ECE 4750 Computer Architecture, Fall 2016

T13 Advanced Processors: Branch Prediction

School of Electrical and Computer Engineering
Cornell University

revision: 2016-11-29-19-25

1. Branch Prediction Overview

Assume incorrect branch prediction in dual-issue I2OL processor.

`bne`													
`opA`													
`opB`													
`opC`													
`opD`													
`opE`													
`opF`													
`opG`													
`opTARG`													

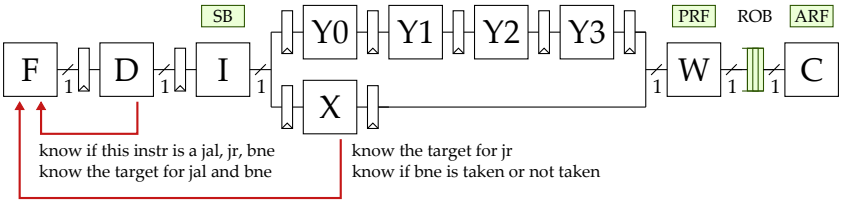
Assume correct branch prediction in dual-issue I2OL processor.

`bne`													
`opA`													
`opTARG`													
`opX`													
`opY`													
`opZ`													

Three critical pieces of information we need to predict control flow:

- (1) Is this instruction a control flow instruction?
- (2) What is the target of this control flow instruction?
- (3) Do we redirect control flow to the target or next instr?

When do we know these critical pieces of information?



	`jal`	`jr`	`bne`
(1) Is this instruction a control flow instruction?	D	D	D
(2) What is the target of this control flow instruction?	D	X	D
(3) Do we redirect ctrl flow to the target or next instr?	D	D	X

What do we need to predict in F stage vs. D stage?

	`jal`	`jr`	`bne`
F stage	predict 1,2,3	predict 1,2,3	predict 1,2,3
D stage	no prediction	predict 2	predict 3

2. Software-Based Branch Prediction

- Static software hints
- Branch delay slots
- Predication

2.1. Static Software Hints

Software provides hints about whether a control flow instruction is likely to be taken or not taken. These hints are part of the instruction and thus are available earlier in the pipeline (e.g., in the D stage).

`bne.t`													
`opA`													
`opTARG`													
`bne.nt`													
`opY`													
`opZ`													

What if the hint is wrong?

`bne.t`													
`opA`													
`opTARG`													
`bne.nt`													
`opA`													
`opB`													

2.2. Branch Delay Slots

Without branch delay slots must squash fall through instructions if branch is taken.

`bne`													
`opA`													
`opB`													
`targ`													

With branch delay slots compiler can put useful work in the slots. Instructions in the delay slots are always executed regardless of branch condition.

`bne`													
`opA`													
`opB`													
`targ`													

2.3. Predication

Not really "prediction". Idea is to turn control flow into dataflow completely eliminating the control hazard.

Conditional move instructions conditionally move a source register to a destination register.

`movn rd, rs1, rs2` if (`R[rs2]` $!= 0$) `R[rd]` $\leftarrow$ `R[rs1]`

`movz rd, rs1, rs2` if (`R[rs2]` $== 0$) `R[rd]` $\leftarrow$ `R[rs1]`

Pseudocode

```
if ( a < b )
  x = a
else
  x = b
```

w/o Predication

```
 slt  x1, x2, x3
 beq  x1, x0, L1
 addi x4, x2, x0
 jal  x0, L2
L1:
 addi x4, x3, x0
L2:
```

w/ Predication

```
slt  x1, x2, x3
movz x4, x2, x1
movn x4, x3, x1
```

Full predication enables almost all instructions toe be executed under a predicate. If predicate is false, instruction should turn into a NOP.

Pseudocode

```
if ( a < b )
  opA
  opB
else
  opC
  opD
```

w/ Predication

```
slt.p p1, x2, x3
( p1) opA
( p1) opB
(!p1) opC
(!p1) opD
```

- What if both sides of branch have many instructions?
- What if one side of branch has many more than the other side?

3. Hardware-Based Branch Prediction

- Fixed branch predictor
- Branch history table (BHT) predictor
- Two-level predictor for temporal correlation
- Two-level predictor for temporal correlation
- Generalized two-level predictors
- Tournament predictor
- Branch target buffer (BTB) predictor

3.1. Fixed Branch Predictor

- Always predict not taken
  - What we have been assuming so far
  - Simple to implement and can perform prediction in F
  - Poor accuracy, especially on very important backwards branch in loops
- Always predict taken
  - Difficult to implement: we don't know if this is a branch until D
  - Difficult to implement: we don't know target until at least D
  - Could predict not taken in F, and then adjust in D
  - Poor accuracy, especially on if/then/else
- Predict taken for backward branches
  and predict not taken for forward branches
  - Difficult to implement: we don't know if this is a branch until D
  - Difficult to implement: we don't know target until at least D
  - Could predict not taken in F, and then adjust in D
  - Better accuracy

```
loop:              <------------------.
 lw   x1, 0(x2)                       | backward
 lw   x3, 0(x4)                       | branches
 slt  x5, x1, x3                      | taken on avg
 beq  x5, x0, L1   --. forward        | 90%
 addi x6, x1, x0     | branches       |
 jal  x0, L2         | taken on avg   |
L1:                <-' 50%            |
 addi x6, x3, x0                      |
L2:                                   |
 sw   x6, 0(x7)                       |
 addi x2, x2, 4                       |
 addi x4, x4, 4                       |
 addi x7, x7, 4                       |
 addi x8, x8, -1                      |
 bne  x8, x0, loop -------------------'
```

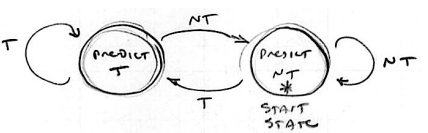
- For now let's focus on conditional branches as opposed to unconditional jumps
- Let's assume we always predict not-taken in the F stage
- In the D stage, we know if the instruction is a branch and we know the target of the branch
- So key goal is to predict whether or not we need to redirect the control flow, i.e., to predict the branch outcome in the D stage instead of waiting until the X stage
- By doing this prediction in the D stage we can reduce the branch misprediction penalty by several cycles although it is still not zero if we predict the branch is taken

3.2. Branch History Table (BHT) Predictor

How can we do better? Exploit structure in the program, namely temporal correlation: the outcomes of specific static branch in the past may be a good indicator of the outcomes of future dynamic instances of the same static branch.

One-Bit Saturating Counter

Remember the previous outcome of a specific static branch and predict the outcome will be the same for the next dynamic instance of the same branch.



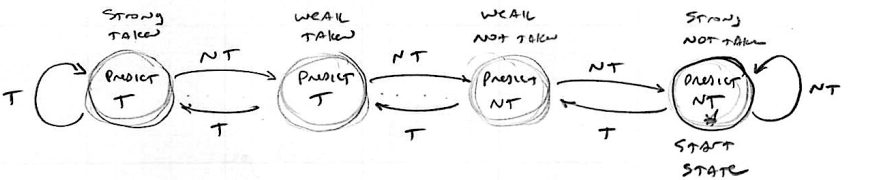
Consider how this saturating counter would be have for a backwards branch in a loop with four iterations. Assume the entire loop is executed several times.

Iteration	Prediction	Actual	Mispredict?
1			
2			
3			
4			
1			
2			
3			
4			

Exploiting temporal correlation works well, but a one-bit saturating counter will always mispredicts the backwards branch in a loop twice. Loops are *very* common!

Two-Bit Saturating Counter

Remember the last *two* outcomes of a specific static branch. Require two consecutive "counter examples" before changing the prediction.

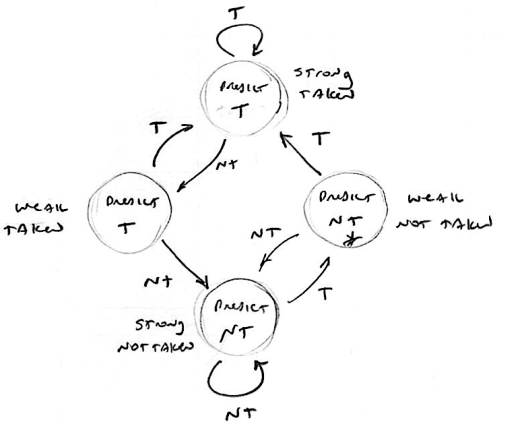


Consider how this saturating counter would be have for a backwards branch in a loop with four iterations. Assume the entire loop is executed several times.

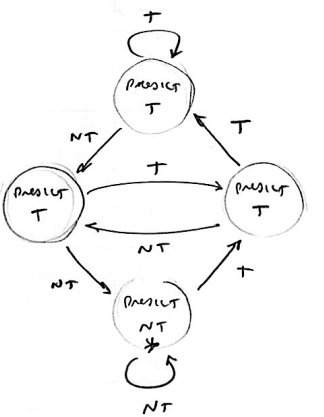
Iteration	Prediction	Actual	Mispredict?	ST	WT	WNT	SNT
1				○	○	○	○
2				○	○	○	○
3				○	○	○	○
4				○	○	○	○
1				○	○	○	○
2				○	○	○	○
3				○	○	○	○
4				○	○	○	○

What if start state is strongly taken?

Other Two-Bit FSM Branch Predictors



JUMP DIRECTLY TO STRONG FROM WEAK

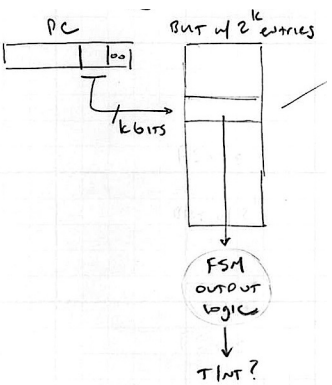


BIASED TOWARDS PREDICTING BRANCHES AS TAKEN

SEE Fig 5.8 IN SHEN + LIPASTI FOR OTHER ALTERNATIVES

Branch History Table

- So far we have focused on a simple FSM that exploits temporal correlation to make a prediction for a *specific static branch*
- To make predictions for many different static branches, we need to keep track of a *dedicated* FSM per static branch
- A branch history table (BHT) is a table where each entry is the state of the FSM for a different static branch.



- Two PC's can "alias" to the same entry in BHT
- Aliasing is similar to a cache conflict
- We could store the PC as a tag along with the FSM state to make sure we don't mix up the FSM state across two static branches
- Storing the PC is too expensive though, so we can just let branches alias and this just reduces the branch prediction accuracy
- Can reduce aliasing with larger BHT

BHT with 4k entries and 2bits/entry = 80–90% accuracy

How do we continue to improve prediction accuracy? Exploit even more complicated temporal correlation.

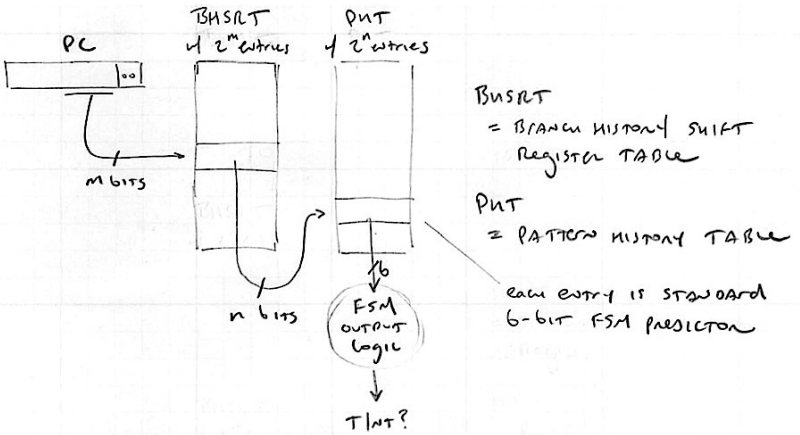
More Complicated Temporal Correlation

Often a branch exhibits more complicated patterns than just "always taken" or "always not taken". Could develop a more complicated FSM, but these patterns vary per branch. We want per branch customized FSMs.

```
void convolve( int B[], int A[], int size ) {
  for (int i = 2; i < size-2; i++)
    for (int j=0; j < 5; j++)
      B[i - (2-j)] = A[i] * COEFF[j]
}
```

Can we predict that every fifth dynamic instance of the backwards loop branch will be not taken?

3.3. Two-Level Predictor For Temporal Correlation



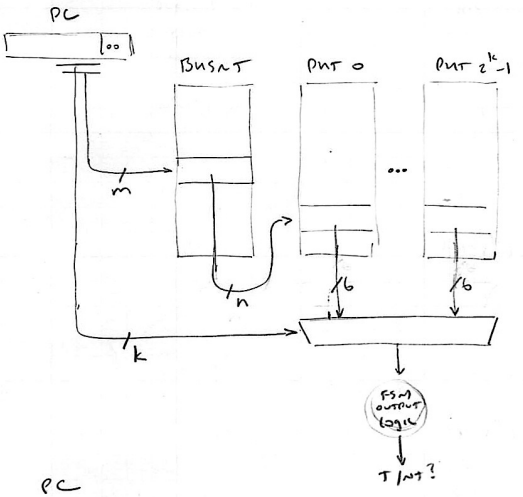
When a branch is taken or not taken we shift in either a one (taken) or a zero (not taken) into the least significant bit of the corresponding BHSR.

Index	Value
...	
0111	ST
1000	WT
1001	WT
1010	WT
1011	ST
1100	WT
1101	ST
1110	ST
1111	SNT

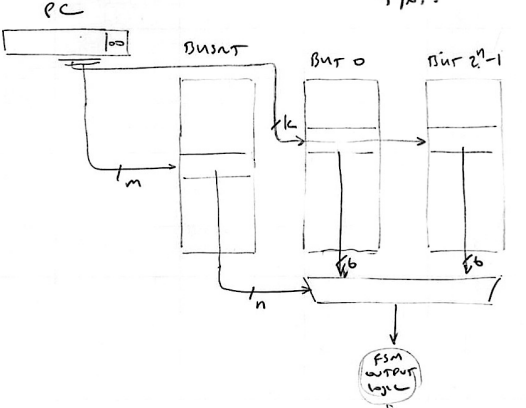
- BHSR captures temporal pattern for that branch
- We use the BHSR to index into the PHT. A BHT has an entry per branch, but a PHT has an entry per branch pattern.
- The PHT says for a given pattern over the past n executions of a branch, should I take or not take the next execution of this branch?
- Once the two-level predictor is warmed up for previous nested loop example, the state of the PHT would be what is shown on the left
- Need at least four bits of "history" to learn this pattern and perfectly predict this branch

PROBLEM: MULTIPLE BRANCHES WITH SAME HISTORY MIGHT NEED DIFFERENT PREDICTIONS. IN OTHER WORDS, ALIASING IN THE PHT CAN RESULT ACCURACY

SOLUTION: ADD MULTIPLE PHTS, USE BITS FROM PC TO CHOOSE WHICH PHT TO USE



ISOMORPHIC - TWO DIFFERENT WAYS OF DRAWING THE SAME TWO-LEVEL STRUCTURE.



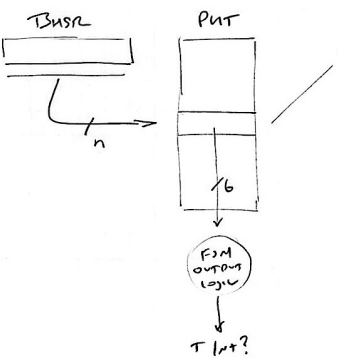
HW BPRED: EXPLOITING SPATIAL CORRELATION

THE WAY ONE BRANCH IS RESOLVED MAY BE A GOOD INDICATION OF THE WAY A LATER (DIFFERENT) BRANCH WILL RESOLVE

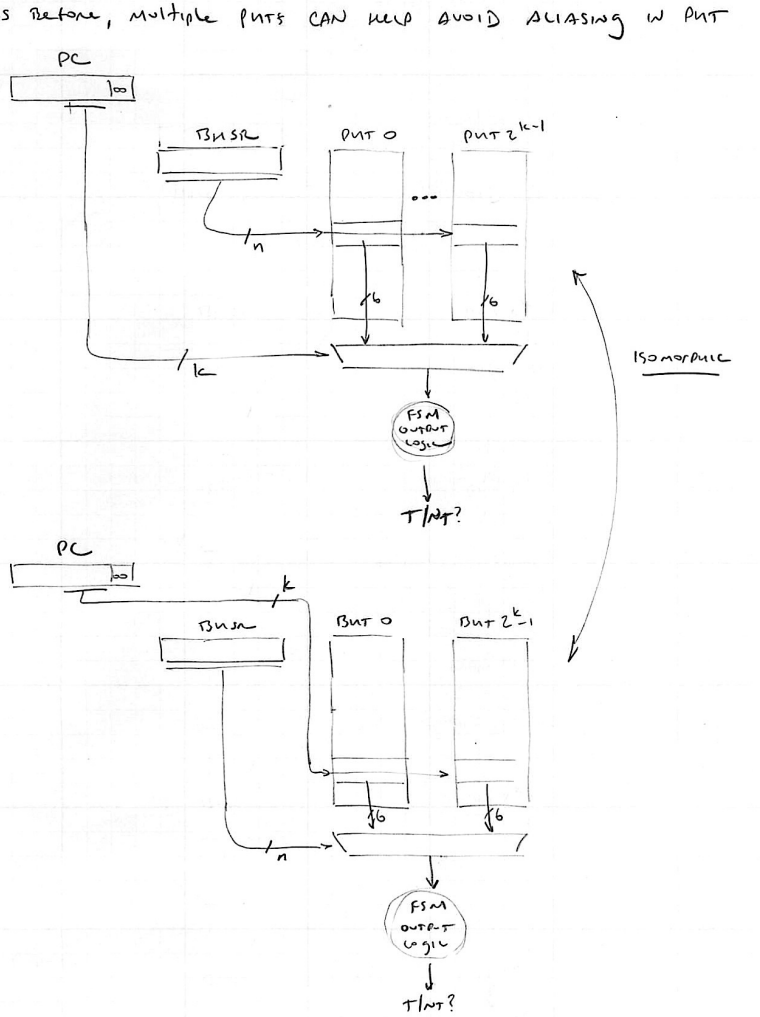
```
if (x < 7)                slt   r2, r1, 7
  Y++                     beq   r2, L1        branch ϕ
if (x < 5)                ADDIU r3, r3, 1
  Z++                   L1:
                          slt   r2, r1, 5
                          beq   r2, L2        branch 1
                          ADDIU r4, r4, 1
                        L2:
```

IF BRANCH ϕ IS TAKEN (IE $x \geq 7$)
THEN BRANCH 1 IS ALWAYS TAKEN (IE x MUST BE ≥ 5)

SO WHETHER BRANCH ϕ IS TAKEN OR NOT TAKEN CAN BE USED TO PREDICT IF WE SHOULD TAKE BRANCH 1

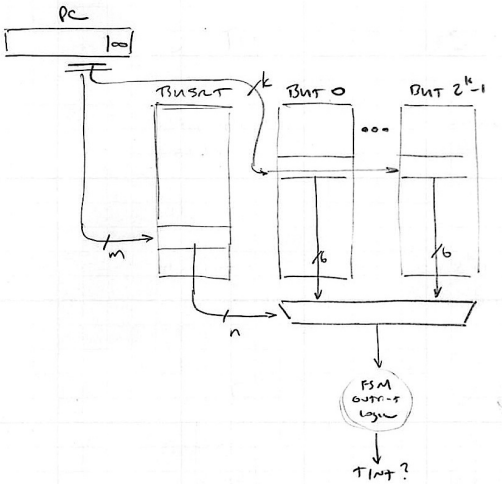


FOR ABOVE EXAMPLE, BHSR WILL CAPTURE HISTORY – SO WE WILL KNOW WHEN FIRST BRANCH IS TAKEN, AND THAT VALUE (5) OF THE BHSR WILL POINT TO AN ENTRY IN THE PHT THAT PREDICTS TAKEN.


AS BEFORE, MULTIPLE PHTS CAN HELP AVOID ALIASING IN PHT
PC
BHSR
PHT 0
PHT 2^k-1
n
k
FSM OUTPUT LOGIC
T/NT?
ISOMORPHIC
PC
k
BHSR
PHT 0
PHT 2^k-1
n
FSM OUTPUT LOGIC
T/NT?

GENERALIZED TWO-LEVEL BHTS

COMBINED APPROACH TO EXPLOIT BOTH COMPLEX TEMPORAL CORRELATION AND SPATIAL CORRELATION



DIFFERENCE FROM DISCUSSION ON COMPLEX TEMPORAL CORRELATION IS THAT WE PURPOSELY CHOOSE A SMALLER m TO CAUSE ALIASING IN THE BHSR SINCE THIS ALIASING ALLOWS US TO CAPTURE SPATIAL CORRELATION

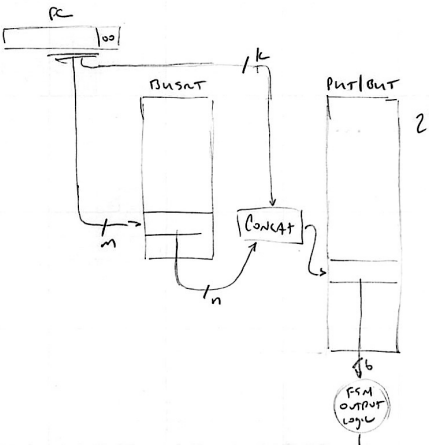
(CHOOSE HIGHER ORDER m BITS)

		ONE PHT $k=0$	PHT for each PC $0<k<30$	PHT for each PC $k=30$
ONE BHSR	$m=0$	GAg	GAs	GAp
	$0<m<30$	PAg	PAs	PAp
ONE BHSR for each PC	$m=30$	SAg	SAs	SAp

97% ACCURACY

g-select

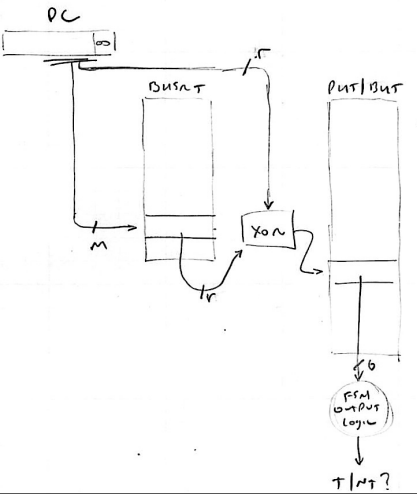
ISOMORPHIC TO PREVIOUS FIGURE



$2^{(k+n)}$ entries

g-share

INSTEAD OF CONCATENATING XOR BITS FROM PC WITH BHSR, HELPS AVOID ALIASING IN THE PHT/BHT MORE EFFECTIVELY

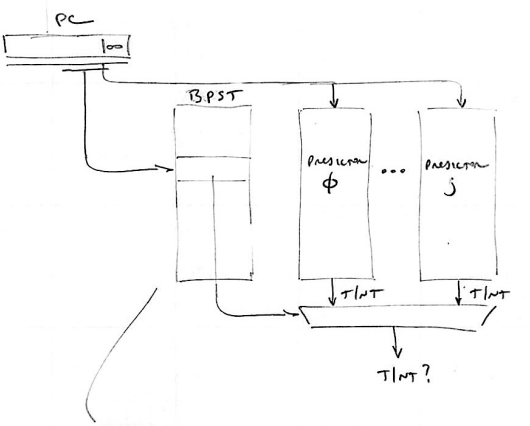


2^r entries

TOURNAMENT PREDICTORS

DIFFERENT PREDICTORS ARE BETTER AT PREDICTING DIFFERENT TYPES OF BRANCHES

- ONE-LEVEL 2-BIT SATURATING COUNTER — LOOPS
- TWO-LEVEL gshare — IRREGULAR CODE

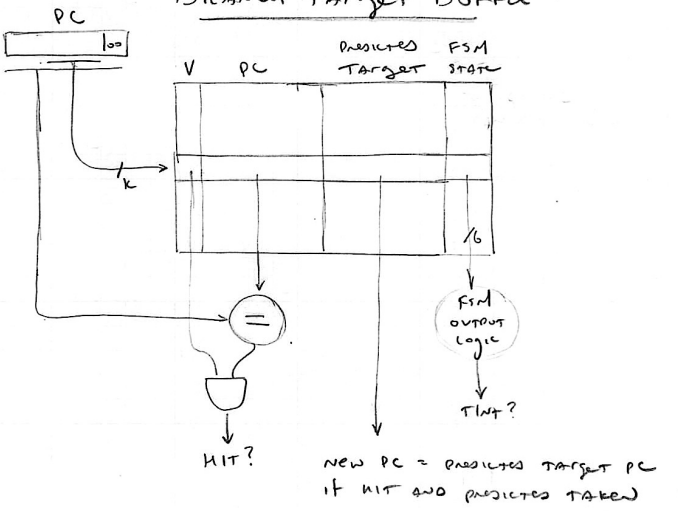


BRANCH PREDICTOR SELECTOR TABLE
— PREDICTS WHICH BRANCH PREDICTOR WE SHOULD USE

HW BPRED: PREDICTING TARGET ADDRESS

EVEN WITH BEST POSSIBLE PREDICTION OF BRANCH OUTCOME STILL NEED TO WAIT FOR TARGET ADDRESS TO BE DETERMINED.

BRANCH TARGET BUFFER

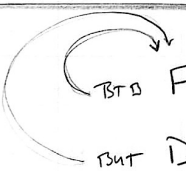


CAN PUT BTB IN FETCH STAGE

- PREDICTING IF PC POINTS TO A BRANCH
- PREDICTING TARGET OF BRANCH
- PREDICTING IF BRANCH IS TAKEN.

SOMETIMES $b=0$, IF HIT THEN ASSUME PREDICT TAKEN

COMBINE BTB AND BHT



BTB is much more expensive than BHT, but BTB earlier in pipeline + can accelerate JR

I

X UPDATE BHT for BRANCHES
UPDATE BTB for JR

COMBINE BTB w/ few entries with BHT w/ many entries

M

W

RETURN ADDRESS STACK PREDICTOR

BTB only works for JR function call returns if always call function from same place (not realistic)

STACK PREDICTOR

- Push Target Address on stack for JAL/JALR
- Pop off Target Address for JR to predict Target

Move STACK PREDICTOR into FETCH and predict which PC's are JR.

USE TOURNAMENT PREDICTOR TO CHOOSE BETWEEN BTB AND STACK PREDICTOR.