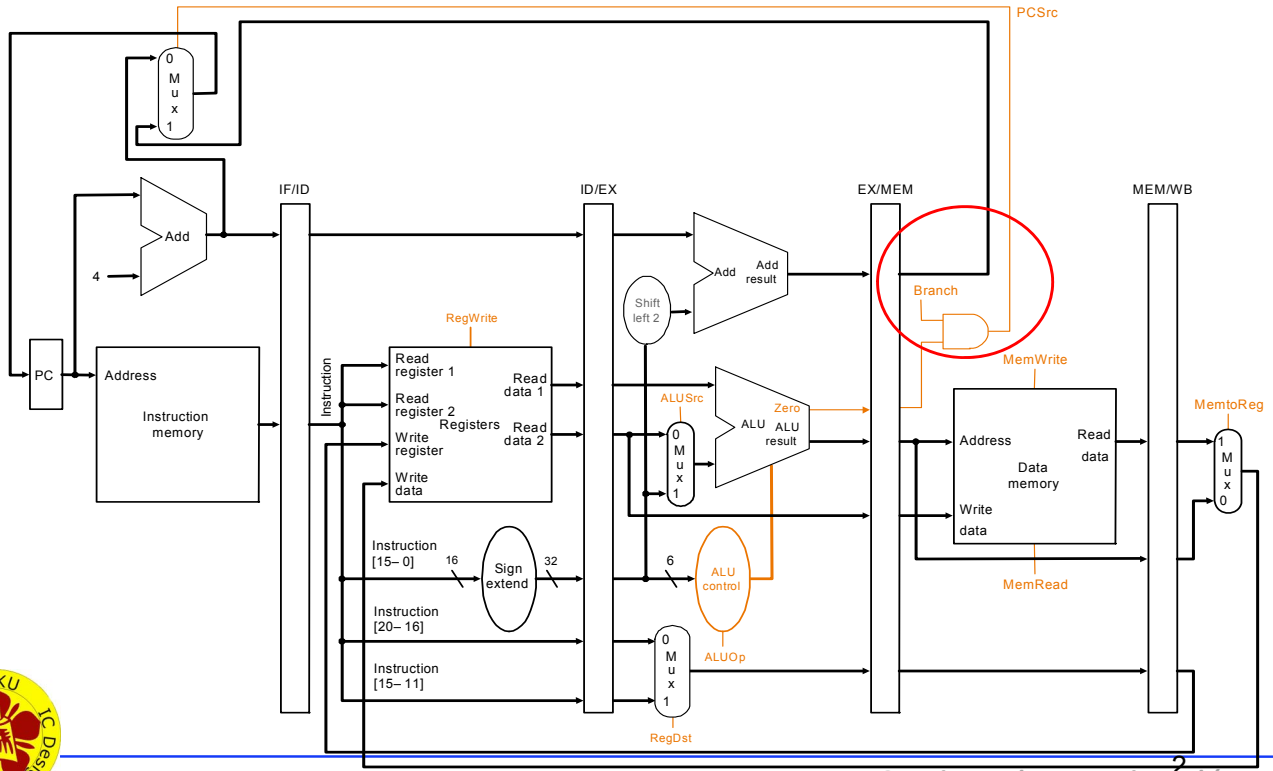
Outline

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

Which stage is the branch decision made?

Case 1: The MEM stage

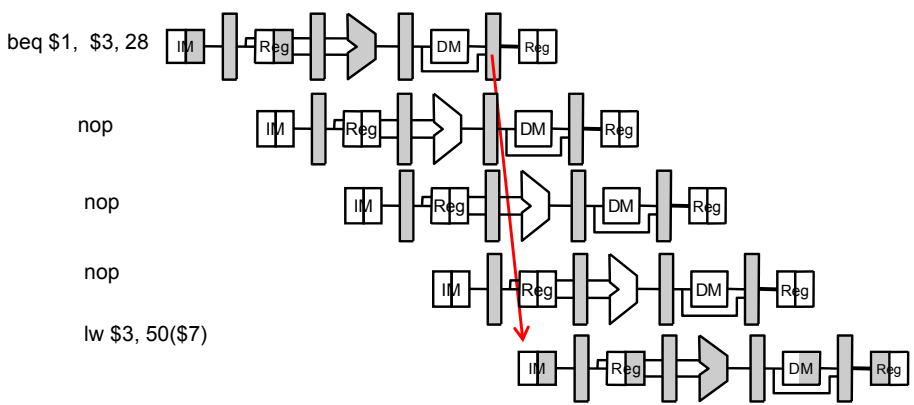


Control (or Branch) Hazards

- Branch decision is made in the MEM stage
  - Unable to determine the following instruction in pipeline immediately
  - Control hazard will occur
- Solution 1: *stall* the pipeline till branch decision is known
  - not efficient, slow the pipeline significantly!

Solution 1 if $1=$3

```
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)
```



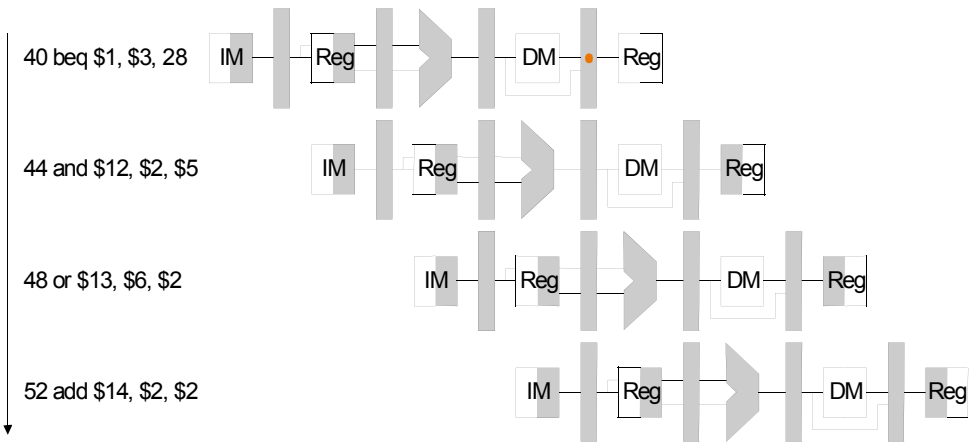
Control Hazards -Solution 2: *predict* branch outcome

- e.g., 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

Assume branch-not-taken and prediction is correct

Pipeline is executed correctly

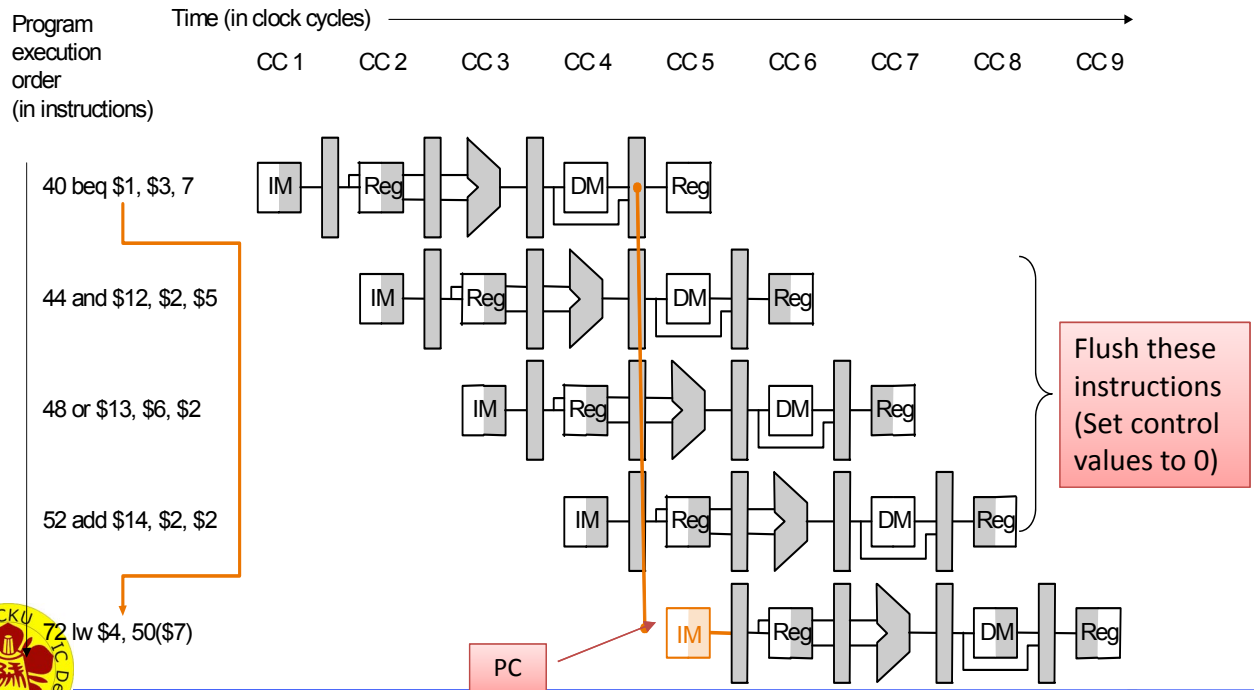
```
Addr
40    beq $1,  $3, 28
44    and $12, $2, $5
48    or $13, $6, $2
52    add $14, $2, $2
....  ……
72    lw $3, 50($7)
```



Incorrect Prediction (Assume Branch-not-taken)

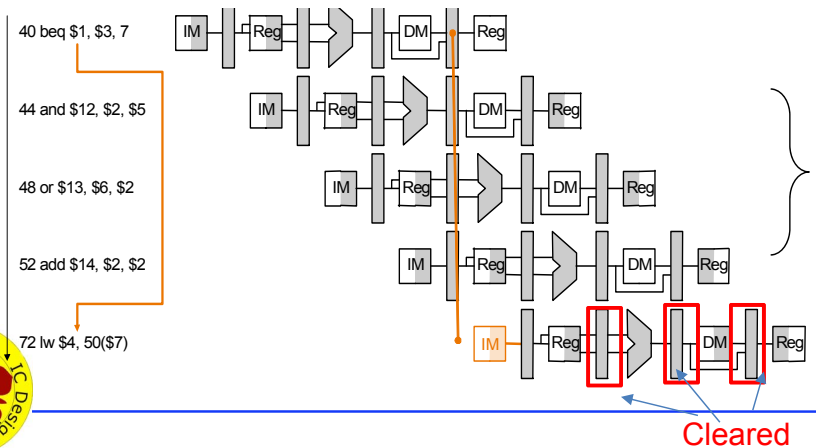
Incorrect prediction => the following three instructions already in the pipeline have to be flushed and execution resumes at lw

Note branch outcome is determined in MEM



How flushing instructions is done?

- Prediction is not 100% accurate
- 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 strategy as for stalling on load-use data hazard (RAW) …



Zero out all the control values (or the instruction itself) in pipeline registers for the instructions

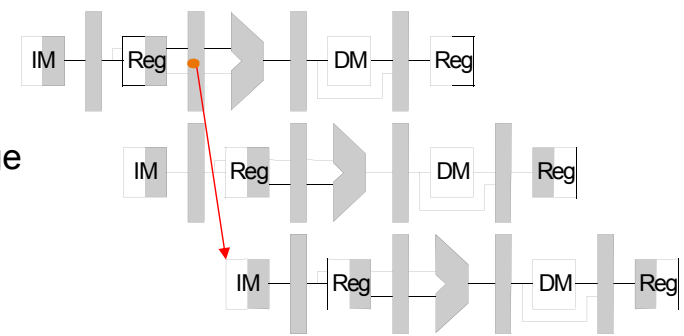
Reducing Branch Delay

- 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 branch decision in ID state

  =>only one instruction is flushed (IF stage)



Hardware moved to ID stage

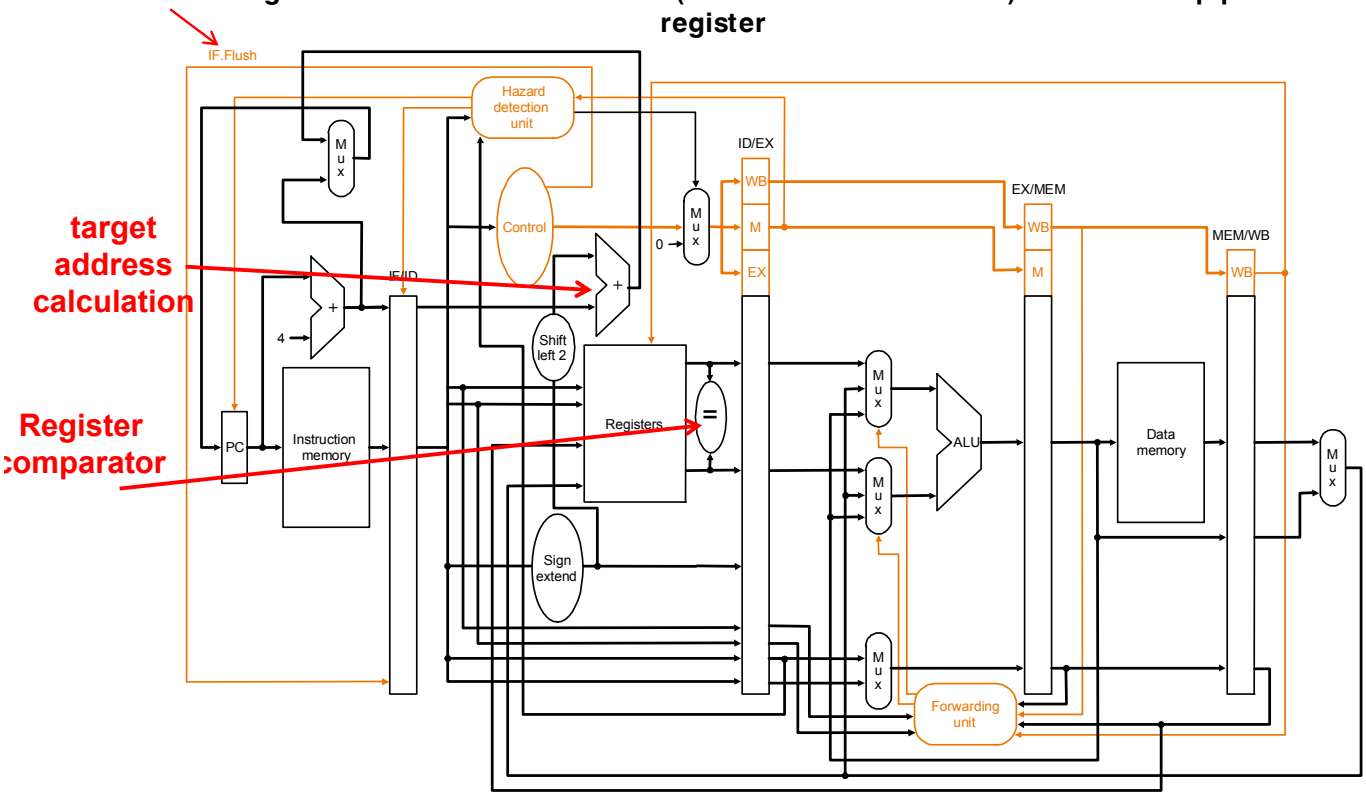
⇒ Target address adder

⇒ Register Comparator

Flush this instruction

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 adder
    - 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

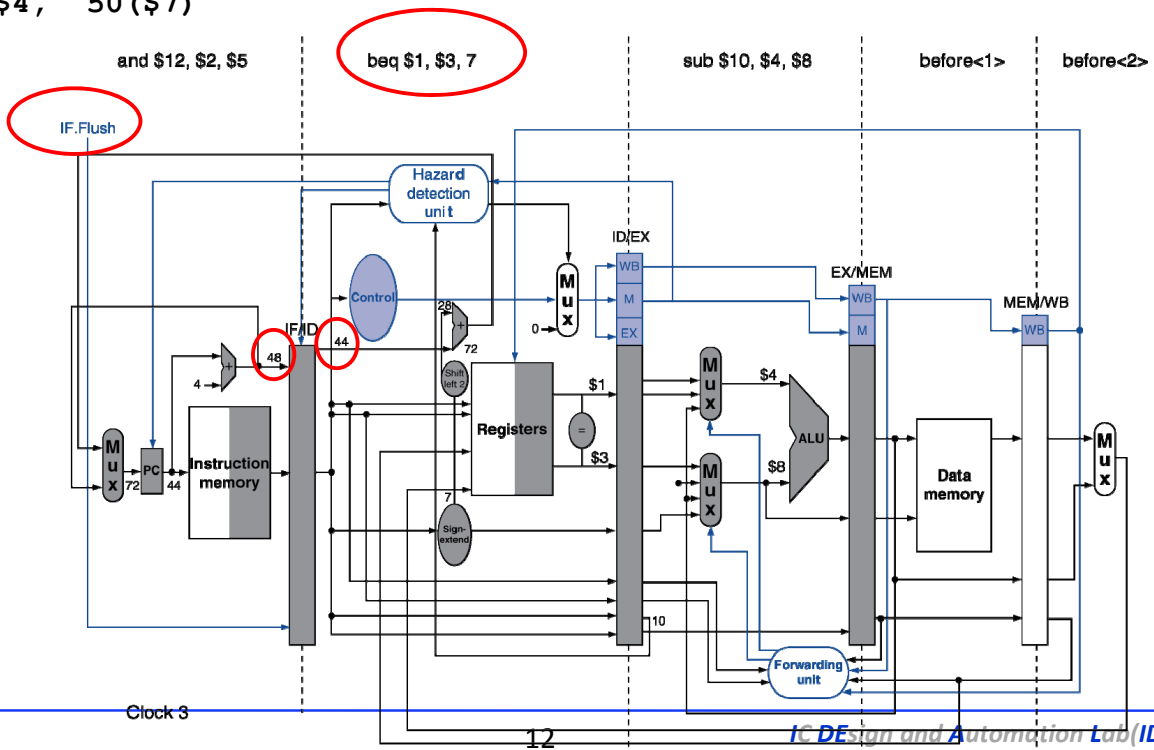
- Example: branch taken

```
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)
```

```
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 Taken

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



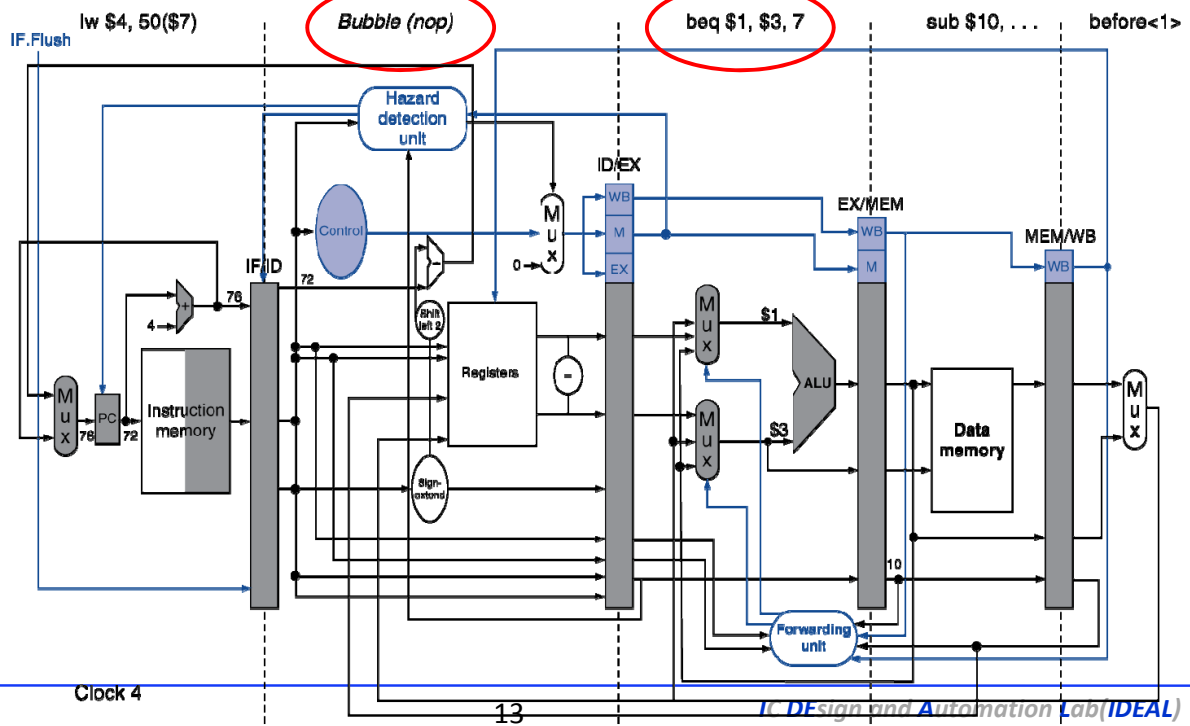
```
36 sub $10, $4, $8
40 beq $1,  $3,  7
44 and $12  $2, $5
48 or  $13  $2, $6
52 add $14, $4, $2
56 slt $15, $6, $7
...
72 lw  $4,  50($7)
```

Example: Branch Taken(cont.)

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

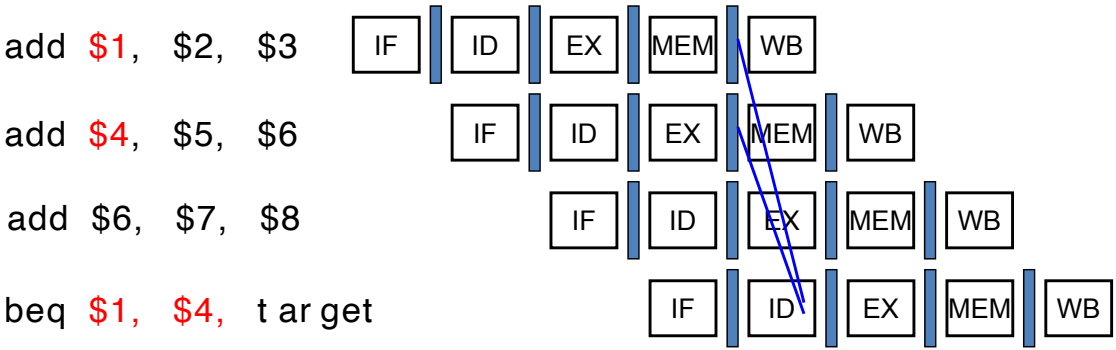
Optimized pipeline with only one bubble penalty

預測錯誤，需加入一個bubble



Data Hazards for Branches

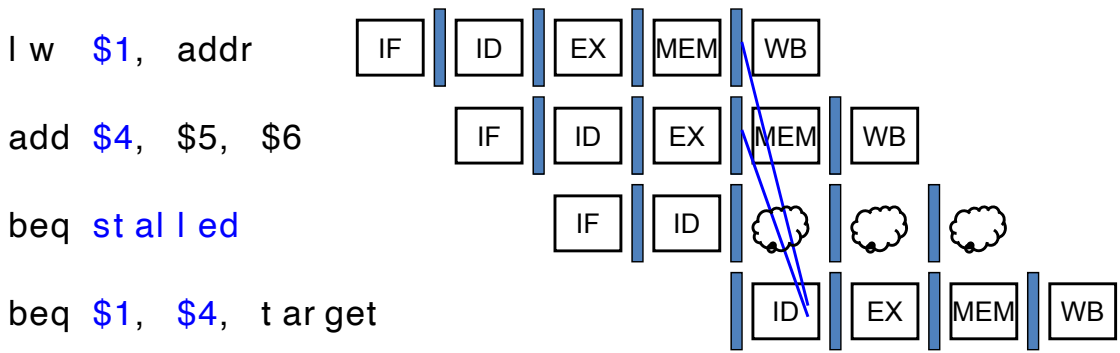
- Are any stalls in need in the following instructions? If so, how many ? Don't forget forwarding.



Solution: no stall, due to forwarding

Data Hazards for Branches-2

- Are any stalls in need in the following instructions? If so, how many ? Don't forget forwarding.



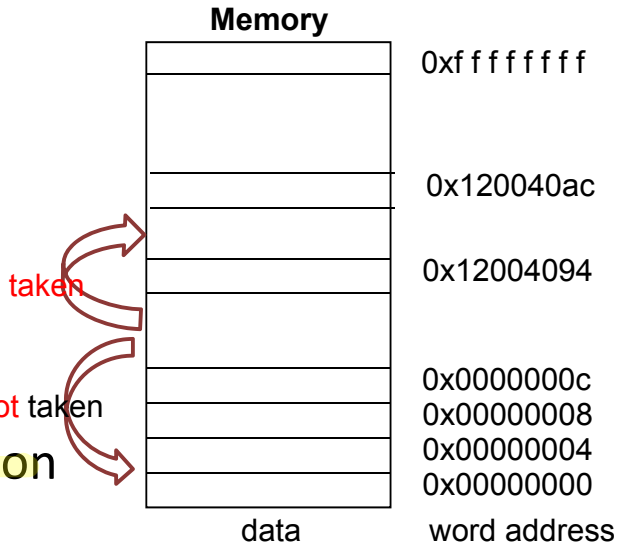
Solution: one stall, even with forwarding

If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction

=> Need 1 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
  - Perdition 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 ?

Dynamic Branch Prediction

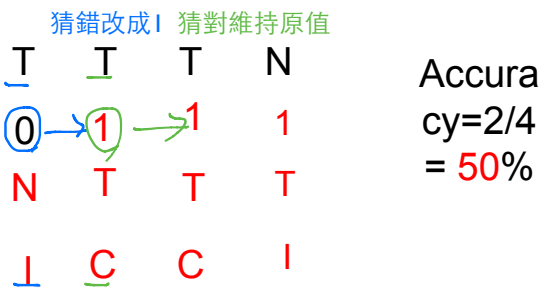
- Improve prediction accuracy
  - Based on the past history
- Use dynamic prediction 紀錄最近發生的情形
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome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

Example: 1-Bit Predictor

- 1-bit predictor:
  - When 0=> predict not taken,
  - When 1=>predict taken,

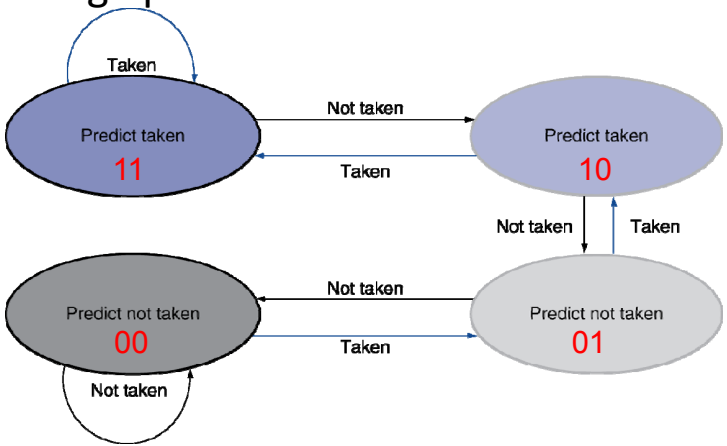
– Change states when incorrectly predicted

– Example: assume four branches are taken, taken, taken, not-taken, 1-bit predictor is initialized to 0, what is the prediction accuracy if 1-bit predictor is used



2-Bit Predictor

- Four states: 00, 01, 10, 11
  - 00,01=> predict not taken, 10, 11=> prediction 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 nine 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:  …
```

9個loop

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%