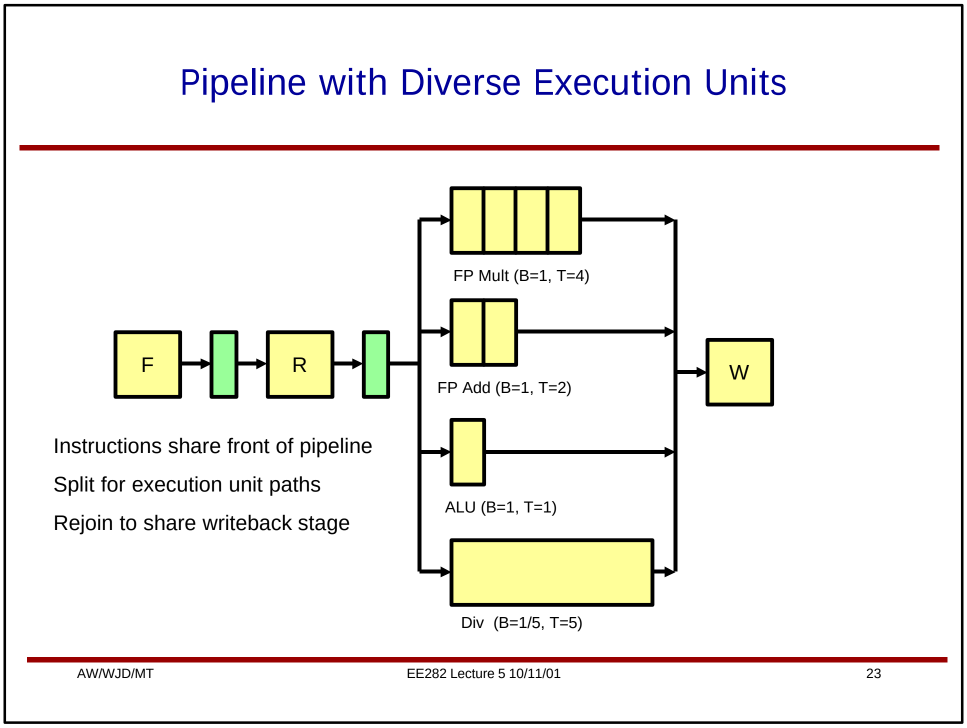
Handling Events

- Internal Events
  - Record any events raised by setting bits of a pipeline register
  - At the *commit point* (usually the WB stage), check register clear to confirm
  - If register not clear, cancel following instructions (and possibly this one) and handle *earliest* occurring event
- External Events
  - Handle by setting event bit in pipeline register at the commit stage
  - Treat as earlier than any internal event depending on priority



Multi-cycle Operations

- Some pipelines may have multi-cycle execution units
  - characterize by bandwidth, B (ops/cycle), and latency, T (cycles)
  - fully pipelined, B = 1
  - sequential, B=1/N
  - partially pipelined
- Instructions cannot be issued until required unit is available
  - For units that are not fully pipelined
- Instructions of different length may contend for the back-end of the pipeline

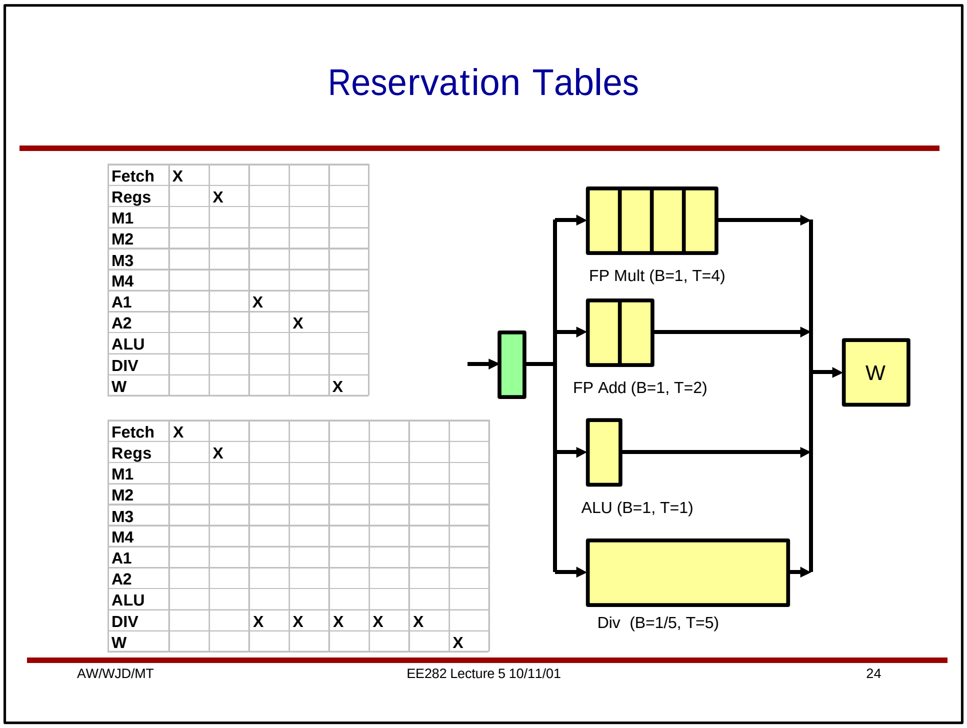


Pipeline with Diverse Execution Units

Instructions share front of pipeline

Split for execution unit paths

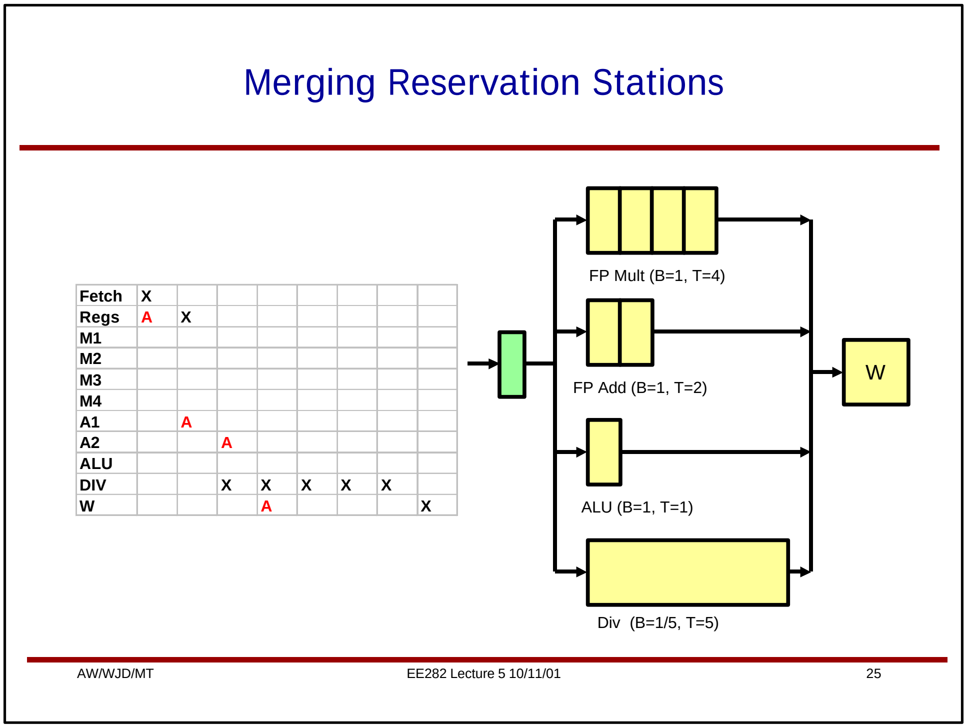
Rejoin to share writeback stage



Reservation Tables

Fetch	X				
Regs		X			
M1					
M2					
M3					
M4					
A1			X		
A2				X	
ALU					
DIV					
W					X

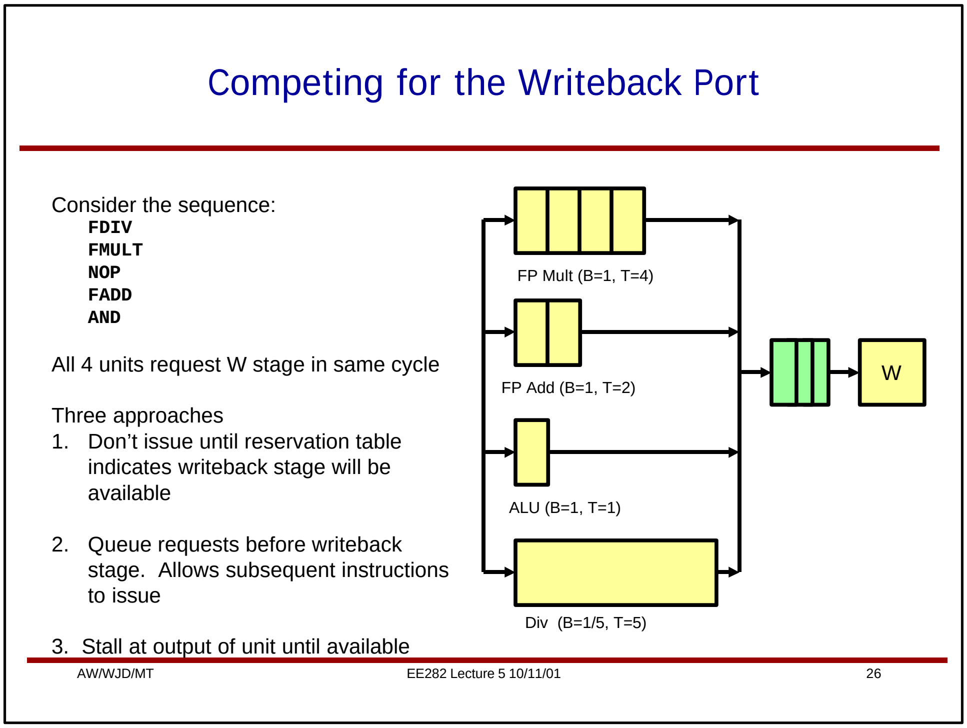
Fetch	X							
Regs		X						
M1								
M2								
M3								
M4								
A1								
A2								
ALU								
DIV			X	X	X	X	X	
W								X



Merging Reservation Stations

Fetch	X							
Regs	A	X						
M1								
M2								
M3								
M4								
A1		A						
A2			A					
ALU								
DIV			X	X	X	X	X	
W				A				X

AW/WJD/MT EE282 Lecture 5 10/11/01 25



Competing for the Writeback Port

Consider the sequence:

```
FDIV
FMULT
NOP
FADD
AND
```

All 4 units request W stage in same cycle

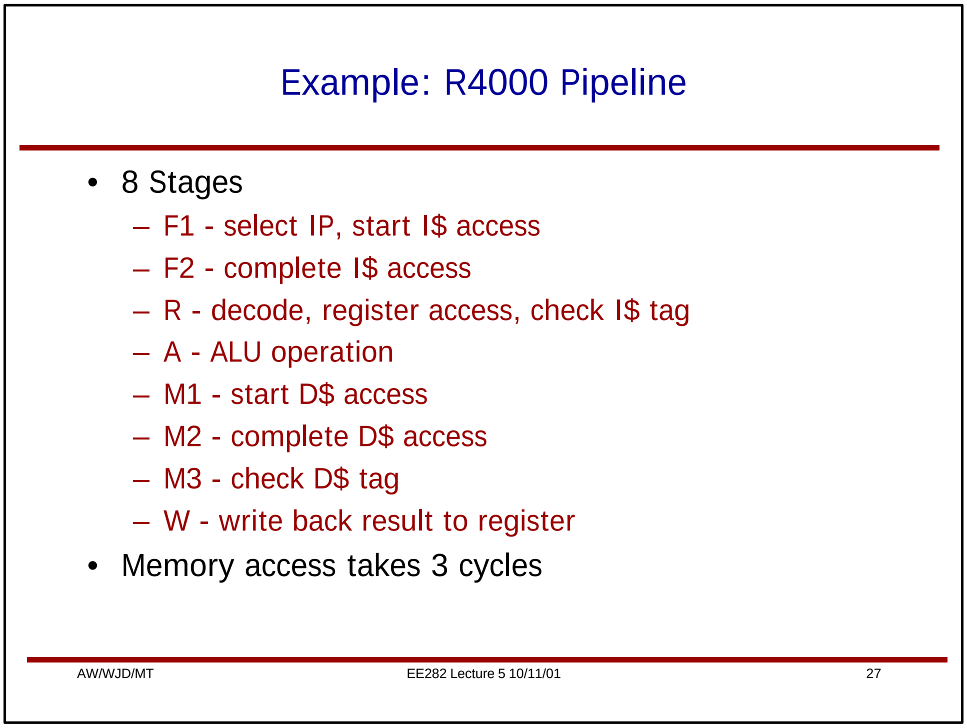
Three approaches

1. Don’t issue until reservation table indicates writeback stage will be available

2. Queue requests before writeback stage. Allows subsequent instructions to issue

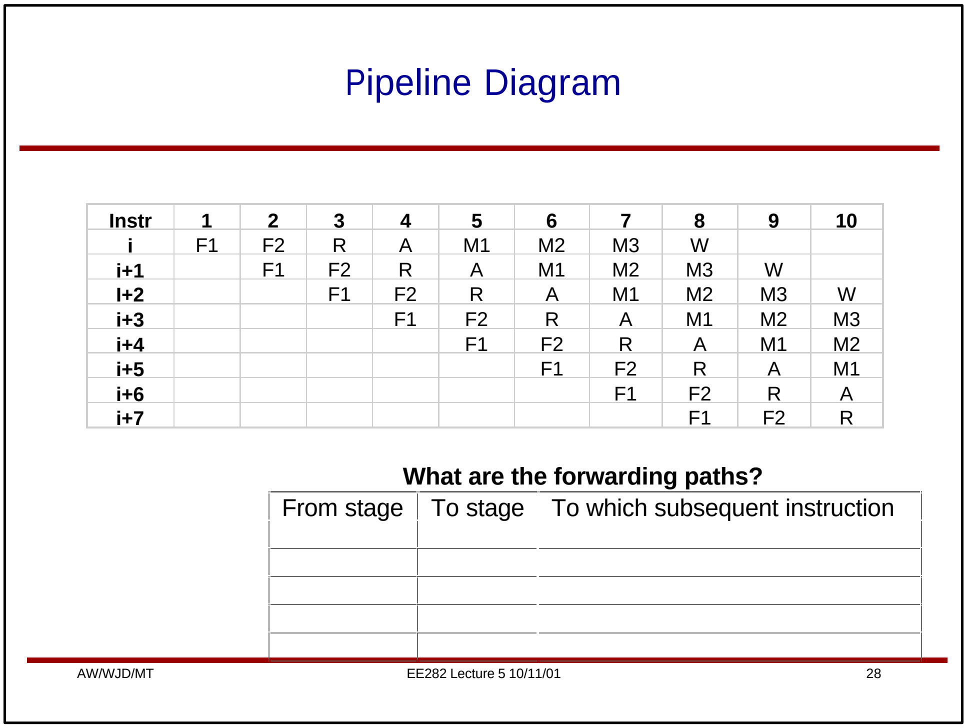
3. Stall at output of unit until available

AW/WJD/MT EE282 Lecture 5 10/11/01 26



Example: R4000 Pipeline

- 8 Stages
  - F1 - select IP, start I$ access
  - F2 - complete I$ access
  - R - decode, register access, check I$ tag
  - A - ALU operation
  - M1 - start D$ access
  - M2 - complete D$ access
  - M3 - check D$ tag
  - W - write back result to register
- Memory access takes 3 cycles

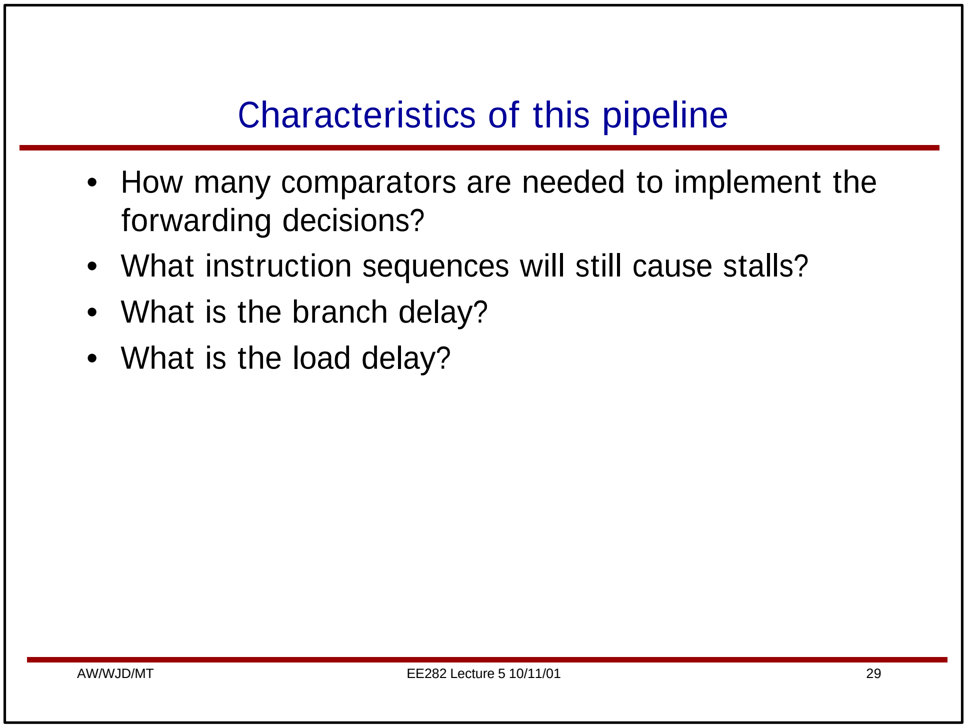


Pipeline Diagram

Instr	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
i	F1	F2	R	A	M1	M2	M3	W		
i+1		F1	F2	R	A	M1	M2	M3	W	
I+2			F1	F2	R	A	M1	M2	M3	W
i+3				F1	F2	R	A	M1	M2	M3
i+4					F1	F2	R	A	M1	M2
i+5						F1	F2	R	A	M1
i+6							F1	F2	R	A
i+7								F1	F2	R

What are the forwarding paths?

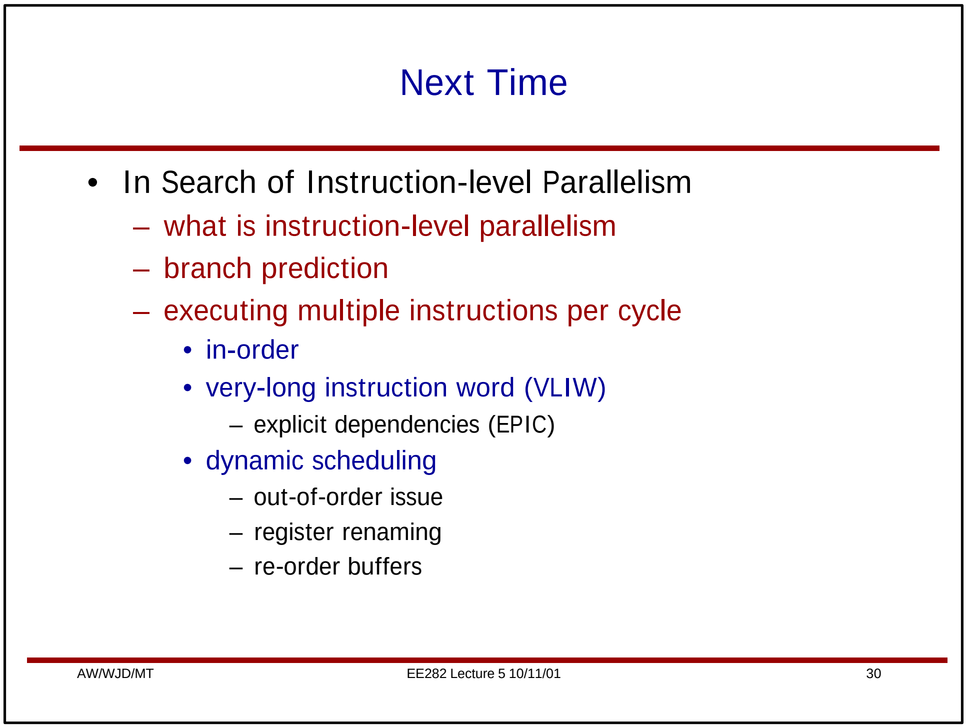
From stage	To stage	To which subsequent instruction



Characteristics of this pipeline

- How many comparators are needed to implement the forwarding decisions?
- What instruction sequences will still cause stalls?
- What is the branch delay?
- What is the load delay?

AW/WJD/MT EE282 Lecture 5 10/11/01 29



Next Time

- In Search of Instruction-level Parallelism
  - what is instruction-level parallelism
  - branch prediction
  - executing multiple instructions per cycle
    - in-order
    - very-long instruction word (VLIW)
      - explicit dependencies (EPIC)
    - dynamic scheduling
      - out-of-order issue
      - register renaming
      - re-order buffers

AW/WJD/MT EE282 Lecture 5 10/11/01 30