CMPE 200
Computer Architecture & Design

Lecture 3. Processor Microarchitecture and Design (4)

Haonan Wang

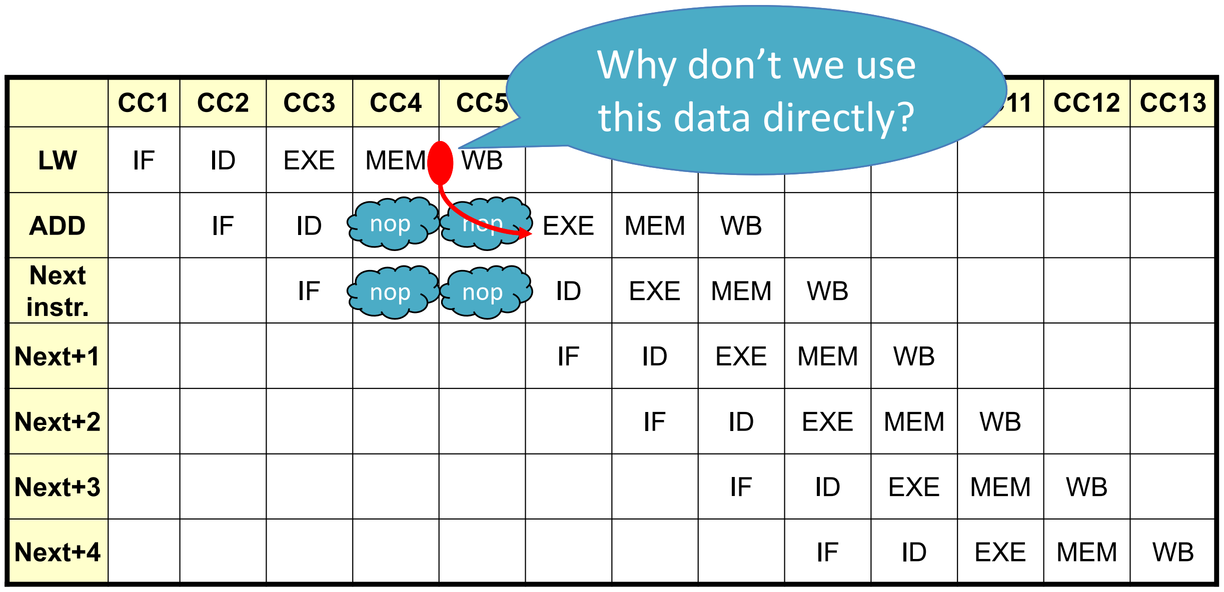
SJSU SAN JOSÉ STATE UNIVERSITY

Cause of Data Hazards

- **Stalling the pipeline to prevent error:**
  - All instructions in front of the stalled instruction can continue
  - All instructions behind the stalled instruction must also stall
- **Stalling inserts “bubbles” / nops (no-operations) into the pipeline**
  - A “nop” is an actual instruction in the MIPS ISA that does NOTHING

Pipeline Stall for Data Hazards

- **ADD can read the updated register value from the register file in the same cycle as LW writes the loaded data to register**
- **Do we really need to wait until LW updates the register file?**


Why don’t we use this data directly?

	CC1	CC2	CC3	CC4	CC5						11	CC12	CC13
LW	IF	ID	EXE	MEM	WB								
ADD		IF	ID	nop	nop	EXE	MEM	WB					
Next instr.			IF	nop	nop	ID	EXE	MEM	WB				
Next+1						IF	ID	EXE	MEM	WB			
Next+2							IF	ID	EXE	MEM	WB		
Next+3								IF	ID	EXE	MEM	WB	
Next+4									IF	ID	EXE	MEM	WB

Data Hazards Solutions

- **Compiler**
  - Reorder Code

- **Hardware**
  - Use forwarding / bypassing

- **Data forwarding**
  - Take results still in the pipeline (not yet written back to a register) and pass them to dependent instructions

  - Forwarding Path
    - Load instruction: from WB to EXE
    - R-type instructions: from MEM to EXE

Data Forwarding

- **As far as earlier instruction's result is in pipeline, the result value can be forwarded to the following instructions**
- **In arithmetic operations, no stall is needed**

```
ADD $t3,$t1,$t2
SUB $t5,$t3,$t4
XOR $t7,$t5,$t3
```

	CC1	**CC2**	**CC3**	**CC4**	**CC5**	**CC6**	**CC7**	**CC8**	**CC9**	**CC10**
ADD	IF	ID	EXE	MEM	WB					
SUB		IF	ID	EXE	MEM	WB				
XOR			IF	ID	EXE	MEM	WB			
Next+1				IF	ID	EXE	MEM	WB		
Next+2					IF	ID	EXE	MEM	WB	
Next+3						IF	ID	EXE	MEM	WB

SJSU SAN JOSÉ STATE UNIVERSITY

Forwarding Logic

- **Forwarding from MEM**
  - if (**we_regM** and (**rsE** ≠ 0) and (**rf_waM** = **rsE**))
    ForwardAE = 10
  - if (**we_regM** and (**rtE** ≠ 0) and (**rf_waM** = **rtE**))
    ForwardBE = 10

rsE & rtE
: source register id in EXE stage
rf_waM
: destination register id in MEM stage
we_regM
: register file write enable in MEM stage

ADD $t3,$t1,$t2
SUB $t5,$t3,$t4
XOR $t7,$t5,$t3

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
ADD	IF	ID	EXE	MEM	WB					
SUB		IF	ID	EXE	MEM	WB				
XOR			IF	ID	EXE	MEM	WB			
Next+1				IF	ID	EXE	MEM	WB		
Next+2					IF	ID	EXE	MEM	WB	
Next+3						IF	ID	EXE	MEM	WB

Forwarding Logic

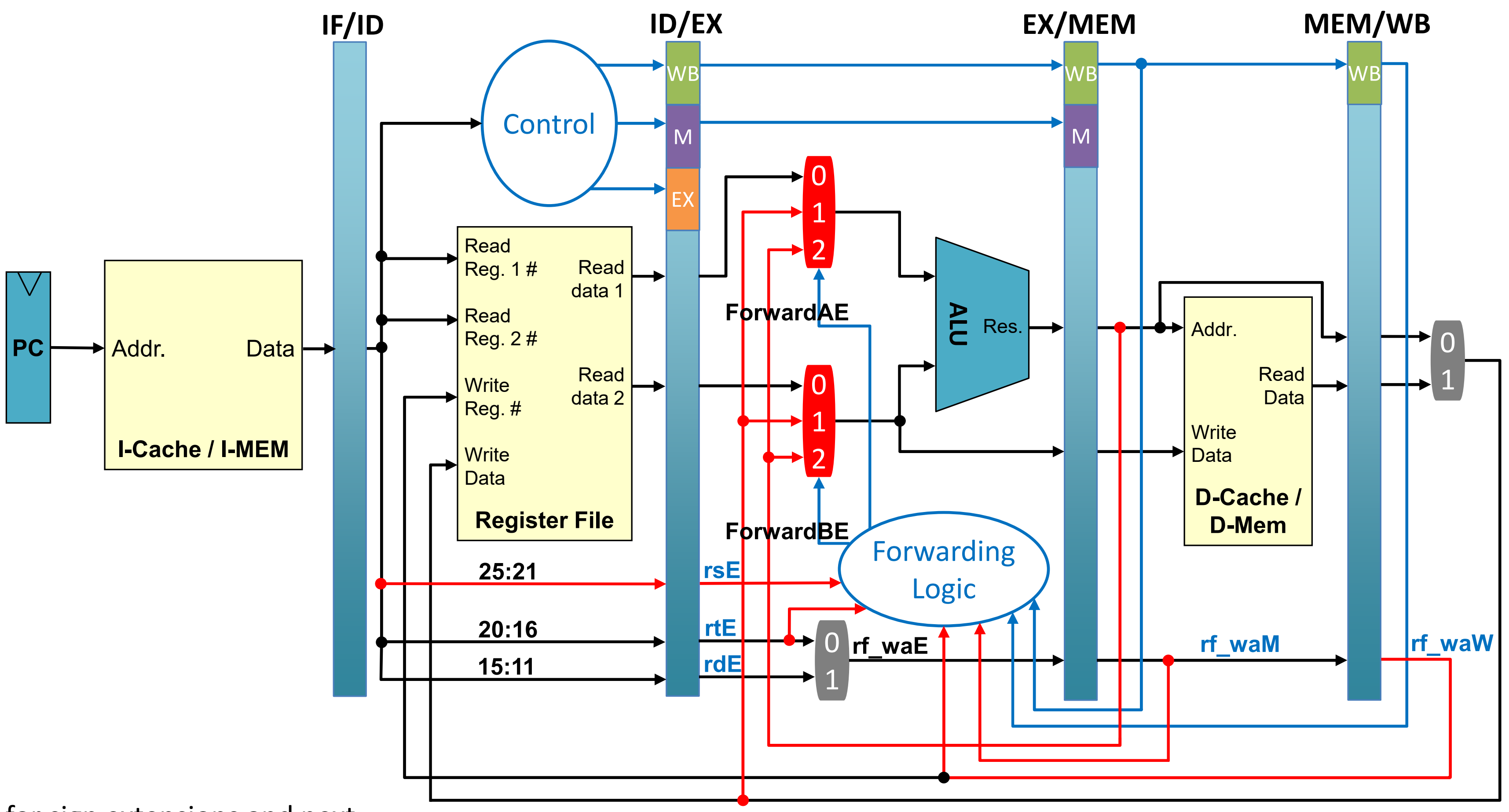
- **Forwarding from WB**
  - if (`we_regW` and (`rsE` ≠ 0) and (`rf_waW` = `rsE`))
    ForwardAE = 01
  - if (`we_regW` and (`rtE` ≠ 0) and (`rf_waW` = `rtE`))
    ForwardBE = 01

`rf_waW`
: destination register id in WB stage
`we_regW`
: register file write enable in WB stage

ADD $t3,$t1,$t2
SUB $t5,$t3,$t4
XOR $t7,$t5,$t3

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
ADD	IF	ID	EXE	MEM	WB					
SUB		IF	ID	EXE	MEM	WB				
XOR			IF	ID	EXE	MEM	WB			
Next+1				IF	ID	EXE	MEM	WB		
Next+2					IF	ID	EXE	MEM	WB	
Next+3						IF	ID	EXE	MEM	WB

Forwarding Path



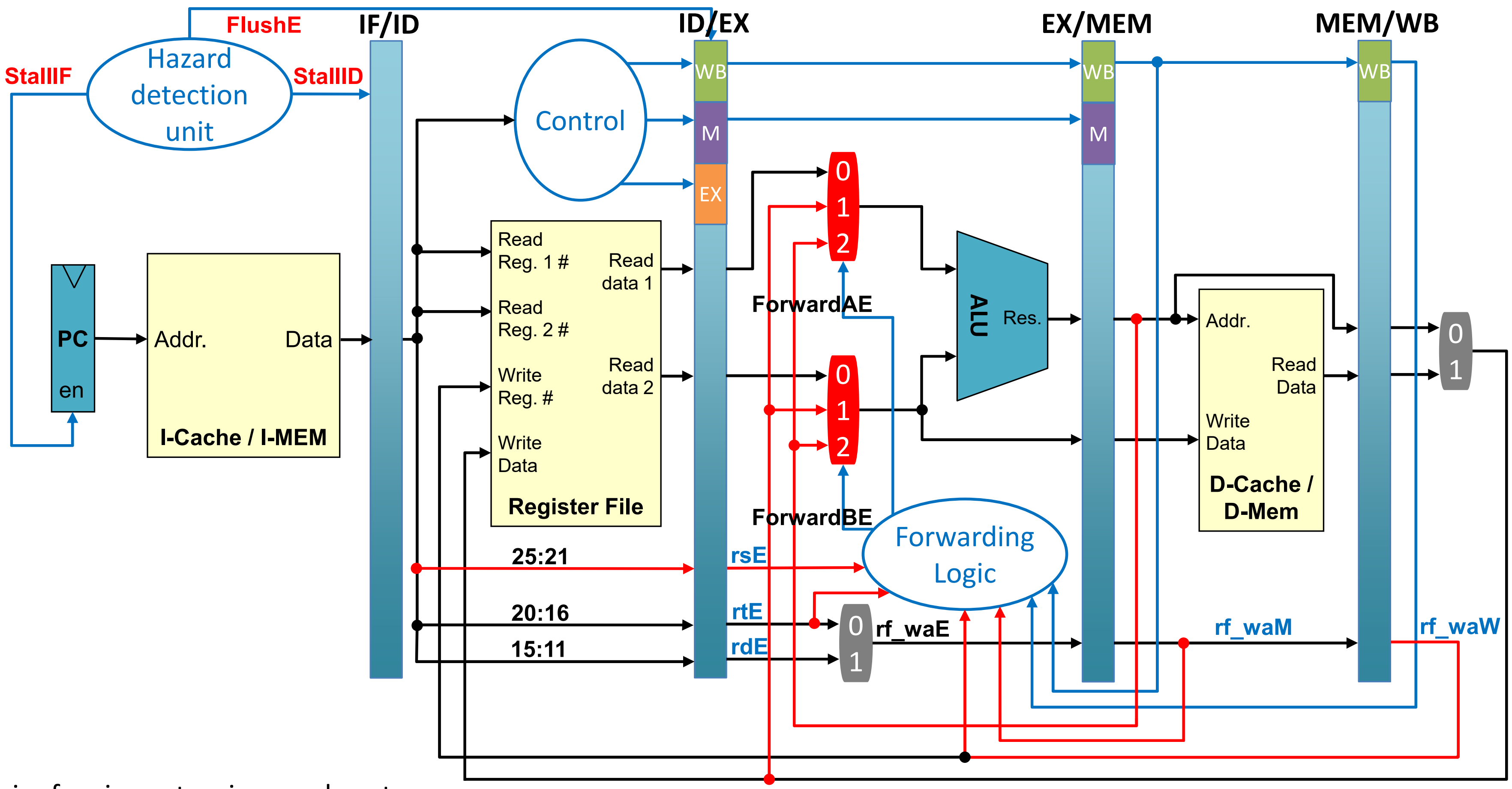
Note: Logics for sign extensions and next pc calculation are omitted for simplicity

Stall With Forwarding

- **For LW sourced dependency, we still need to stall for one cycle**
  - Freeze the PC and IF/ID pipeline register for one clock cycle
  - Flush the ID/EXE pipeline register

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC14
LW	IF	ID	EXE	MEM	WB								
ADD		IF	ID	nop	EXE	MEM	WB						
Next instr.			IF	nop	ID	EXE	MEM	WB					
Next+1					IF	ID	EXE	MEM	WB				
Next+2						IF	ID	EXE	MEM	WB			
Next+3							IF	ID	EXE	MEM	WB		
Next+4								IF	ID	EXE	MEM	WB	

Forwarding Path



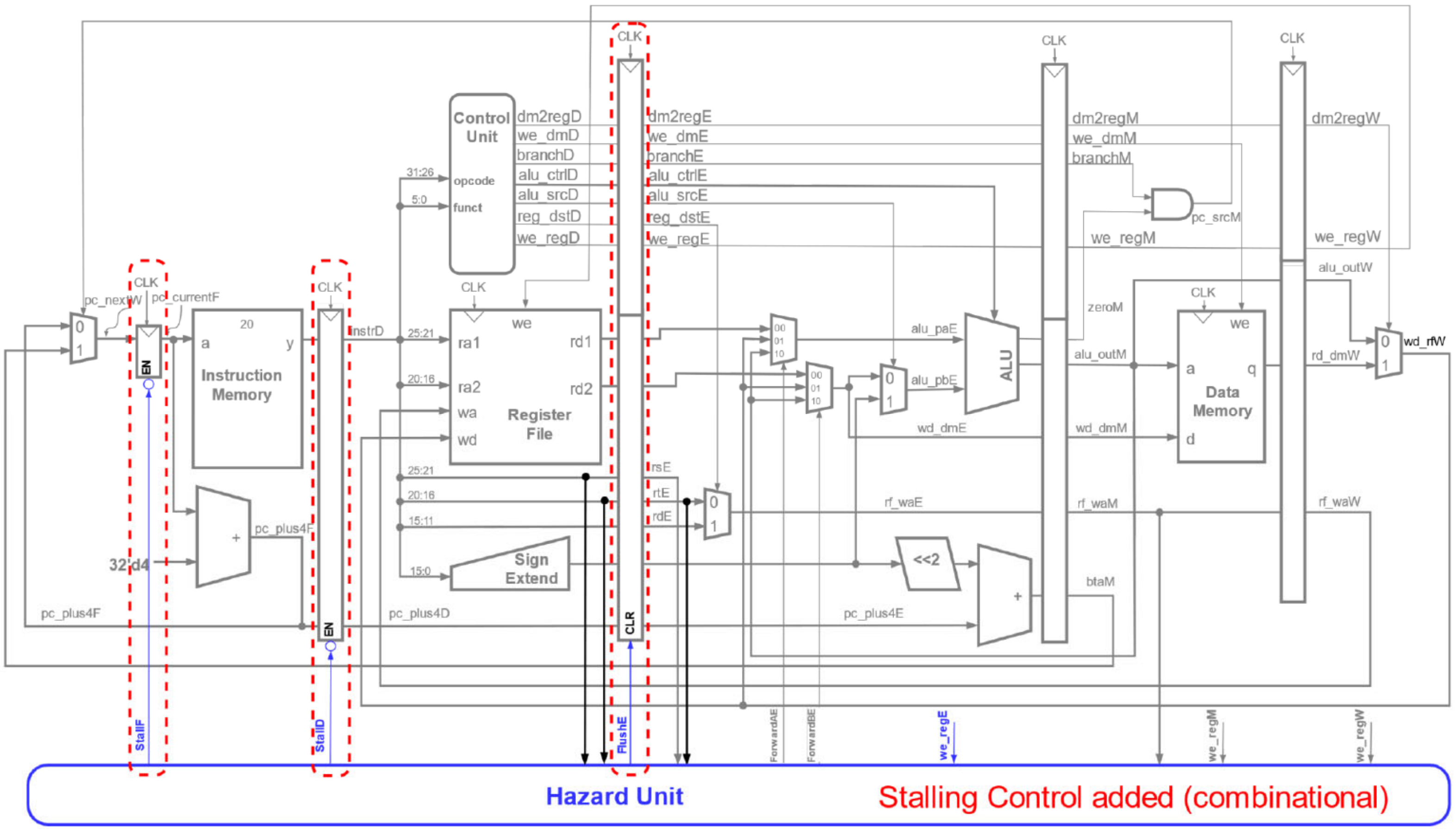
Note: Logics for sign extensions and next pc calculation are omitted for simplicity

Stalling Condition

- **Stalling Logic**
  - `lwstall` = ((`rsD` == `rtE`) or (`rtD` == `rtE`)) and `dm2regE`
  - `stallF = stallD = FlushE = lwstall`

`rsD` & **`rtD`**
: source register id in ID stage
`rtE`
: destination register in EXE stage
`dm2regE`
: data memory to register write signal of the instruction in EXE stage

Current Design



SJSU SAN JOSÉ STATE UNIVERSITY

Pipeline Latency with Forwarding/Stall

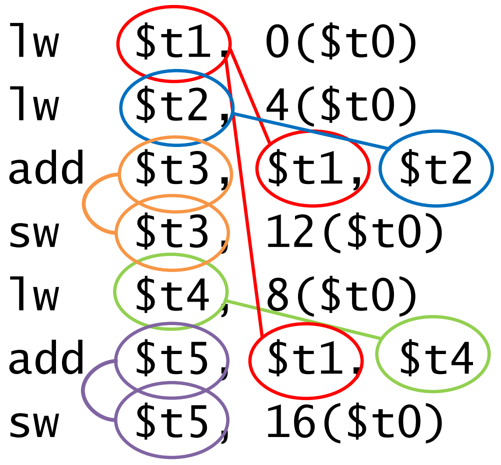
- **Number of cycles we must stall to execute a dependent instruction:**

Instruction that is the dependency source	w/o Forwarding (cycles)	w/ Forwarding (cycles)
LW	2	1
Other instructions that update register file	2	0

- **Data Hazards Solutions:**
  - Hardware: stalling & forwarding
  - Software: compiler -- code reordering

Code Scheduling to Avoid Stalls

- **Compiler can reorder code to avoid the stalls**
- **C code for** `A = B + E; C = B + F;`

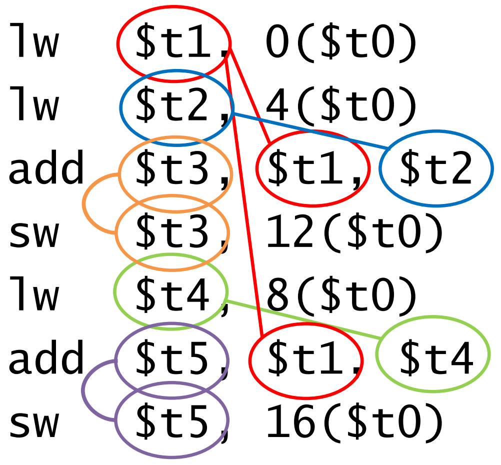


```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Find RAW dependencies and check if stall is needed

Code Scheduling to Avoid Stalls

- **Compiler can reorder code to avoid the stalls**
- **C code for** `A = B + E; C = B + F;`



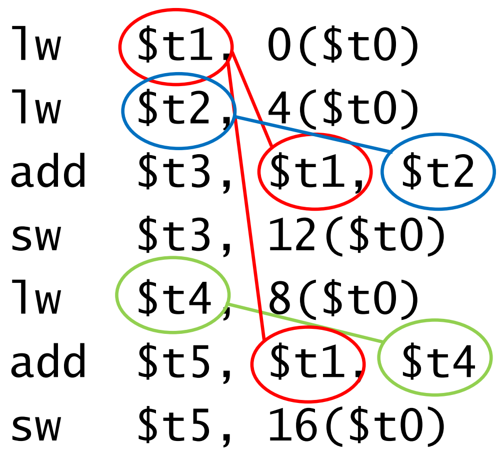
```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Find RAW dependencies and check if stall is needed

When forwarding is used, only lw instruction causes stall
→ Leave lw instructions only

Code Scheduling to Avoid Stalls

- **Compiler can reorder code to avoid the stalls**
- **C code for** `A = B + E; C = B + F;`



```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Find RAW dependencies and check if stall is needed

lw instructions cause 1-cycle stalls
→ Leave instructions that are executed back-to-back only

Code Scheduling to Avoid Stalls

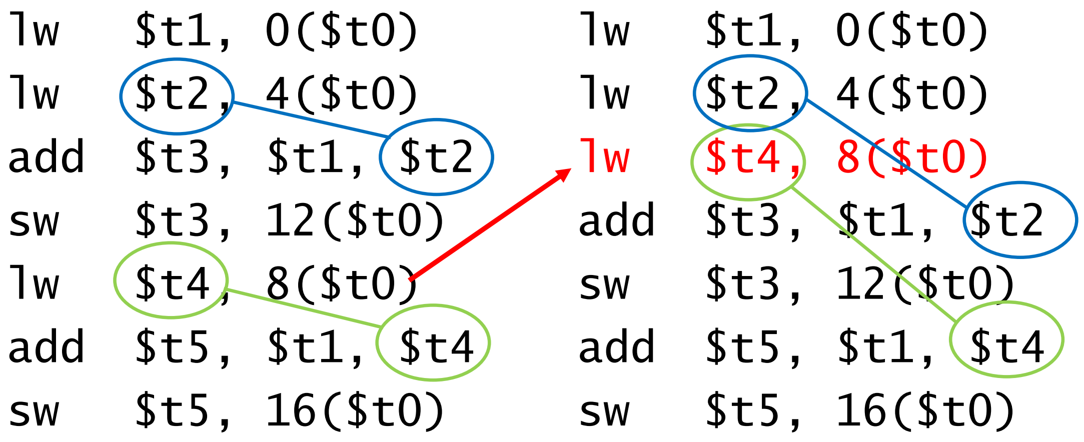
- **Compiler can reorder code to avoid the stalls**
- **C code for** `A = B + E; C = B + F;`

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Now, check if the lw and dependent instructions **must be** executed back-to-back

Code Scheduling to Avoid Stalls

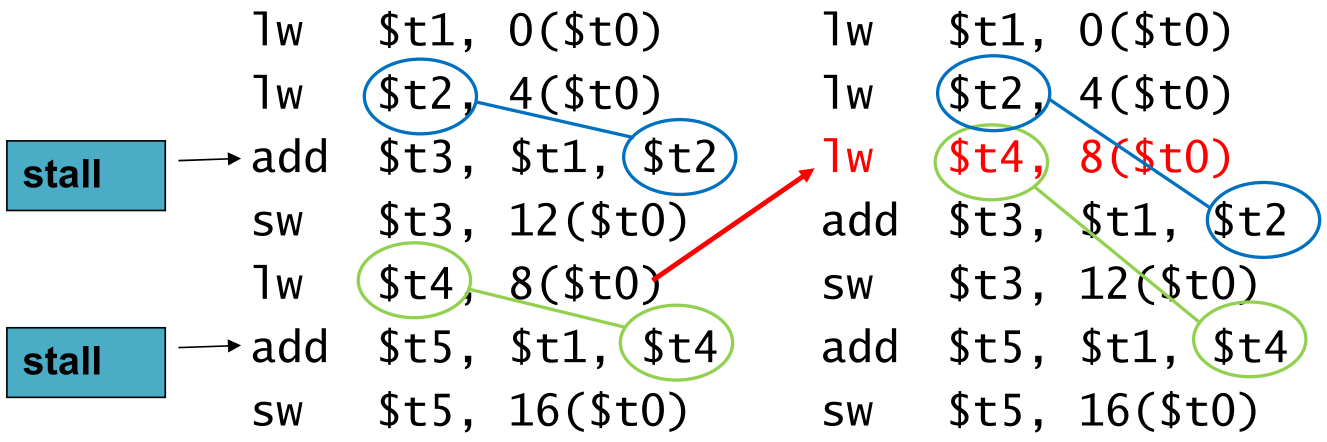
- **Compiler can reorder code to avoid the stalls**
- **C code for** `A = B + E; C = B + F;`



```
lw   $t1, 0($t0)         lw   $t1, 0($t0)
lw   $t2, 4($t0)         lw   $t2, 4($t0)
add  $t3, $t1, $t2       lw   $t4, 8($t0)
sw   $t3, 12($t0)        add  $t3, $t1, $t2
lw   $t4, 8($t0)         sw   $t3, 12($t0)
add  $t5, $t1, $t4       add  $t5, $t1, $t4
sw   $t5, 16($t0)        sw   $t5, 16($t0)
```

Code Scheduling to Avoid Stalls

- **Compiler can reorder code to avoid the stalls**
- **C code for `A = B + E; C = B + F;`**



```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Total execution time

5 cycles for the first inst +
6 cycles for the remaining insts +
2 x 1 stall cycles = 13 cycles

5 cycles for the first inst +
6 cycles for the remaining insts +
0 stall cycles = 11 cycles

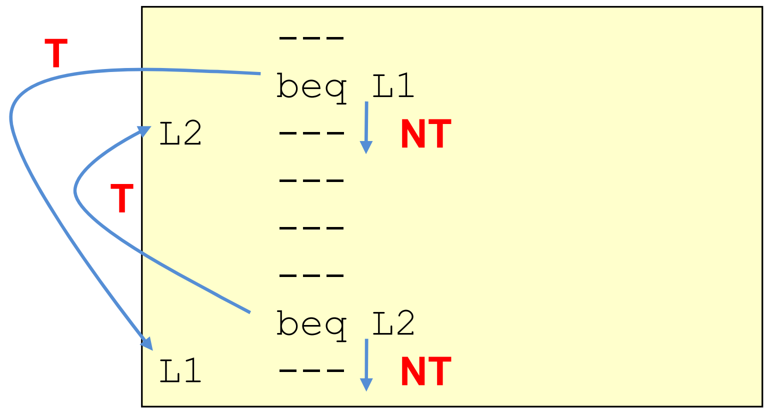
SJSU SAN JOSÉ STATE UNIVERSITY

Control Hazard

- **Branch outcomes: Taken (T) or Not-Taken (NT)**

- **Not known until late in the pipeline**
  - Prevents us from fetching future instructions
  - Solution: rather than stall, we can predict the outcome and keep fetching
  - We only need to correct the pipeline if we guess wrong

- **Static Predictions**
  - Predict Taken
  - Predict Not Taken

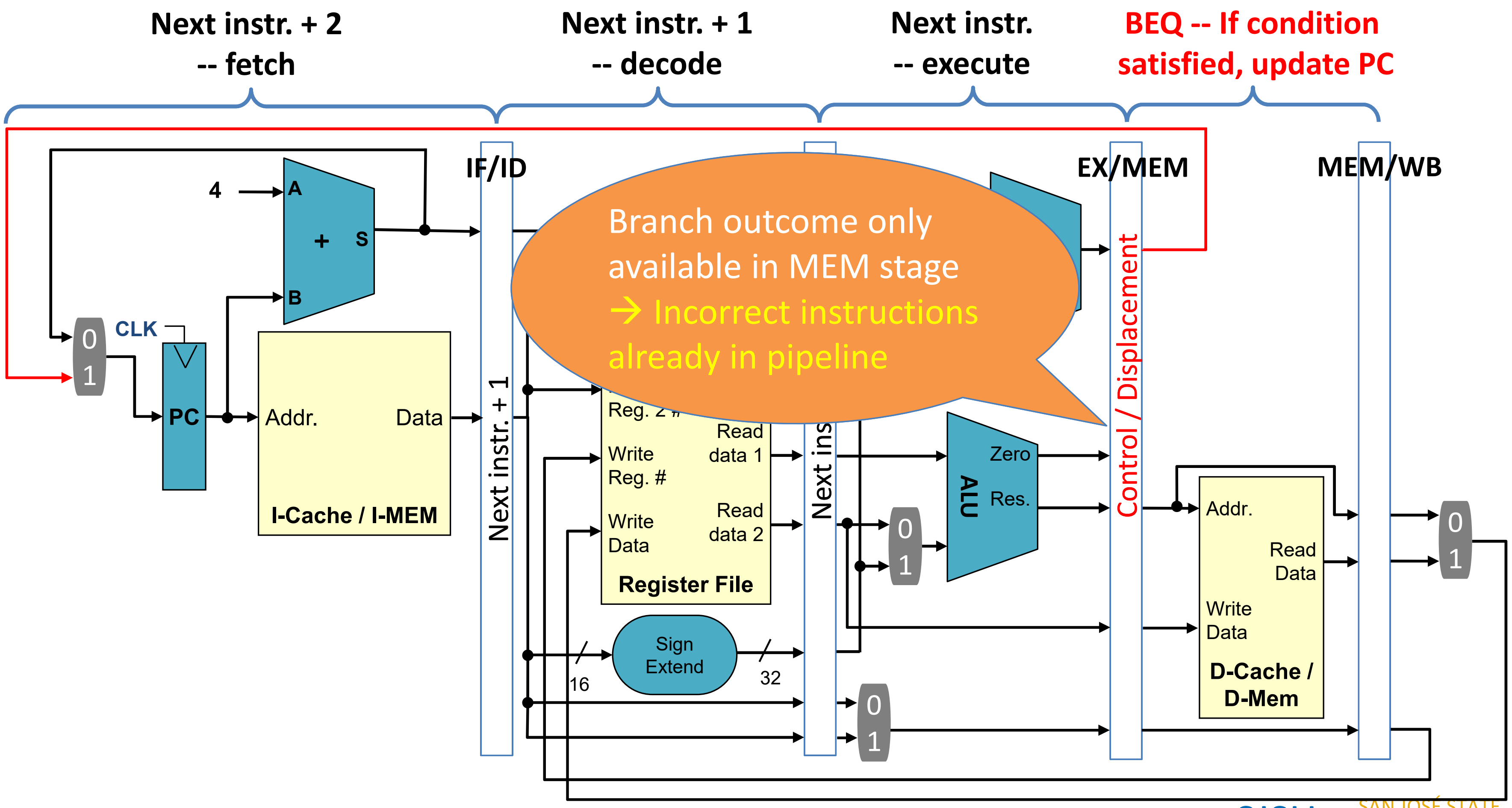


```
      ---
      beq L1
L2    ---
      ---
      ---
      ---
      beq L2
L1    ---
```

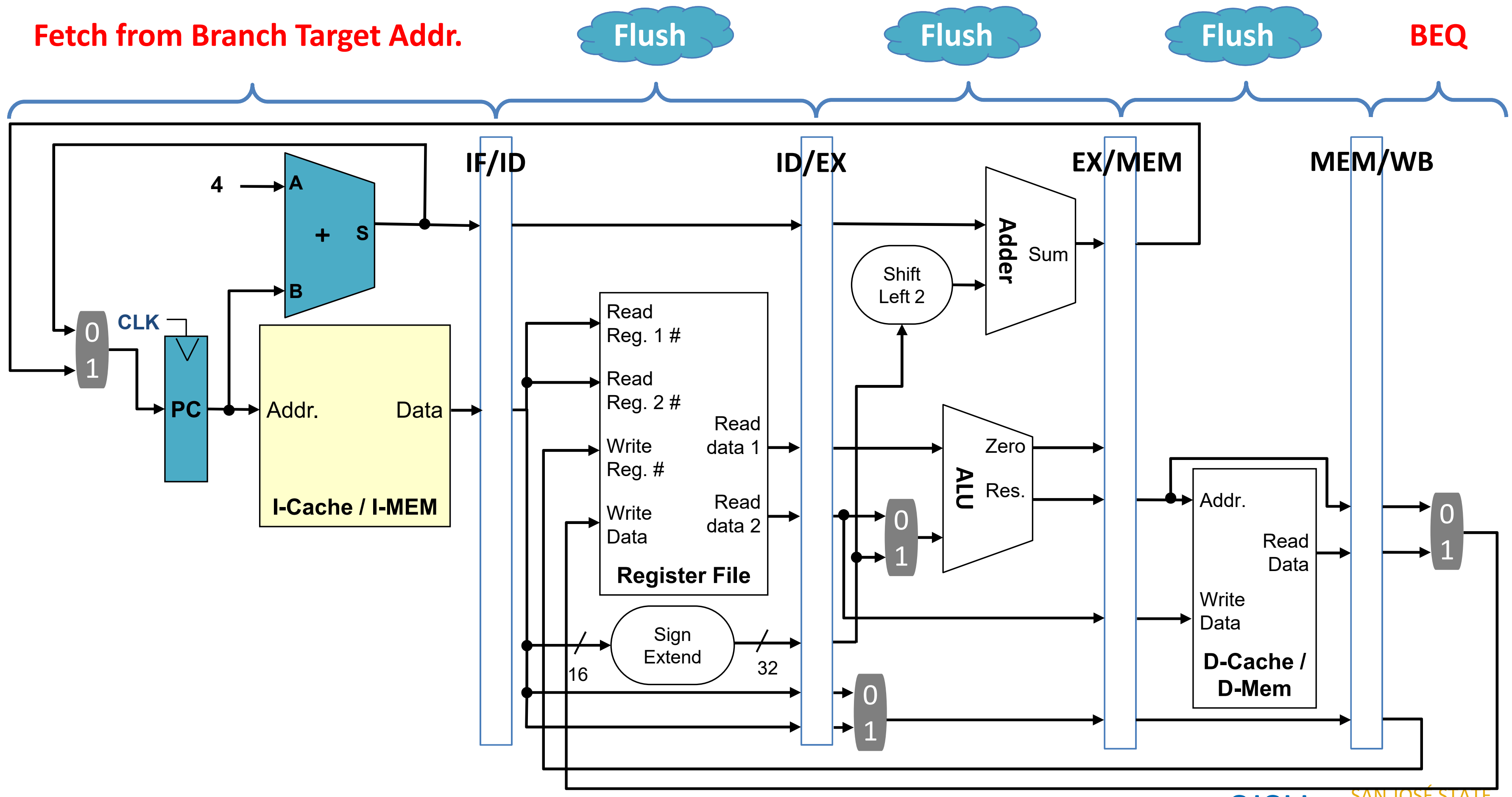
Statically Predict Not Taken

- Keep fetching instructions from the PC + 4 by assuming that the branch will be not taken
- Once branch is turned out to be taken, flush the incorrectly fetched instructions
- In the following cycle, fetch the instruction from the branch target address

Statically Predict Not Taken: Issues



Statically Predict Not Taken: Issues



Static Branch Prediction Penalty

- **Penalty = number of instructions that need to be flushed on misprediction**

- **For static branch prediction:**
  - The branch outcome and target address is available at the MEM stage and passed back to the Fetch stage
  - If mispredicted, instructions of the correct path will be fetched on the next cycle

- **A 3-cycle branch penalty when mispredicted**

Statically Predict Not Taken

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

NT!

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM						
ADD		IF	ID	EXE						
SUB			IF	ID						
OR				IF						

Statically Predict Not Taken

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM					
SUB			IF	ID	EXE					
OR				IF	ID					
BNE					IF					

SJSU SAN JOSÉ STATE UNIVERSITY

Statically Predict Not Taken

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM				
OR				IF	ID	EXE				
BNE					IF	ID				
AND						IF				

Statically Predict Not Taken

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM	WB			
OR				IF	ID	EXE	MEM	WB		
BNE					IF	ID	EXE	MEM	T!	
AND						IF	ID	EXE		
SW							IF	ID		
LW								IF		

Flush

Statically Predict Not Taken

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

Instruction in the target address (L2)

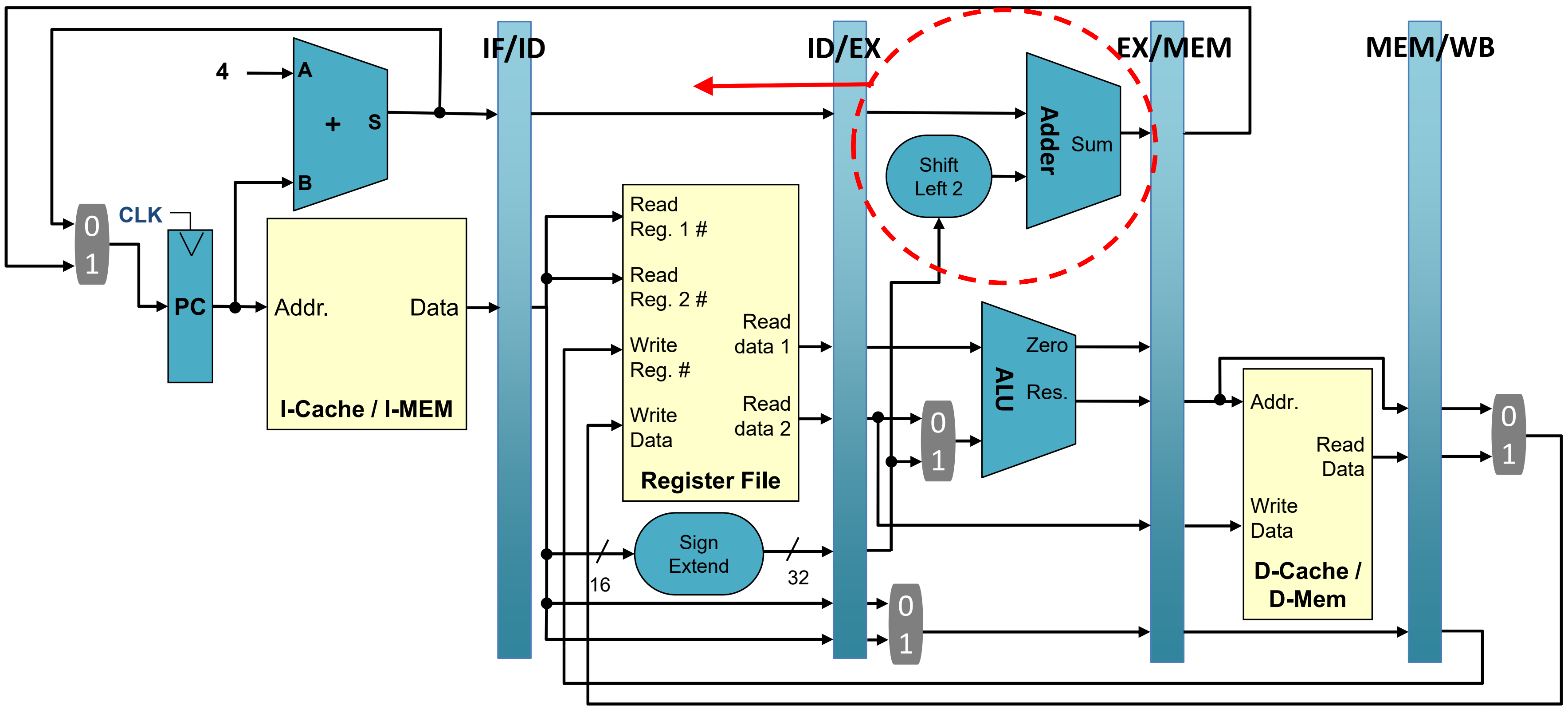
	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM	WB			
OR				IF	ID	EXE	MEM	WB		
BNE					IF	ID	EXE	MEM	WB	
AND						IF	ID	EXE	nop	nop
SW							IF	ID	nop	nop
LW								IF	nop	nop
ADD									IF	ID
SUB										IF

Predict Taken?

- **It is not possible to statically predict all the branches to be taken**
  - The branch target address is not computed until the EX stage, so we don’t have the PC values even needed to predict taken
- **Dynamic branch prediction + branch target buffer enables to predict taken**
  - Predict untaken a few times to collect branch target address of each branch and then predict taken based on the history
- **Can we reduce branch penalty with change of datapath?**

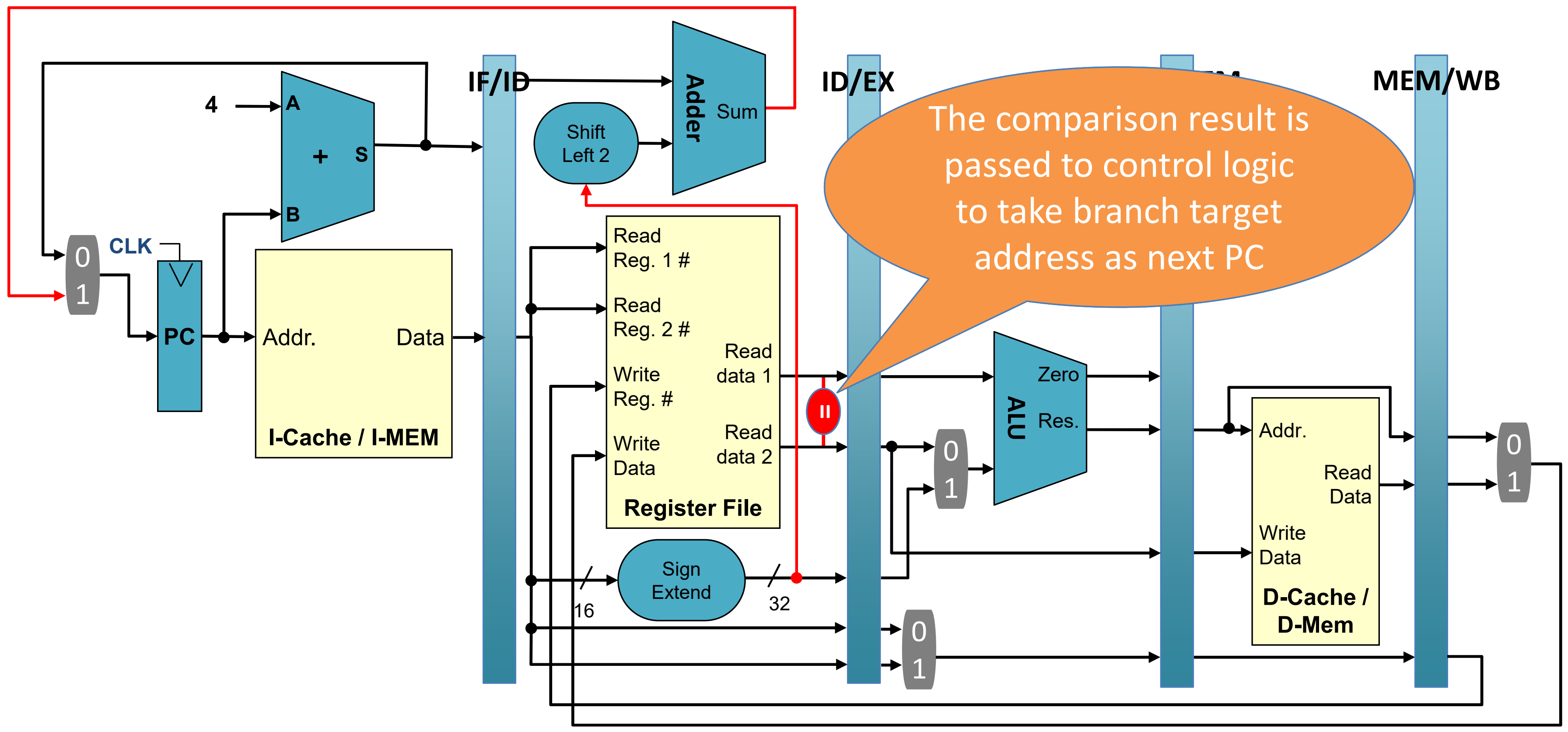
Early Branch Determination

- Target address can be calculated earlier by moving shift-left-2 and Adder



Early Branch Determination

- Target address can be calculated earlier by moving shift-left-2 and Adder



SJSU SAN JOSÉ STATE UNIVERSITY

Early Determination w/ Predict NT

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

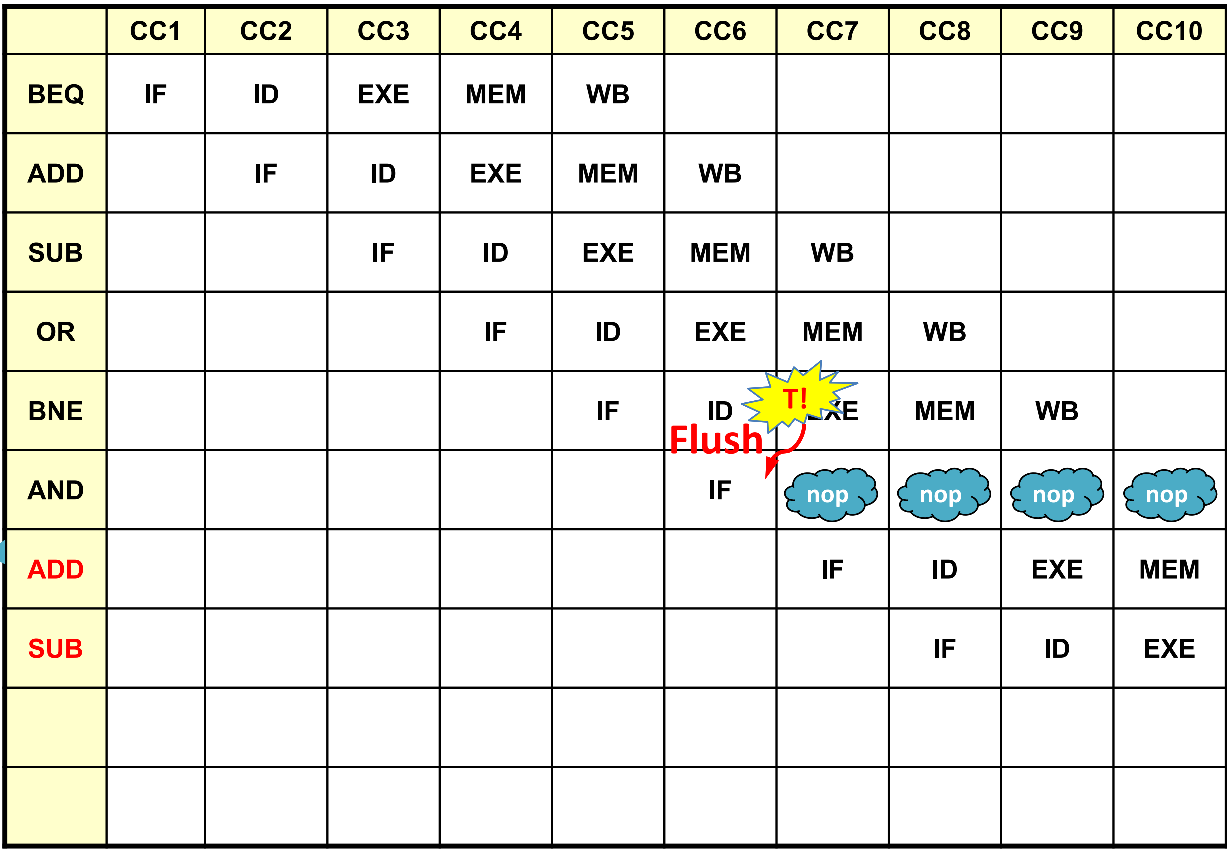
	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM				
OR				IF	ID	EXE				
BNE					IF	ID T! Flush				
AND						IF				

Early Determination w/ Predict NT

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

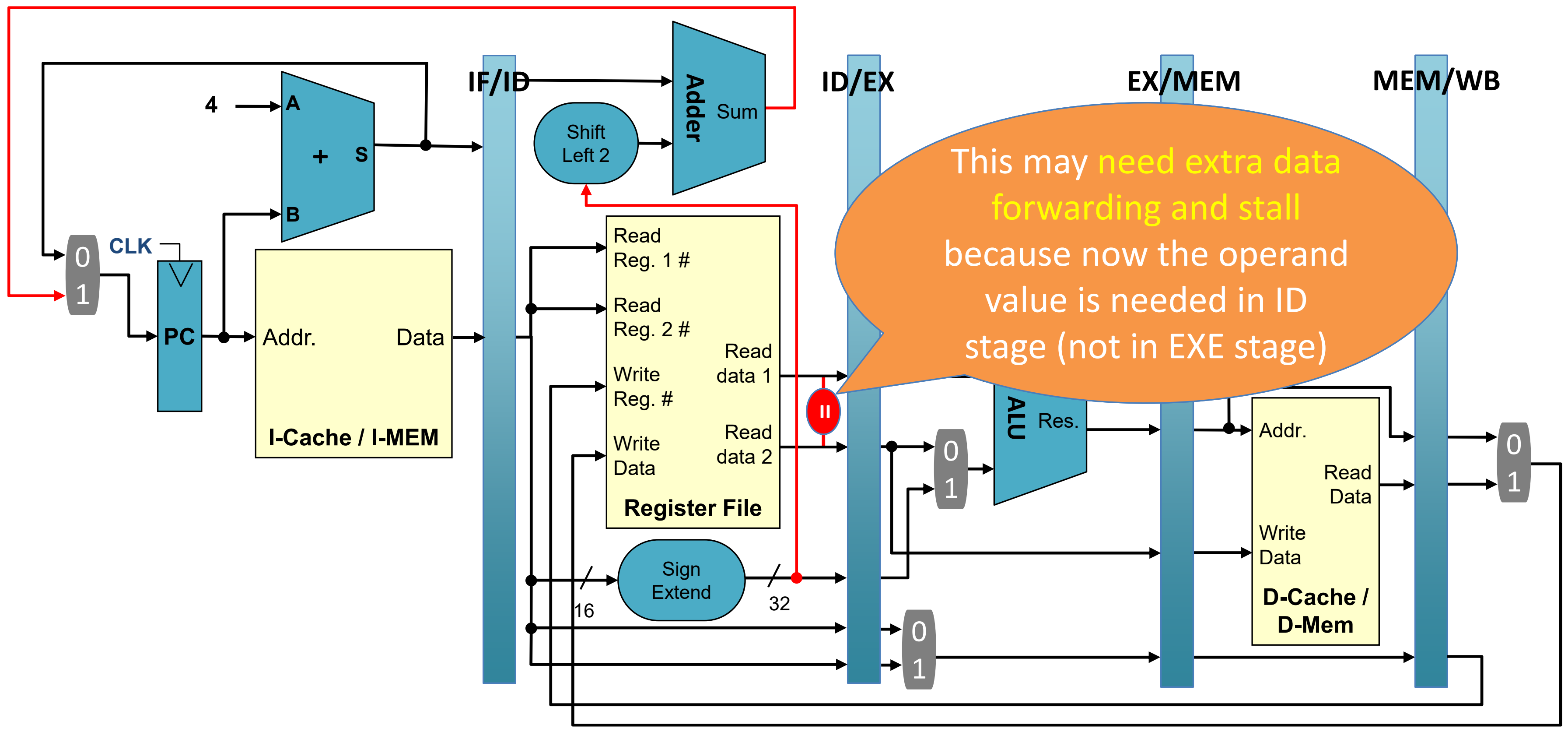
Instruction in the target address (L2)



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM	WB			
OR				IF	ID	EXE	MEM	WB		
BNE					IF	ID	EXE	MEM	WB	
AND						IF	nop	nop	nop	nop
ADD							IF	ID	EXE	MEM
SUB								IF	ID	EXE

Early Branch Determination: Issues

- **Target address can be calculated earlier by moving shift-left-2 and Adder**



Early Determination w/ Predict NT

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $s0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM				
OR				IF	ID	EXE				
BNE					IF	ID				
AND						IF				

Early Determination w/ Predict NT

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $s0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	MEM	WB				
SUB			IF	ID	EXE	MEM	WB			
OR				IF	ID	EXE	MEM			
BNE					IF	ID	ID T!			
AND						IF	IF			

SJSU SAN JOSÉ STATE UNIVERSITY

Early Determination w/ Predict NT

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $s0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

Instruction in the target address (L2)

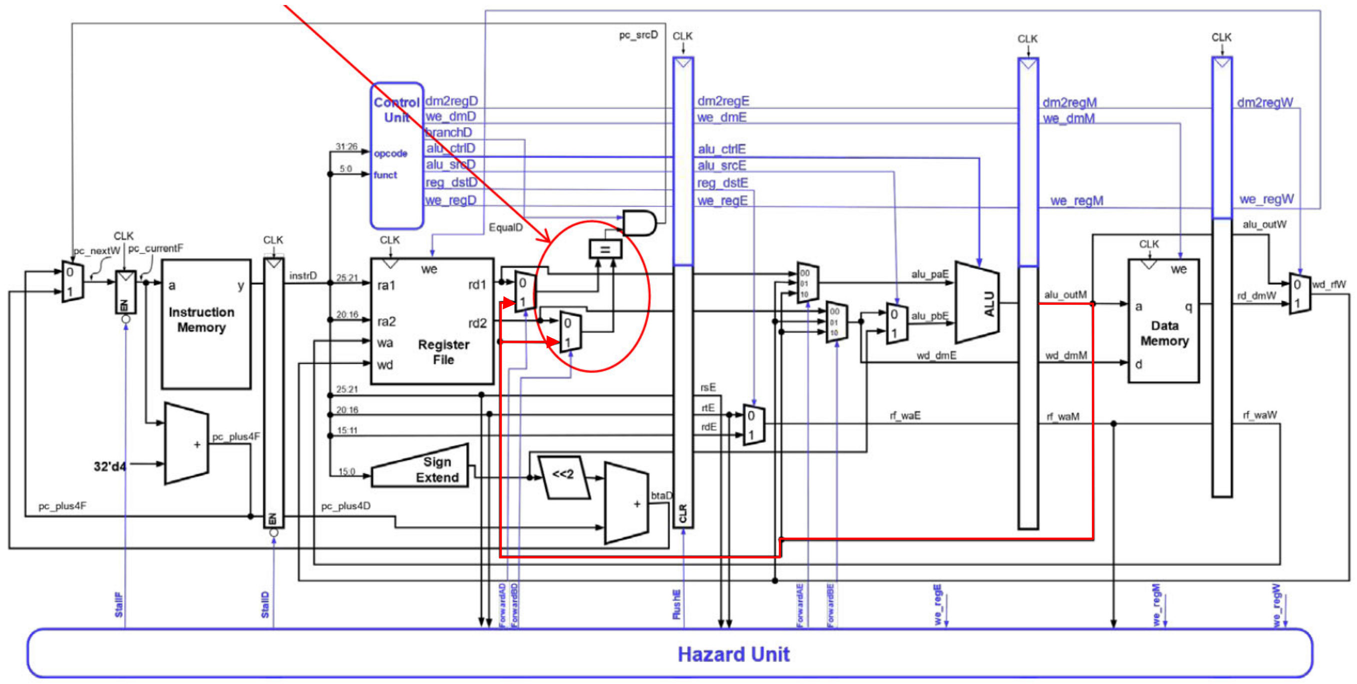
	CC1	CC2	CC3							0
BEQ	IF	ID	EXE							
ADD		IF	ID	E						
SUB			IF	ID	EXE	M				
OR				IF	ID	EXE		WB		
BNE					IF	ID	ID	T! XE	MEM	WB
AND						IF	IF	nop	nop	nop
ADD								IF	ID	EXE
SUB									IF	ID

If BNE has dependency with its preceding instruction, OR, one cycle stall is required to get the OR's result value

→ Still better than normal 3 cycle flushing

Early Branch Determination

- **Forwarding logic for early branch determination**
  - `ForwardAD = branchD` and `we_regM` and (`rsD` ≠ 0) and (`rf_waM` == `rsD`)
  - `ForwardBD = branchD` and `we_regM` and (`rtD` ≠ 0) and (`rf_waM` == `rtD`)



SJSU SAN JOSÉ STATE UNIVERSITY

Early Branch Determination

- **Stalling logic for early branch determination**
  - `branchstall` =
    `branchD` and `we_regE` and ((`rf_waE` == `rsD`) or (`rf_waE` == `rtD`))
    or
    `branchD` and `dm2regM` and ((`rf_waM` == `rsD`) or (`rf_waM` == `rtD`))
  - `StallF = StallD = FlushE = lwstall OR branchstall`

Branch Delay Slot

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    OR   $s0,$t6,$t7
    BNE  $a0,$s1,L2  (T)
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

Instruction in the target address (L2)

Find an instruction that
1) has to be executed regardless branch outcome
2) independent to branch

The instruction will be placed in the branch delay slot (right after branch instruction) instead of fetching wrong instruction and flushing

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
				MEM	WB					
					MEM	WB				
					EXE	MEM	WB			
OR				IF	ID	EXE	MEM	WB		
BNE					IF	ID Flush	T! EXE	MEM	WB	
AND						IF	nop	nop	nop	nop
							IF	ID	EXE	MEM
								IF	ID	EXE

Branch Delay Slot

Actual Branch Outcome

```
    BEQ  $a0,$a1,L1  (NT)
L2: ADD  $s1,$t1,$t2
    SUB  $t3,$t0,$s0
    BNE  $a0,$s1,L2  (T)
    OR   $s0,$t6,$t7
L1: AND  $t3,$t6,$t7
    SW   $t5,0($s1)
    LW   $s2,0($s5)
```

Instruction in the target address (L2)

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
BEQ	IF	ID	EXE	MEM	WB					
ADD		IF	ID	EXE	ME					
SUB			IF	ID	EXE	MEM	WB			
BNE				IF	ID	EXE	MEM	WB		
OR					IF	ID	EXE	MEM	WB	
ADD						IF	ID	EXE	MEM	WB
SUB							IF	ID	EXE	MEM

No Flush & Better Performance

SJSU SAN JOSÉ STATE UNIVERSITY

Branch Delay Slot

- **Require implementation on both the CPU and the compiler**
  - If no suitable instruction can be found for the slot, then a NOP will be filled

- **Branch delay slot is more beneficial for unconditional branches such as `j`, `jal`, `jr` because they are always taken**

- **For example, if `jal` uses branch delay slot,**
  - After `jal`, `$ra` is updated with PC + 8
  - The instruction in PC + 4 is executed after jal before entering the subfunction

Does branch delay slot make branch prediction useless?

Conclusion Time

What is the use of ForwardAE and ForwardBE Muxes?

- **Selecting ALU input from different stages**

How many cycles can be saved from data forwarding?

- **1 or 2**

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY