Chapter 3 – Instruction-Level Parallelism and its Exploitation (Part 2)

ILP vs. Parallel Computers

Dynamic Scheduling (Section 3.4, 3.5)

Dynamic Branch Prediction (Section 3.3, 3.9, and Appendix C)

Hardware Speculation and Precise Interrupts (Section 3.6)

Multiple Issue (Section 3.7)

Static Techniques (Section 3.2, Appendix H)

Limitations of ILP

Multithreading (Section 3.11)

Putting it Together (Mini-projects)

Dynamic Branch Prediction

Reducing penalties from control dependences

Basic idea

Hardware guesses

* Whether branch will be taken/not taken

* Where the branch will go

Especially important for multiple issue processors

Desirable properties

Good prediction rate

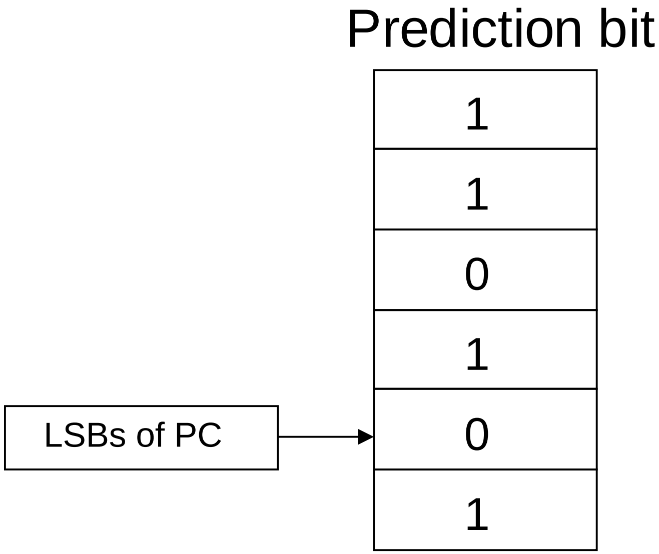
Make correct prediction fast

Don't slow too much on misprediction

Branch Prediction Buffer (Appendix C)

Maintain a buffer with prediction bits

Index buffer with LSBs of branch instruction PC



Predict based on indexed bit, change bit on misprediction

Accessed in ID stage (not useful for simple 5-stage pipeline)

Limitation of 1-bit predictor?

Variations on Branch Prediction Buffer

Variations

- n-bit predictor
- Correlating predictors
- Tournament predictors

N-bit Predictor

Contains n-bit saturating counter

Count up if taken, down if not taken

Predict taken if ≥ 2**(n-1); predict not taken if < 2**(n-1)

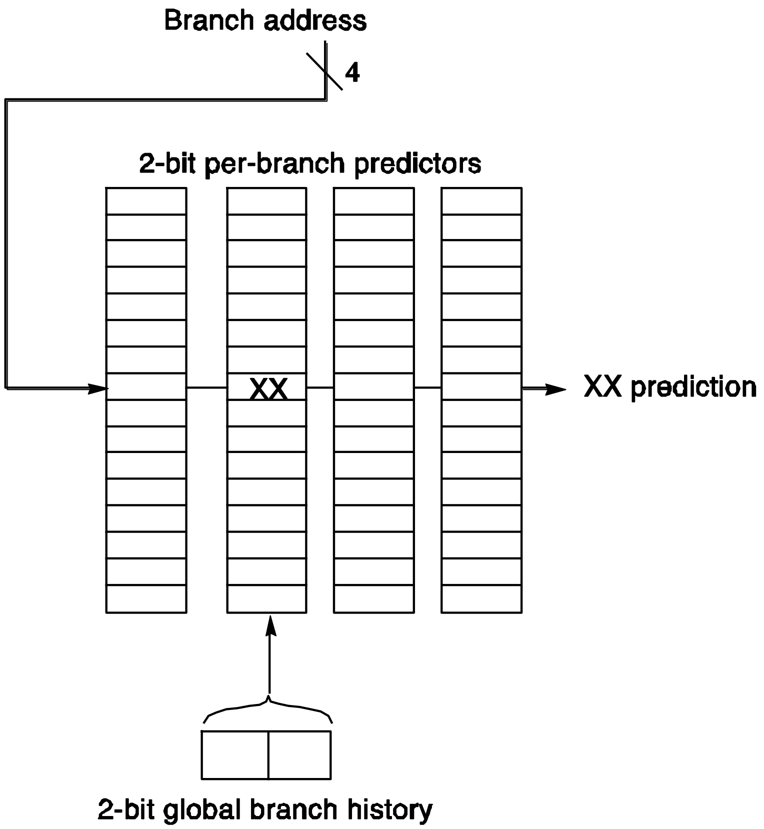
2-bit good for loops

Correlating Predictors: (m,n) Predictor

Use outcome of previous m branches and n-bit predictors

For each branch, the prediction buffer contains

- An entry for each possible history of previous m branches
- Each entry is an n-bit predictor



* This figure has been taken from Computer Architecture, A Quantitative Approach, 3rd Edition Copyright 2003 by Elsevier Inc. All rights reserved. It has been used with permission by Elsevier Inc.

*Correlating Predictors: (m,n) Predictor***

Use outcome of previous m branches and n-bit predictors

For each branch, the prediction buffer contains

- An entry for each possible history of previous m branches
- Each entry is an n-bit predictor

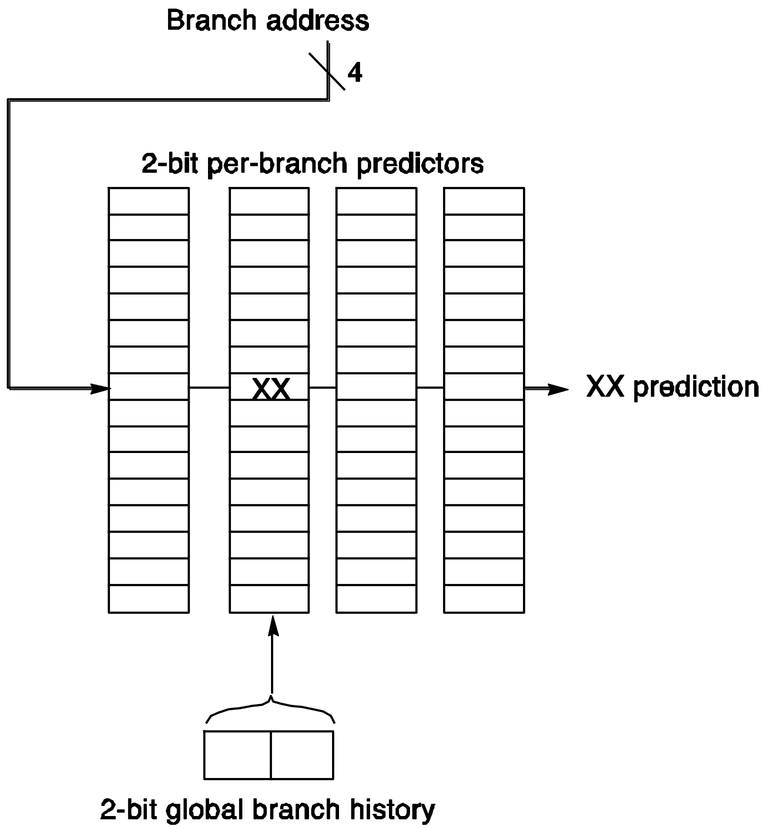


Figure 3.14 A(2,2) branch-prediction buffer uses a 2-bit global history to choose from among four predictors for each branch address. Each predictor is in turn a 2-bit predictor for that particular branch. The branch-prediction buffer shown here has a total of 64 entries; the branch address is used to choose four of these entries, and the global history is used to choose one of the four. The 2-bit global history can be implemented as a shifter register that simply shifts in the behavior of a branch as soon as it is known.

* This figure has been taken from Computer Architecture, A Quantitative Approach, 3rd Edition Copyright 2003 by Elsevier Inc. All rights reserved. It has been used with permission by Elsevier Inc.

Correlating Predictors (Cont.)

(1,1) predictor

Prediction based on 1 previous branch,

1 bit predictor

Number of prediction entries per branch = ??

Number of bits per prediction entry = ??

Correlating Predictors (Cont.)**

(1,1) predictor

Prediction based on 1 previous branch,

1 bit predictor

Number of prediction entries per branch = 2

(first for previous branch taken, second for not taken)

Number of bits per prediction entry = 1

(because 1 bit predictor)

Correlating Predictors Example

```
Loop:
    If a == 1          /* b1 */
        a = 0
    If a == 0          /* b2 */
        …
```

Let a = 1, 3, 1, 3, 1, 3, …

Notation: N=not taken; T=taken

Initialize (1,1) prediction buffer entries of b2 to NT

(1st entry for previous branch taken, 2nd for not taken)

Direction of b1:

Direction of b2:

History at b2:

Prediction entries of b2:

Prediction for b2:

Correlating Predictors Example**

```
Loop:
    If a == 1           /* b1 */
        a = 0
    If a == 0           /* b2 */
        …
```

Let a = 1, 3, 1, 3, 1, 3, …

Notation: N=not taken; T=taken

Initalize (1,1) prediction buffer entries of b2 to NT

(1st entry for previous branch taken, 2nd for not taken)

Direction of b1: T, N, T, N, T, N, T, N, …

Direction of b2:

History at b2:

Prediction entries of b2:

Prediction for b2:

Correlating Predictors Example**

```
Loop:
    If a == 1          /* b1 */
       a = 0
    If a == 0          /* b2 */
       …
```

Let a = 1, 3, 1, 3, 1, 3, …

Notation: N=not taken; T=taken

Initalize (1,1) prediction buffer entries of b2 to NT

(1st entry for previous branch taken, 2nd for not taken)

Direction of b1: T, N, T, N, T, N, T, N, …

Direction of b2: T, N, T, N, T, N, T, N, …

History at b2:

Prediction entries of b2:

Prediction for b2:

*Correlating Predictors Example***

```
Loop:
    If a == 1           /* b1 */
       a = 0
    If a == 0           /* b2 */
       …
```

Let a = 1, 3, 1, 3, 1, 3, …

Notation: N=not taken; T=taken

Initalize (1,1) prediction buffer entries of b2 to NT

(1st entry for previous branch taken, 2nd for not taken)

Direction of b1:	T,	N,	T,	N,	T,	N,	T,	N, …
Direction of b2:	T,	N,	T,	N,	T,	N,	T,	N, …
History at b2:	T,	N,	T,	N,	T,	N,	T,	N, …
Prediction entries of b2:	NT,							
Prediction for b2:								

*Correlating Predictors Example***

```
Loop:
        If a == 1           /* b1 */
            a = 0
        If a == 0           /* b2 */
            …
```

Let a = 1, 3, 1, 3, 1, 3, …

Notation: N=not taken; T=taken

Initalize (1,1) prediction buffer entries of b2 to NT

(1st entry for previous branch taken, 2nd for not taken)

Direction of b1:	T,	N,	T,	N,	T,	N,	T,	N,	…
Direction of b2:	T,	N,	T,	N,	T,	N,	T,	N,	…
History at b2:	T,	N,	T,	N,	T,	N,	T,	N,	…
Prediction entries of b2:	NT,	TT,	TN,	TN,	TN,	TN,	TN,	TN,	…
Prediction for b2:	N,	T,	T,	N,	T,	N,	T,	N,	…

Correlating Predictors – Misprediction Rate**

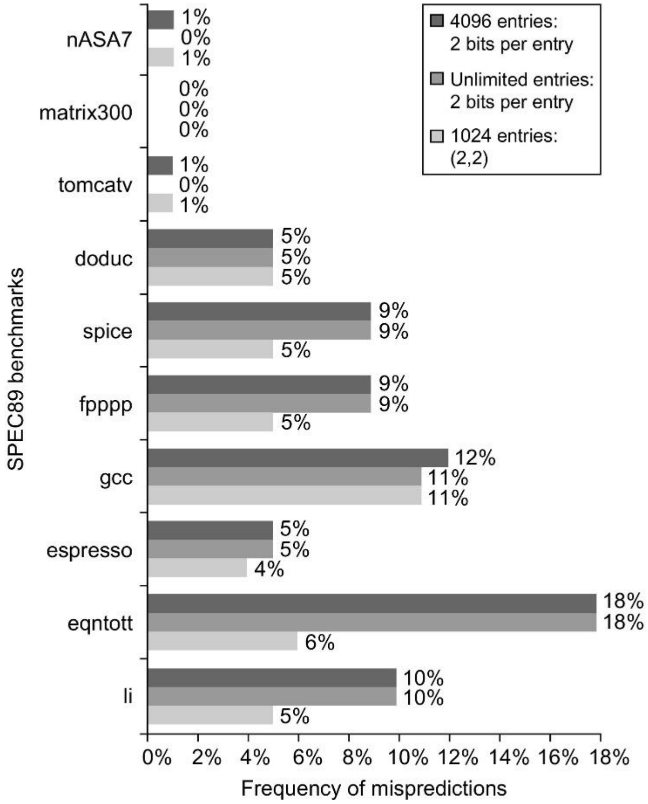


Figure 3.3 Comparison of 2-bit predictors. A noncorrelating predictor for 4096 bits is first, followed by a noncorrelating 2-bit predictor with unlimited entries and a 2-bit predictor with 2 bits of global history and a total of 1024 entries. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks would show similar differences in accuracy.

© 2019 Elsevier Inc. All rights reserved.

*Correlating Predictor Variation – gshare***

Alloyed or hybrid predictor – combines global branch history with local branch information

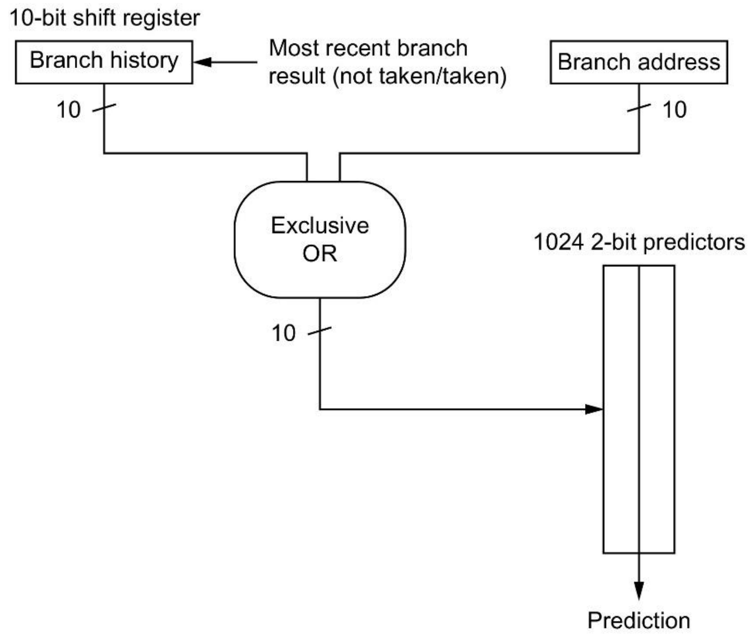
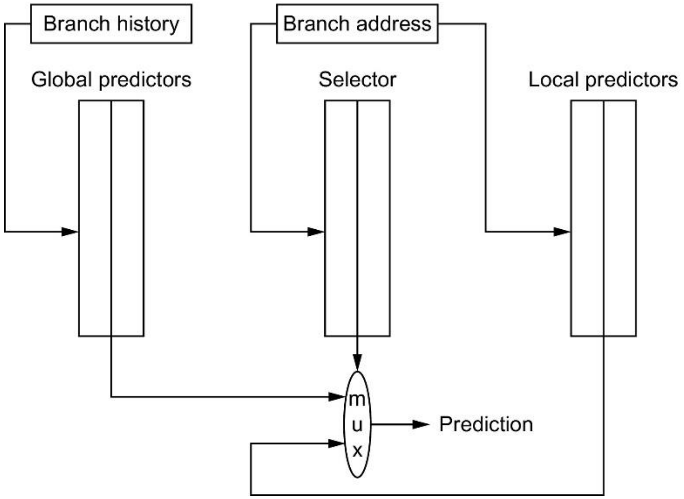


Figure 3.4 A gshare predictor with 1024 entries, each being a standard 2-bit predictor.

© 2019 Elsevier Inc. All rights reserved.

Tournament Predictor

- Combine multiple predictors with a selector
  - Often combine a global predictor and a local predictor
  - Selector typically two bit saturating counter
    - Increment when predicted predictor correct, other incorrect



© 2019 Elsevier Inc. All rights reserved.

Tournament Predictor**

Combine multiple predictors with a selector

- Often combine a global predictor and a local predictor
- Selector typically two bit saturating counter
  - Increment when predicted predictor correct, other incorrect

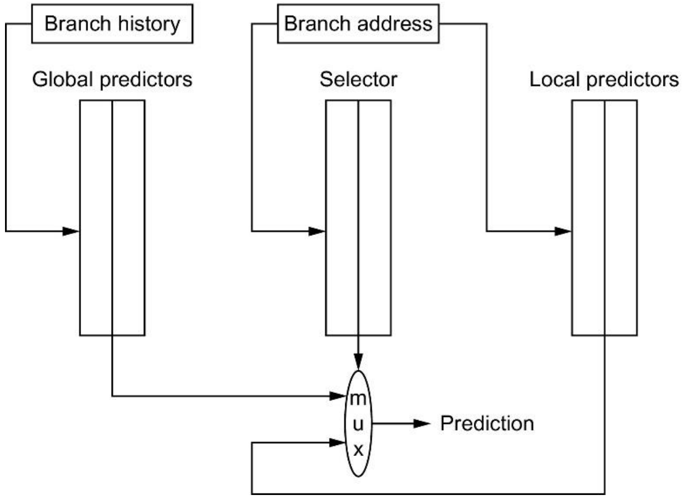


Figure 3.5 A tournament predictor using the branch address to index a set of 2-bit selection counters, which choose between a local and a global predictor. In this case, the index to the selector table is the current branch address. The two tables are also 2-bit predictors that are indexed by the global history and branch address, respectively. The selector acts like a 2-bit predictor, changing the preferred predictor for a branch address when two mispredicts occur in a row. The number of bits of the branch address used to index the selector table and the local predictor table is equal to the length of the global branch history used to index the global prediction table. Note that misprediction is a bit tricky because we need to change both the selector table and either the global or loc al predictor.

© 2019 Elsevier Inc. All rights reserved.

Tournament Predictor Example - Alpha 21264

Uses 4K 2-bit counters to choose from global and local predictor

Global predictor

- 4K entries of 2-bit predictors
- Indexed by history of last 12 branches

Local predictor is a two-level predictor

- History table with 1K 10-bit entries (for that branch)
  - Each entry gives 10 most recent branch outcomes
- Indexes table of 1K entries with 3-bit counters

Total of 29K bits

Misprediction rate

- SPECfp95 – 1 per 1000
- SPECint95 – 11.5 per 1000

Tournament Predictor – Misprediction Rate**

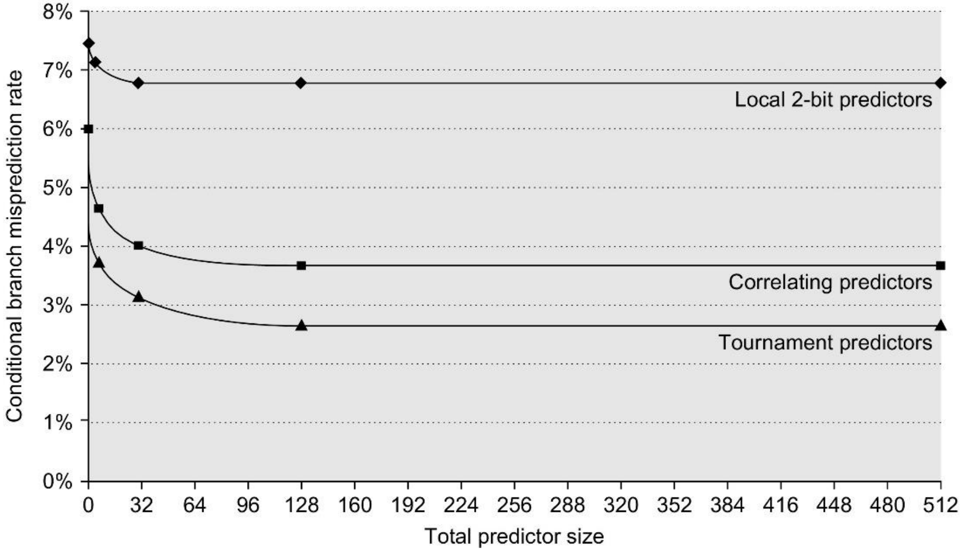


Figure 3.6 The misprediction rate for three different predictors on SPEC89 versus the size of the predictor in kilobits. The predictors are a local 2-bit predictor, a correlating predictor that is optimally structured in its use of global and local information at each point in the graph, and a tournament predictor. Although these data are for an older version of SPEC, data for more recent SPEC benchmarks show similar behavior, perhaps converging to the asymptotic limit at slightly larger predictor sizes.

© 2019 Elsevier Inc. All rights reserved.

More Predictors

Lots of work on branch prediction

International Branch Prediction Competition!

More Predictors – Tagged Hybrid Predictors**

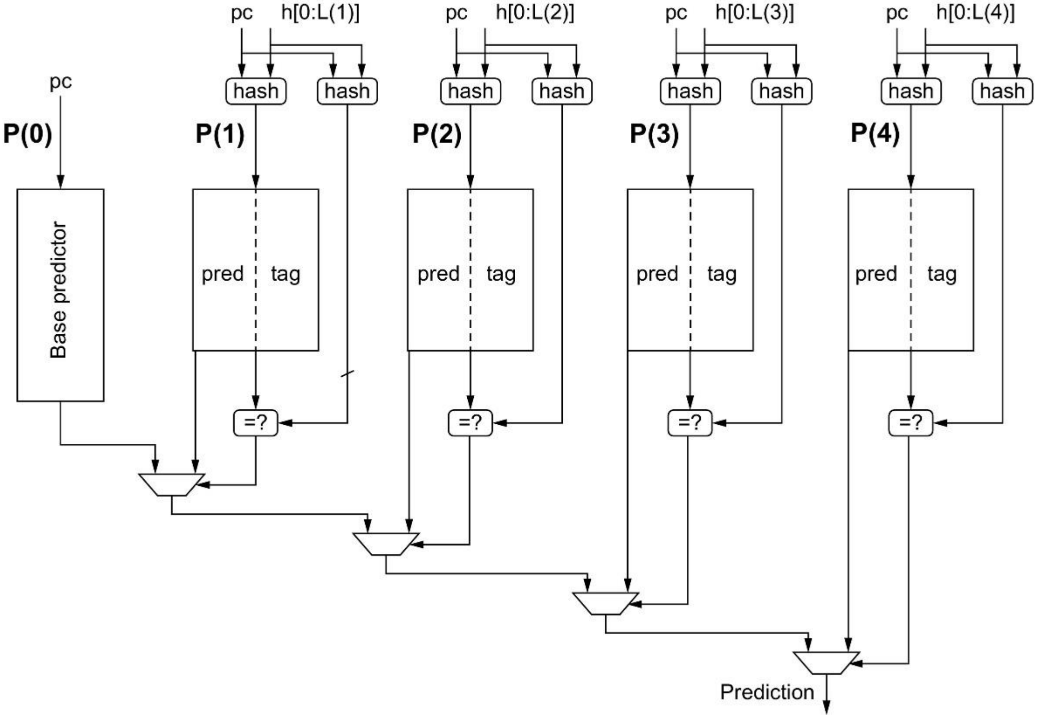


Figure 3.7 A five-component tagged hybrid predictor has five separate prediction tables, indexed by a hash of the branch address and a segment of recent branch history of length 0–4 labeled "h" in this figure. The hash can be as simple as an exclusive-OR, as in gshare. Each predictor is a 2-bit (or possibly 3-bit) predictor. The tags are typically 4–8 bits. The chosen prediction is the one with the longest history where the tags also match.

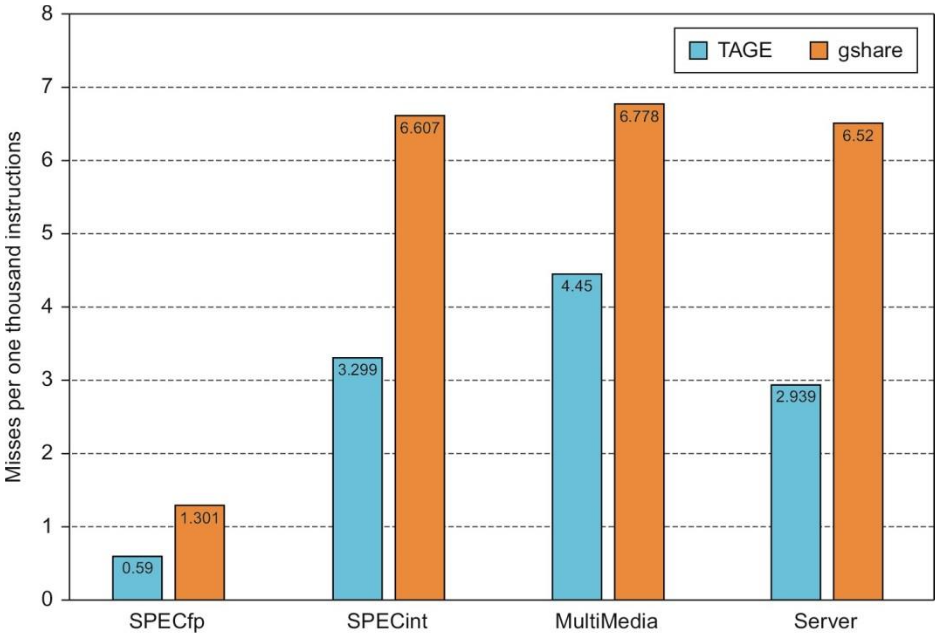


Figure 3.8 A comparison of the misprediction rate (measured as mispredicts per 1000 instructions executed) for tagged hybrid versus gshare. Both predictors use the same total number of bits, although tagged hybrid uses some of that storage for tags, while gshare contains no tags. The benchmarks consist of traces from SPECfp and SPECint, a series of multimedia and server benchmarks. The latter two behave more like SPECint.

© 2019 Elsevier Inc. All rights reserved.

Branch Prediction Buffer Strategies: Limitations

Limitations

- May use bit from wrong PC
- Target must be known when branch resolved

Branch Target Buffer or Cache (Section 3.9)

Store target PC along with prediction

Accessed in IF stage

Next IF stage uses target PC

No bubbles on correctly predicted taken branch

Must store tag

More state

Can remove not-taken branches?

Branch Target Buffer or Cache**

Store target PC along with prediction

Accessed in IF stage

Next IF stage uses target PC

No bubbles on correctly predicted taken branch

Must store tag

More state

Can remove not-taken branches?

N-bit predictors must update state for not-taken branches

*Branch Target Buffer or Cache***

Store target PC along with prediction

Accessed in IF stage

Next IF stage uses target PC

No bubbles on correctly predicted taken branch

Must store tag

More state

Can remove not-taken branches?

N-bit predictors must update state for not-taken branches

Maintain branch prediction buffer to remove not-taken branches from target buffer

Branch Target Cache With Target Instruction

Store target instruction along with prediction

Send target instruction instead of branch into ID

Zero cycle branch - branch folding

Used for unconditional jumps

E.g., ARM Cortex A-53

Return Address Stack (Section 3.9)

Hardware stack for addresses for returns

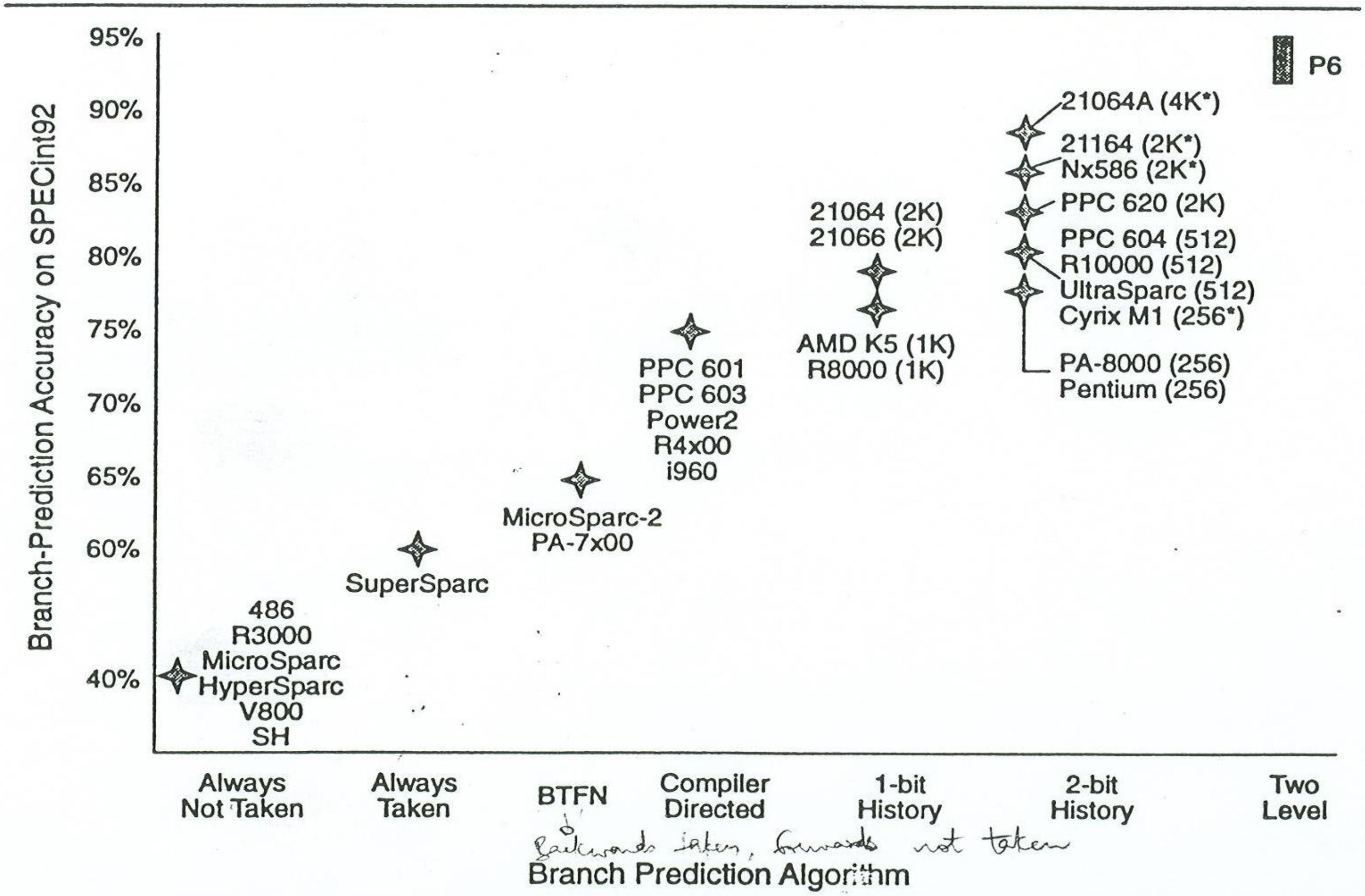
Call pushes return address in stack

Return pops the address

Perfect prediction if stack length ≥ call depth

Static vs. Dynamic Branch Prediction Accuracy**

MICROPROCESSOR REPORT



backwards taken, forwards not taken

Figure 4. As processors use more complex algorithms, branch-prediction accuracy increases. (Number of history-table entries in parentheses.) *also uses return-address stack.

Speculative Execution

How far can we go with branch prediction?

- Speculative fetch?
- Speculative issue?
- Speculative execution?
- Speculative write?

Speculative Execution

Allows instructions after branch to *execute* before knowing if branch will be taken

Must be able to undo if branch is not taken

Often try to combine with dynamic scheduling

Key insight: Split Write stage into Complete and Commit

- Complete out of order
    - No state update
- Commit in order
    - State updated (instruction no longer speculative)

Use reorder buffer

Reorder Buffer

Overview

- Instructions complete out-of-order
- Reorder buffer reorganizes instructions
- Modify state in-order

	Entry	Busy	Type	Dest	Result	State	Excep
	1	0					
head →	2	1	LD	4		Exec	0
	3	1	BR			Exec	0
tail →	4	1	ADD	6	75	Compl	0
	5	0					
		0					
	N	0					

Instruction tag now is reorder buffer entry

Re-order Buffer Pipeline

Issue:

Execute:

Complete:

Commit:

*Re-order Buffer Pipeline***

Issue:

- Allocate reorder buffer entry (RB) and reservation station (RS)
- Make RS and register result status point to RB
- Read operands from registers or reorder buffer if available

Execute:

Complete:

Commit:

Re-order Buffer Pipeline**

Issue:

- Allocate reorder buffer entry (RB) and reservation station (RS)
- Make RS and register result status point to RB
- Read operands from registers or reorder buffer if available

Execute:

- Execute when operands available
- (Monitor CDB if not available)

Complete:

Commit:

*Re-order Buffer Pipeline***

Issue:

- Allocate reorder buffer entry (RB) and reservation station (RS)
- Make RS and register result status point to RB
- Read operands from registers or reorder buffer if available

Execute:

- Execute when operands available
- (Monitor CDB if not available)

Complete:

- Write result to CDB, RB entry pointed to by RS, other RS waiting for this operand (no write in register file)

Re-order Buffer Pipeline (Cont.)**

Commit: When instruction reaches head of reorder buffer:

- Write result in register file (for all but branch and store)
- For store, do memory write
- For branch,
  - if mispredict, flush all entries in reorder buffer and restart
- Make RB entry free

Precise Interrupts Again

Precise interrupts hard with dynamic scheduling

Consider our canonical code fragment:

```
LF F6,34(R2)
LF F2,45(R3)
MULTF F0,F2,F4
SUBF F8,F6,F2
DIVF F10,F0,F6
ADDF F6,F8,F2
```

What happens if DIVF causes an interrupt?

- ADDF has already completed

Out-of-order completion makes interrupts hard

But reorder buffer can help!

Reorder Buffer for Precise Interrupts

*Reorder Buffer for Precise Interrupts***

Take interrupt only after instruction reaches the head of the reorder buffer

Flush all remaining instructions and restart

Ok since no registers updated or stores sent to memory

Re-order Buffer Drawback

Operands need to be read from reorder buffer or registers

Alternative: Rename registers

Rename Registers + Reorder Buffer

Many current machines

- More physical registers than logical registers
- Reorder buffer does not have values
- Read all values from registers

Rename mechanism

- Rename map stores mapping from logical to physical registers
  - (Logical register Rl mapped to physical register Rp)
  - On issue, Rl mapped to Rp-new
  - On completion, write to Rp-new
  - On commit, old mapping of Rl discarded (free Rp-old)
  - On misprediction, new mapping of Rl discarded (free Rp-new)