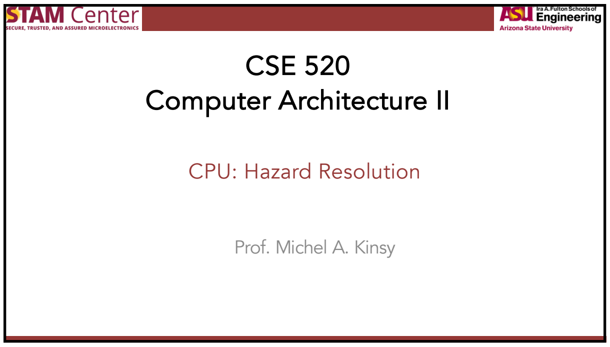
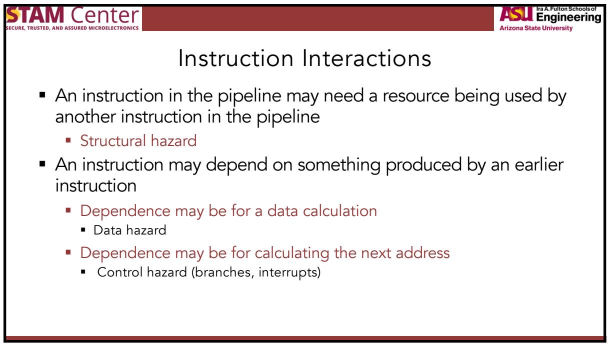


CSE 520
Computer Architecture II

CPU: Hazard Resolution

Prof. Michel A. Kinsy

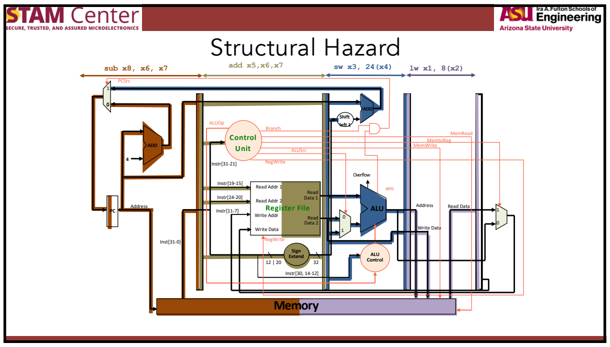
1



Instruction Interactions

- 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 calculation
    - Data hazard
  - Dependence may be for calculating the next address
    - Control hazard (branches, interrupts)

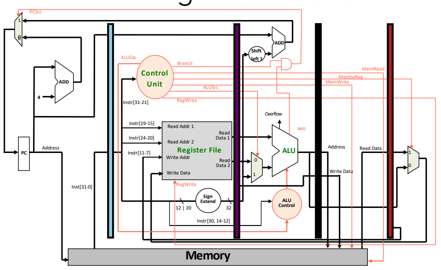
2



Structural Hazard

3

Multi-Stage RISC-V CPU



4

Stalled Stages and Pipeline

	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$_2$
(I$_3$)			IF$_3$	IF$_3$	IF$_3$	IF$_3$	ID$_3$	EX$_3$
(I$_4$)							IF$_4$	ID$_4$
(I$_5$)								IF$_5$

stalled stages

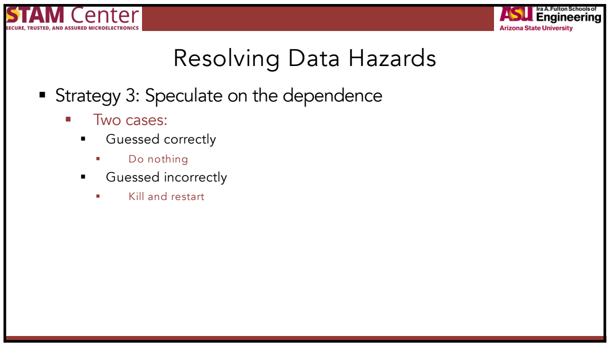
		time								
		t0	t1	t2	t3	t4	t5	t6	t7	. . .
Resource Usage	IF	I$_1$	I$_2$	I$_3$	I$_3$	I$_3$	I$_3$	I$_4$	I$_5$	
	ID		I$_1$	I$_2$	I$_2$	I$_2$	I$_2$	I$_3$	I$_4$	
	EX			I$_1$	nop	nop	nop	I$_2$	I$_3$	
	MA				I$_1$	nop	nop	nop	I$_2$	
	WB					I$_1$	nop	nop	nop	

5

Resolving Data Hazards

- Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  - Interlocks
- Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - Bypass

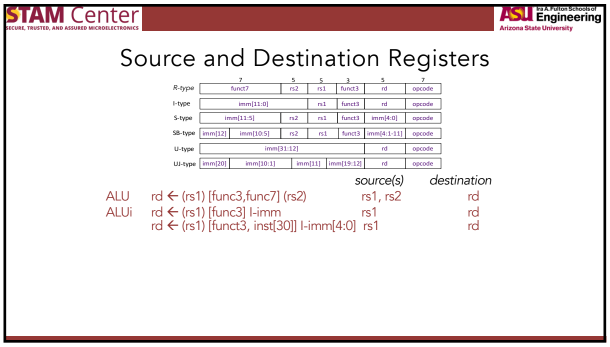
6



Resolving Data Hazards

- Strategy 3: Speculate on the dependence
  - Two cases:
    - Guessed correctly
      - Do nothing
    - Guessed incorrectly
      - Kill and restart

7

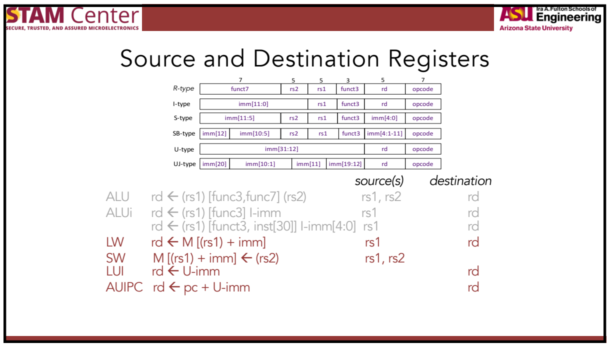


Source and Destination Registers

	7	5	5	3	5	7
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	imm[11:0]		rs1	funct3	rd	opcode
S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB-type	imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1-11]	opcode
U-type	imm[31:12]				rd	opcode
UJ-type	imm[20] imm[10:1]		imm[11]	imm[19:12]	rd	opcode

		source(s)	*destination*
ALU	rd ← (rs1) [func3,func7] (rs2)	rs1, rs2	rd
ALUi	rd ← (rs1) [func3] I-imm	rs1	rd
	rd ← (rs1) [funct3, inst[30]] I-imm[4:0]	rs1	rd

8

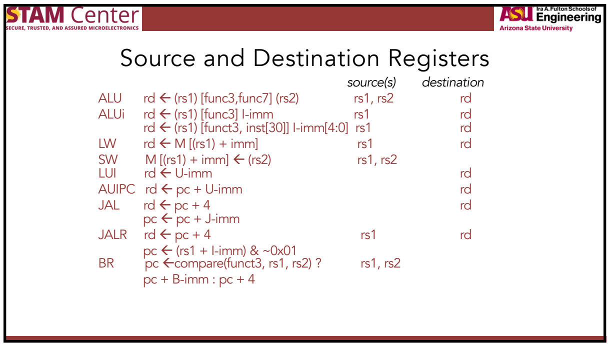


Source and Destination Registers

	7	5	5	3	5	7
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	imm[11:0]		rs1	funct3	rd	opcode
S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB-type	imm[12] imm[10:5]	rs2	rs1	funct3	imm[4:1-11]	opcode
U-type	imm[31:12]				rd	opcode
UJ-type	imm[20] imm[10:1]		imm[11]	imm[19:12]	rd	opcode

		source(s)	*destination*
ALU	rd ← (rs1) [func3,func7] (rs2)	rs1, rs2	rd
ALUi	rd ← (rs1) [func3] I-imm	rs1	rd
	rd ← (rs1) [funct3, inst[30]] I-imm[4:0]	rs1	rd
LW	rd ← M [(rs1) + imm]	rs1	rd
SW	M [(rs1) + imm] ← (rs2)	rs1, rs2	
LUI	rd ← U-imm		rd
AUIPC	rd ← pc + U-imm		rd

9



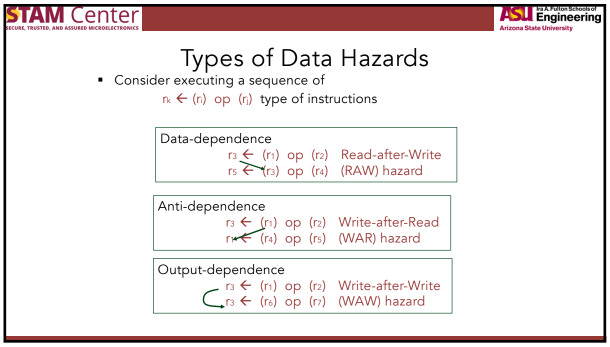
STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

Ira A. Fulton Schools of Engineering Arizona State University

Source and Destination Registers

		source(s)	*destination*
ALU	rd ← (rs1) [func3,func7] (rs2)	rs1, rs2	rd
ALUi	rd ← (rs1) [func3] I-imm	rs1	rd
	rd ← (rs1) [funct3, inst[30]] I-imm[4:0]	rs1	rd
LW	rd ← M [(rs1) + imm]	rs1	rd
SW	M [(rs1) + imm] ← (rs2)	rs1, rs2	
LUI	rd ← U-imm		rd
AUIPC	rd ← pc + U-imm		rd
JAL	rd ← pc + 4		rd
	pc ← pc + J-imm		
JALR	rd ← pc + 4	rs1	rd
	pc ← (rs1 + I-imm) & ~0x01		
BR	pc ←compare(funct3, rs1, rs2) ?	rs1, rs2	
	pc + B-imm : pc + 4		

10



STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

Ira A. Fulton Schools of Engineering Arizona State University

Types of Data Hazards

- Consider executing a sequence of
  r_k ← (r_i) op (r_j) type of instructions

Data-dependence

r_3 ← (r_1) op (r_2) Read-after-Write
r_5 ← (r_3) op (r_4) (RAW) hazard

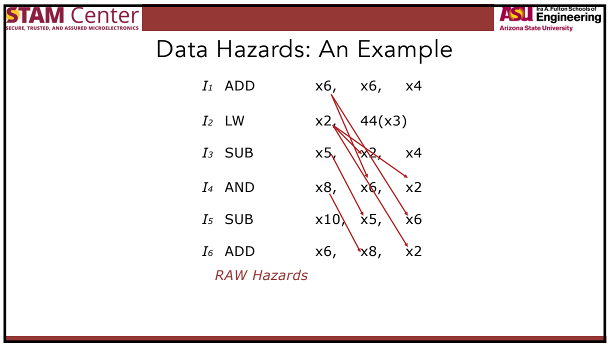
Anti-dependence

r_3 ← (r_1) op (r_2) Write-after-Read
r_1 ← (r_4) op (r_5) (WAR) hazard

Output-dependence

r_3 ← (r_1) op (r_2) Write-after-Write
r_3 ← (r_6) op (r_7) (WAW) hazard

11



STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

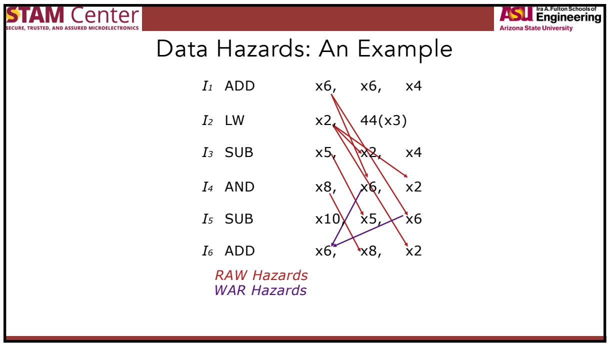
Ira A. Fulton Schools of Engineering Arizona State University

Data Hazards: An Example

```
I1  ADD    x6,   x6,   x4
I2  LW     x2,   44(x3)
I3  SUB    x5,   x2,   x4
I4  AND    x8,   x6,   x2
I5  SUB    x10,  x5,   x6
I6  ADD    x6,   x8,   x2
```

RAW Hazards

12



Data Hazards: An Example

I_1 ADD x6, x6, x4

I_2 LW x2, 44(x3)

I_3 SUB x5, x2, x4

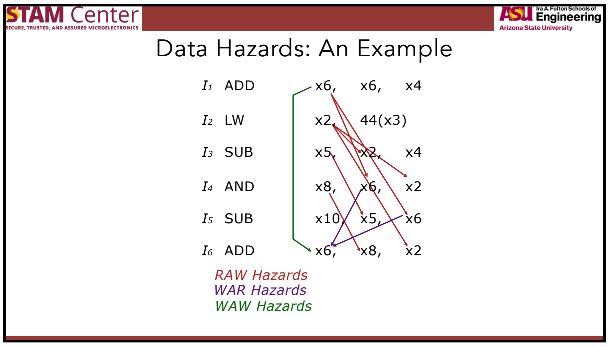
I_4 AND x8, x6, x2

I_5 SUB x10, x5, x6

I_6 ADD x6, x8, x2

RAW Hazards
WAR Hazards

13



Data Hazards: An Example

I_1 ADD x6, x6, x4

I_2 LW x2, 44(x3)

I_3 SUB x5, x2, x4

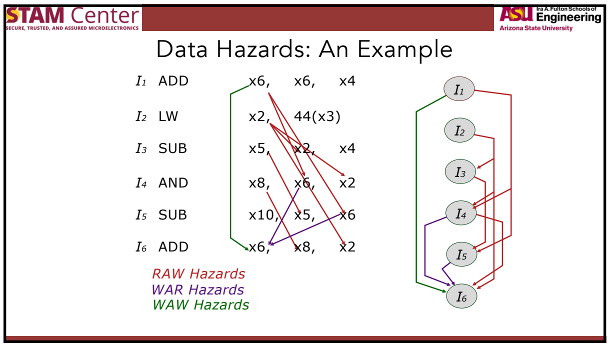
I_4 AND x8, x6, x2

I_5 SUB x10, x5, x6

I_6 ADD x6, x8, x2

RAW Hazards
WAR Hazards
WAW Hazards

14



Data Hazards: An Example

I_1 ADD x6, x6, x4

I_2 LW x2, 44(x3)

I_3 SUB x5, x2, x4

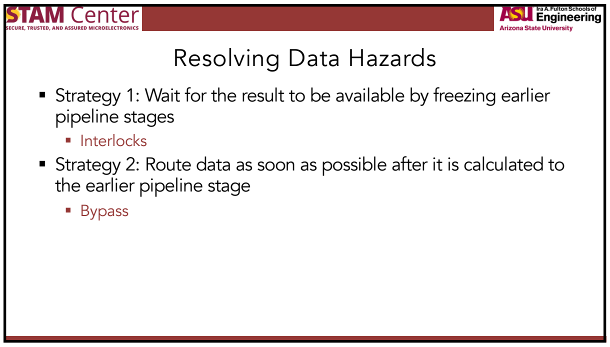
I_4 AND x8, x6, x2

I_5 SUB x10, x5, x6

I_6 ADD x6, x8, x2

RAW Hazards
WAR Hazards
WAW Hazards

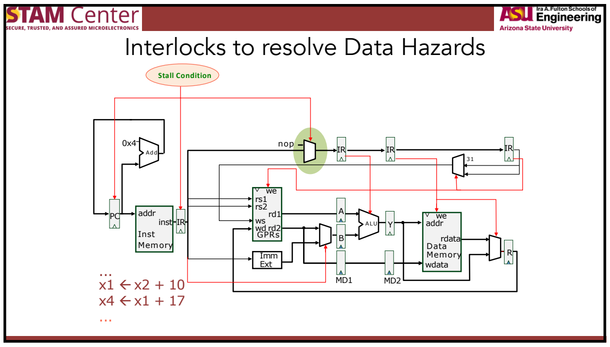
15



Resolving Data Hazards

- Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  - Interlocks
- Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - Bypass

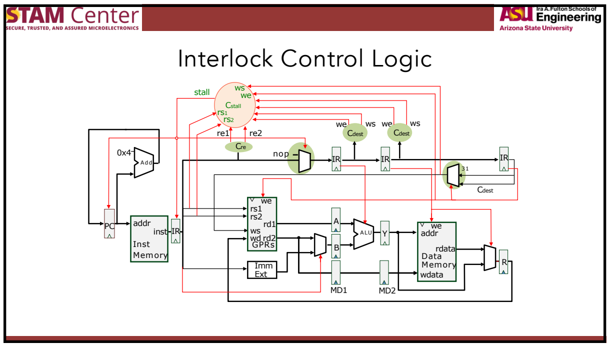
16



Interlocks to resolve Data Hazards

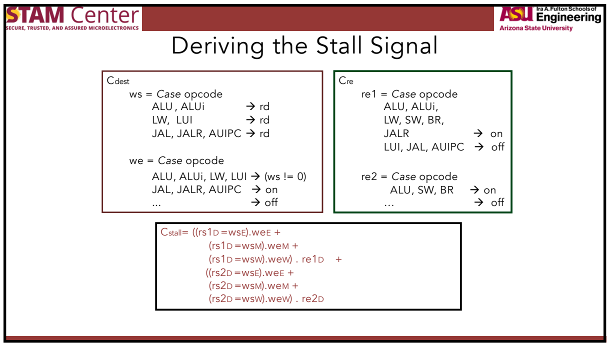
...
x1 ← x2 + 10
x4 ← x1 + 17
...

17



Interlock Control Logic

18



STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS — ASU Ira A. Fulton Schools of Engineering, Arizona State University

Deriving the Stall Signal

C_{dest}

```
ws = Case opcode
    ALU , ALUi          → rd
    LW,  LUI            → rd
    JAL, JALR, AUIPC    → rd

we = Case opcode
    ALU, ALUi, LW, LUI  → (ws != 0)
    JAL, JALR, AUIPC    → on
    ...                 → off
```

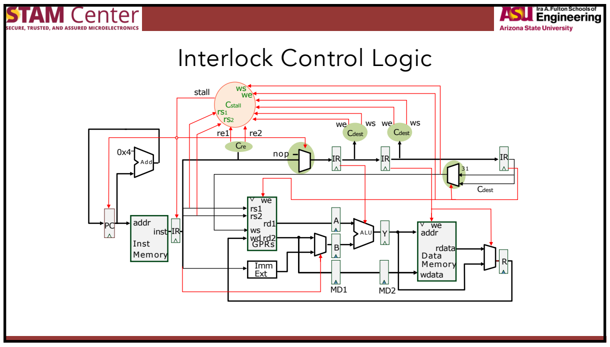
C_{re}

```
re1 = Case opcode
    ALU, ALUi,
    LW, SW, BR,
    JALR                → on
    LUI, JAL, AUIPC     → off

re2 = Case opcode
    ALU, SW, BR         → on
    ...                 → off
```

$$
\begin{aligned}
C_{stall} = \; & ((rs1_D = ws_E).we_E + \\
& (rs1_D = ws_M).we_M + \\
& (rs1_D = ws_W).we_W) \,.\, re1_D \; + \\
& ((rs2_D = ws_E).we_E + \\
& (rs2_D = ws_M).we_M + \\
& (rs2_D = ws_W).we_W) \,.\, re2_D
\end{aligned}
$$

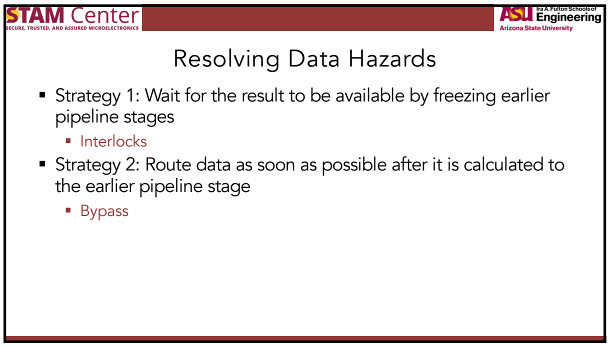
19



STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS — ASU Ira A. Fulton Schools of Engineering, Arizona State University

Interlock Control Logic

20



STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS — ASU Ira A. Fulton Schools of Engineering, Arizona State University

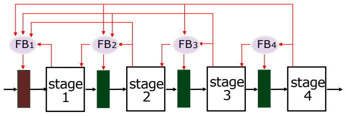
Resolving Data Hazards

- Strategy 1: Wait for the result to be available by freezing earlier pipeline stages
  - Interlocks
- Strategy 2: Route data as soon as possible after it is calculated to the earlier pipeline stage
  - Bypass

21

Feedback to Resolve Hazards

- Later stages provide dependence information to earlier stages which can stall (or kill) instructions
- Controlling a pipeline in this manner works provided the instruction at stage i+1 can complete without any interference from instructions in stages 1 to i
  - Otherwise deadlocks may occur



22

Bypassing

	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
(I_3)			IF_3	IF_3	IF_3	IF_3	ID_3	EX_3
(I_4)							IF_4	ID_4
(I_5)								IF_5

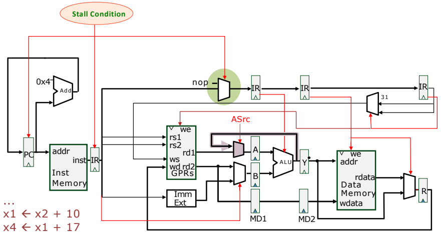
stalled stages

- Each stall or kill introduces a bubble in the pipeline → CPI > 1

	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	
(I_5)					IF_5	ID_5	EX_5	

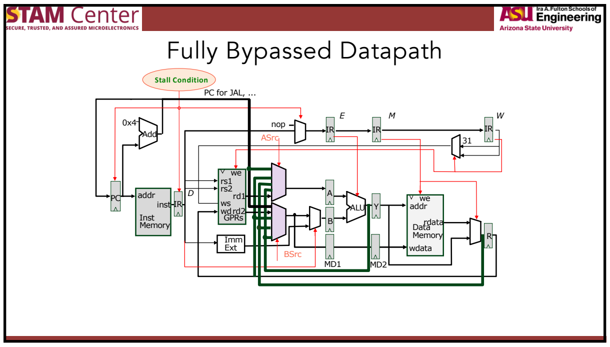
23

Adding a Bypass



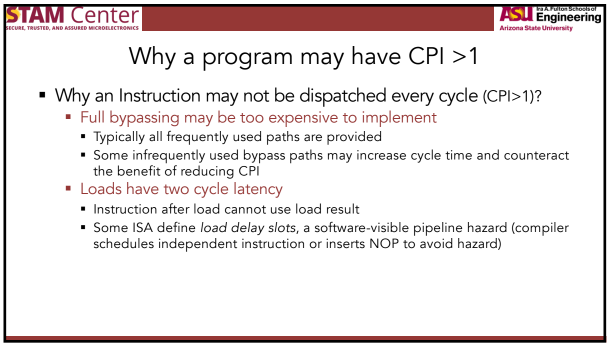
…
x1 ← x2 + 10
x4 ← x1 + 17
…

24



Fully Bypassed Datapath

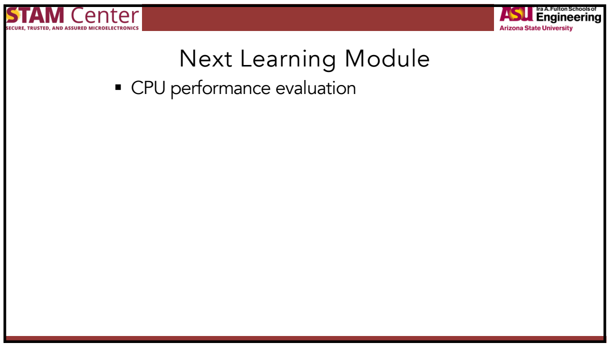
25



Why a program may have CPI >1

- Why an Instruction may not be dispatched every cycle (CPI>1)?
  - Full bypassing may be too expensive to implement
    - Typically all frequently used paths are provided
    - Some infrequently used bypass paths may increase cycle time and counteract the benefit of reducing CPI
  - Loads have two cycle latency
    - Instruction after load cannot use load result
    - Some ISA define *load delay slots*, a software-visible pipeline hazard (compiler schedules independent instruction or inserts NOP to avoid hazard)

26



Next Learning Module

- CPU performance evaluation

27