

CSE 141:

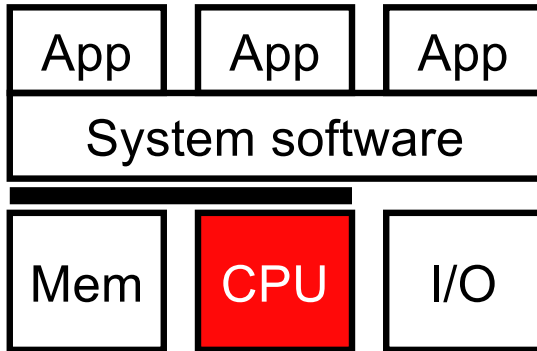
Introduction to Computer Architecture

Branch Prediction 1

Jishen Zhao (<https://cseweb.ucsd.edu/~jzhao/>)

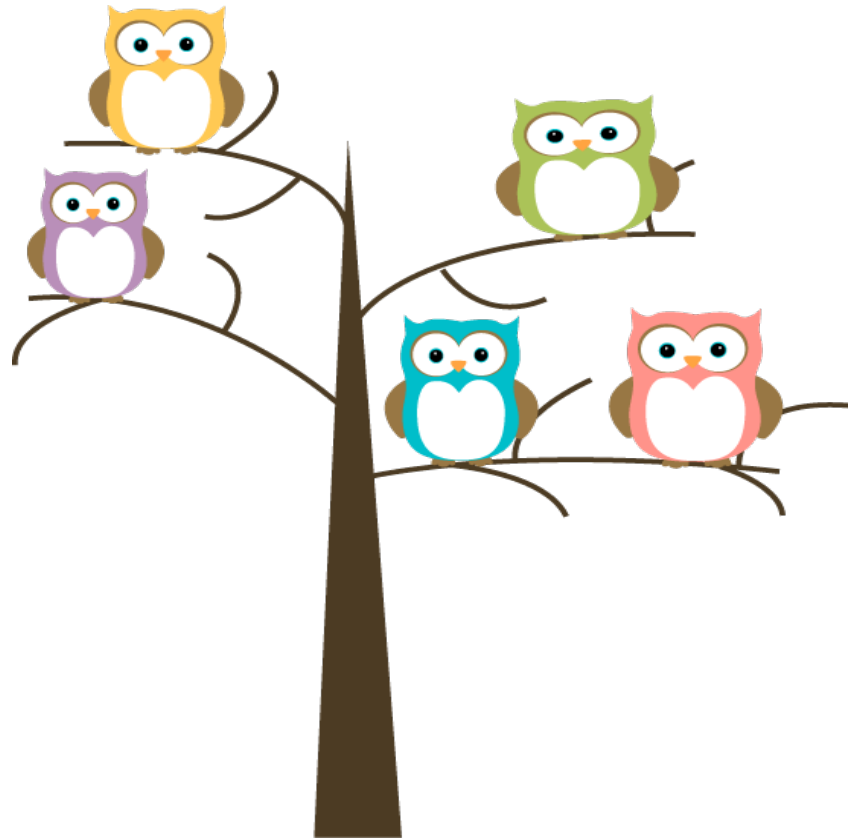
[Adapted in part from Dean Tullsen, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

What we learned



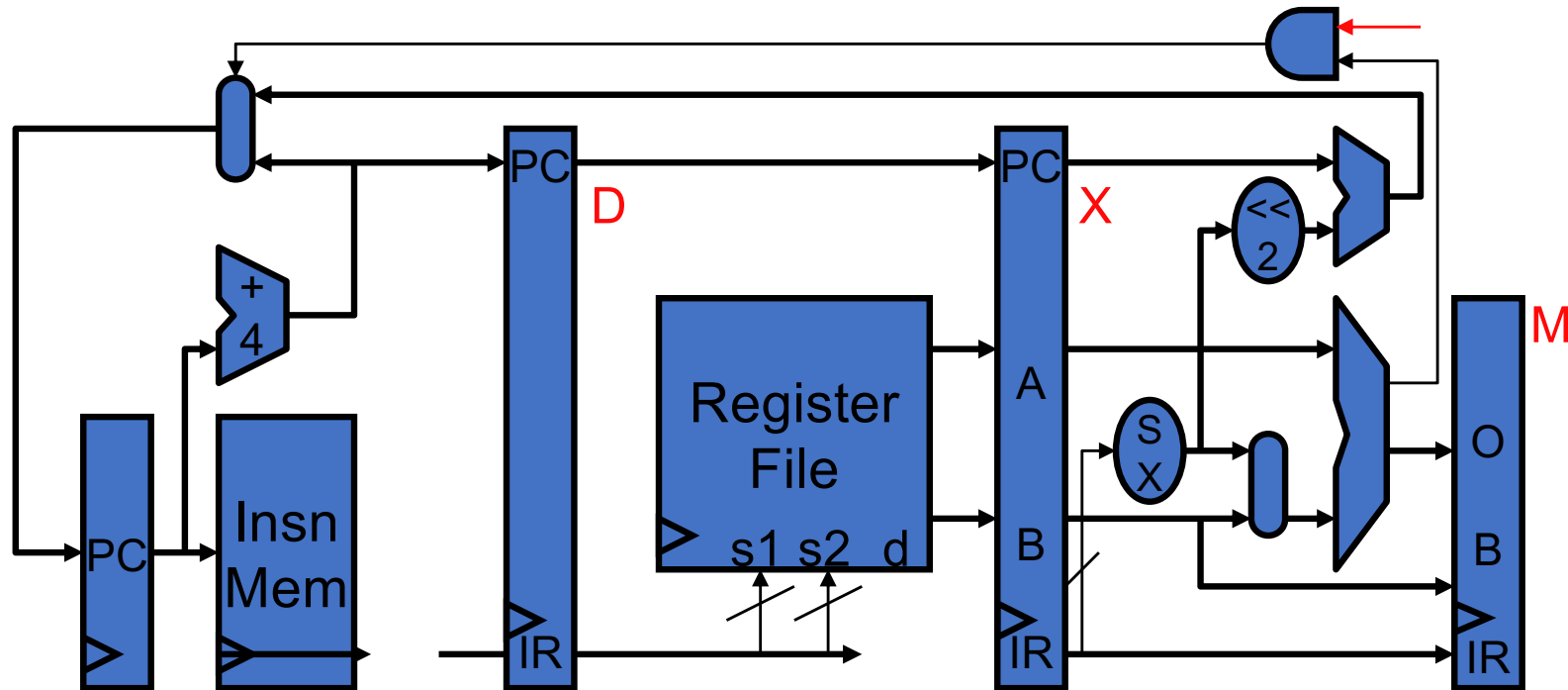
- Single-cycle datapaths ✓
- Latency vs. throughput & ✓
- performance
- Basic pipelining ✓
- Data hazards ✓
 - Bypassing
 - Load-use stalling
- Pipelined multi-cycle operations ✓
- Control hazards
 - Branch prediction

`bnez r3, target`



Control Dependences and Branch Prediction

What About Branches?



bnez r3, target

The next inst

F D X ~~M W~~

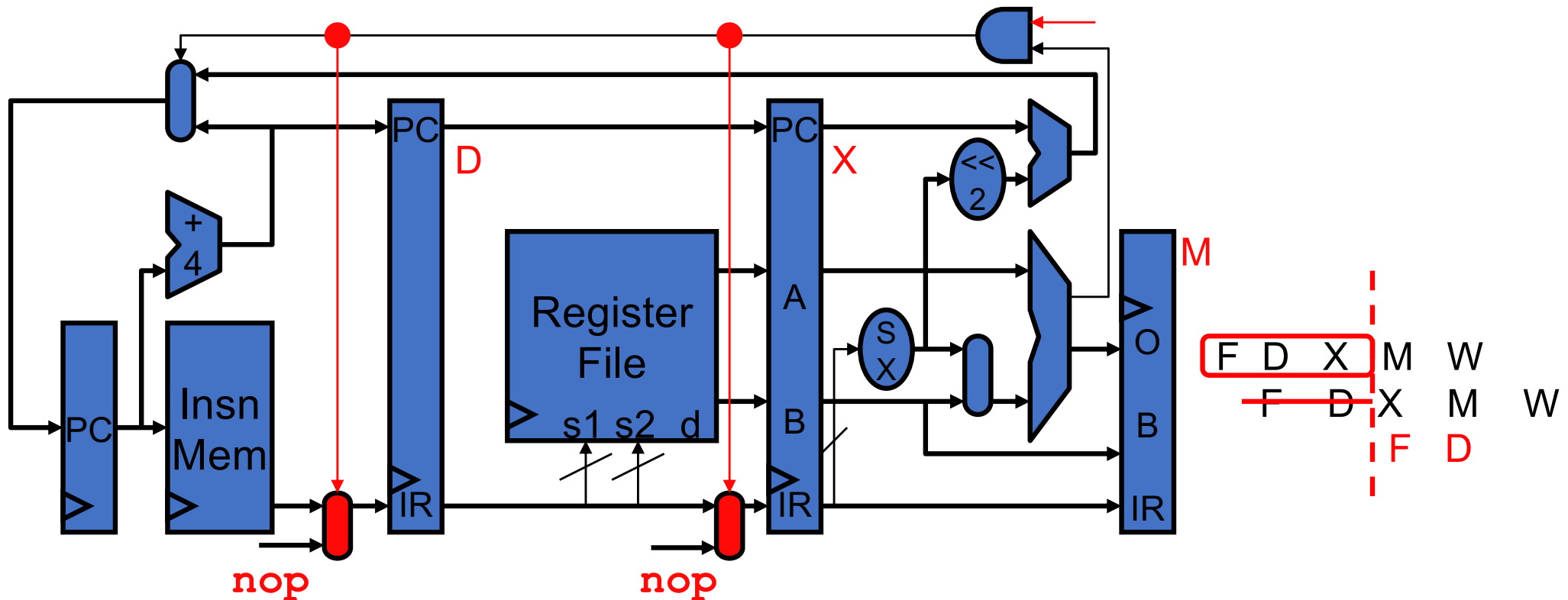
F D X M W

F D X M W

F D X M W

- Could just stall to wait for branch outcome (two-cycle penalty)
- **Branch speculation:** speculatively fetch some instruction before branch outcome is known
 - Default: assume "**not-taken**" (at fetch, can't tell it's a branch)

Branch Recovery



- **Branch recovery:** what to do when branch is actually taken
 - Insns that will be written into D and X are wrong
 - **Flush them**, i.e., replace them with **nops**
- + They haven't written permanent state yet (regfile, DMem)
 - Two cycle penalty for taken branches

Branch Speculation and Recovery

Speculation:

	1	2	3	4	5	6	7	8	9
addi r1,1→r3	F	D	X	M	W				
bnez r3,targ		F	D	X	M	W			
st r6→[r7+4]			F	D	X	M	W		
mul r8,r9→r10				F	D	P0	P1	P2	...

speculative

- **Mis-speculation recovery**: what to do on wrong guess
 - Not too painful in a short, in-order pipeline
 - Branch resolves in X
 - + Younger insns (in F, D) haven't changed permanent state
 - **On next cycle, flush** insns in D and X → **2 cycle overhead**

Recovery:

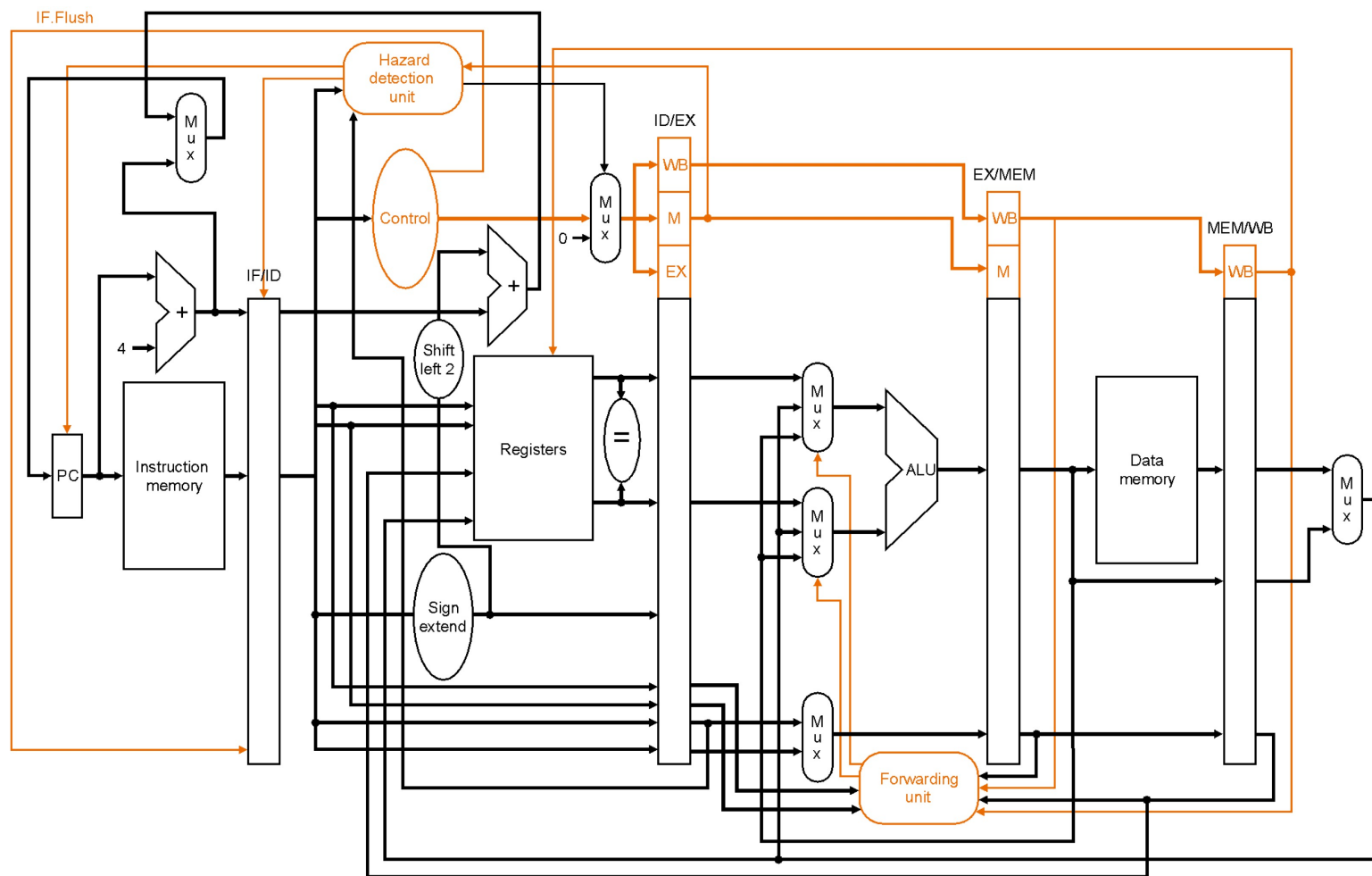
	1	2	3	4	5	6	7	8	9
addi r1,1→r3	F	D	X	M	W				
bnez r3,targ		F	D	X	M	W			
st r6→[r7+4]			F	D	--	--	--		
mul r8,r9→r10				F	--	--	--	--	
targ:add r4,r5→r4					F	D	X	M	W

Exercise: Branch Performance

- Back of the envelope calculation
 - **Branch: 20%**, load: 20%, store: 10%, other: 50%
 - Speculation: not-taken
 - But actually, **75% of branches are taken**
- $\text{CPI} = ?$ (assuming $\text{CPI}_{\text{base}} = 1$)
- $\text{CPI} = 1 + 20\% * 75\% * 2 = 1.3$
- **Branches cause 30% slowdown**
 - Worse with deeper pipelines (higher misprediction penalty)
- Can we do better than assuming branch is not taken?

One-cycle branch misprediction penalty

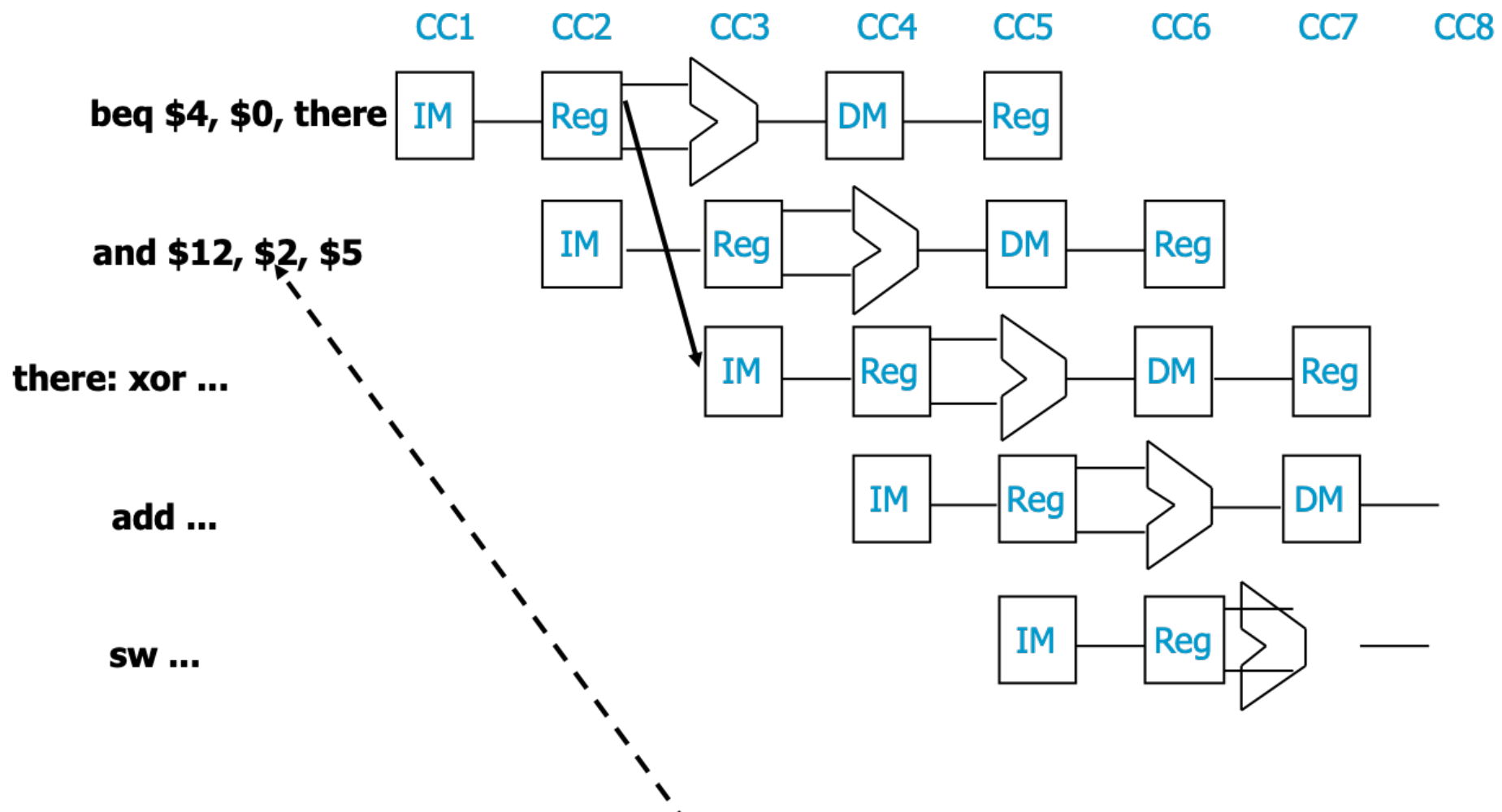
- Target computation & equality check in instruction decode (ID) phase
 - This figure also shows flushing hardware



Eliminating the Branch Stall

- There's no rule that says we have to branch immediately. We could wait an extra instruction before branching
- The original SPARC and MIPS processors used a **branch delay slot** to eliminate single-cycle stalls after branches
- The instruction after a branch is always executed in those machines, whether the branch is taken or not!

Branch delay slot



Branch delay slot instruction (next instruction after a branch) is executed even if the branch is taken.

Filling the branch delay slot

- The branch delay slot is only useful if you can find something to put there
 - Need earlier instruction that doesn't affect the branch
- If you can't find anything, you must put a nop to ensure correctness
- Worked well for early RISC machines
 - Doesn't help recent processors much
 - E.g. MIPS R10000, has a 5-cycle branch penalty, and executes 4 instructions per cycle
- Meanwhile, delayed branch is a permanent part of the ISA

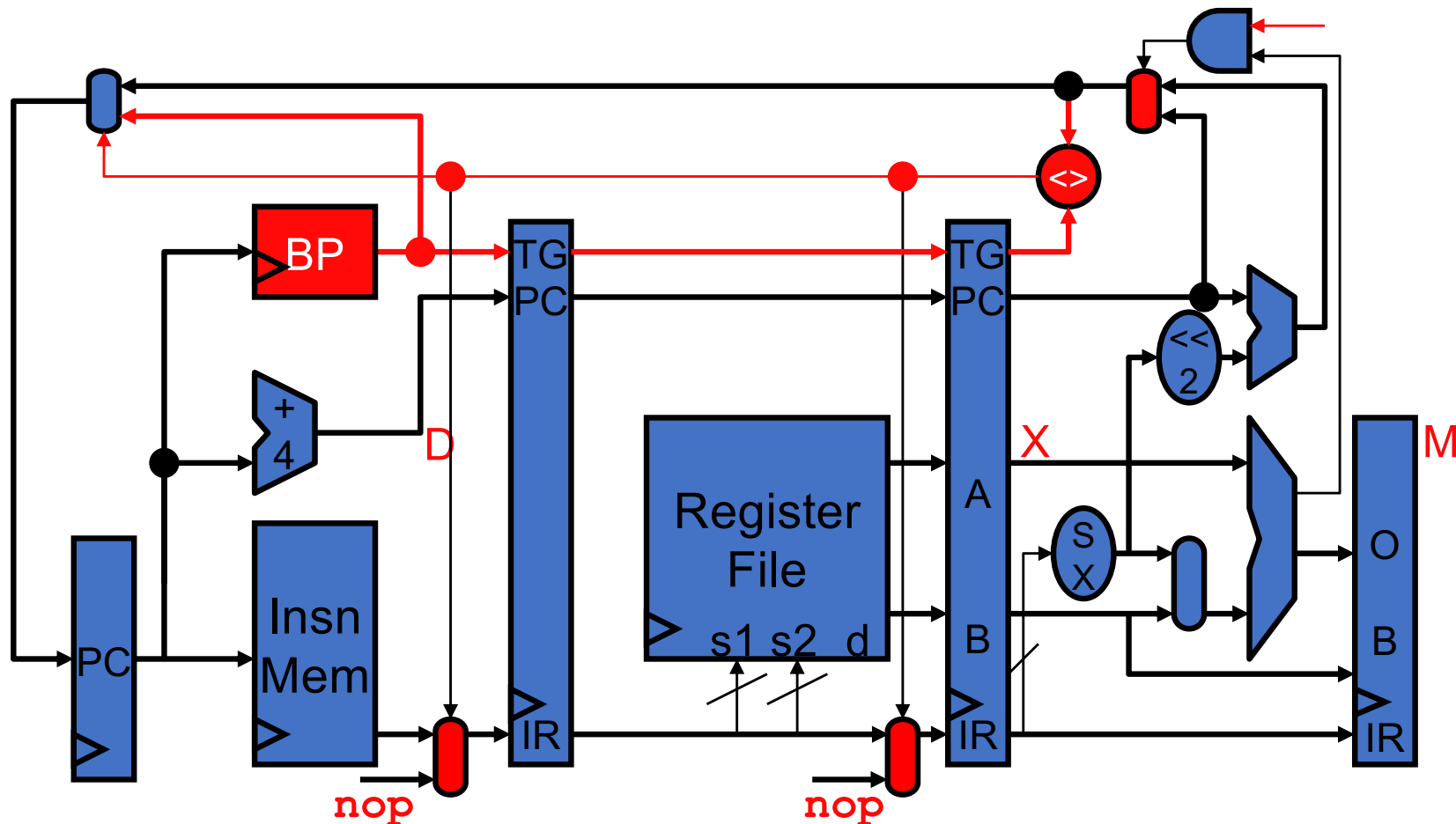
Big Idea: Speculative Execution

- **Speculation:** “risky transactions on chance of profit”
- **Speculative execution**
 - Execute before all parameters known with certainty
 - **Correct speculation**
 - + Avoid stall, improve performance
 - **Incorrect speculation (mis-speculation)**
 - Must abort/flush/squash incorrect insns
 - Must undo incorrect changes (recover pre-speculation state)
- **“Control speculation”:**
 - speculation aimed at control hazards

Control Speculation Mechanics

- Guess branch target, start fetching at guessed position
 - Doing nothing is implicitly guessing target is PC+length_in_bytes (e.g., PC+4 for 32-bit instructions)
 - Can actively guess other targets: **dynamic branch prediction**
- Execute branch to verify (check) guess
 - Correct speculation? keep going
 - Mis-speculation? Flush mis-speculated insns
 - Hopefully haven't modified permanent state (Regfile, DMem)
 - + Happens naturally in in-order 5-stage pipeline

Dynamic Branch Prediction



- **Dynamic branch prediction:** hardware guesses outcome
 - Start fetching from guessed address
 - Flush on **mis-prediction**

Exercise: Branch Prediction Performance

- Parameters
 - **Branch: 20%**, load: 20%, store: 10%, other: 50%
 - 75% of branches are taken
- Dynamic branch prediction
 - Branches predicted with 95% accuracy, i.e., 5% mis-prediction
 - CPI = ?

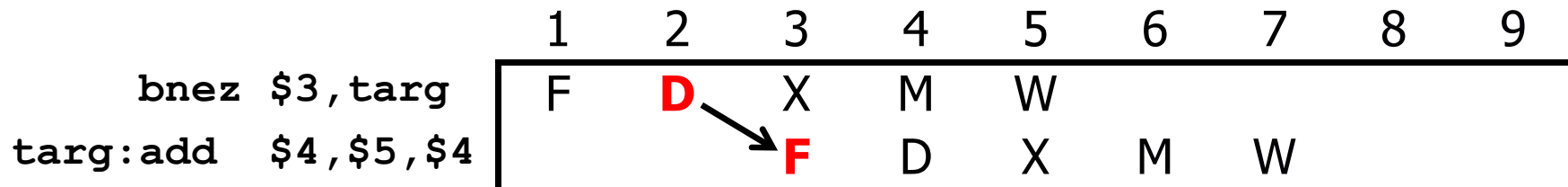
Exercise: Branch Prediction Performance

- Parameters
 - **Branch: 20%**, load: 20%, store: 10%, other: 50%
 - 75% of branches are taken
- Dynamic branch prediction
 - Branches predicted with 95% accuracy, i.e., 5% mis-prediction
 - CPI = ?
 - $\text{CPI} = 1 + 20\% * 5\% * 2 = \mathbf{1.02}$

When to perform branch prediction?

❑ During Decode

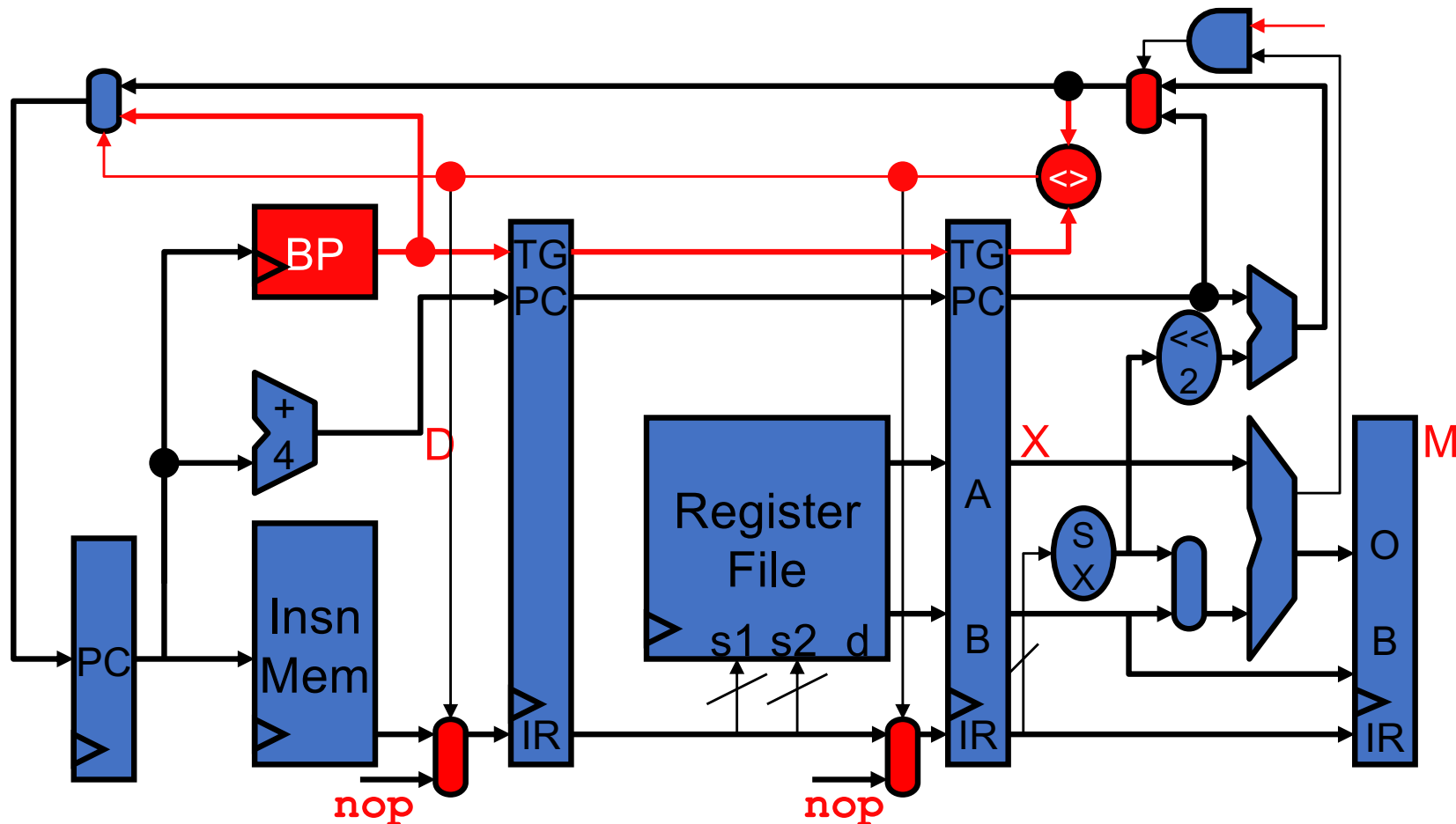
- Look at instruction opcode to determine branch instructions
- Can calculate next PC from instruction (for PC-relative branches)
- One cycle “mis-fetch” penalty **even if branch predictor is correct**



❑ During Fetch?

- How do we do that?
- Branch predictor locates at F stage

Where is branch predictor (BP)?

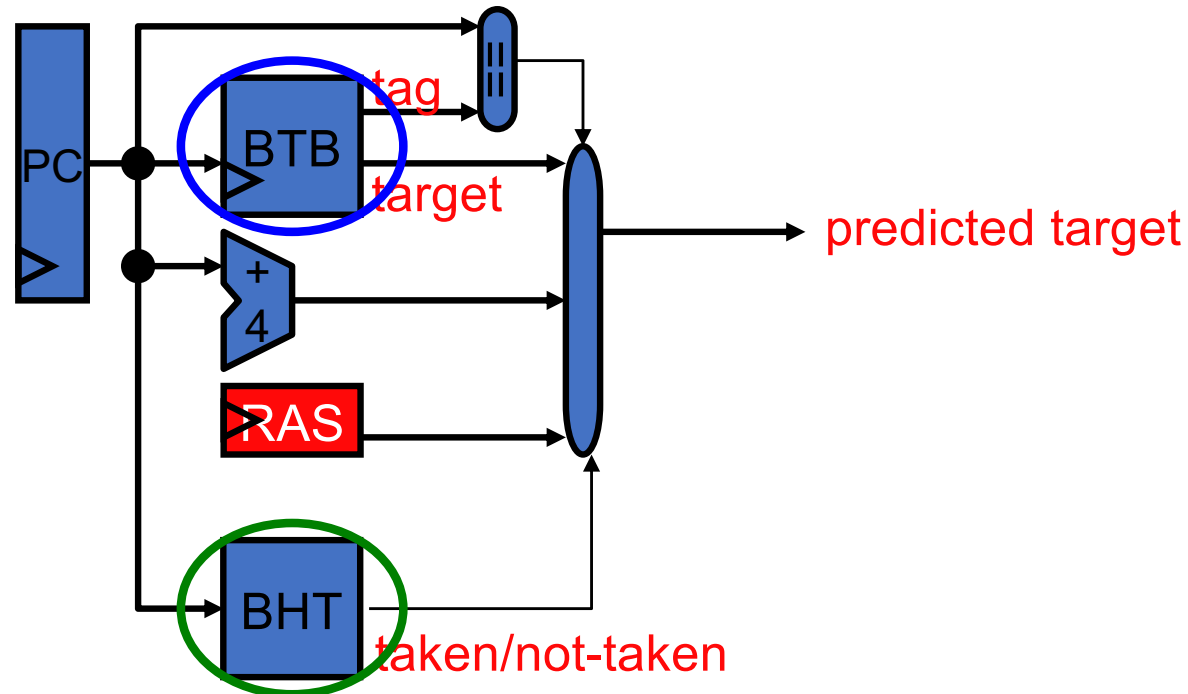


- **Dynamic branch prediction:** hardware guesses outcome
 - Start fetching from guessed address
 - Flush on **mis-prediction**

Branch predictor

What's inside a branch predictor?

- BTB & branch direction predictor during fetch



- Step #1: is it a branch? BTB (branch target buffer)
- Step #2: is the branch taken or not taken? BHT (branch history table)
- Step #3: if the branch is taken, where does it go? BTB, RAS (return address stack)

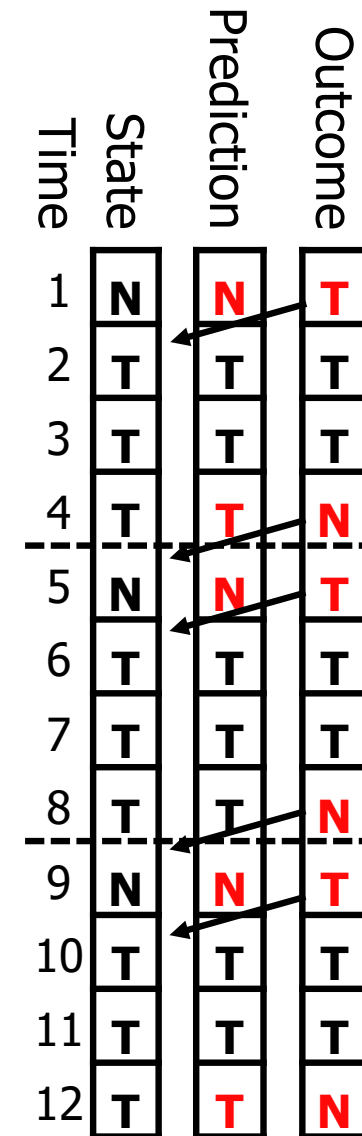
One-bit history-based branch prediction

- **Branch history table (BHT):**
simplest direction predictor
 - PC indexes table of bits (0 = N, 1 = T)
 - Essentially: branch will go same way it went last time
 - Problem: **inner loop branch** below

```
for (i=0; i<100; i++)
    for (j=0; j<3; j++)
        // loop body
```

 - Two “built-in” mis-predictions per inner loop iteration
 - Branch predictor “changes its mind too quickly”

Time	State	Prediction	Outcome
1	N	N	T
2	T	T	T
3	T	T	T
4	T	T	N
5	N	N	T
6	T	T	T
7	T	T	T
8	T	T	N
9	N	N	T
10	T	T	T
11	T	T	T
12	T	T	N



Two-bit history-based branch prediction

- **Two-bit saturating counters (2bc)**

[Smith 1981]



















- Replace each single-bit prediction
 - $(0,1,2,3) = (N,n,t,T)$
- Adds "hysteresis"
 - Force predictor to mis-predict twice before "changing its mind"
- One misprediction each loop execution (rather than two)
 - + Fixes this pathology (which is not contrived, by the way)
- Can we do even better?





Time	State	Prediction	Outcome
1	N	N	T
2	n	N	T
3	t	T	T
4	T	T	N
5	t	T	T
6	T	T	T
7	T	T	T
8	T	T	N
9	t	T	T
10	T	T	T
11	T	T	T
12	T	T	N


2-bit Branch Prediction Concept - Example

Edgar Bernal - Wednesday, November 9, 2016

8:44 PM

Time	State		Prediction	Outcome	Result?	
1	N		N	T	Wrong	
2	n		N	T	Wrong	
3	t		T	T	Correct	
4	T		T	N	Wrong	
5	t		T	T	Correct	
6	T		T	T	Correct	
7	T		T	T	Correct	
8	T		T	N	Wrong	
9	t		T	T	Correct	

N	
n	
t	
T	

Correct:	
Wrong:	