This set under construction.

Outline

- Branch Prediction Overview
- Bimodal (One-Level) Predictor
- Correlating (Two-Level) Predictors: local, global, gshare
- Other topics to be added.
- Sample Problems

### Motivation

Branches occur frequently in code.

At best, one cycle of branch delay; more with dependencies.

Therefore, impact on CPI is large.

Techniques

Branch Prediction: Predict outcome of branch. (Taken or not taken.)

Branch Target Prediction: Predict branch or other CTI's target address.

Branch Folding: Replace branch or other CTI with target instruction.

# Branch Prediction

## Methods Covered

One-level predictor, a.k.a. bimodal.

Two-Level Predictors

GAg, a.k.a. Global History.

gshare.

Local History, a.k.a. PAg.

Idea: Predict using past behavior.

Example:

LOOP:																		
lw	r1,	0(r2)	#	Lo	bad	d o	rand	lom	nι	ımb	ber,	eit	her	c O or	1	•		
addi	r2,	r2, 4																
slt	r6,	r2, r7																
beq	r1,	r0 SKIP	#	Т	N	N	Т	N	Т	Т	T N		#	Rando	m,	no	pattern	•
nop																		
addi	r3,	r3, 1																
SKIP:																		
bneq	r6,	r0 LOOP	#	Т	Т	Т	• • •	Т	N '	ΤI	ΓТ	#	99	9 T's,	1	N,	99 T's,	• • •
nop																		

Second branch, bneq, taken 99 out of 100 executions.

Pattern for  $\mathtt{bneq:}~T~T~T~\ldots~T~N~T~T$ 

First branch shows no pattern.

SPEC89 benchmarks on IBM POWER (predecessor to PowerPC).



FIGURE 4.14 Prediction accuracy of a 4096-entry two-bit prediction buffer for the SPEC89 benchmarks.



# Branch Prediction Terminology

*Outcome:* [of a branch instruction execution]. The outcome of the execution of a branch instruction.

T: A taken branch.

*N*:

A branch that is not taken.

*Prediction:* [made by branch prediction hardware]. The predicted outcome of a branch.

Misprediction: An incorrectly predicted outcome.

*Prediction Accuracy:* [of a branch prediction scheme]. The number of correct predictions divided by the number of predictions.

# Branch Prediction Terminology (Continued)

### Speculative Execution:

The execution of instructions following a predicted branch.

### Misprediction Recovery:

Undoing the effect of speculatively executed instructions  $\dots$ 

... and re-starting instruction fetch at the correct address.

Idea: maintain a *branch history* for each branch instruction.

# Branch History: Information about past behavior of the branch.

Branch histories stored in a branch history table (BHT).

Often, branch history is sort of number of times branch taken... ... minus number of times not taken.

Other types of history possible.

Branch history read to make a prediction.

Branch history updated when branch outcome known.

If a counter used, branch history incremented when branch taken...

 $\ldots$  and decremented when branch not taken.

Symbol n denotes number of bits for branch history.

To save space and for performance reasons ...

... branch history limited to a few bits, usually n = 2.

Branch history updated using a saturating counter.

A saturating counter is an arithmetic unit that can add or subtract one ...

... in which 
$$x + 1 \to x + 1$$
 for  $x \in [0, 2^n - 2] \dots$   
...  $x - 1 \to x - 1$  for  $x \in [1, 2^n - 1] \dots$   
...  $(2^n - 1) + 1 \to 2^n - 1 \dots$   
... and  $0 - 1 \to 0$ .

For an *n*-bit counter, predict taken if counter  $\geq 2^{n-1}$ .

# One-Level Branch Predictor Hardware

Illustrated for 5-stage MIPS implementation ... ... even though prediction not very useful.

Branch Prediction Steps

1: Predict.

12-11

Read branch history, available in IF.

2: Determine Branch Outcome

Execute predicted branch in usual way.

**3**: Recover (If necessary.)

Undo effect of speculatively executing instructions, start fetching from correct path.

4: Update Branch History

### Branch History Table

Stores branch histories,

Implemented using a memory device.

Address (called index) is hash of branch address (PC).

For  $2^m$ -entry BHT, hash is *m* lowest bits of branch PC skipping alignment.

			BHT Addr		Align.	
Branch address:						0
	31	m+2	m+1	2	1	0

Data input and output of BHT is branch history.

Outcomes for individual branches, categorized by pattern, sorted by frequency.

Branches running T<sub>E</sub>X text formatter compiled for SPARC (Solaris).

Arbitra	cy, pat	60288, b	r732164,	0.7743	3 0.717	70 0.71	99 (0.19675)	
	% Pa	tterns #	Branches	gshre	local	corr	Local History	
0:	fe7f	0.0004	1397	0.912	0.916	0.896	TTTTTTTTNNTTTTTTT	0
1:	ff3f	0.0004	1323	0.924	0.909	0.900	TTTTTTNNTTTTTTTT	0
2:	fcff	0.0004	1317	0.949	0.939	0.948	TTTTTTTTTNNTTTTTT	0
3:	ff9f	0.0003	1245	0.910	0.905	0.898	TTTTTNNTTTTTTTTT	0
4:	f9ff	0.0003	1235	0.955	0.950	0.955	TTTTTTTTTTNNTTTTT	0
5:	ffcf	0.0003	1188	0.926	0.921	0.923	TTTTNNTTTTTTTTTT	0
6:	60	0.0003	1163	0.873	0.829	0.854	NNNNNTTNNNNNNNNN	0
7:	180	0.0003	1159	0.955	0.914	0.926	NNNNNNTTNNNNNN	0
8:	300	0.0003	1158	0.949	0.926	0.934	NNNNNNNTTNNNNN	0
9:	c0	0.0003	1155	0.944	0.917	0.926	NNNNNNTTNNNNNNN	0

Loop, pa	t 124, br	137681	, 0.890	0.90	055 0.7	441 (0.03700)	
% Pa	tterns #	Branches	gshre	local	corr	Local History	
5555	0.0040	14753	0.987	0.981	0.912	TNTNTNTNTNTNTNTNTN	1
aaaa	0.0040	14730	0.859	0.978	0.461	NTNTNTNTNTNTNTNTNT	1
9249	0.0022	8062	0.997	0.992	0.988	TNNTNNTNNTNNTNNT	1
4924	0.0022	8055	0.997	0.998	0.998	NNTNNTNNTNNTNNTN	1
2492	0.0022	8047	0.993	0.991	0.009	NTNNTNNTNNTNNTNN	1
db6d	0.0013	4864	0.713	0.915	0.065	TNTTNTTNTTNTTNTT	1
b6db	0.0013	4713	0.862	0.903	0.926	TTNTTNTTNTTNTTNT	1
6db6	0.0012	4640	0.991	0.978	0.970	NTTNTTNTTNTTNTTN	1
bbbb	0.0008	3061	0.896	0.936	0.949	TTNTTTNTTTNTTTNT	1
	Loop, pa % Pa 5555 aaaa 9249 4924 2492 db6d b6db 6db6 bbbb	Loop, pat 124, br % Patterns # 5555 0.0040 aaaa 0.0040 9249 0.0022 4924 0.0022 2492 0.0022 db6d 0.0013 b6db 0.0013 6db6 0.0012 bbbb 0.0008	Loop, pat 124, br 137681 % Patterns # Branches 5555 0.0040 14753 aaaa 0.0040 14730 9249 0.0022 8062 4924 0.0022 8055 2492 0.0022 8047 db6d 0.0013 4864 b6db 0.0013 4713 6db6 0.0012 4640 bbbb 0.0008 3061	Loop, pat 124, br 137681, 0.890 % Patterns # Branches gshre 5555 0.0040 14753 0.987 aaaa 0.0040 14730 0.859 9249 0.0022 8062 0.997 4924 0.0022 8055 0.997 2492 0.0022 8047 0.993 db6d 0.0013 4864 0.713 b6db 0.0013 4713 0.862 6db6 0.0012 4640 0.991 bbbb 0.0008 3061 0.896	Loop, pat 124, br 137681, 0.8908 0.90 % Patterns # Branches gshre local 5555 0.0040 14753 0.987 0.981 aaaa 0.0040 14730 0.859 0.978 9249 0.0022 8062 0.997 0.992 4924 0.0022 8055 0.997 0.998 2492 0.0022 8047 0.993 0.991 db6d 0.0013 4864 0.713 0.915 b6db 0.0013 4713 0.862 0.903 6db6 0.0012 4640 0.991 0.978 bbbb 0.0008 3061 0.896 0.936	Loop, pat 124, br 137681, 0.8908 0.9055 0.7 % Patterns # Branches gshre local corr 5555 0.0040 14753 0.987 0.981 0.912 aaaa 0.0040 14730 0.859 0.978 0.461 9249 0.0022 8062 0.997 0.992 0.988 4924 0.0022 8055 0.997 0.998 0.998 2492 0.0022 8047 0.993 0.991 0.009 db6d 0.0013 4864 0.713 0.915 0.065 b6db 0.0013 4713 0.862 0.903 0.926 6db6 0.0012 4640 0.991 0.978 0.970 bbbb 0.0008 3061 0.896 0.936 0.949	Loop, pat 124, br 137681, 0.8908 0.9055 0.7441 (0.03700) % Patterns # Branches gshre local corr Local History 5555 0.0040 14753 0.987 0.981 0.912 TNTNTNTNTNTNTNTN aaaa 0.0040 14730 0.859 0.978 0.461 NTNTNTNTNTNTNTNT 9249 0.0022 8062 0.997 0.992 0.988 TNNTNNTNNTNNTN 4924 0.0022 8055 0.997 0.998 0.998 NNTNNTNNTNNTN 2492 0.0022 8047 0.993 0.991 0.009 NTNNTNNTNNTNN db6d 0.0013 4864 0.713 0.915 0.065 TNTTNTNTNTNTNTNT b6db 0.0013 4713 0.862 0.903 0.926 TTNTTNTTNTTNTTNT 6db6 0.0012 4640 0.991 0.978 0.970 NTTNTTNTTNTTNTNTNTNTNTNTNTNTNTNTNTNTNT

Long Loop?, pat 32, br 185795, 0.9170 0.9052 0.9096 (0.04993)

0:	fffe	0.0025	9204	0.902	0.930	0.913	NTTTTTTTTTTTTTTTT	2
1:	8000	0.0025	9198	0.654	0.700	0.705	NNNNNNNNNNNNNN	2
2:	7fff	0.0022	8052	0.890	0.817	0.818	TTTTTTTTTTTTTTTT	2
3:	ffbf	0.0018	6800	0.933	0.908	0.920	TTTTTTTNTTTTTTTTT	2
4:	feff	0.0018	6782	0.946	0.938	0.942	TTTTTTTTTTTTTTTTT	2
5:	ff7f	0.0018	6778	0.949	0.946	0.950	TTTTTTTTTTTTTTTTTT	2
6:	fdff	0.0018	6738	0.947	0.941	0.946	TTTTTTTTTTTTTTTTT	2
7:	1	0.0018	6690	0.955	0.945	0.942	TNNNNNNNNNNNNNN	2
8:	fffd	0.0018	6667	0.968	0.966	0.967	TNTTTTTTTTTTTTTT	2

Change,	pat 26, b	r 48190,	0.8453	3 0.904	40 0.84	470 (0.01295)	
% Pa	atterns #	Branches	gshre	local	corr	Local History	
c000	0.0012	4554	0.653	0.777	0.680	NNNNNNNNNNNNTT	3
e000	0.0009	3420	0.714	0.859	0.758	NNNNNNNNNNNTTT	3
f000	0.0008	2942	0.756	0.888	0.788	NNNNNNNNNNNTTTT	3
fffc	0.0008	2878	0.908	0.960	0.959	NNTTTTTTTTTTTTTTT	3
f800	0.0007	2642	0.786	0.917	0.827	NNNNNNNNNNTTTTT	3
3	0.0007	2572	0.968	0.952	0.951	TTNNNNNNNNNNNNN	3
fc00	0.0007	2435	0.815	0.933	0.854	NNNNNNNNNTTTTTT	3
fe00	0.0006	2225	0.836	0.936	0.876	NNNNNNNNTTTTTTT	3
ff00	0.0006	2140	0.856	0.947	0.931	NNNNNNNTTTTTTTT	3
ff80	0.0006	2061	0.854	0.941	0.934	NNNNNNTTTTTTTTT	3
	Change, % Pa c000 e000 f000 fffc f800 3 fc00 fe00 ff00 ff80	Change, pat 26, b % Patterns # c000 0.0012 e000 0.0009 f000 0.0008 fffc 0.0008 fffc 0.0008 f800 0.0007 fc00 0.0007 fe00 0.0006 ff00 0.0006 ff80 0.0006	Change, pat 26, br 48190, % Patterns # Branches c000 0.0012 4554 e000 0.0009 3420 f000 0.0008 2942 fffc 0.0008 2878 f800 0.0007 2642 3 0.0007 2572 fc00 0.0007 2435 fe00 0.0006 2225 ff00 0.0006 2140 ff80 0.0006 2061	Change, pat 26, br 48190, 0.8453 % Patterns # Branches gshre c000 0.0012 4554 0.653 e000 0.0009 3420 0.714 f000 0.0008 2942 0.756 fffc 0.0008 2878 0.908 f800 0.0007 2642 0.786 3 0.0007 2572 0.968 fc00 0.0007 2435 0.815 fe00 0.0006 2225 0.836 ff00 0.0006 2140 0.856 ff80 0.0006 2061 0.854	Change, pat 26, br 48190, 0.8453 0.904 % Patterns # Branches gshre local c000 0.0012 4554 0.653 0.777 e000 0.0009 3420 0.714 0.859 f000 0.0008 2942 0.756 0.888 fffc 0.0008 2878 0.908 0.960 f800 0.0007 2642 0.786 0.917 3 0.0007 2572 0.968 0.952 fc00 0.0007 2435 0.815 0.933 fe00 0.0006 2225 0.836 0.936 ff00 0.0006 2140 0.856 0.947 ff80 0.0006 2061 0.854 0.941	Change, pat 26, br 48190, 0.8453 0.9040 0.84 % Patterns # Branches gshre local corr c000 0.0012 4554 0.653 0.777 0.680 e000 0.0009 3420 0.714 0.859 0.758 f000 0.0008 2942 0.756 0.888 0.788 fffc 0.0008 2878 0.908 0.960 0.959 f800 0.0007 2642 0.786 0.917 0.827 3 0.0007 2572 0.968 0.952 0.951 fc00 0.0007 2435 0.815 0.933 0.854 fe00 0.0006 2225 0.836 0.936 0.876 ff00 0.0006 2140 0.856 0.947 0.931 ff80 0.0006 2061 0.854 0.941 0.934	Change, pat 26, br 48190, 0.8453 0.9040 0.8470 (0.01295) % Patterns # Branches gshre local corr Local History c000 0.0012 4554 0.653 0.777 0.680 NNNNNNNNNNTT e000 0.0009 3420 0.714 0.859 0.758 NNNNNNNNNNNTT f000 0.0008 2942 0.756 0.888 0.788 NNNNNNNNNNNTTT fffc 0.0008 2878 0.908 0.960 0.959 NNTTTTTTTTT f800 0.0007 2642 0.786 0.917 0.827 NNNNNNNNNNTTTT 3 0.0007 2572 0.968 0.952 0.951 TTNNNNNNNNNNNN fc00 0.0006 225 0.836 0.936 0.876 NNNNNNNNNTTTT ff00 0.0006 2140 0.856 0.947 0.931 NNNNNNNNTTTTT ff80 0.0006 2061 0.854 0.941 0.934 NNNNNNNTTTTTT

One Way,	pat	2, br	2617433,	0.9917 0.9934 0.9897 (0.70337)	
0:	ffff	0.5151	1916950	0.993 0.996 0.993 TTTTTTTTTTTTTTT	4
1:	0	0.1882	700483	0.988 0.986 0.982 NNNNNNNNNNNNNN	4

# Two-Level Correlating Predictors

Idea: Base branch decision on ...

- $\ldots$  the address of the branch instruction (as in the one-level scheme)  $\ldots$
- ... and the most recent branch outcomes.

#### History:

The outcome (taken or not taken) of the most recent branches. Usually stored as a bit vector with 1 indicating taken.

### Pattern History Table (PHT):

Memory for 2-bit counters, indexed (addressed) by some combination of history and the branch instruction address.

Some Types of Two-Level Predictors

Global, a.k.a. GAg.

History is global (same for all branches), stored in a global history register (GHR). PHT indexed using history only.

#### gshare

History is global (same for all branches), stored in a global history register (GHR). PHT indexed using history exclusive-ored with branch address.

#### gselect

History is global (same for all branches), stored in a global history register (GHR). PHT indexed using history concatenated with branch address. Local, a.k.a., PAg.

History is local, BHT stores history for each branch.

PHT indexed using history only.

```
! Loop always iterates 4 times.
! Branch below never taken.
bneq r2, SKIP
                   Ν
                                         Ν
addd f0, f0, f2
SKIP:
addi
      r1, r0, #4
LOOP:
multd f0, f0, f2
subi r1, r1, #1
                            T T N ... T T T N ...
bneq r1, LOOP
                        Т
                     10 20 30 40 50 110 120 130 140 150
! Cycle
ļ
! Global History (m=4), X: depends on earlier branches.
! 10
     XXXN
           Human would predict taken.
! 20
     XXNT
           Human would predict taken.
! 30
     XNTT
           Human would predict taken.
! 40
           Human would predict not taken.
     NTTT
! 50
     TTTN
```

Register r1 not available until cycle ten<sup>1</sup>.

Cycle 1: When branch in ID, read BHT and make prediction.

Cycle 1: (Optional) Backup (checkpoint) register map (if present).

Cycle 10: Execute branch in usual way and check prediction.

Cycle 11: If prediction correct, update BHT when branch commits.

Cycle 11: If pred. wrong, start recovery process (does not occur here).

! Predict not ta	ken,	not	taken							
Cycle:	0	1	2	3			10	11	12	13
bneq r1, TARGET	IF	ID	Q		•	• •	В	WC		
xor r2, r3, r4		IF	ID	Q	EX	WB			С	
										С
•••										
TARGET:										
and r5, r6, r7										

<sup>&</sup>lt;sup>1</sup> Perhaps due to a cache miss, or maybe it depended on a long-latency floating-point operation, the reason is not important

12-20

### BHT use when branch taken, correctly predicted.

Register r1 not available until cycle 10.

Cycle 1: When branch in ID, compute target, read BHT and make prediction.

Cycle 10: Execute branch in usual way.

Cycle 11: Check outcome. Correctly predicted.

Cycle 23: Commit branch after div.

! Predict taken,	take	n.									
Cycle:	0	1	2	3		10	11	•••	21	22	23
div f0,f2, f4	ID	Q	DIV						DIV	WC	
bneq r1, TARGET	IF	ID	Q		•••	В	WB				C
xor r2, r3, r4		IFx									
••••											
TARGET:											
and r5, r6, r7			IF	•							C

BHT use when branch taken, incorrectly predicted, register map not backed up.

Register r1 not available until cycle 10.

Cycle 1: When branch in ID, compute target, read BHT and make prediction.

Cycle 10: Compute branch condition.

Cycle 11: Misprediction "discovered." Because register map not backed up, recovery must wait until commit.

Cycle 23: Start recovery: Squash instructions in reorder buffer, start fetching correct path.

! Predict not take	en,	take	n. Re	gis	ter	map	not	backed	d up.				
Cycle:	0	1	2	3				10	11	•••	21	22	23
div f0,f2, f4	ID	Q	DIV								DIV	WC	
bneq r1, TARGET	IF	ID	Q			•••		В	WB				С
xor r2, r3, r4		IF	ID	Q	EX	•••							
•••													
TARGET:													
and r5, r6, r7													IF

BHT use when branch taken, incorrectly predicted, register map backed up.

Register r1 not available until cycle 10.

Cycle 1: When branch in ID, backup (checkpoint) register map, compute target, read BHT and make prediction.

Cycle 10: Compute branch condition.

Cycle 11: Misprediction discovered. Squash reorder buffer past branch, switch to backed up register map, start fetching correct path.

Cycle 23: Branch commits.

! Predict not tak	en,	take	n. Re	gis	ter	map	back	ed up	•				
Cycle:	0	1	2	3				10	11	•••	21	22	23
div f0,f2, f4	ID	Q	DIV								DIV	WC	
bneq r1, TARGET	IF	ID	Q			•••		В	WB				С
xor r2, r3, r4		IF	ID	Q	EX								
• • •													
TARGET:													
and r5, r6, r7									IF	•••	•		

Global history must be accurate.

Why that's a problem:

! First branch:	Predi	ct	not	taken,	taken.	Registe	r map	backed	up.				
Cycle:	0	1	2	3		10	11	12	13	•••	21	22	23
div f0,f2, f4	ID	Q	DI	V							DIV	WC	
bneq r1, TARGET	IF	ID	Q		• • •	В	WB						С
beqz r2, SKIP		IF	' ID	Q	в.								
xor r2, r3, r4			IF	ID	EX								
TARGET:													
and r5, r6, r7							IF	ID	Q	EX			
beqz r4, LINE1								IF	ID	Q			
Cycle:	0	1	2	3		10	11	12	13	• • •	21	22	23

Cycle 2: beqz should see global history with bneq not taken.

Global history includes assumption that **bneq** not taken.

! First	branch:	Predi	ct	not	taken,	tak	en.	Re	gister	c map	backed	up.				
Cycle:		0	1	2	3				10	11	12	13	•••	21	22	23
div f0	,f2, f4	ID	Q	DI	V									DIV	WC	
bneq r	I, TARGEI	IF IF	ID	Q			•		В	WB						С
beqz r2	2, SKIP		IF	7 ID	) Q	В		•••								
xor r2	, r3, r4			IF	' ID	Q	EX.	•••								
TARGET:																
and r5	, r6, r7									IF	ID	Q	•••			
beqz r4	1, LINE1										IF	ID	•••			
Cycle:		0	1	2	3				10	11	12	13	•••	21	22	23

Cycle 3: Now global history includes assumption that bneq and first beqz not taken.

Cycle 11: Ooops, bneq misprediction discovered.

Global history has two incorrect assumptions ...

... unless they're fixed prediction for second beqz won't be accurate.

Cycle 12: beqz should see global history with bneq taken.

Global History in Two-Level Predictor with Dynamic Execution

Global history backed up (checkpointed) at each branch.

Predicted outcome shifted into global history.

If misprediction discovered, global history restored from backup ... ... just as the register map can be. Target Prediction: Predicting the outcome and target of a branch.

Branch Target Buffer: A table indexed by branch address holding a predicted target address.

Target Prediction

Put BTB in IF stage.

Use PC to read an entry from BTB.

If valid entry found, replace PC with predicted target.

With target correctly predicted, zero branch delay.

Static scheduled system (for clarity).

Cycle:		0	1	2	3	4	10	11	12	13	14
bneq r1,	TARGET	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB
xor r2,	r3, r4								IF	ID	EX
TARGET:											
and r5,	r6, r7		IF	ID	EX	MEM WB		IF	Х		

Cycle 0

BTB lookup and prediction. Predict taken.

Target from BTB will be clocked into PC.

### 12-29

### Target Prediction Example, continued.

Static scheduled system (for clarity).

Cycle: bneq	r1,	TARG	ET	0 IF	1 ID	2 EX	3 MEM	4 WB	10 IF	11 ID	12 EX	13 MEM	14 WB
xor	r2,	r3,	r4								IF	ID	EX
TARGET	:												
and	r5,	r6,	r7		IF	ID	EX	MEM WB		IF	Х		

Cycle 1

Start fetching predicted target.

Execute branch instruction (in ID).

Check predicted outcome and predicted target.

Correct predictions, continue execution.

-30	Τa	arget	: Pre	edict	ion Examj	ple, con	tinu	ed.			12-
Cycle:	0	1	2	3	4	10	11	12	13	14	
bneq r1, TARGET	IF	ID	EX	MEM	WB	IF	ID	EX	MEM	WB	
xor r2, r3, r4								IF	ID	EX	
TARGET:											
and r5, r6, r7		IF	ID	EX	MEM WB		IF	Х			

Cycle 10

BTB lookup and prediction. Predict taken.

Target from BTB will be clocked into PC.

Cycle 11

Start fetching predicted target.

Execute branch instruction (in ID).

Ooops, incorrect outcome prediction ... ... replace target with nop ... ... and clock correct target into PC. What BTB predicts for branch instructions:

That instruction will be a CTI.

If CTI is a branch, that branch is taken.

CTI target.

For branches and non-indirect jumps (j, jal)...
... predicting target is easy, since target always same.
bneq r1, LOOP ! Target always PC + 4 + 4 \* LOOP
j LINEJ ! Target always PC + 4 + 4 \* LINEJ

For register-indirect jumps (jr, jalr) ...
... prediction depends on predictable behavior.
jr r1 ! Target is in r1. Can be different each time.
jalr r1 ! Target is in r1. Can be different each time.

# Behavior of Register-Indirect Jumps

Predictability depends on how jumps used.

Major Uses

• Procedure Passed as Parameter

For example, function passed to the C library's qsort.

These rarely change so target is predictable.

• Case Statements

These change, and so prediction more difficult.

12-32

Separate techniques used for procedure returns and other indirect jumps.

Return Address Prediction

Keep a stack of (what appear to be) return addresses.

Other Indirect Jumps Prediction

Predict last target.

Use global branch history to index BTB.

#### Operation

Hardware keeps a stack of return addresses.

BTB stores whether instruction is a return.

When a call instruction encountered push return address on stack.

When BTB identifies instruction as a return target address is popped off stack.

#### Effectiveness

Works fairly well.

Can be confused when returns skipped (as with long jumps).

Costly to implement precisely with dynamic scheduling.

Can be used for everything except return instructions.

Last time instruction executed target address stored in BTB.

If entry found and predicted taken (for a branch), last target address used.

Effectiveness:

Perfect for non-indirect jumps and branches (if taken).

Reasonably effective on indirect branches.

Use Global History

Can be used for everything except return instructions.

Much more effective on than last target.

Consider code for C switch statement:

```
! Possible code for a switch statement.
! switch( r2 ) { case 0: foo(); break; case 1: bar(); break; ... }
! Set r1 to base of switch address table.
lhi r1, #0x1234
ori r1, r1, #0x5670
! Multiply switch index by stride of table (4 bytes per address).
slli r3, r2, #2
! Get address of case code address.
add r1, r1, r3
! Get case code address.
lw r4, 0(r1)
! Jump to case code.
jr r4
```

If r2 rarely changes, jr predictable.

## Possible BTB Contents

Target address.

History information (replaces BHT).

Tag, to detect collisions.