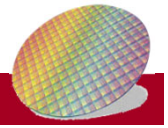
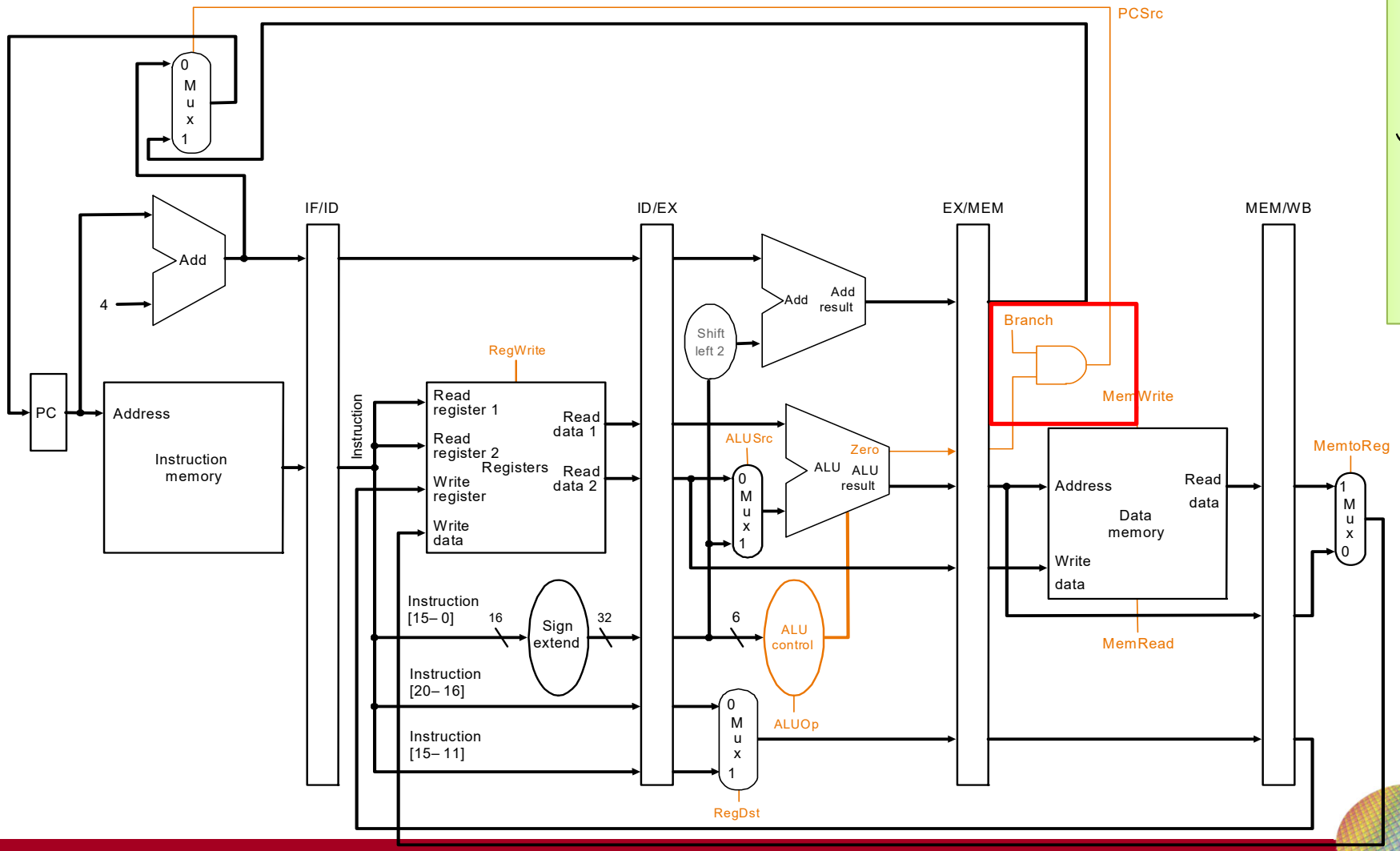
Outline

- A pipelined datapath
- Pipelined control
- Data hazards and forwarding
- Data hazards and stalls
- Branch (control) hazards



Which stage is the branch decision made?

Case 1: The MEM stage

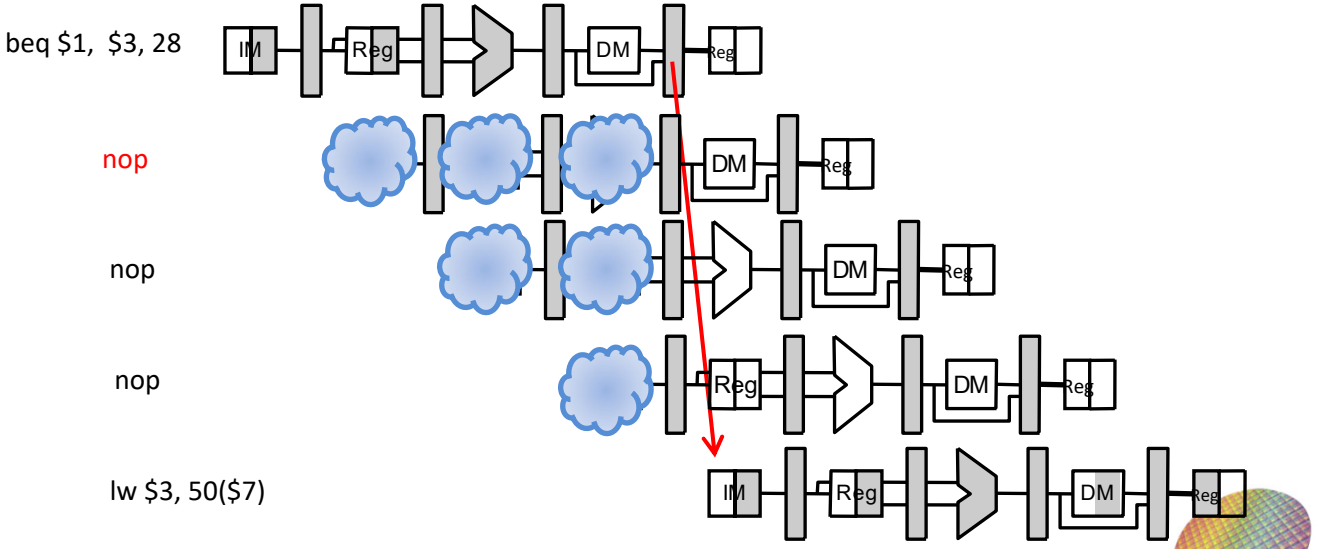


Control (or Branch) Hazards

- Branch decision is made in the MEM stage
  - Unable to determine the following instruction in pipeline immediately
  - Control hazard will occur
- *Stall* the pipeline utill branch decision is known
  - not efficient, slow the pipeline significantly!

if $1=$3 Branch decision is made in the MEM stage

```
40  beq $1,  $3, 28
44  and $12, $2, $5
48  or $13, $6, $2
52  add $14, $2, $2
      ……
72  lw $3, 50($7)
```



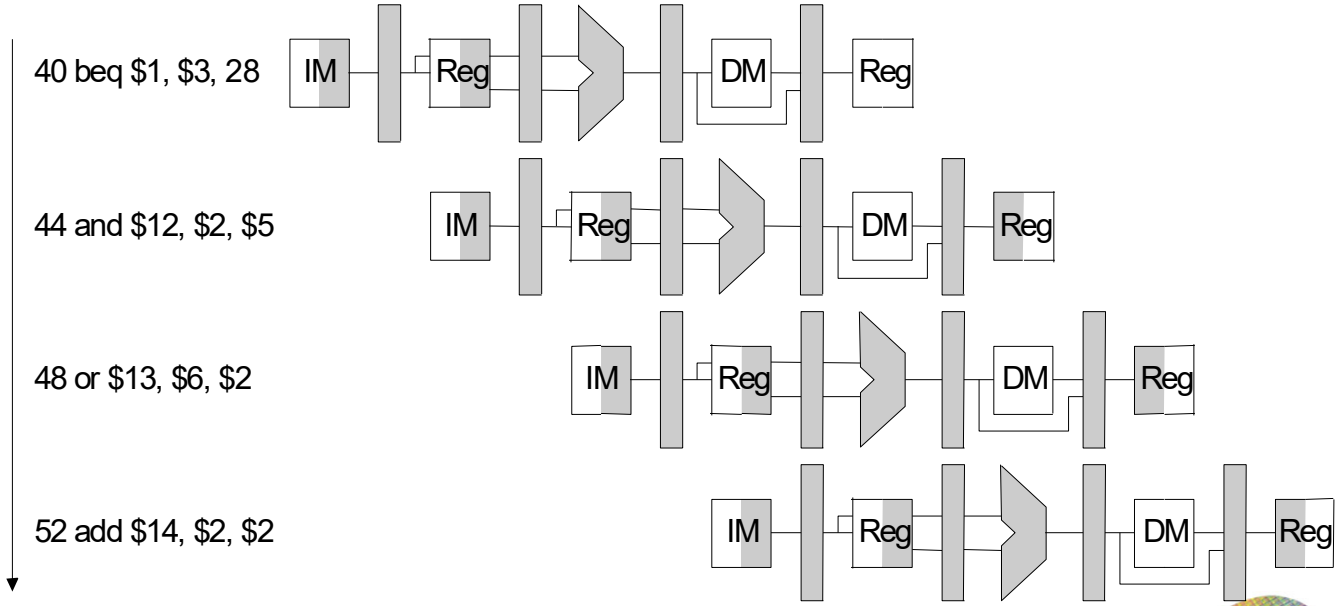
Better solution 1: *predict* branch outcome

- *Predict branch-not-taken => continue with next sequential instructions*
  - Correct prediction => no penalty and save time
  - Incorrect prediction => have to flush the pipeline behind the branch (see next slide

Case 1: Assume branch-not-taken and prediction is correct

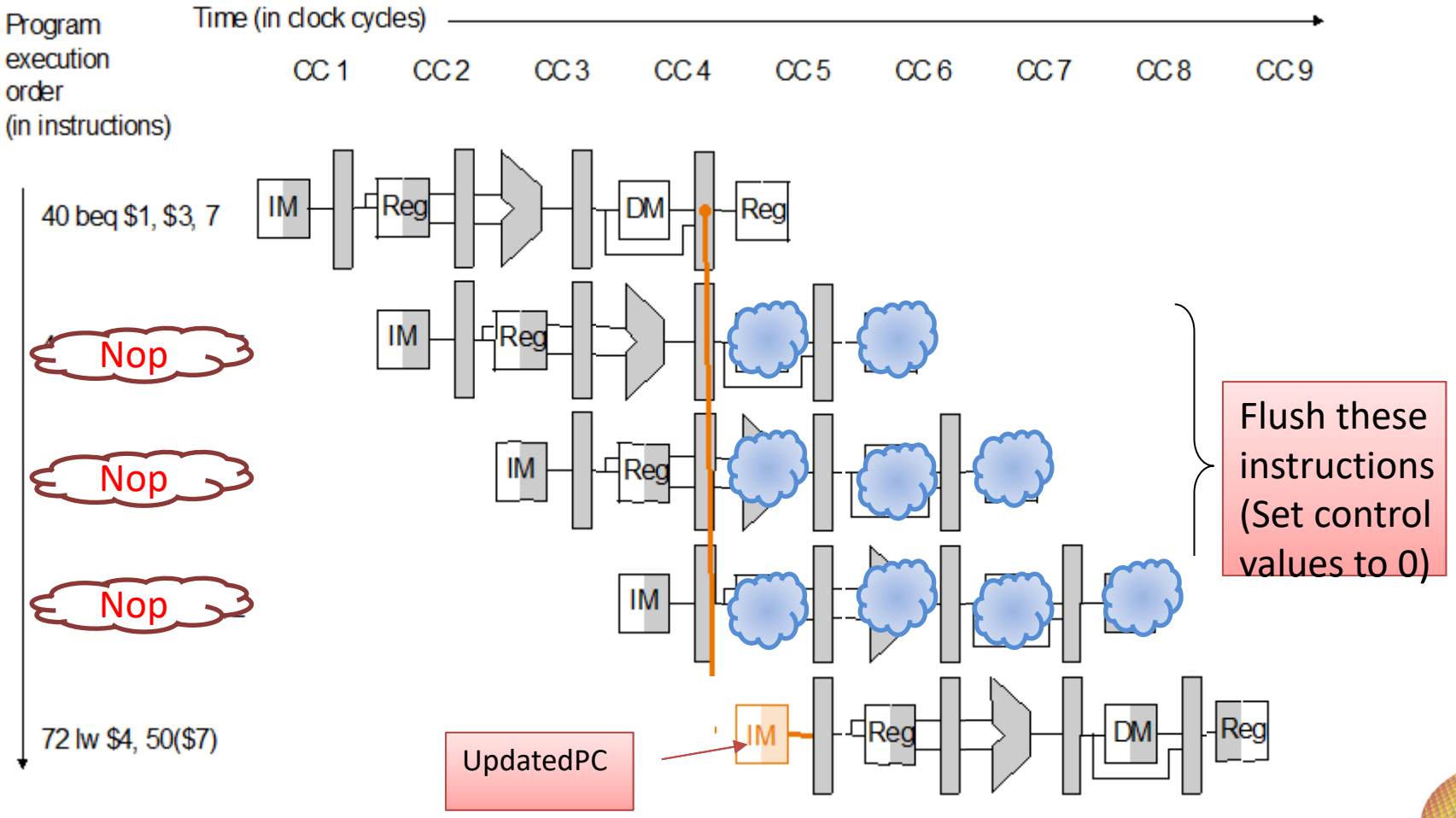
⇨ Pipeline is executed as normal

```
Addr
40    beq $1,  $3, 28
44    and $12, $2, $5
48    or $13, $6, $2
52    add $14, $2, $2
....  .........
72    lw $3, 50($7)
```



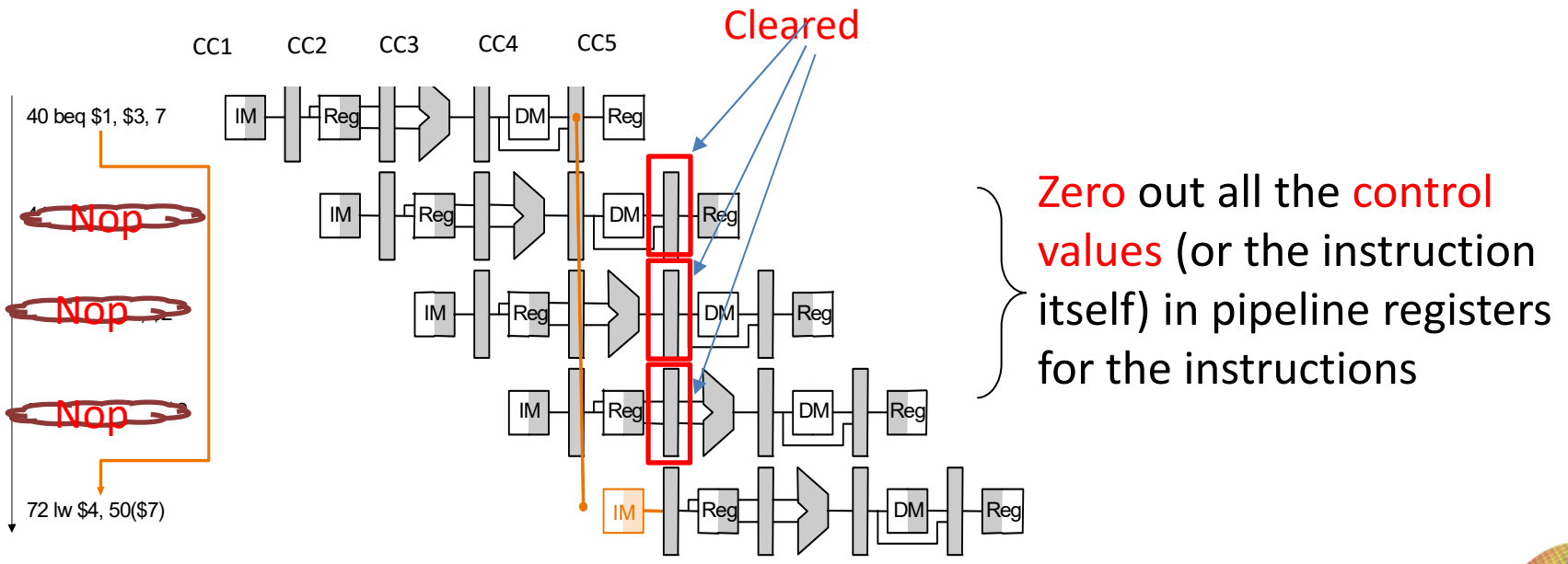
Better solution 1: *predict* branch outcome -2

Case 2: Assume Branch-not-taken, Incorrect Prediction (branch outcome is determined in MEM)



How flushing instructions is done?

- When misprediction occurs
  - Flush: Zero out all the control values (or the instruction itself) in pipeline registers for the instructions following the branch that are already in the pipeline
  - Similar to the stall on load-use data hazard (RAW) ...



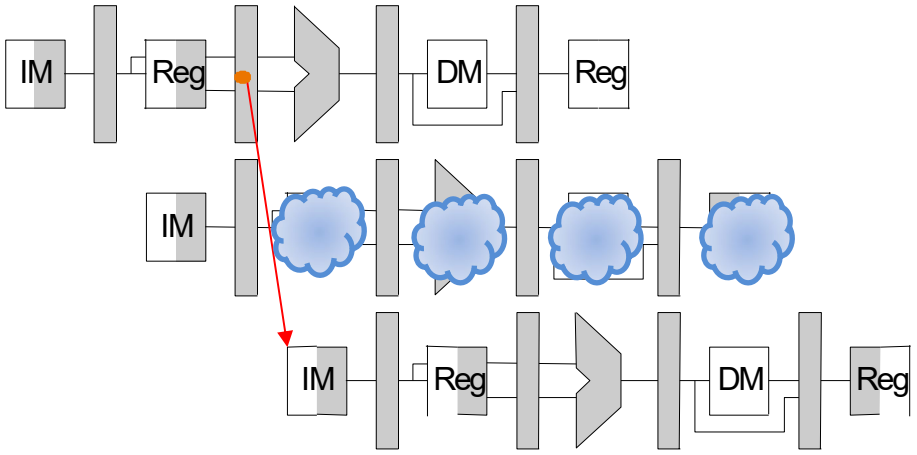
Better Solution 2:Reducing Branch Delay

- If branch decision is made at MEM stage, three instructions are flushed if misprediction occurs
- How to reduce Brach delay

  =>Decide branch outcome earlier (make decision in ID stage)

  =>only one instruction is flushed (IF stage)

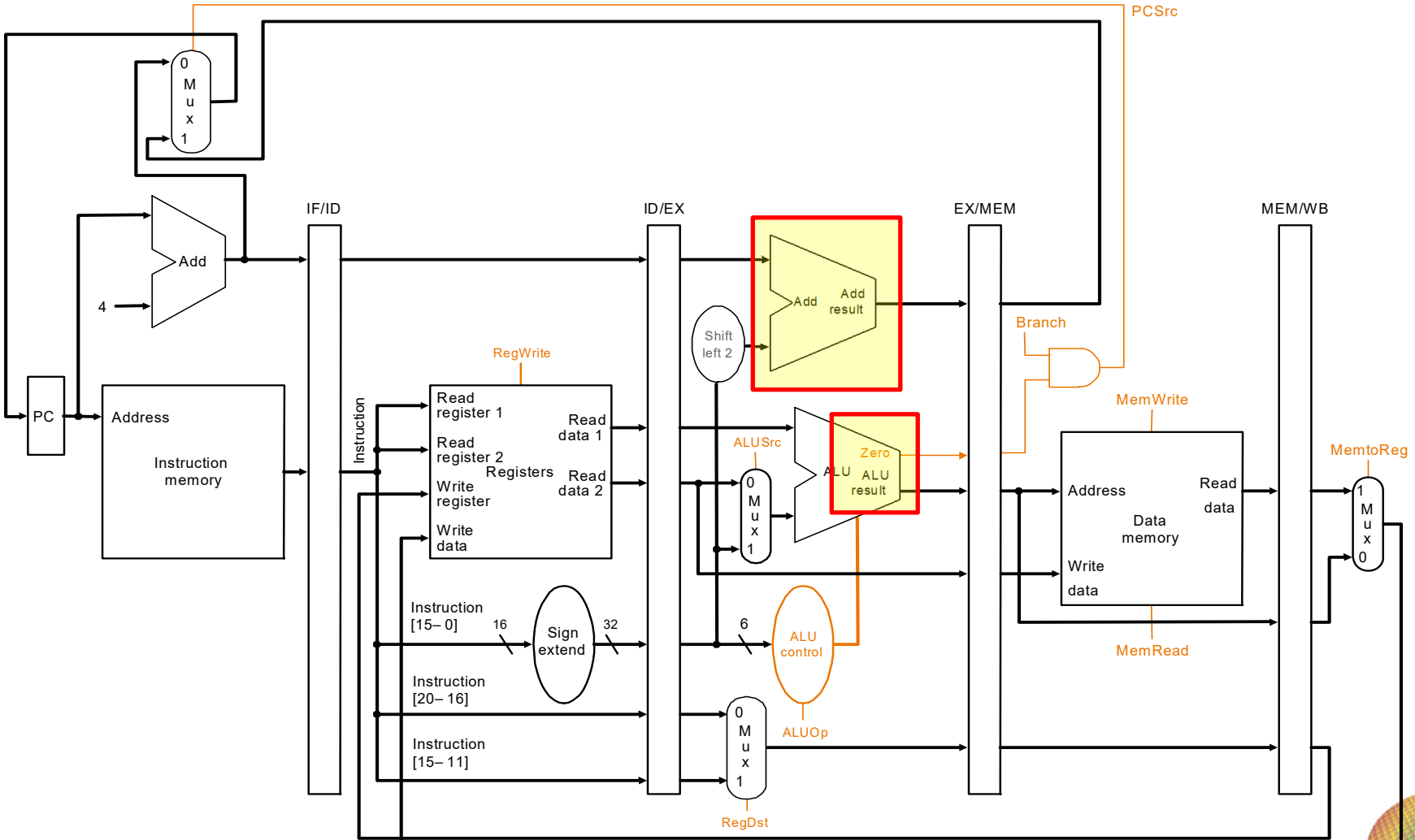
Flush this instruction



Original branch decision

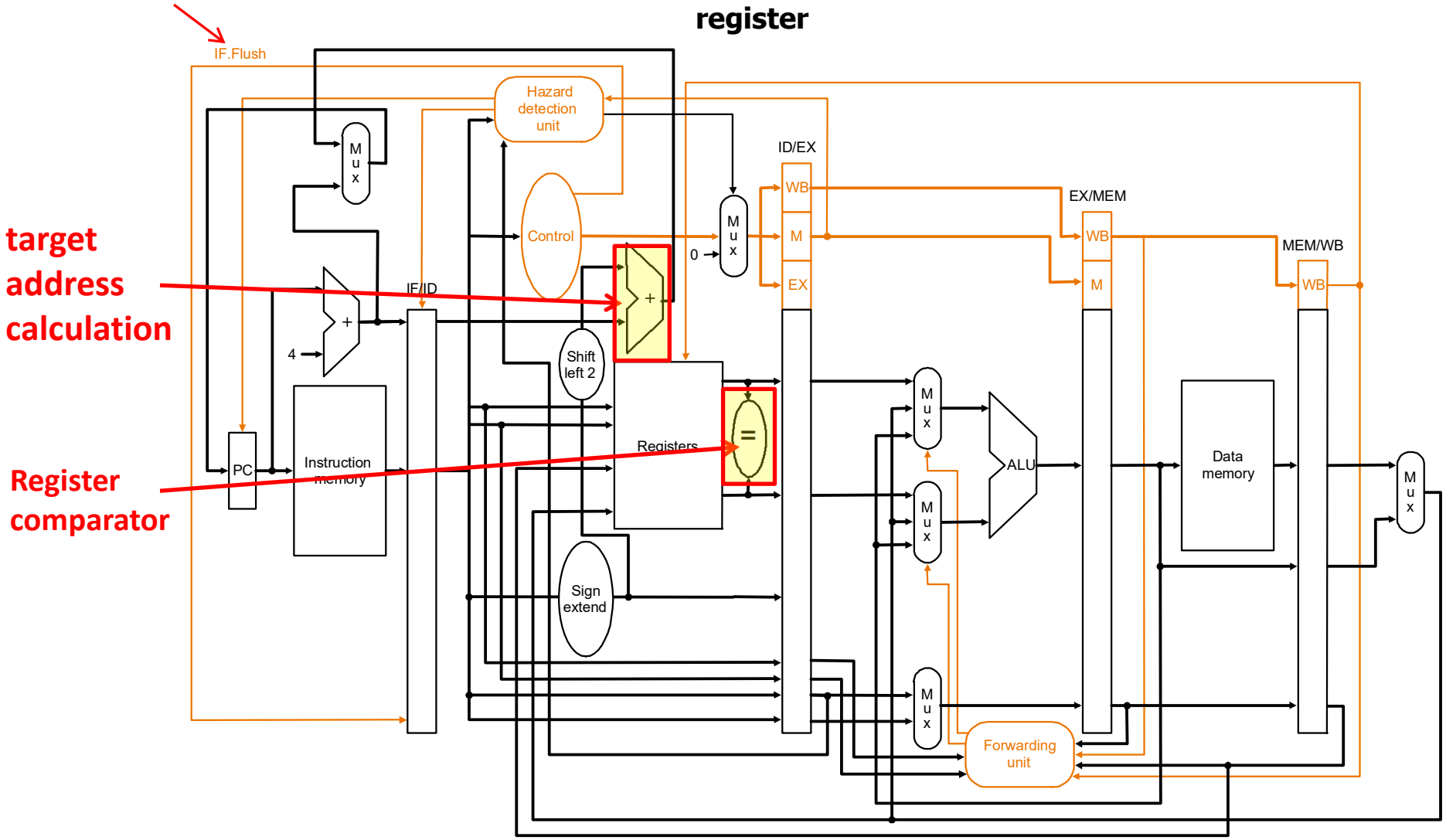
Hardware that are moved to ID stage

=> Target address calculation & Register Comparator



Optimized Datapath for Branch

IF.Flush signal zeros out the instruction (which follows the branch) in the IF/ID pipeline register

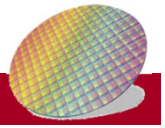


Branch decision is moved from the MEM stage to the ID stage – simplified drawing not showing enhancements to the forwarding and hazard detection units

Reducing Branch Delay by detecting at ID stage

- Two changes are needed to move the branch decision to the ID stage
  - Target address calculation
    - calculating the branch target address in ID stage, inputs to this adder, the PC value and the immediate fields are already available in the IF/ID pipeline register)
  - Register comparator
    - calculating the branch decision in ID stage,
    - for equality test, by XORing respective bits and then ORing all the results and inverting, rather than using the ALU to subtract and then test for zero (when there is a carry delay)

Also modify the forwarding and hazard detection units to forward to or stall the branch at the ID stage in case the branch decision depends on an earlier result

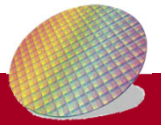


Reducing Branch Delay

- Example: branch taken

```
36:   sub   $10, $4, $8
40:   beq   $1,  $3, 7
44:   and   $12, $2, $5
48:   or    $13, $2, $6
52:   add   $14, $4, $2
56:   slt   $15, $6, $7
      ...
72:   lw    $4, 50($7)
```

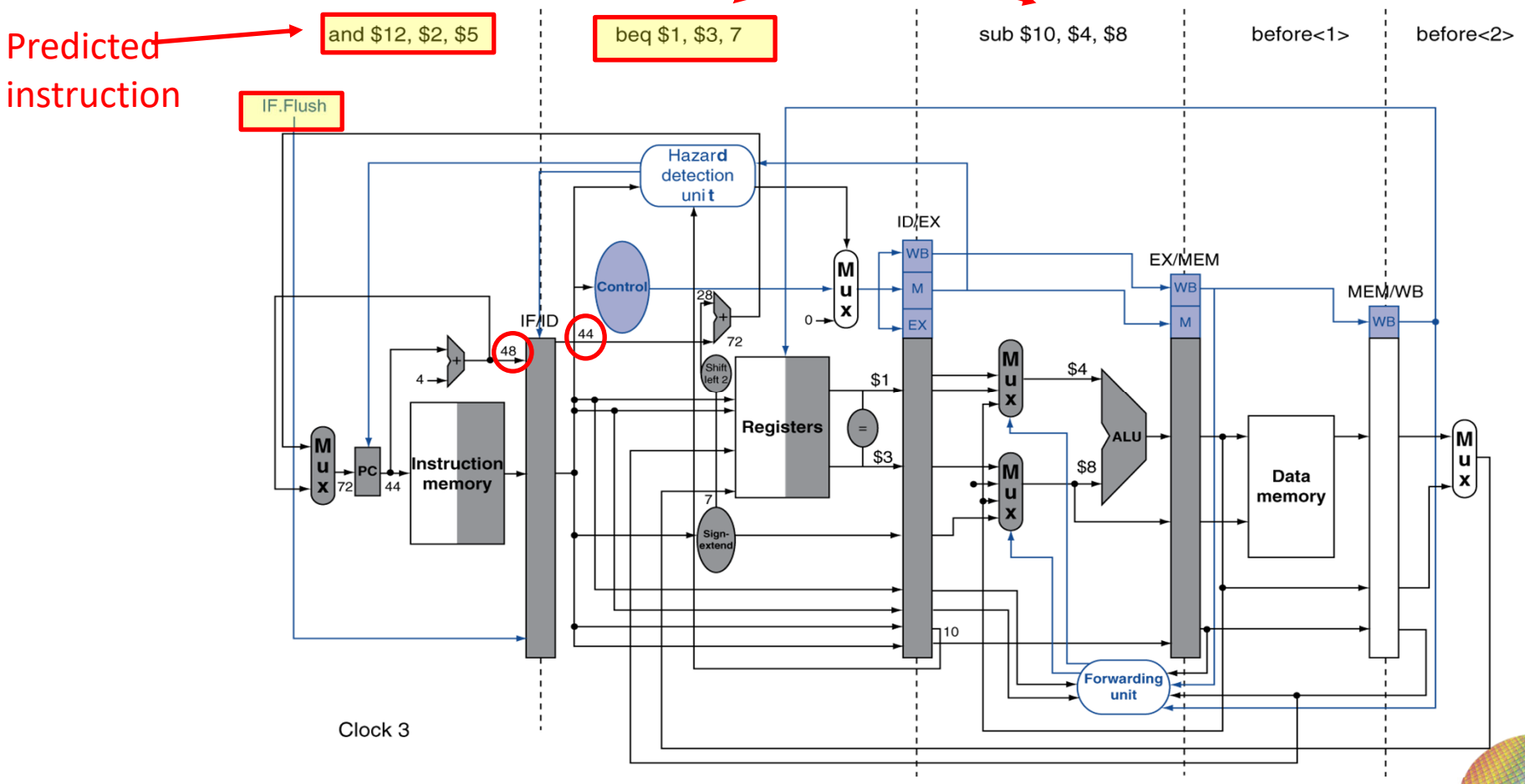
Branch is predicted not taken, but finally it is taken



```
36 sub $10, $4, $8
40 beq $1,  $3,  7
44 and $12  $2, $5
48 or  $13  $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
…
72 lw  $4,  50($7)
```

Example: Branch is predicted not taken, but actually taken

Assume $1 == $3, and predict not taken, but prediction is incorrect)

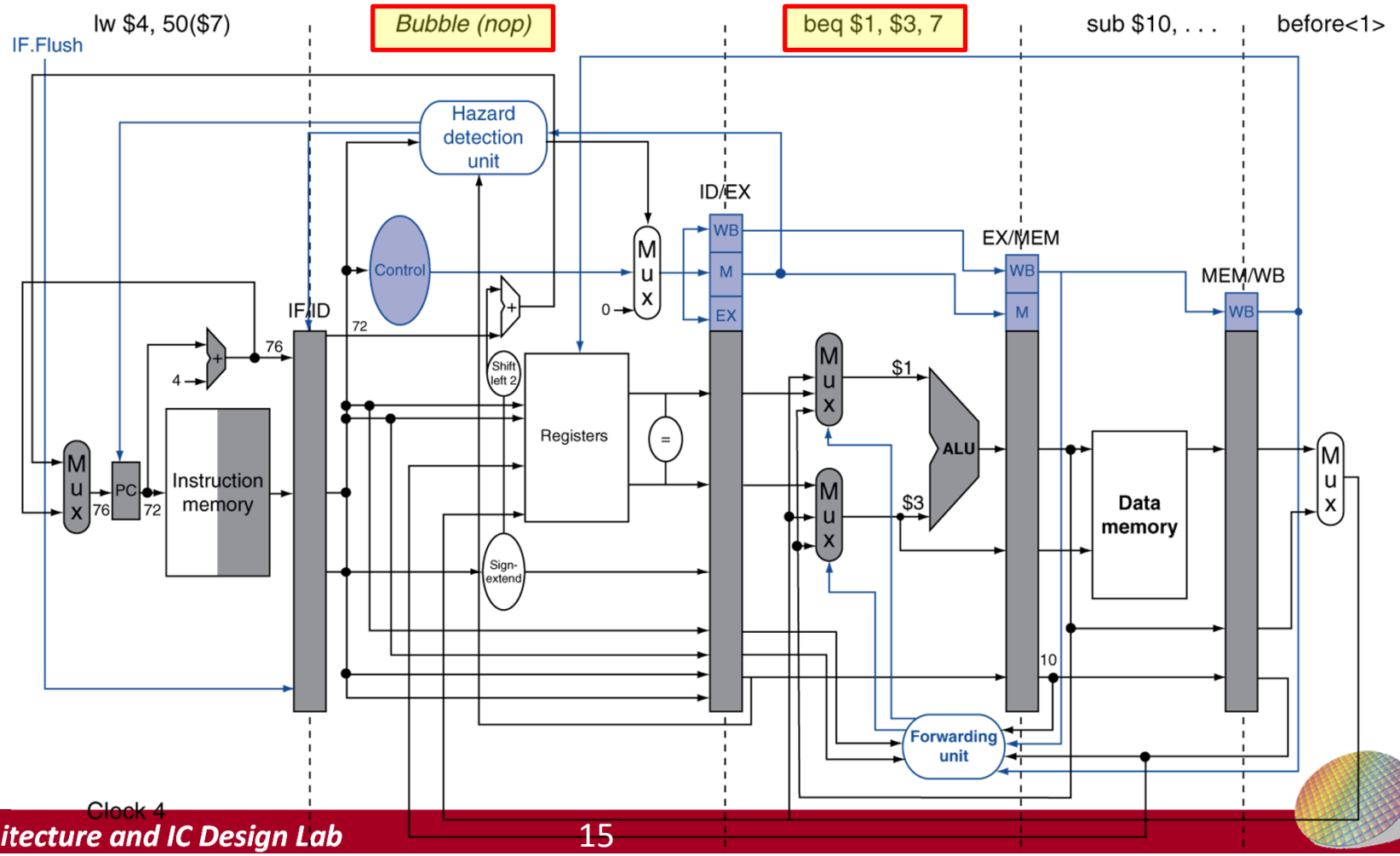


```
36 sub $10, $4, $8
40 beq $1,  $3,  7
44 and $12  $2, $5
48 or  $13  $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
…
72 lw  $4,  50($7)
```

Example: Branch is predicted not taken, but actually taken

Assume $1 == $3, and predict not taken (incorrect prediction)

Optimized pipeline with only one bubble penalty

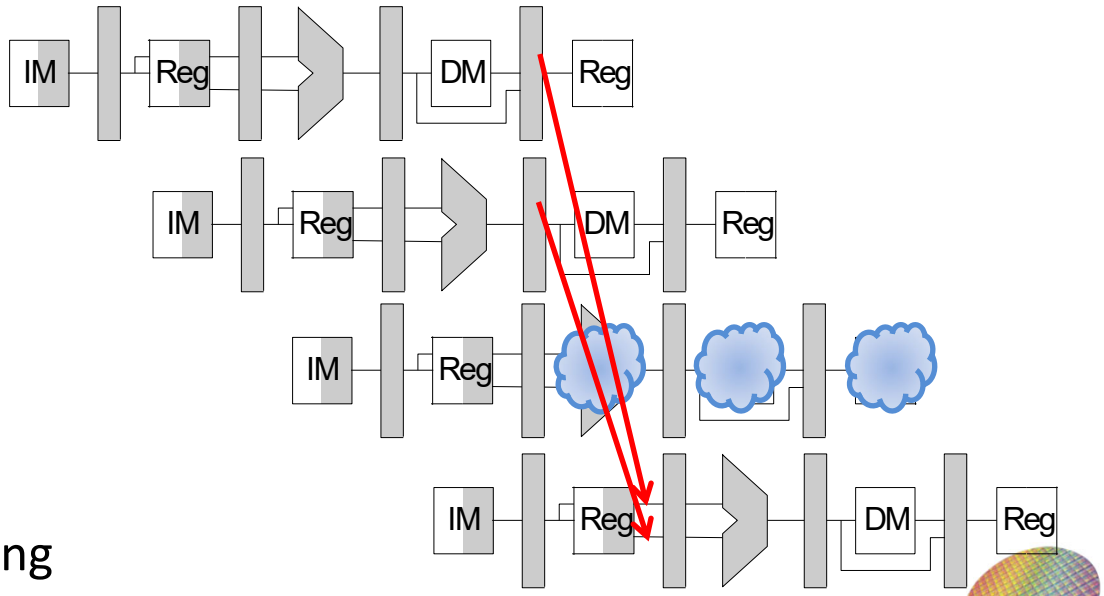


Data Hazards for Branches

If forwarding is used, are there any stalls in need in the following instructions ? If so, how many ?

```
add $1, $2, $3
add $4, $5, $6
beq $1, $4, target
```

```
add $1, $2, $3
add $4, $5, $6
nop
beq $1, $4, target
```

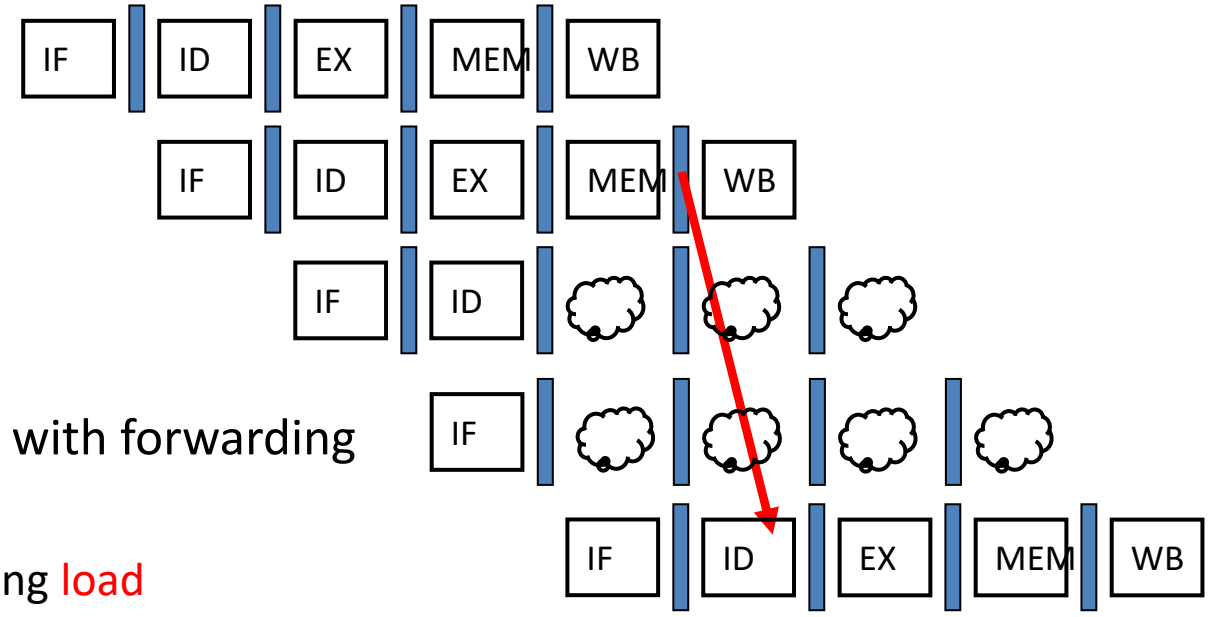


Solution: one stall with forwarding

Data Hazards for Branches-2

If forwarding is used, are there any stalls in need in the following instructions ? If so, how many ?

```
add $4, $5, $6
lw  $1, addr
beq $1, $4, target
```



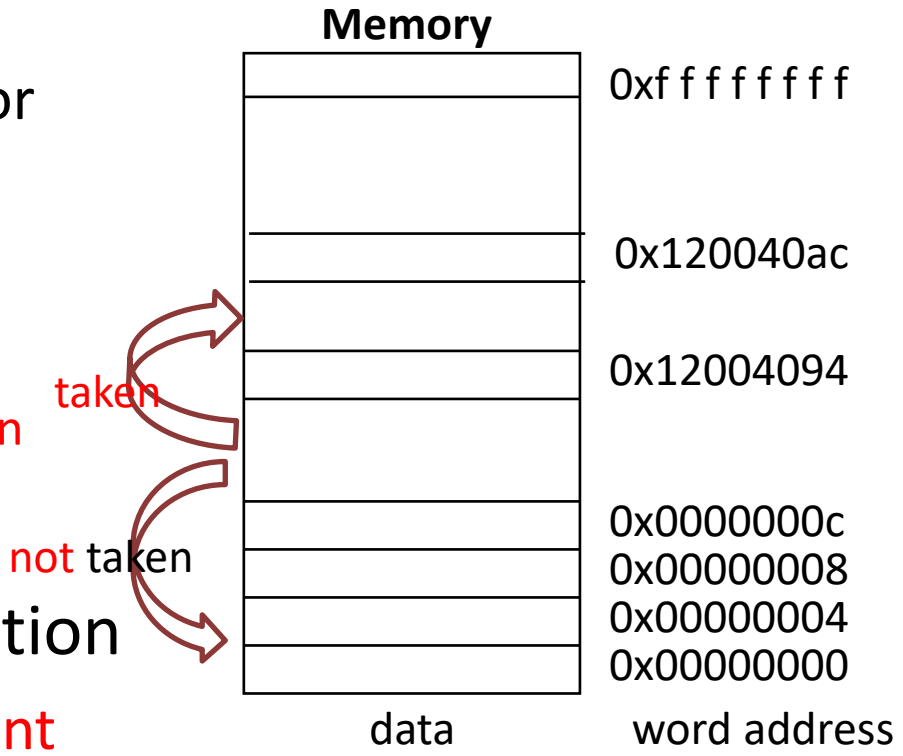
Solution: Two stall, even with forwarding

Forwarding from 2nd preceding load instruction

=> Need 2 stall cycle

(Recap) Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken



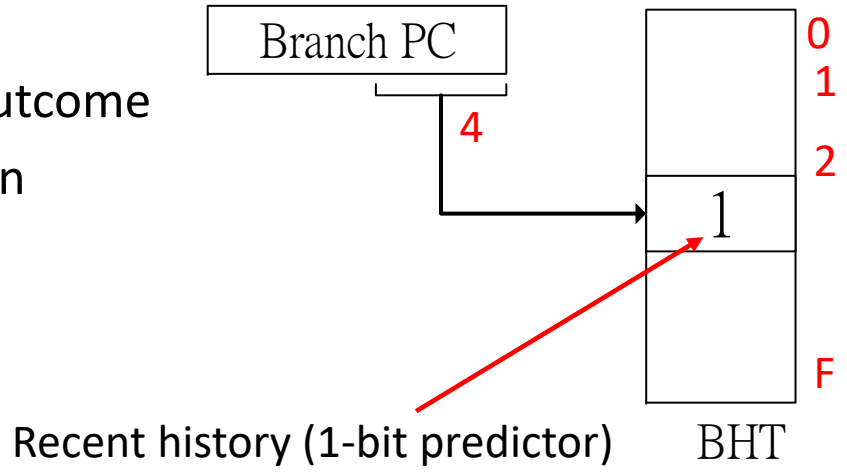
- Today: Dynamic branch prediction
  - Prediction based on record recent history of each branch
  - Hardware measures actual branch behavior

Taken, Taken, Taken, Taken

What is the next prediction?
Taken for Not Taken ?

Better Solution 3: Dynamic Branch Prediction

- Improve prediction accuracy
  - Based on the past history
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Based the table, make prediction



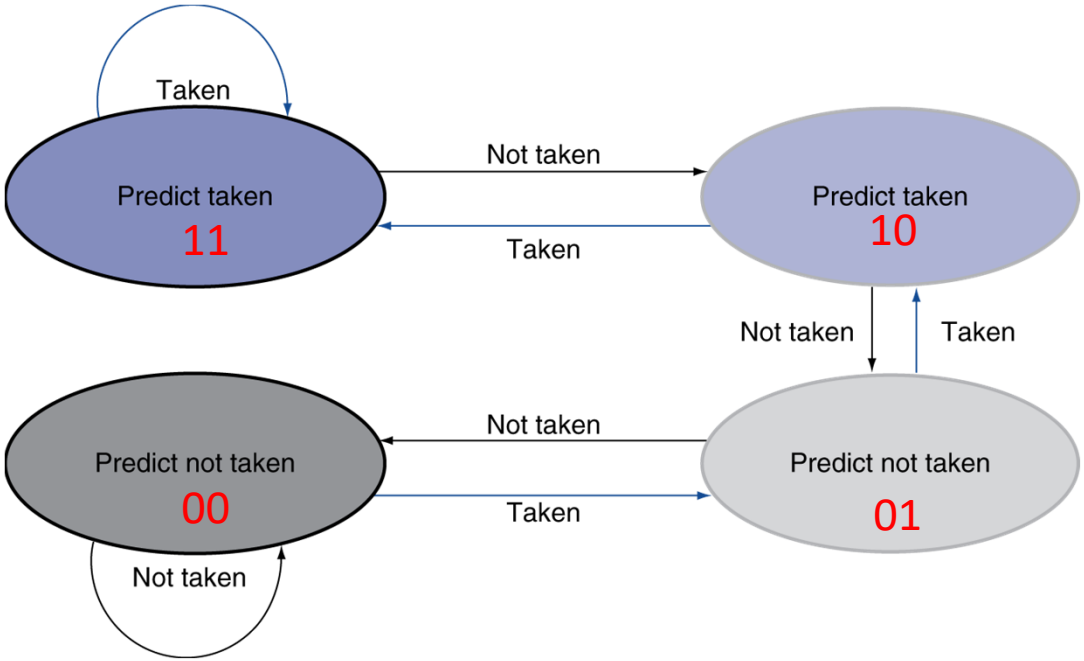
Example: 1-Bit Predictor

- 1-bit predictor:
  - When 0=> predict not taken,
  - When 1=>predict taken,
- Change states when incorrectly predicted
- Example: assume four branches are taken(T), taken(T), taken(T), not-taken(N), 1-bit predictor is initialized to 0, what is the prediction accuracy if 1-bit predictor is used

Execution pattern	T	T	T	N	Accurac
Predictor value	0	1	1	1	y=2/4 =
Predicted branch	N	T	T	T	50%
Correct or incorrect	I	C	C	I	

2-Bit Predictor

- Four states: 00, 01, 10, 11
  - 00, 01=> predict not taken,
  - 10, 11=> predict taken
- Only change prediction on two successive mispredictions



2-bit predictor is initialized to 0

Execution Pattern	T	T	N	T	T	T	N	T
Predictor value at time of prediction	0	1	2	1	2	3	3	2
Predicted branch	N	N	T	N	T	T	T	T
Prediction result in steady state	I	I	I	I	C	C	I	C

Another Example:

- Consider the following loop branch that branches 9 times in a row (taken), and then is not taken once.
  - Prediction accuracy for 1-bit predictor
  - Prediction accuracy for 2-bit predictor

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

1-bit predictor

Execution Pattern:	T T T T T T T T T N T T T T T T T T T N ….
Predictor value at time of prediction	0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 …
Predicted branch	N T T T T T T T T T N T T T T T T T T T …
Prediction result in steady state	I C C C C C C C C I I C C C C C C C C I …

Prediction accuracy (on average) for many loops: 80%

2-bit predictor

Execution Pattern:	T T T T T T T T T N T T T T T T T T T N T
Predictor value at time of prediction	0 1 2 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 2
Predicted branch	N N T T T T T T T T T T T T T T T T T T T
Prediction result in steady state	I I C C C C C C C C I C C C C C C C C C I C

Prediction accuracy (on average) for many loops: 90%