Computer Science 146
Computer Architecture

Fall 2019

Harvard University

Instructor: Prof. David Brooks
dbrooks@eecs.harvard.edu

Lecture 7: Dynamic Branch Prediction

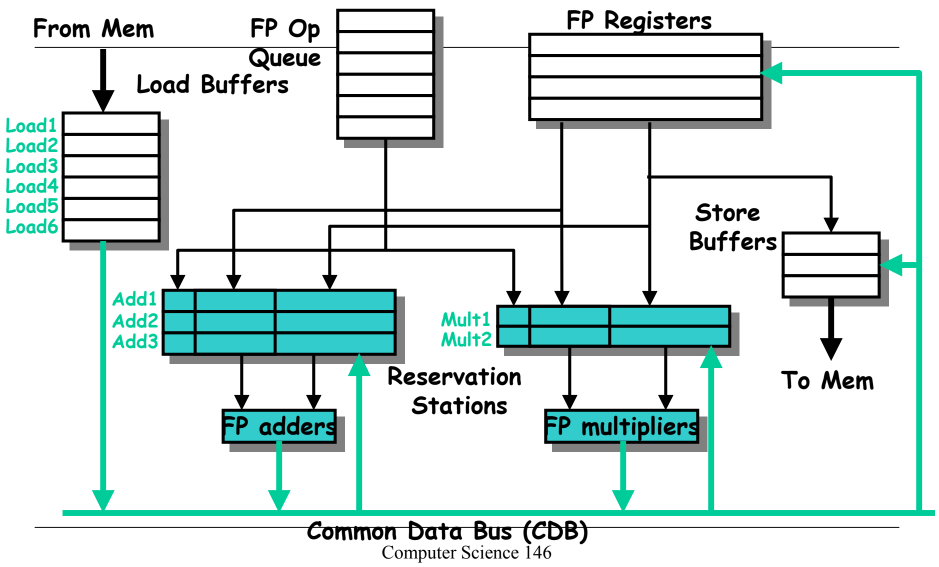
Lecture Outline

- Tomasulo's Algorithm Review (3.1-3.3)
- Pointer-Based Renaming (MIPS R10000)
- Dynamic Branch Prediction (3.4)
  - Yeh + Patt Paper
- Other Front-end Optimizations (3.5)
  - Branch Target Buffers/Return Address Stack

Tomasulo Review

- Reservation Stations
  - Distribute RAW hazard detection
  - Renaming eliminates WAW hazards
  - Buffering values in Reservation Stations removes WARs
  - Tag match in CDB requires many associative compares
- Common Data Bus
  - Achilles heal of Tomasulo
  - Multiple writebacks (multiple CDBs) expensive
- Load/Store reordering
  - Load address compared with store address in store buffer

Tomasulo Organization



Tomasulo Review

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LD F0, 0(R1)	Iss	M1	M2	M3	M4	M5	M6	M7	M8	Wb										
MUL F4, F0, F2		Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Ex	Ex	Ex	Ex	Wb					
SD 0(R1), F0			Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	M1	M2	M3	Wb	
SUBI R1, R1, 8				Iss	Ex	Wb														
BNEZ R1, Loop					Iss	Ex	Wb													
LD F0, 0(R1)						Iss	Iss	Iss	Iss	M	Wb									
MUL F4, F0, F2							Iss	Iss	Iss	Iss	Iss	Ex	Ex	Ex	Ex	Wb				
SD 0(R1), F0								Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	Iss	M1	M2	M3	Wb
SUBI R1, R1, 8									Iss	Ex	Wb									
BNEZ R1, Loop										Iss	Ex	Wb								
LD F0, 0(R1)											Iss	M1	M2	M3	M4	M5	M6	M7	M8	Wb
MUL F4, F0, F2																Iss	Iss	Iss	Iss	Iss
SD 0(R1), F0																	Iss	Iss	Iss	Iss

Register Renaming: Pointer-Based

- MIPS R10K, Alpha 21264, Pentium 4, POWER4
- Mapper/Map Table: Hardware to hold these mappings
  - Register Writes: Allocate new location, note mapping in table
  - Register Reads: Look in map table, find location of most recent write
- Deallocate mappings when done

Register Renaming: Example

- Mapper/Map Table: Hardware to hold these mappings
  - Register Writes: Allocate new location, note mapping in table
  - Register Reads: Look in map table, find location of most recent write
- Deallocate mappings when done

- Assume
  - 4 Architected/Logical Registers (F1,F2,F3,F4) “names”
  - 8 Physical/Rename Registers (P1—P8) “locations”
- Code – Lots of Potential WAR/WAW, also RAWs

```
ADD     R1, R2, R4
SUB     R4, R1, R2
ADD     R3, R1, R3
ADD     R1, R3, R2
```

Register Renaming: Example

Initial Mapping

```
ADD     R1, R2, R4
SUB     R4, R1, R2
ADD     R3, R1, R3
ADD     R1, R3, R2
```

Map Table			
R1	R2	R3	R4
P1	P2	P3	P4
P5	P2	P3	P4
P5	P2	P3	P6
P5	P2	P7	P6
P8	P2	P7	P6

```
ADD     P5, P2, P4
SUB     P6, P5, P2
ADD     P7, P5, P3
ADD     P8, P7, P2
```

Control Hazards

- Key to performance in current microprocessors
- Almost every design decision changes if we assume “perfect” rather than realistic branch prediction

Strategies to reduce control hazards

- Compiler techniques reduce branch frequency
- Hardwired strategies for responding to branches – “assume not taken”
- Delayed branches
- Nullifying branches
- Compiler hints to suggest likely outcomes
- Dynamic hardware branch prediction

Compiler techniques to reduce branch frequency

- Loop unrolling
  - Will discuss in detail in Chapter 4
- Constant propagation

```
N=0;
…
~~A=b*N;~~ A=0;
…
~~If~~(A==0) {
}
```

- Procedure inlining/cloning

```
foo(int i) {
  return(2*i);
}
a=foo(b);
Inlining => a=2*b;
```

Branch prediction methods

- When is information about branches gathered/applied?
  - When the machine is designed
  - When the program is compiled ("compile-time") (ch.4)
  - When a "training run" of the program is executed ("profile-based")
  - As the program is executing ("dynamic")

Why predict? Speculative Execution

- Execute *beyond* branch boundaries before the branch is resolved
- Correct Speculation
  - Avoid stall, result is computed early, performance++
- Incorrect Speculation
  - Abort/squash incorrect instructions, complexity+
  - Undo any incorrect state changes, complexity++
- Performance gain is weighed vs. penalty
- Speculation accuracy = branch prediction accuracy

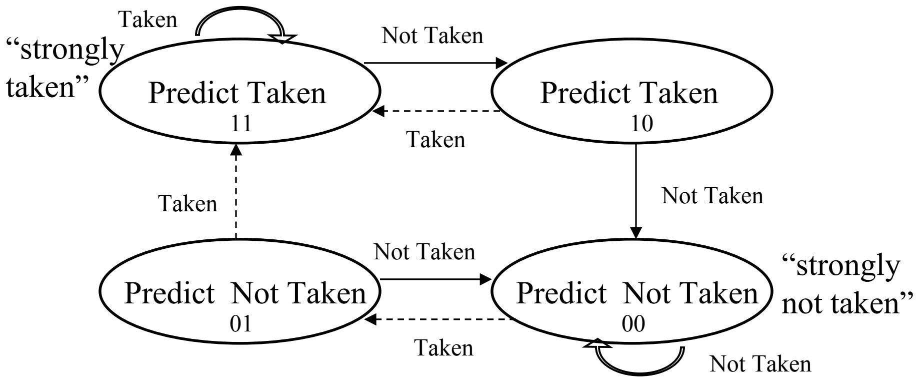
Dynamic Hardware Branch Prediction

- Branch behavior is monitored during program execution
  - History data can influence prediction of future executions of the branch instruction
- Branches instruction execution has two tasks/predictions
  - Condition evaluation (taken or not-taken)
  - Target address calculation (where to go when taken)
- Target prediction also applies to unconditional branches
- Branch Direction Prediction: 3 levels of complexity
  - Branch history tables, Two-level tables, hybrid predictors

Branch Direction Prediction

- Basic idea: Hope that future behavior of the branch is correlated to past behavior
  - Loops
  - Error-checking conditionals
- For a single branch PC
  - Simplest possible idea: Keep 1 bit around to indicate taken or not-taken
  - 2nd simplest idea: Keep 2 bits around, saturating counter

Two-bit Saturating Counters



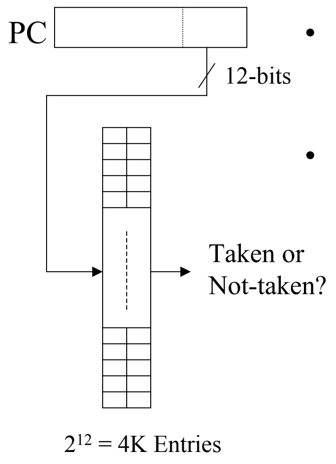
- 2-bit FSMs mean prediction must miss twice before change
- N-bit predictors are possible, but after 2-bits not much benefit

Example: Two-bit Vs. 1-bit Branch Prediction

Branch Outcome	T	T	T	N	T	T	T	N	T	T	T	N	% predict rate
1-bit Prediction	N	T	T	T	N	T	T	T	N	T	T	T	
1-bit Mis-Predict?	Y			Y	Y			Y	Y			Y	~50%
2-bit Prediction	n	T	T	t	T	T	T	t	T	T	T	t	
2-bit Mis-Predict?	Y			Y				Y				Y	~75%

- 2-bit “hysterisis” helps

Branch Prediction Buffer (branch history table, BHT)



- Small memory indexed with low bits of the branch instruction’s address
  - Why the low bits?
- Implementation
  - Separate memory accessed during IF phase
  - 2-bits attached to each block in the Instruction Cache
    - Caveats: Cannot separately size I-Cache and BHT
    - What about multiple branches in a cache line?
  - Does this help our simple 5-stage pipeline?

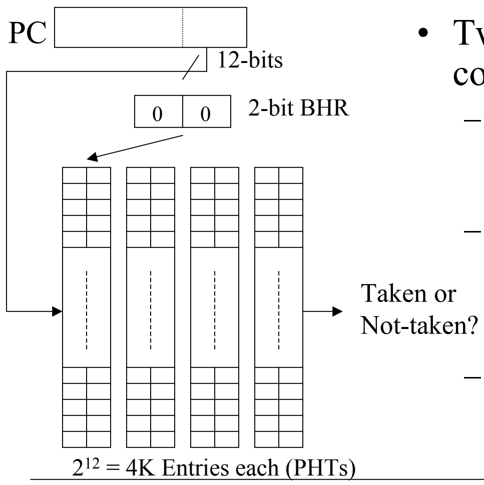
Correlating Predictors

- 2-bit scheme only looks at branch's *own* history to predict its behavior
- What if we use other branches to predict it as well?

```
if (aa==2)aa=0;    // Branch #1
if (bb==2)bb=0;    // Branch #2
if (aa!=bb){..}    // Branch #3
```

- Clearly branch #3 depends on outcome of #1 and #2
- Prediction must be a function of own branch as well as recent outcomes of other branches

Two-level Adaptive Branch Prediction (Correlating Predictor)



- Two-level BP requires to main components
  - Branch history register (BHR): recent outcomes of branches (last **k** branches encountered)
  - Pattern History Table (PHT): branch behavior for last **s** occurrences of the specific pattern of these **k** branches
  - In effect, we concatenate BHR with Branch PC bits
    - Can also XOR (GSHARE), etc

Branch History Register

- Simple shift register
  - Shift in branch outcomes as they occur
  - 1 => branch was taken
  - 0 => branch was not-taken
  - k-bit BHR => 2^k patterns
  - Use these patterns to address into the Pattern History Table

Pattern History Table

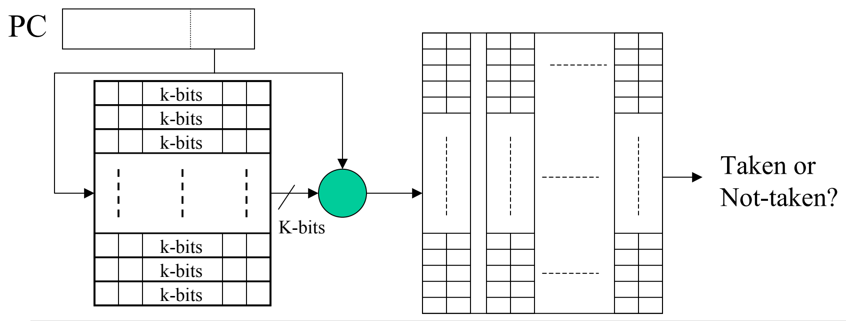
- Has 2^k entries
- Usually uses a 2-bit counter for the prediction
- Each entry summarizes branch results for the last **s** times that BHR pattern was seen
  - Not a shift register, usually an FSM
- BHR is used to address the PHT

Variations on 2-Level BP

- See Yeh + Patt for details
- Variations depend on
  - How many branches share a BHR
  - How many branches share a PHT
- 3 possibilities for each: global, per-address, per-set
- 9 total!
  - GAg, GAs, GAp
  - PAg, PAs, PAp
  - SAg, SAs, SAp

2-level Branch History

- Global history -- 1 Branch History Register (BHR)
- Per-address/set history
  - Per-Address/set Branch History Table holds many BHRs



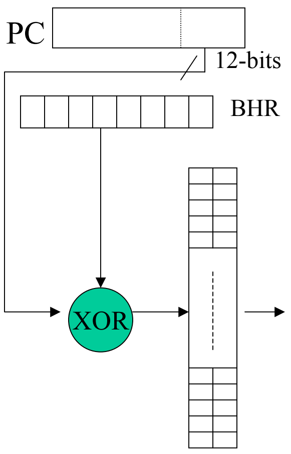
Hardware Costs of 2-level predictions

- (m,n) predictor ➔ m-bits of global history, n-bit predictor
- 2^m*n*Number of prediction entries
- Say you have m-bits of history (m=2)
- n-bits of predictor per entries (n=2)

(2,2) predictor with 1K prediction entries

2^2*2*1024 = 8K-bits

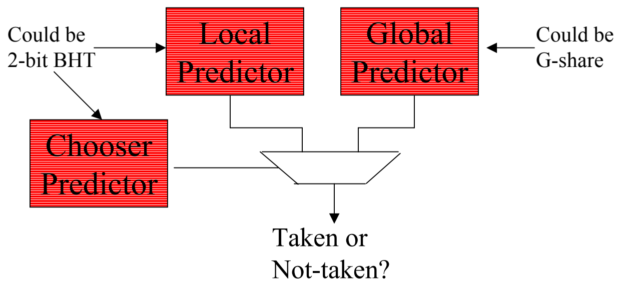
Variations on the basics -- GSHARE



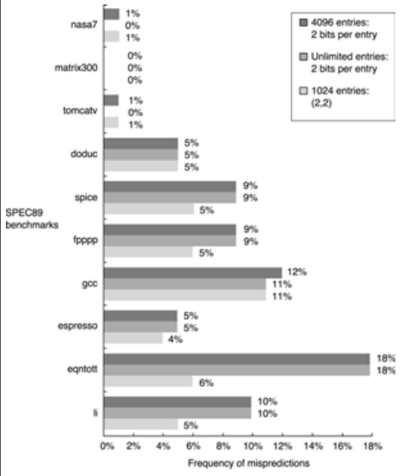
- Gshare a variant on GAg
- Don’t use BHR directly to address PHT
- Instead, XOR bits of BHR with bits of PC (branch address) and use that to index PHT
- Tries to separate out the behaviors/predictions associated with different branches, without extra hardware of PA and SA schemes

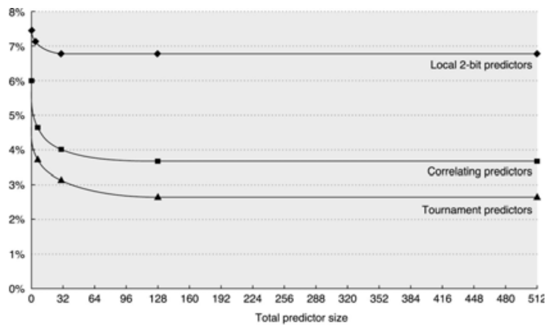
Hybrid Branch Predictors

- Tournament predictors: Adaptively combine local and global predictors
- Different schemes work better for different branches



Branch Predictor Performance





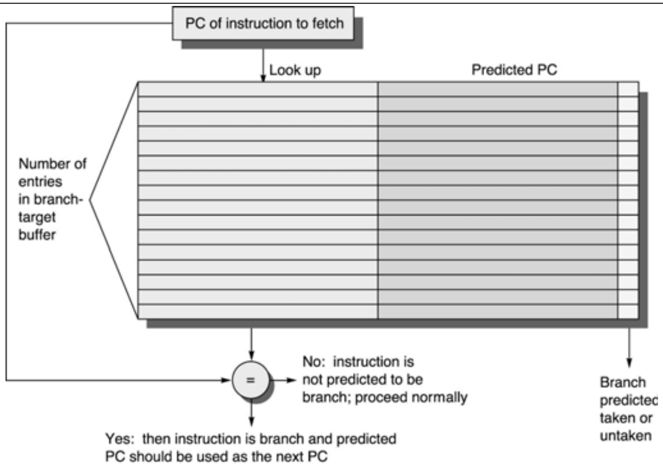
Branch Target Prediction

- So far we have only talked about predicting *direction*
- We still need to predict the *address*
  - Branch Target Buffer (BTB)
    - Useful for conditional/unconditional branches
  - Return Address Stack (RAS)
    - Useful for procedure returns

Branch Target Buffer

- Simple pipeline resolves stages in ID
  - We'd really like to know by the end of IF so we can proceed without a bubble
- Idea:
  - As part of IF use the instruction address (every instruction) to do a lookup in the BTB
  - For *N* recently executed branches, hold the predicted PC value (may also hold additional prediction bits)
  - If instruction is not a branch, don't add to BTB
  - If BTB fails revert to earlier method
    - Either instruction is not a branch
    - Or, there is no predictor entry for that branch
  - Many more bits per entry than BHT

Branch Target Buffer


PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be branch; proceed normally
Yes: then instruction is branch and predicted PC should be used as the next PC
Branch predicted taken or untaken

Branch Target Cache

- Similar to BTB, but we also want to know the target instruction!
  - Prediction returns not just the direction address, but also the instruction stored there
  - Allows zero-cycle branches (branch-folding)
    - Send target-instruction to ID rather than branch
    - Branch is not sent into pipe

Return Address Stack

- Included in many recent processors
  - Alpha 21264 => 12 entry RAS
- Procedure returns account for ~85% of indirect jumps
- Like a hardware stack, LIFO
  - Procedure Call => Push Return PC onto stack
  - Procedure Return => Prediction off of top of stack, Pop it
- RAS tends to work quite well since call depths are typically not large
- Problem: Speculative state! More next time

For next time

- Multiple Issue Machines
- Hardware Speculation
  - Performance and Precise Interrupts