Lecture 18
Control Hazards

School of Computer Science and Engineering
Soongsil University

4. The Processor

4.8 Control Hazards

- **Control hazard or branch hazard**
  - When the proper instruction cannot execute in the proper pipeline clock cycle because the instruction that was fetched is not the one that is needed
  - That is, the flow of instruction addresses is not what the pipeline expected.
  - Pipeline can’t always fetch correct instruction
  - The delay in determining the proper instruction to fetch
- **Solutions**
  1. Stall on branch
  2. Branch prediction
  3. Delayed branch

Impact of the Pipeline on the Branch

- When branch outcome is determined in MEM stage

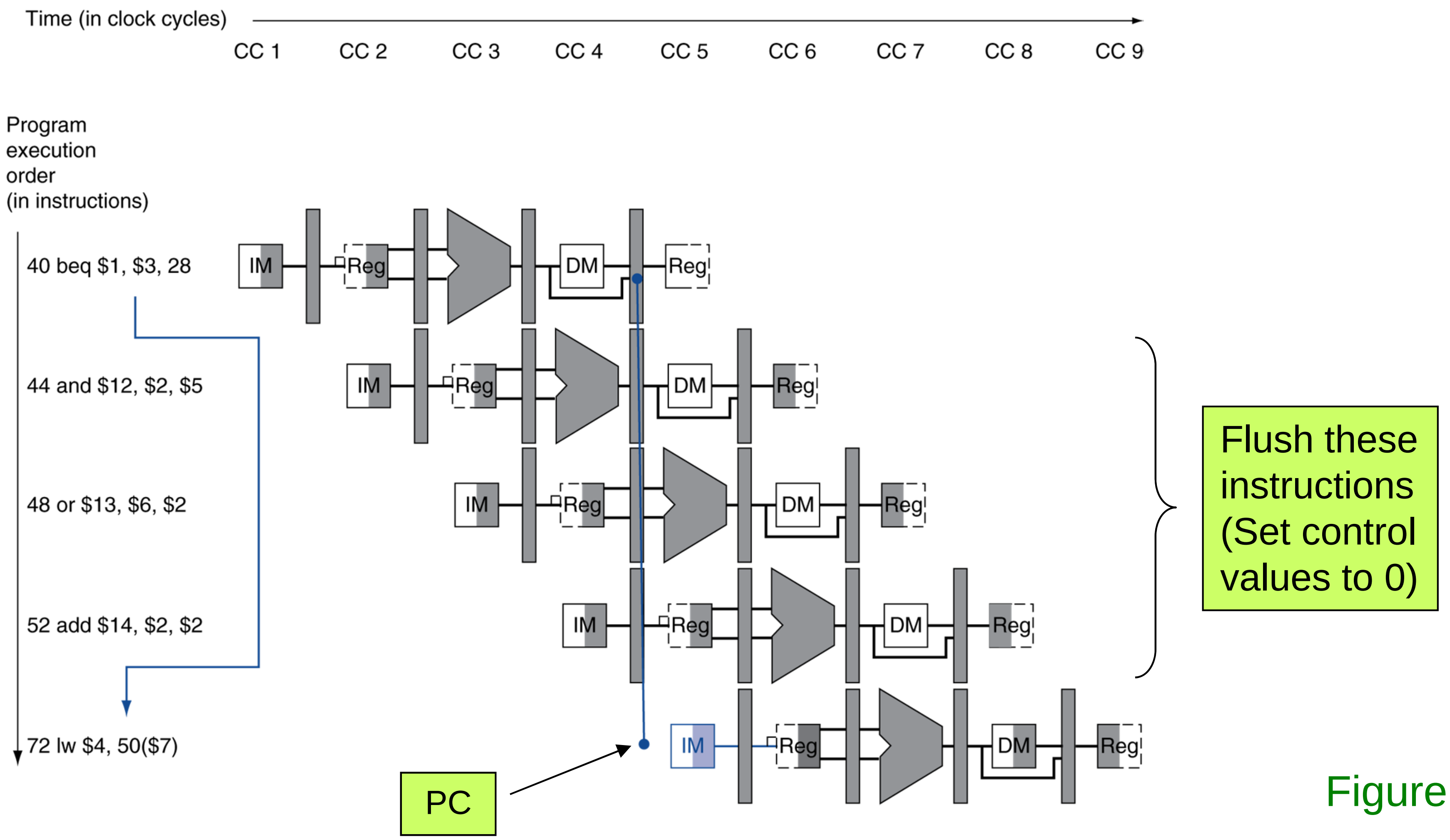


Figure 4.61

Solution 1 – Stall on Branch

- Wait until branch outcome determined before fetching next instruction
  - See § 4.5
- A penalty of 3 clock cycles for each branch
  - With branch execution in MEM stage (Figure 4.61)
- 1 cycle penalty with branch execution in ID stage

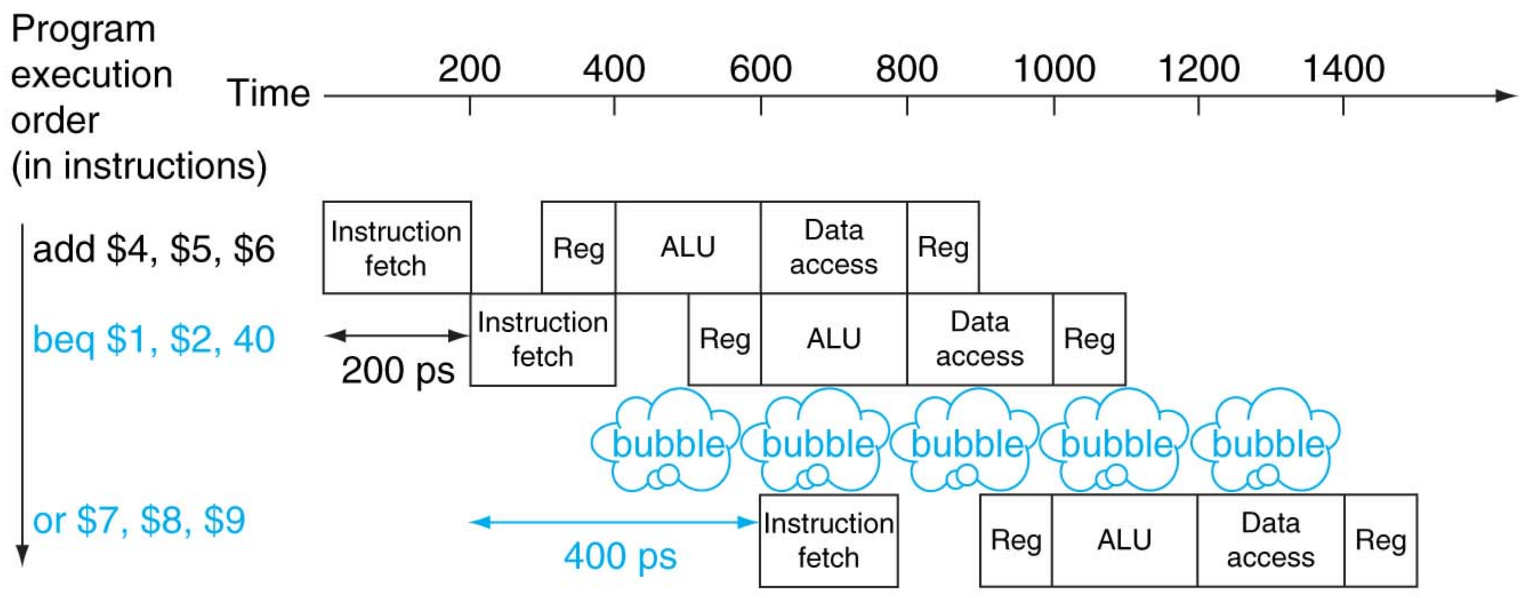


Figure 4.31

Reducing the Delay of Branches

- **Branch execution in ID stage, not in MEM stage**
  - Only 1 instruction in IF stage should be flushed.
  - 1 clock cycle of penalty
- **Modifications of the datapath**
  - Moving branch adder to ID stage
  - Inserting comparator in ID stage
- **New control signal : IF.Flush**
  - Zeroing the instruction field of IF/ID register

    (cf) nop = $0000\ 0000_{hex}$

Final Datapath and Control

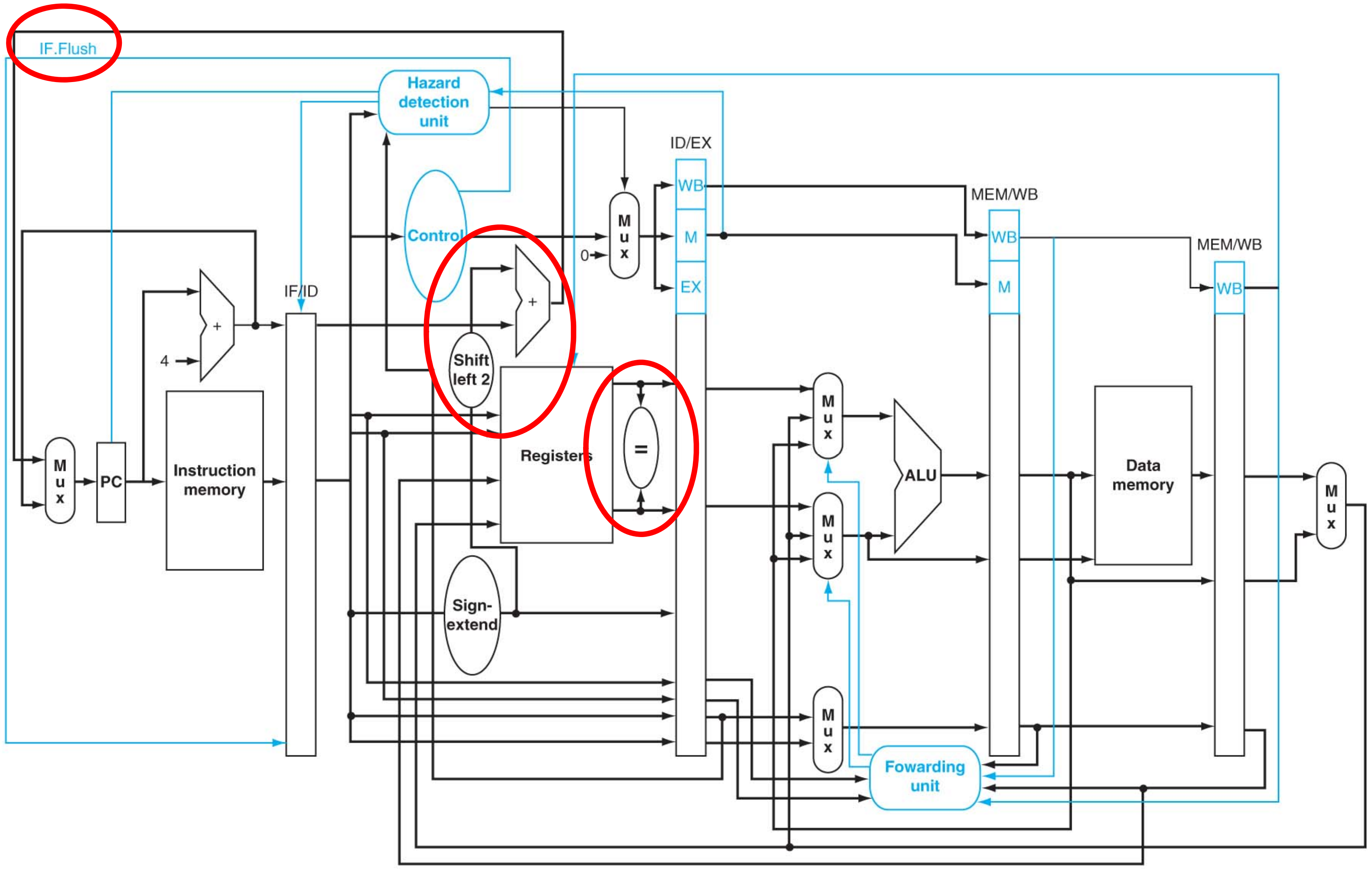


Figure 4.65

Example in p.270 (p.283 of Korean edition) Performance of “Stall on Branch”

- **Estimate the impact on CPI of stalling on branches.**

[Answer]

Frequency of branches: 17% in SPECint2006

CPI of branch = 1 clock + 1 extra clock for the stall

Other instructions = 1 clock

Thus,

Average CPI = 1 + 0.17x1 = 1.17

Solution 2 – Branch Prediction

- Longer pipelines can’t readily determine branch outcome early
    - Stall penalty becomes unacceptable
- **Branch Prediction**
    - A method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.
    - Predict outcome of branch
    - Only stall if prediction is wrong

Static and Dynamic Branch Prediction

- **Static branch prediction**
  - predicts before a programs runs
  - using either compile time heuristics or profiling
- **Dynamic branch prediction**
  - predicts at run-time
  - by recording information, in hardware, of past branch history during a program's execution

Static prediction

1. Assume branch taken
2. Assume branch not taken
3. Prediction by opcode
4. Prediction by direction

Dynamic prediction

1. Branch prediction buffer
   - 1-bit predictor
   - 2-bit predictor
2. Correlating branch predictor
3. Tournament branch predictor
4. Branch target buffer

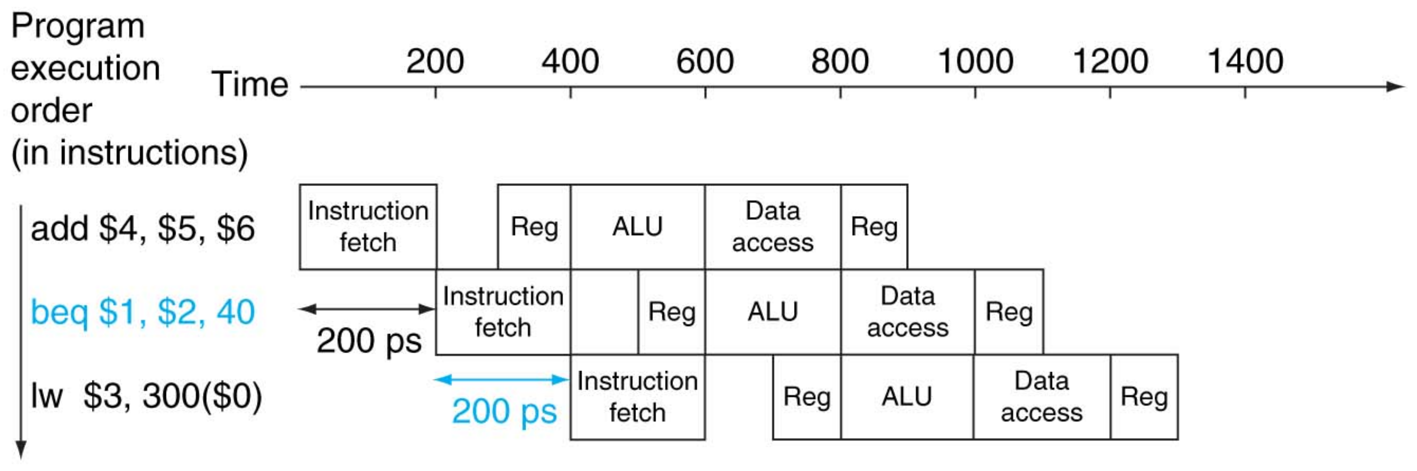
Assume Branch Not Taken

- Continue execution down the sequential instruction stream
- If branch taken,

  discard the instructions in the pipeline.

  - Changing the original control values to 0s
  - Flushing the 3 instructions in the IF, ID and EX stages when the branch reaches MEM stage

Misprediction Penalty



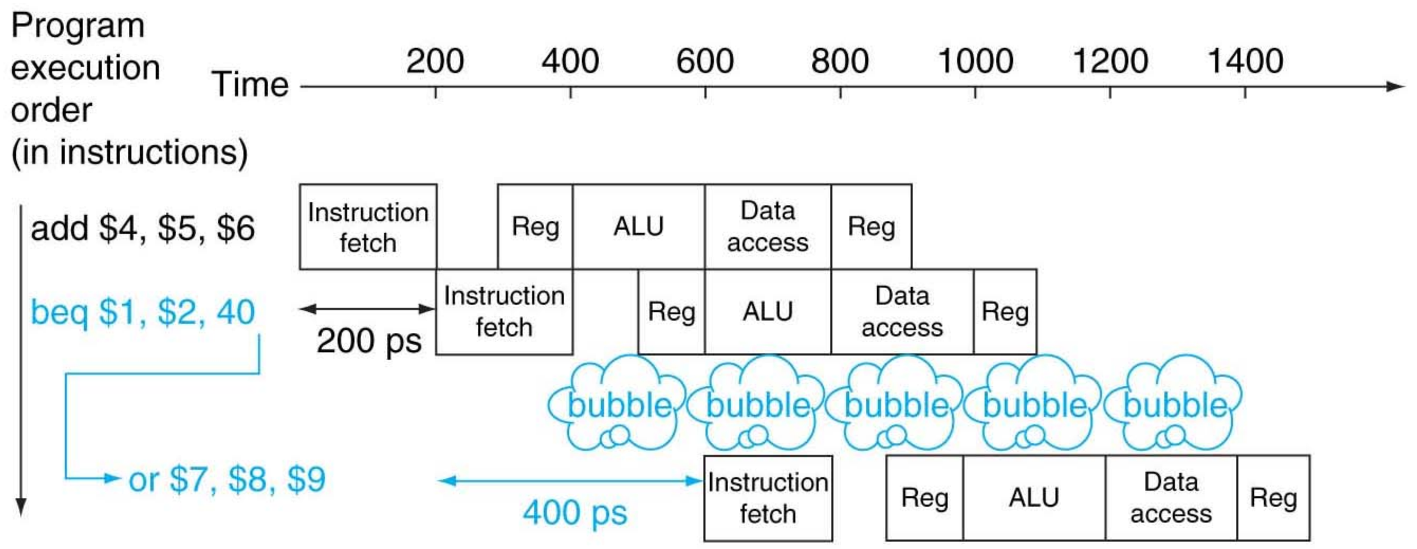


Figure 4.32

Example: Pipelined Branch

- **Show what happens both when the branch is taken and when not taken.**
- Assume the optimization on branch not taken.

```
36    sub  $10, $4, $8
40    beq  $1,  $3, 7   # pc-relative branch
                        # to 40+4+7*4=72
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)
```

[Answer: When branch taken] - Clock 3

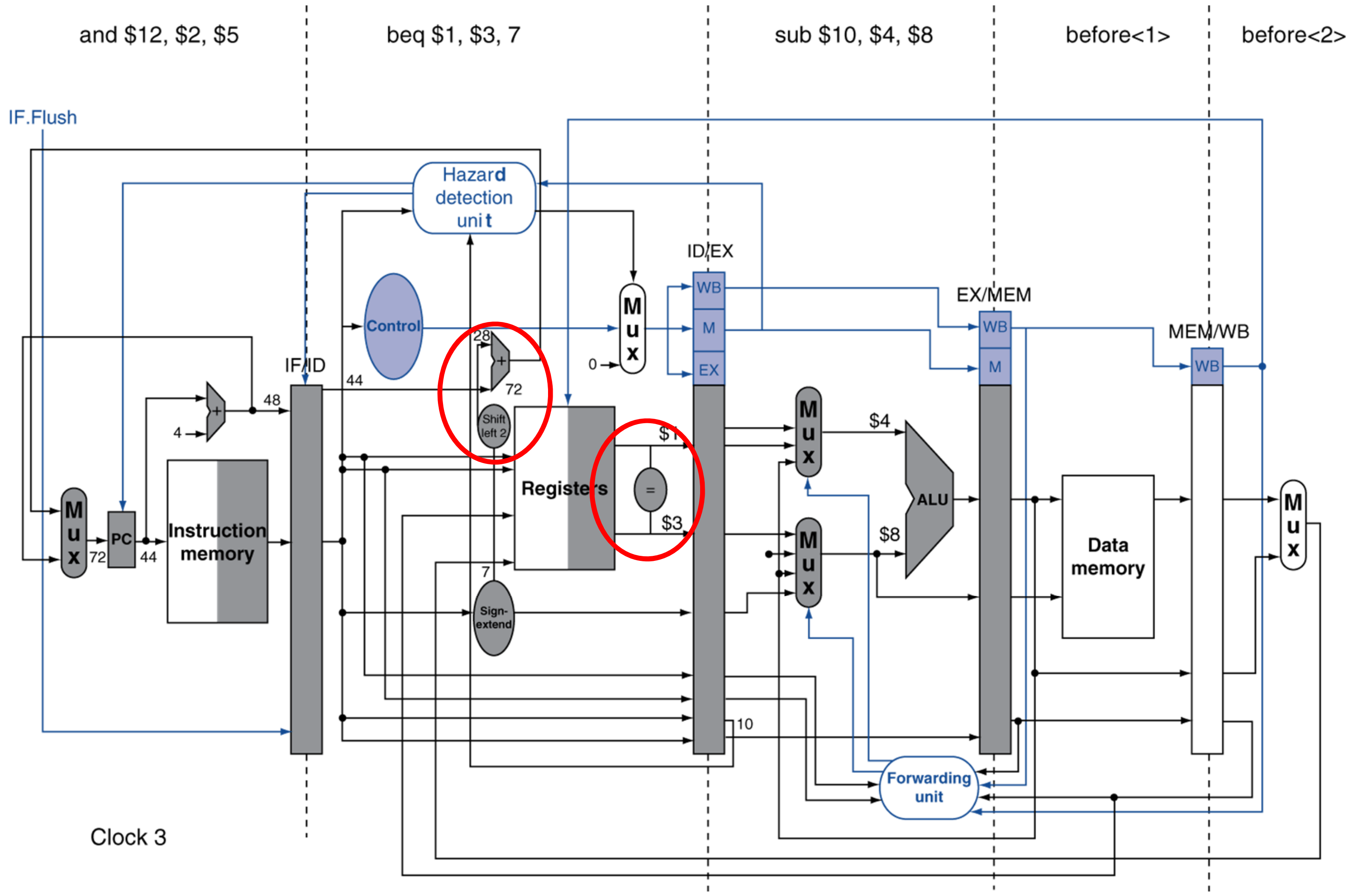


Figure 4.62-upper

[Answer: When branch taken] - Clock 4

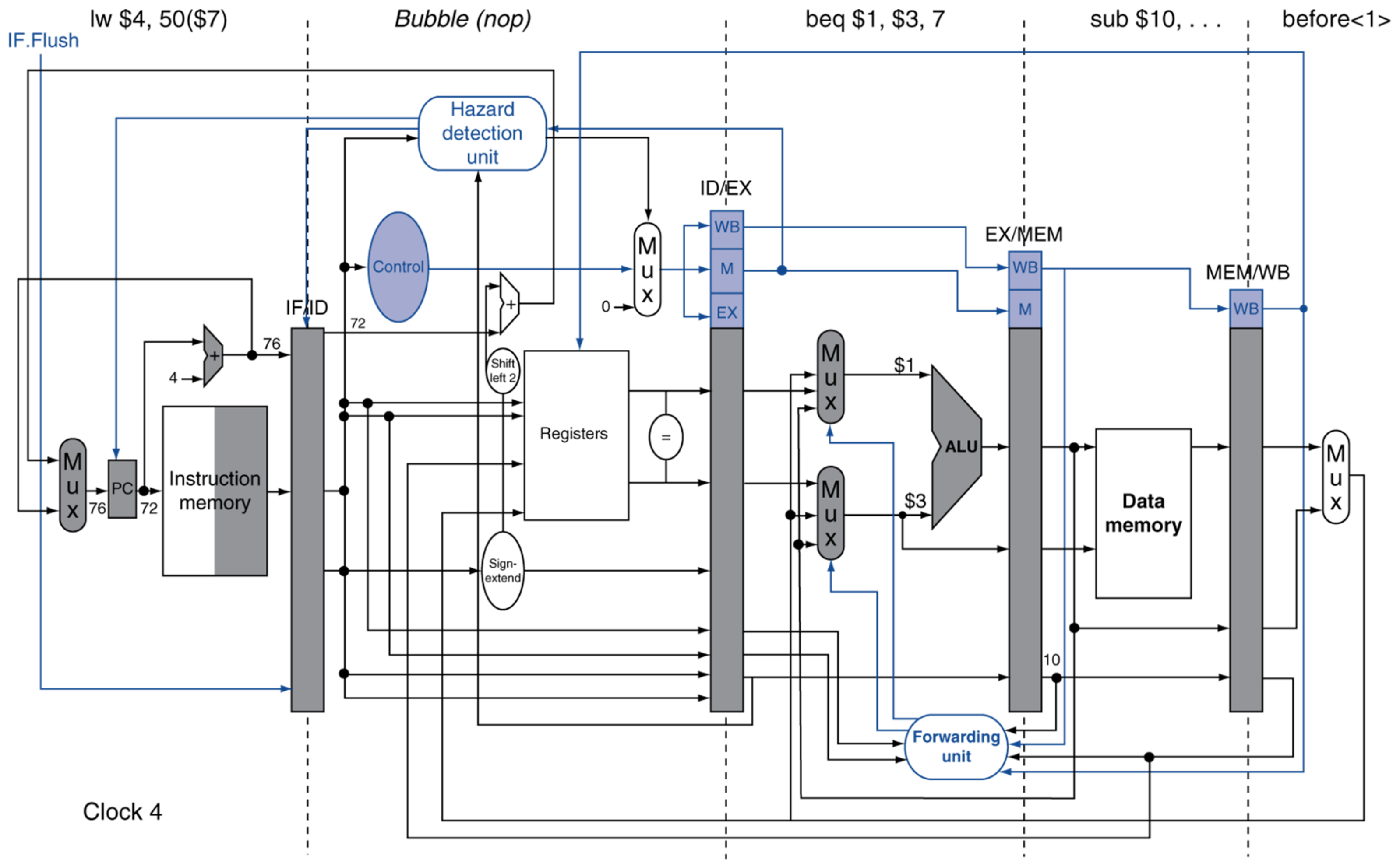


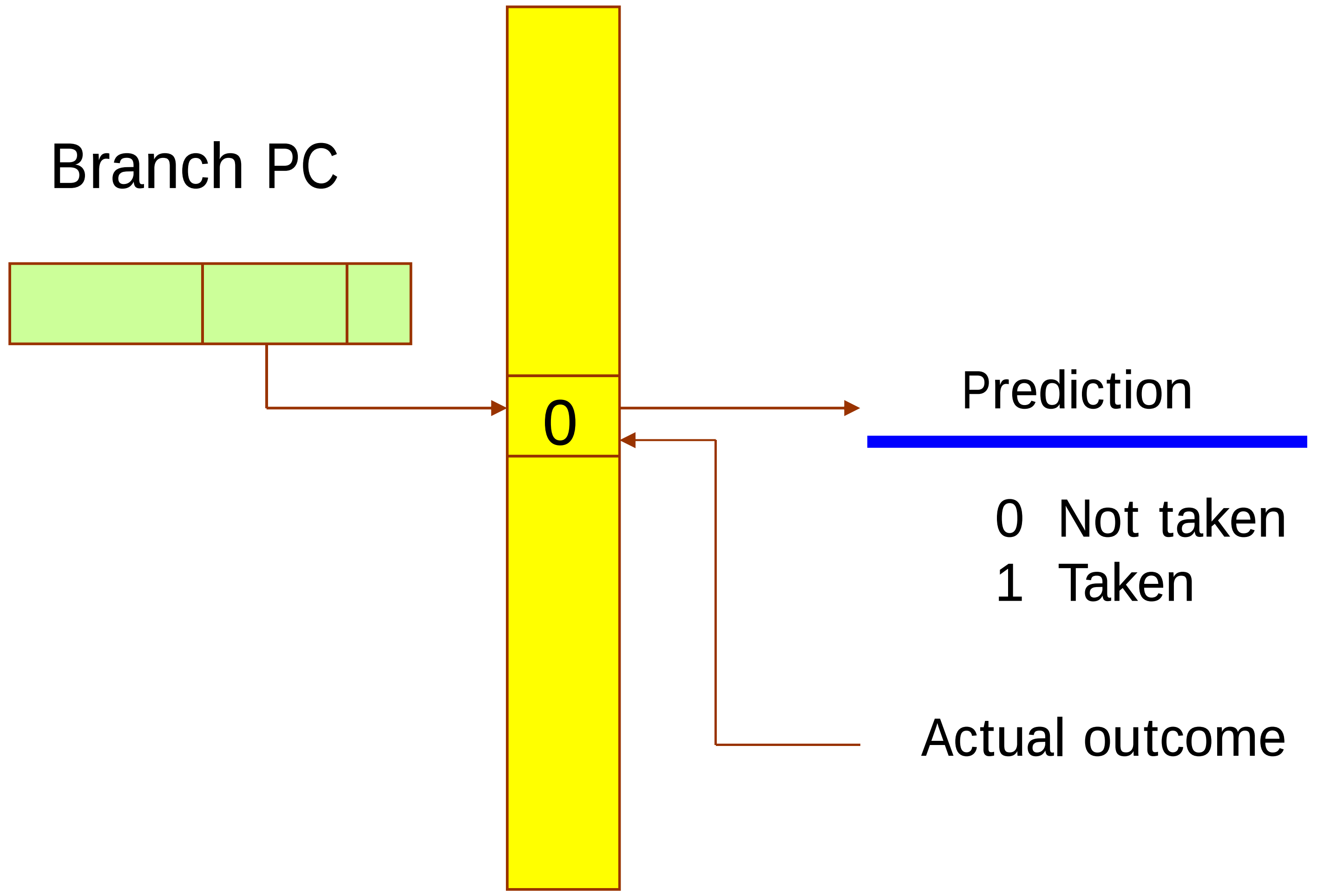
Figure 4.62-lower

Dynamic Branch Prediction

- **Dynamic branch prediction**
  - Prediction of branches at runtime using run time information
  - Look up the address of the instruction to see if a branch was taken the last time this instruction was executed
  - If so, to begin fetching new instructions from the same place as the last time
- **Branch prediction buffer (aka branch history table)**
  - Small table indexed by the lower portion of the address of the branch instruction
  - Contains a bit that says whether the branch was recently taken or not
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

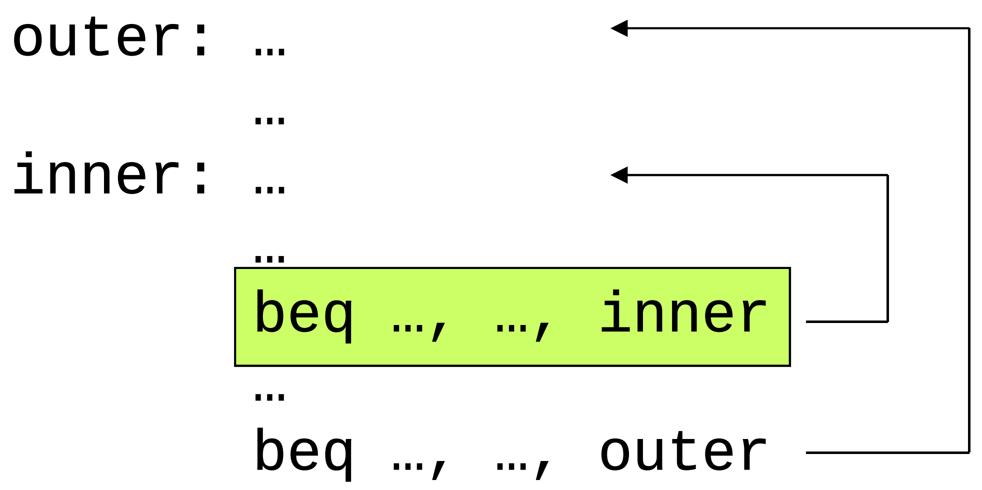
Branch History Table (BHT)

- Accessed early in the pipeline using the branch instruction PC
- Updated using the actual outcome



Example: Loops and Prediction

- **Loop branch**
  - 1 not taken after 9 taken branches
- **What is the prediction accuracy?**



```
outer: …
       …
inner: …
       …
       beq …, …, inner
       …
       beq …, …, outer
```

[Answer]

- End of loop case, when it exits instead of looping as before
- First time through loop on next time through code, when it predicts exit i nstead of looping
- Only 80% accuracy even if loop 90% of the time

2-bit Branch Prediction Scheme

- Changing prediction only if get misprediction ***twice***

[ref] "Branch prediction strategies and branch target buffer design," IEEE Computer, Vol. 17, No.1, Jan. 1984, pp.6-22.

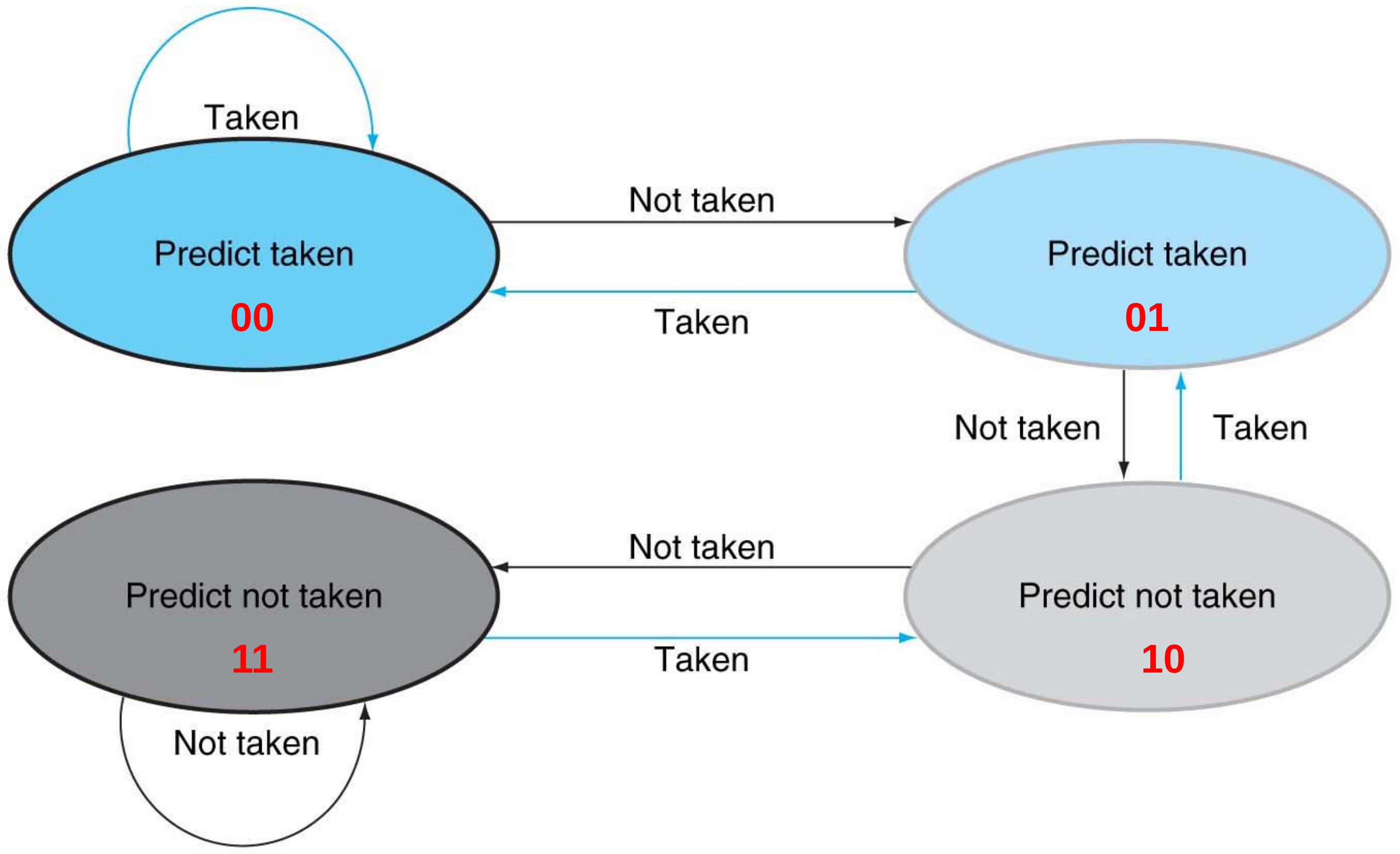
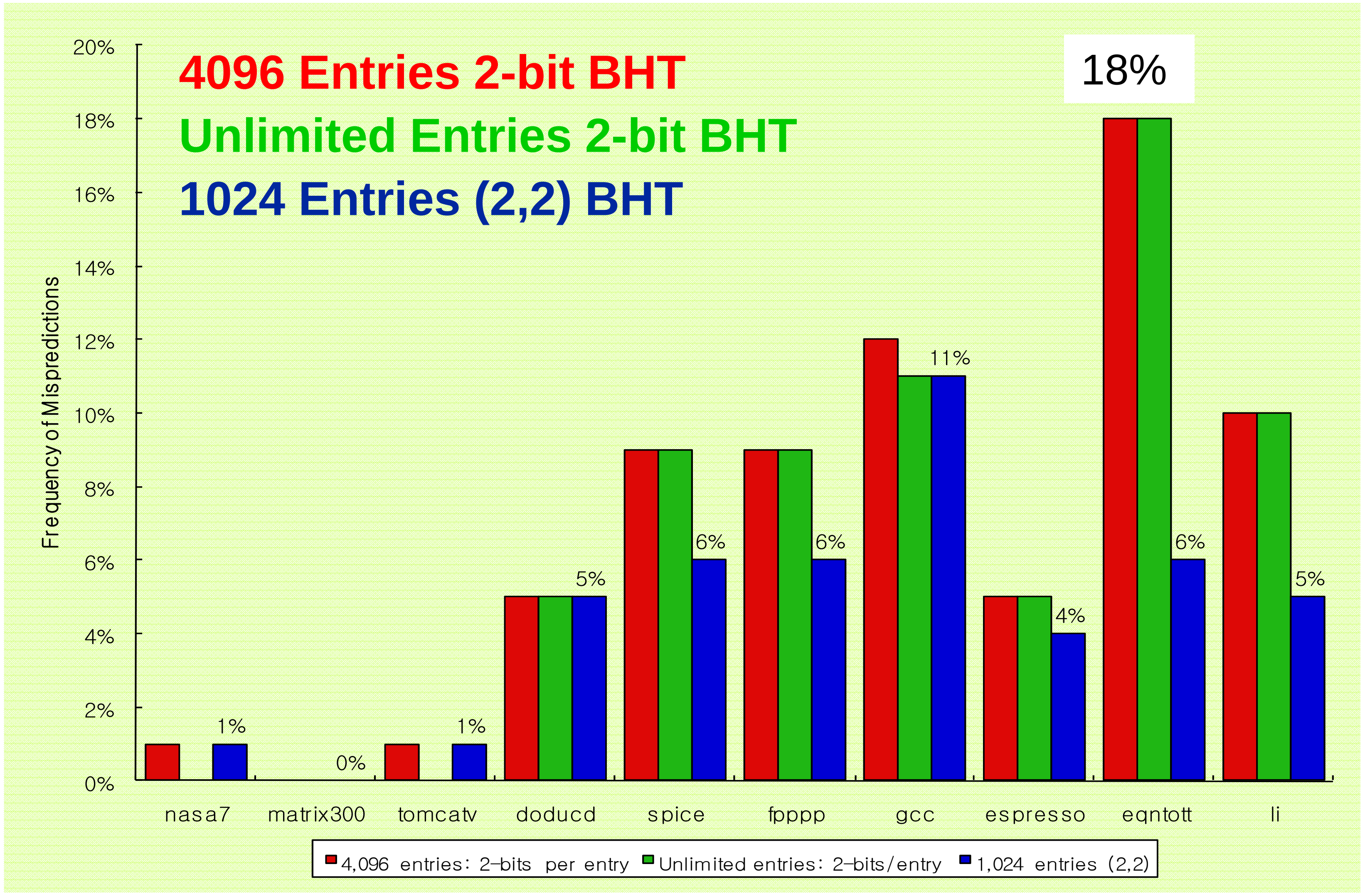


Figure 4.63

Accuracy of Different Schemes



Solution 3 – Delayed Branch

- **Delayed branch**
  - Always executes the next sequential instruction,
    with the branch taking place ***after that one instruction delay***
- **Branch delay slot**
  - The slot directly after a delayed branch instruction
  - Filled by a safe instruction
- **Delayed branch vs. dynamic branch prediction**
  - Delayed branch was a simple and effective solution for a 5-stage pipeline issuing one instruction each clock cycle.
  - As processors go to both longer pipelines and issuing multiple instructions per clock cycle, the branch delay becomes longer, and a single delay slot is insufficient.
  - Hence, delayed branch has lost popularity compared to more expensive but more flexible dynamic approaches

Supplement

Example: Prob. 2 of 2011-1 Terminal Exam

- Always-not-taken branch prediction
- Branch is executed in EX stage

```
       sub $1,$2,$3
Label: and $4,$5,$1
       lw $6,100($7)
       add $8,$9,$6
       beq $3,$3,Label
       sw $10,200($11)
       slt $12,$13,$14
       or $15,$16,$17
```

	IF	ID	EX	MEM	WB
CC0	sub				
CC1	and	sub			
CC2	lw	and	sub		
CC3	add	lw	and	sub	
CC4	beq	add	lw	and	sub
CC5	beq	add	(bubble)	lw	and
CC6	sw	beq	add	(bubble)	lw
CC7	slt	sw	beq	add	(bubble)
CC8	and	(bubble)	(bubble)	beq	add
CC9	lw	and	(bubble)	(bubble)	beq

Delayed Branch

- Assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.
- Branch is executed in MEM stage.

	IF	ID	EX	MEM	WB
CC0	sub				
CC1	and	sub			
CC2	lw	and	sub		
CC3	add	lw	and	sub	
CC4	beq	add	lw	and	sub
CC5	beq	add	(bubble)	lw	and
CC6	sw	beq	add	(bubble)	lw
CC7	slt	sw	beq	add	(bubble)
CC8	or	slt	sw	beq	add
CC9	and	(bubble)	(bubble)	sw	beq