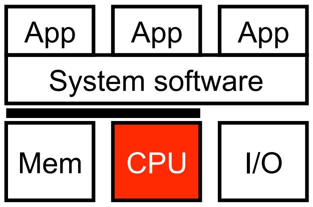
CMPE 110: Computer Architecture

Week 4
Pipelining IV

Jishen Zhao (http://users.soe.ucsc.edu/~jzhao/)

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

Pipelining



- Single-cycle datapaths vs. pipelined datapath
  - Basic pipelining: F, D, X, M, W
  - Base CPI = 1
  - Pipeline diagram (table)
- Data hazards
  - **Stalling**
  - Bypassing (forwarding)
- Structural hazards
  - **Stalling**
  - Add more hardware resources
- Multi-cycle operations
- Control hazards
  - Branch prediction

Review: 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	…

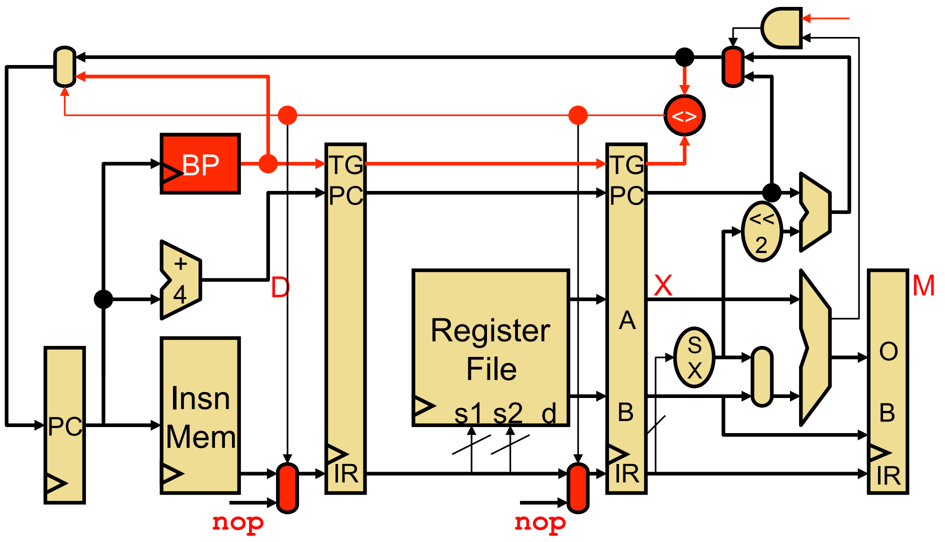
speculative

- **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

Review: Dynamic Branch Prediction



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

When to perform branch prediction?

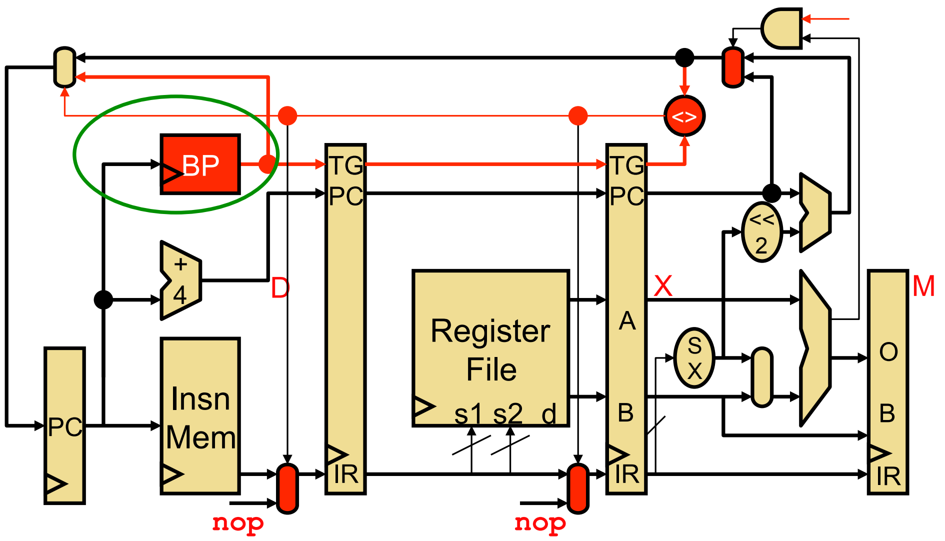
- During Decode
  - Look at instruction opcode to determine branch instructions
  - Can calculate next PC from instruction (for PC-relative branches)
  - One cycle "mis-fetch" penalty **even if branch predictor is correct**

	1	2	3	4	5	6	7	8	9
`bnez $3,targ`	F	**D**	X	M	W				
`targ:add $4,$5,$4`			**F**	D	X	M	W		

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

Branch predictor

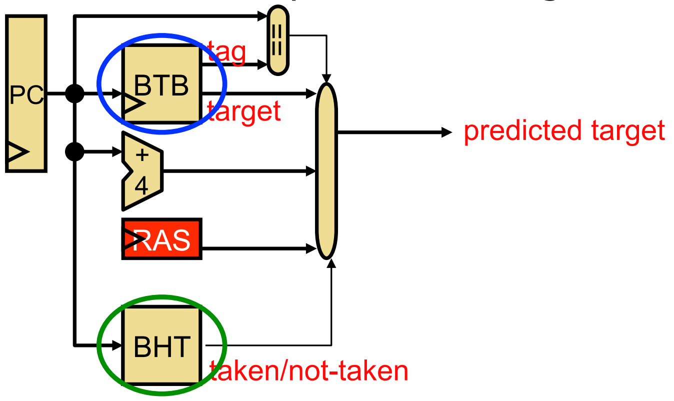
Where is branch predictor (BP)?



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

What's inside a branch predictor?

- BTB & branch direction predictor during fetch



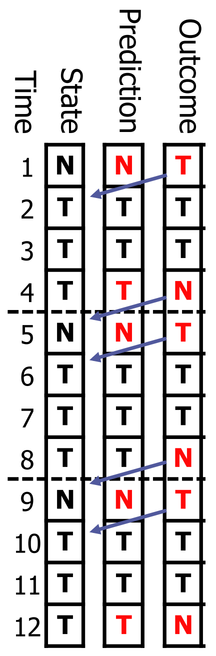
- Step #1: is it a branch? BTB
- Step #2: is the branch taken or not taken? BHT
- Step #3: if the branch is taken, where does it go? BTB, RAS

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
```

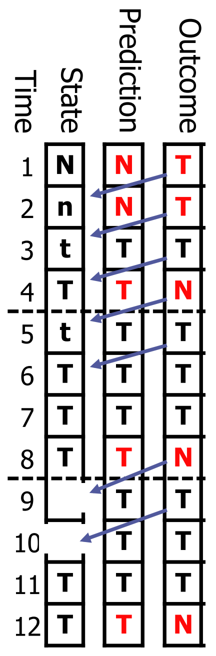
    – Two "built-in" mis-predictions per inner loop iteration
    – Branch predictor "changes its mind too quickly"
    – 6 wrong predictions



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

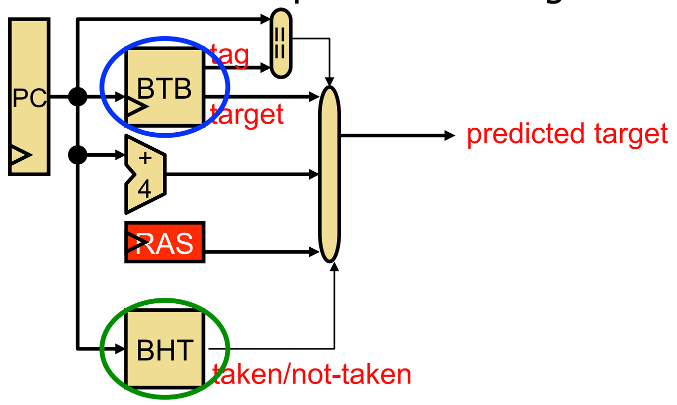
- **Two-bit saturating counters (2bc)** [Smith 1981]
  - Replace each single-bit prediction
    - (0,1,2,3) = (N,n,t,T)
  - Adds "hysteresis"
    - Force predictor to mis-predict twice before "changing its mind"
  - One misprediction each loop execution (rather than two)
    + Fixes this pathology (which is not contrived, by the way)
    - Can we do even better?
    - 5 wrong predictions



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
10		T	T
11	T	T	T
12	T	T	N

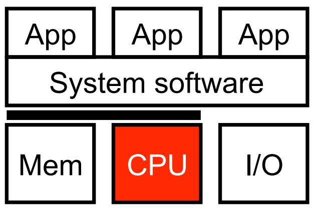
Summary: inside a branch predictor?

- BTB & branch direction predictor during fetch



- Step #1: is it a branch? BTB
- Step #2: is the branch taken or not taken? BHT
- Step #3: if the branch is taken, where does it go? BTB, RAS

Summary of pipelining



- Single-cycle datapaths vs. pipelined datapath
  - Basic pipelining
- Data hazards
- Structural hazards
- Multi-cycle operations
- Control hazards
- Fine-grained multithreading

Any other methods on handling data and control hazards?

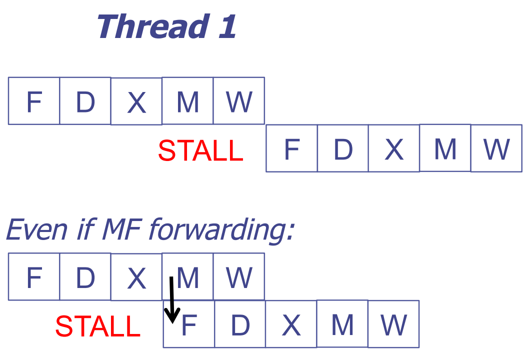
Do something else: fine-grained multithreading

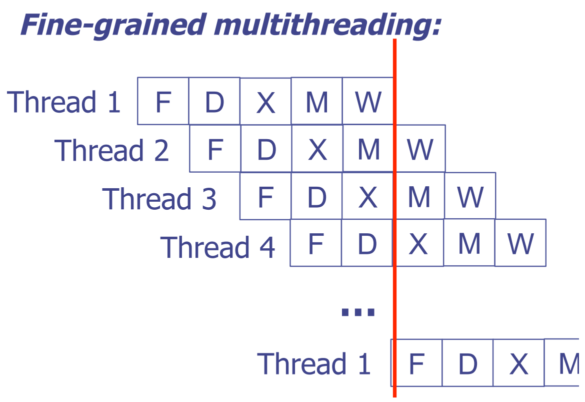
Fine-Grained Multithreading

- Idea: Hardware has multiple thread contexts (hardware threads). Each cycle, fetch an instruction from a different thread. → the same pipeline, multiple threads
  - By the time the fetched branch instruction resolves, no instruction is fetched from the same thread
  - But…Branch instruction resolution latency can overlap with execution of other threads' instructions

Fine-Grained Multithreading

```
        Thread 1               Thread 2                Other threads…
     bne R3, R6, L3       add R10<- R5, R8        Has no dependency
     addi R1<- R1, #1     addi R10<- R10, #1
L3:  add R1<- R2, R4
```



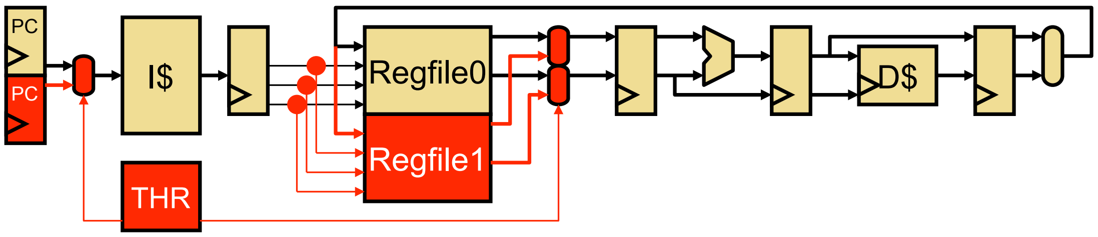


Pros and cons

- Advantages
  - + No need for dependency checking between instructions (only one instruction in pipeline from a single thread)
  - + No need for branch prediction logic
  - + Otherwise-stall cycles used for executing useful instructions from different threads
  - + Improved system throughput and pipeline utilization
- Disadvantages
  - - Extra hardware complexity: multiple hardware contexts (PCs, register files, …), thread selection logic
  - - Reduced single thread performance (one instruction fetched every N cycles from the same thread)
  - - Resource contention between threads in caches and memory
  - - Some dependency checking logic *between* threads remains (load/store)

Hardware Multithreading

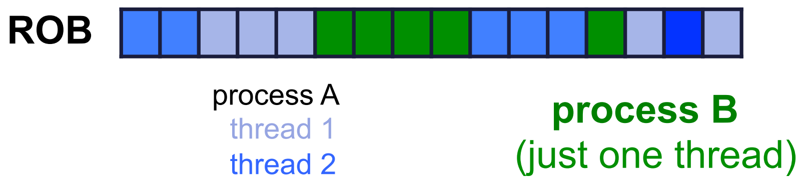
- **Not** the same as software multithreading!
- A **hardware thread** is a sequential stream of insns
  - could be a software thread or a single-threaded process



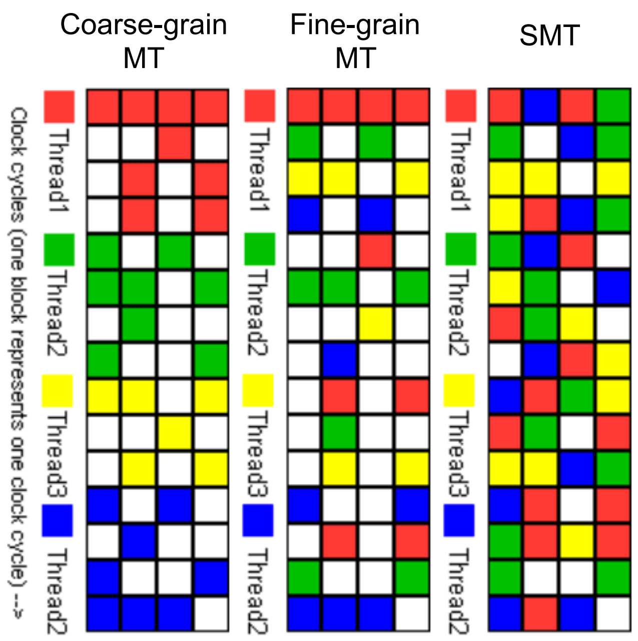
- **Hardware Multithreading (MT)**
  - Multiple hardware threads dynamically share a single pipeline
  - Replicate only per-thread structures: program counter & registers
  - Hardware interleaves instructions

Hardware Multithreading

- Why use hw multithreading?
  - + **Multithreading improves utilization and throughput**
    - Single programs utilize <50% of pipeline (branch, cache miss)
    - allow insns from different hw threads in pipeline at once
  - **Multithreading does not improve single-thread performance**
    - Individual threads run as fast or even slower
  - **Coarse-grain MT**: switch on cache misses   Why (will discuss later)?
  - **Simultaneous MT**: no explicit switching, fine-grain interleaving
    - Simultaneous multithreading (SMT): Issue multiple instructions from multiple threads in one cycle. The processor must be superscalar to do so.
    - Intel's "hyperthreading"



Hardware Multithreading


Coarse-grain MT
Fine-grain MT
SMT
Thread1
Thread2
Thread3
Thread2
Clock cycles (one block represents one clock cycle) -->