

Principles of Computer Architecture

CSE 240A

Fall 2024

Hadi Esmaeilzadeh

hadi@ucsd.edu

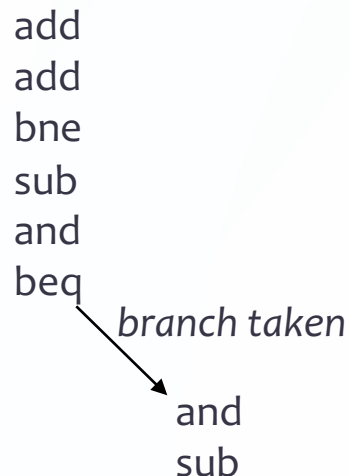
University of California, San Diego



Pipeline Hazards

Control Hazards

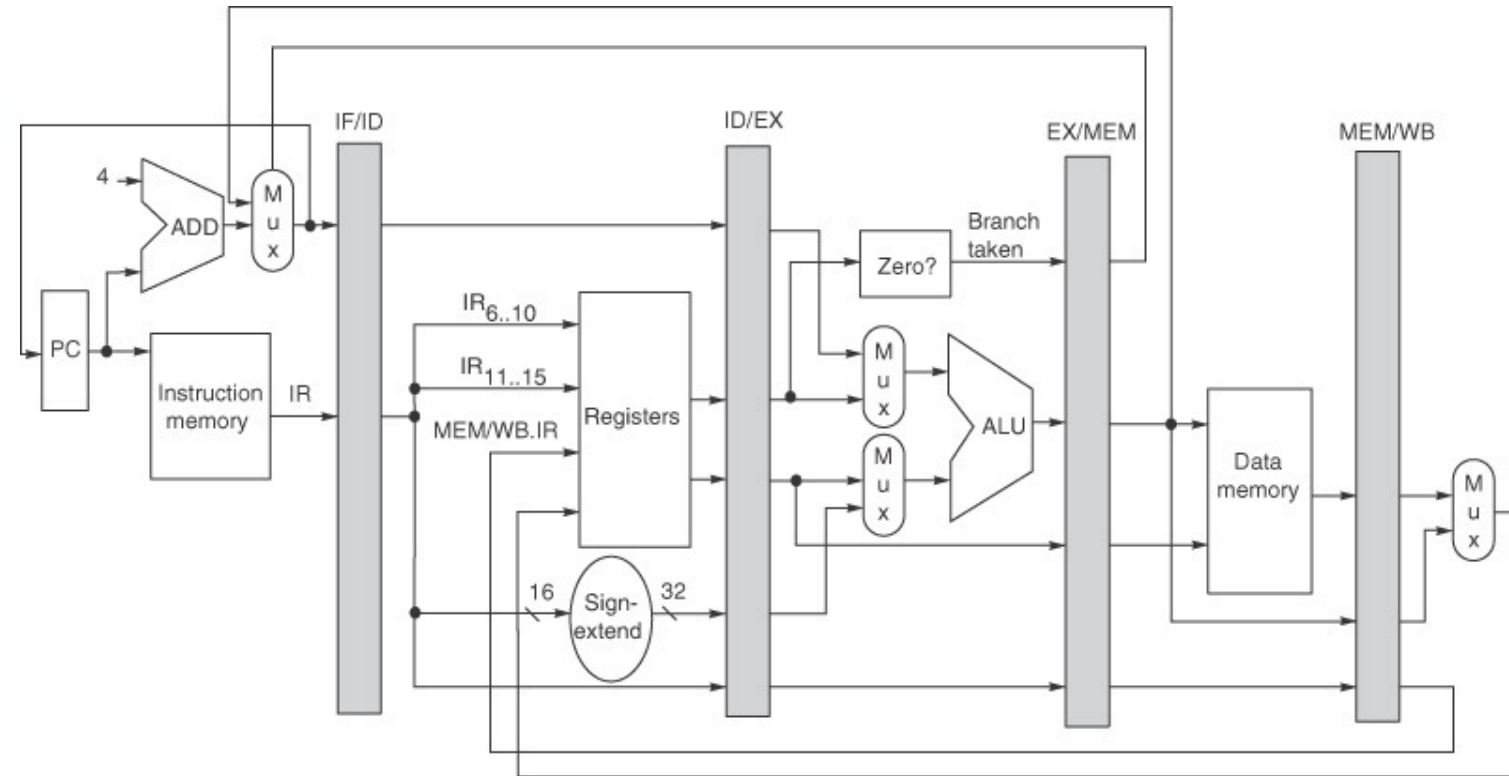
- Result from *branch* or *control* _____
- Instructions are not only dependent on instructions that produce their operands, but also on all previous control flow (branch, jump) instructions that lead to that instruction.



Branch or control dependence
may lead to
Branch or control hazard

Stalling the pipeline

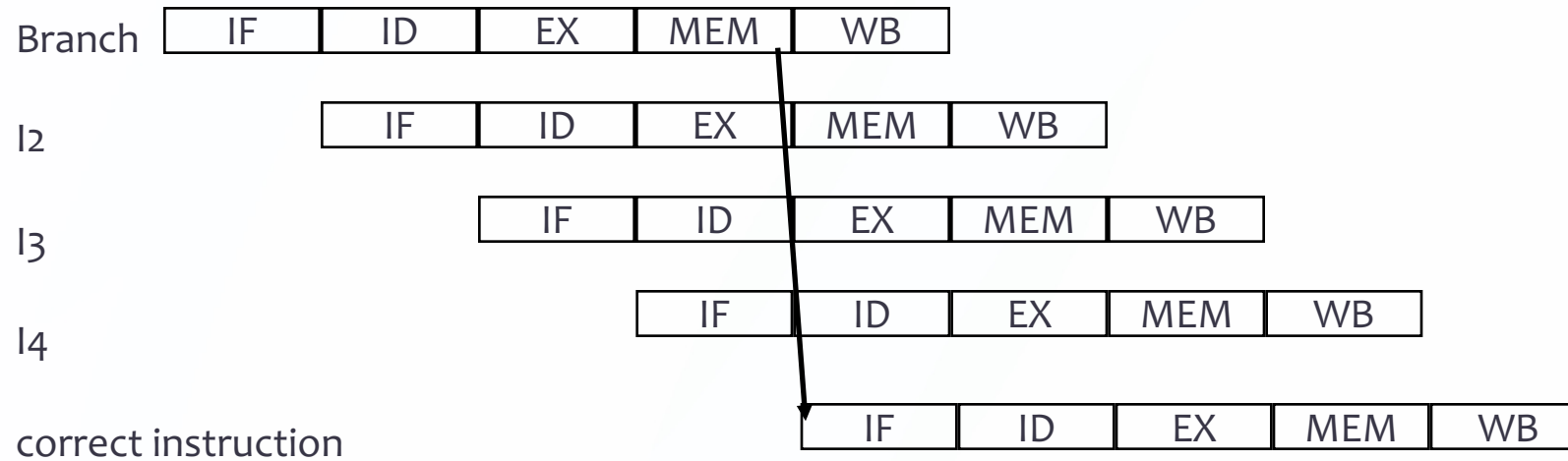
Given our current pipeline – let's assume we stall until we know the branch outcome. How many cycles will you lose per branch?



© 2007 Elsevier, Inc. All rights reserved.

Selection	cycles
A	0
B	1
C	2
D	3
E	4

Branch Hazards



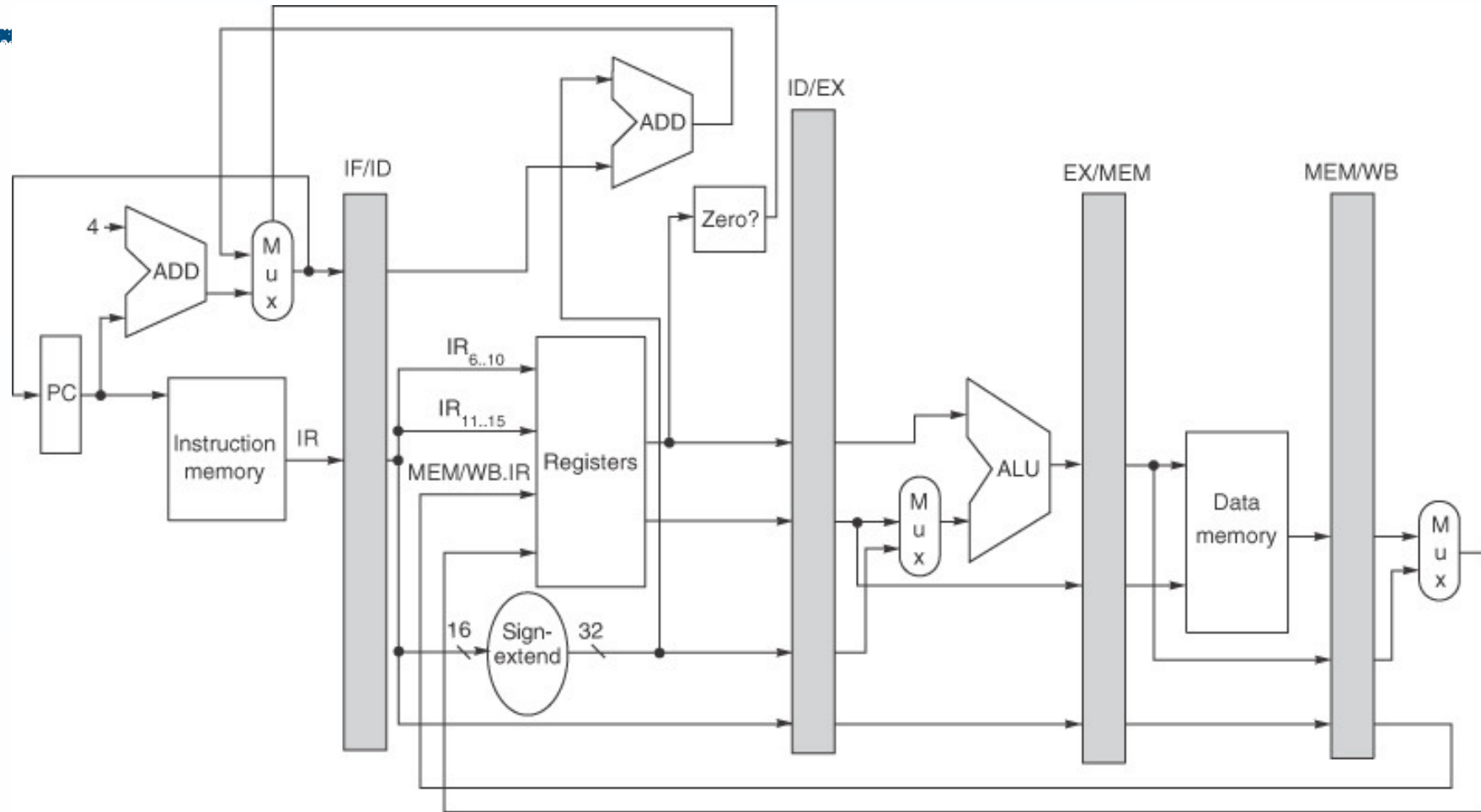
Branch Stall Impact

If CPI = 1, 30% branch, Stall 3 cycles => new CPI = ????

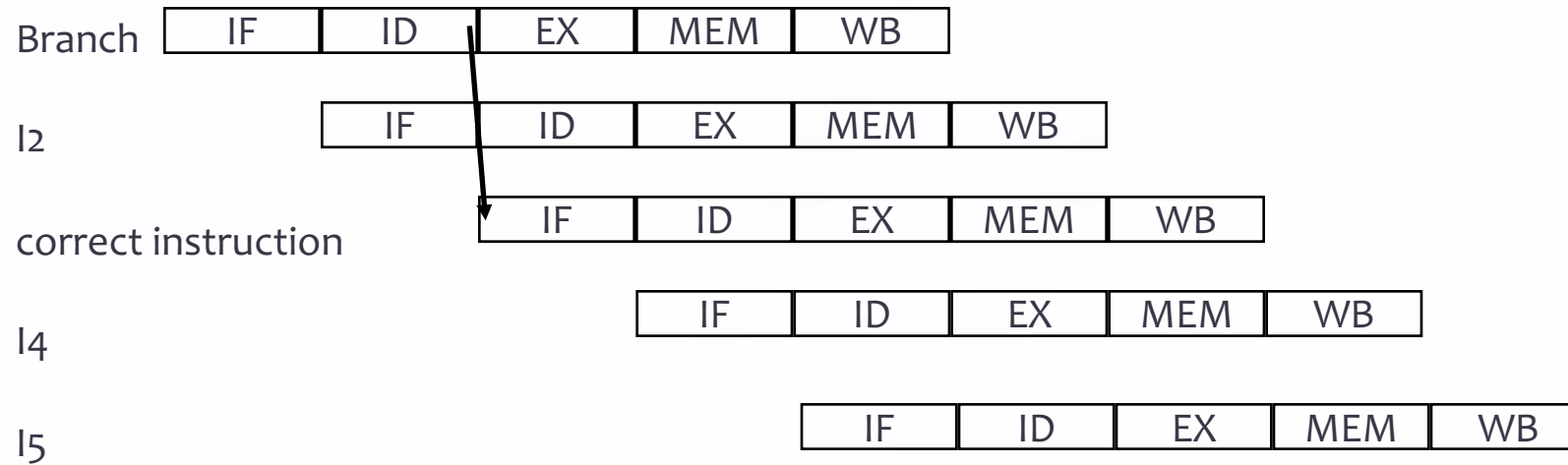
Branch Stall Impact

- If $CPI = 1$, 30% branch, Stall 3 cycles \Rightarrow new $CPI = \text{????}$
- Two part solution:
 - Determine branch taken or not sooner, AND
 - Compute taken branch address earlier
- (limited MIPS) branch tests if register = 0 or $\neq 0$
- MIPS Solution:
 - Move Zero test to ID/RF stage
 - Adder to calculate new PC in ID/RF stage
 - 1 clock cycle penalty for branch versus 3

New Datapath



Branch Hazards



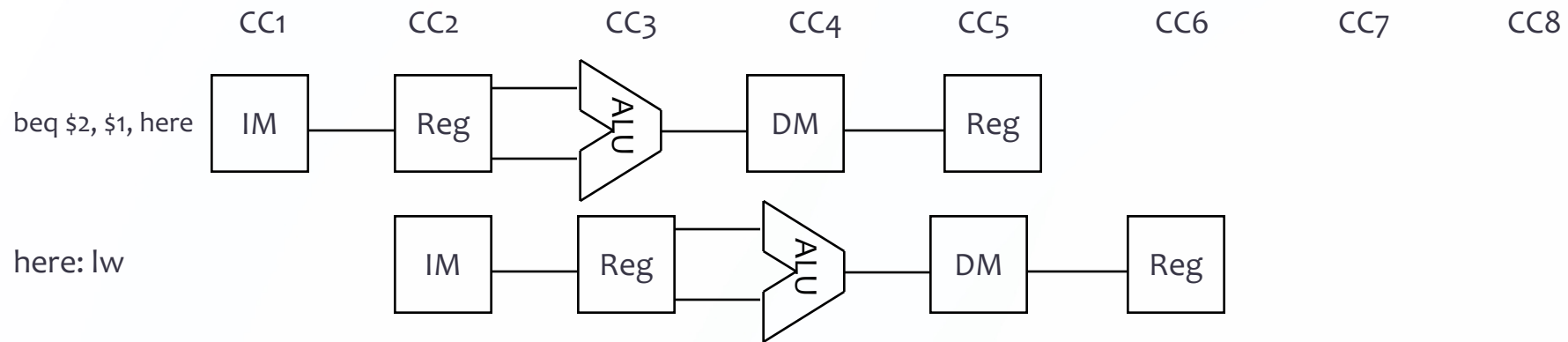
Still a branch hazard!!

What We Know About Branches

- more conditional branches than unconditional
- 67% of branches taken
- backward branches taken 80%

But is predicting taken as easy as NT

Branch Hazards – Predicting Taken



Required information to predict Taken:

1. An instruction is a branch before decode
2. The target of the branch
3. The outcome of the branch

Selection	Required knowledge
A	2,3
B	1,2,3
C	1,2
D	2
E	None of the above

Branch Target Buffer

- Keeps track of the PCs of recently seen branches and their targets.
- Consult during Fetch (in parallel with Instruction Memory read) to determine:
 - Is this a branch?
 - If so, what is the target
- We'll discuss this more...

Four Branch Hazard Alternatives

Stall until direction clear (branch resolved)

predict branch not taken

predict branch taken

Next: delayed branch

Fourth Branch Hazard Alternatives

-- Delayed Branch

- Define branch to take place *AFTER* a following instruction

```
branch instruction
  sequential successor1
  sequential successor2
  .....
  sequential successorn
branch target if taken
```



Branch delay of length n

- 1 slot delay allows proper decision and branch target address in 5 stage pipeline
- MIPS uses this

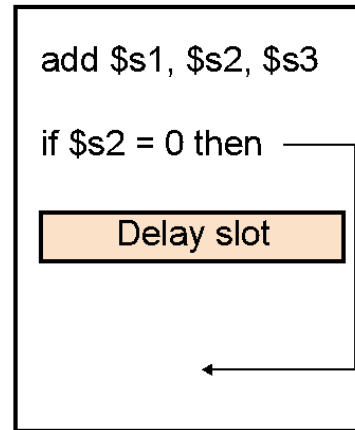
Delayed Branch

- Where to get instructions to fill branch delay slot?
 - Before branch instruction
 - From the target address: only valuable when branch taken
 - From fall through: only valuable when branch not taken
 - Cancelling branches allow more slots to be filled

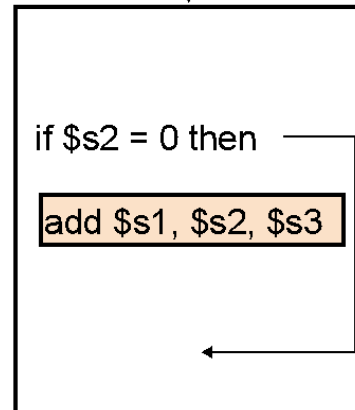
Filling the branch delay slot

- The branch delay slot is only useful if you can find something to put there.
- If you can't find anything, you must put a *noop* to insure correctness.

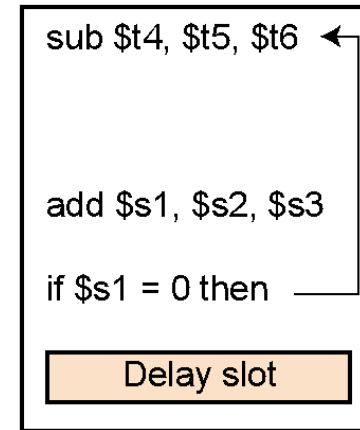
a. From before



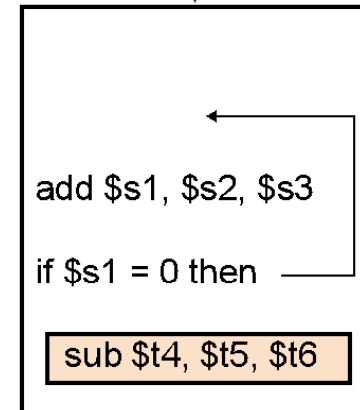
Becomes



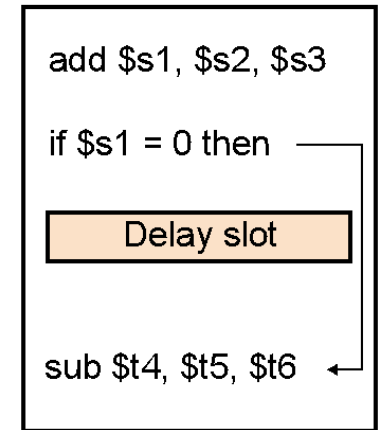
b. From target



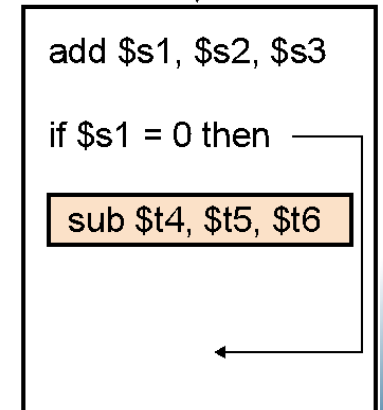
Becomes



c. From fall through



Becomes



Filling the branch delay slot

```
1 add $5, $3, $7
2 add $9, $1, $3
3 sub $6, $1, $4
4 and $7, $8, $2
5 beq $6, $7, there
  nop /* branch delay slot */
6 add $9, $1, $2
7 sub $2, $9, $5
  ...
  there:
8 mult $2, $10, $11
  ...
```

* It is not safe to assume anything about the ... code

Selection	Safe instructions
A	1,2
B	2,6
C	6,8
D	1,2,7,8
E	None of the above

Filling the branch delay slot

1	add \$5, \$3, \$7	No-R7 WAR
2	add \$9, \$1, \$3	Safe, \$1 and \$3 are fine
3	sub \$6, \$1, \$4	No-R6
4	and \$7, \$8, \$2	No-R7
5	beq \$6, \$7, there	
	nop /* branch delay slot */	
6	add \$9, \$1, \$2	Not safe (\$9 on not taken path)
7	sub \$2, \$9, \$5	Not safe (needs \$9 not yet produced)
	...	
	there:	
8	mult \$2, \$10, \$11	Not safe (\$2 is used before overwritten)
	...	E is the correct answer

* It is not safe to assume anything about the ... code

Selection	Safe instructions
A	1,2
B	2,6
C	6,8
D	1,2,7,8
E	None of the above

Delayed Branch

- Compiler effectiveness for single branch delay slot:
 - Fills about 60% of branch delay slots
 - About 80% of instructions executed in branch delay slots useful in computation
 - About 50% ($60\% \times 80\%$) of slots usefully filled

Key Points

- Pipeline improves throughput rather than latency
- Pipelining gets parallelism without replication
- $ET = IC * CPI * CT$
- Keeping the pipeline full is no easy task
 - structural hazards, data hazards, control hazards
- Data Hazards require dependent instructions to wait for the producer instruction
 - Most of the problem handled with forwarding (bypassing)
 - Sometimes stall still required (especially in modern processors)
- Control hazards require control-dependent (post-branch) instructions to wait for the branch to be resolved