Cornell University

ECE4750/CS4420 Computer Architecture L10: Branch Prediction

Edward Suh
Computer Systems Laboratory
suh@csl.cornell.edu

Announcements

- Lab2 and prelim grades
- Back to the regular office hours

Overview

- Dynamic scheduling handles various types of data hazards
  - BUT, how about control hazard?
- Today – dealing with control hazards through prediction
  - Branch prediction
    - Temporal correlation
    - Spatial correlation
  - Branch target buffer (BTB)
- Reading
  - H&P Chapter 2.3
  - Scott McFarling, "Combining Branch Predictors", WRL Technical Note TN-36


Cornell University

Run-Length Between Branches

Average dynamic instruction mix from SPEC92:

	SPECint92	SPECfp92
ALU	39 %	13 %
FPU Add		20 %
FPU Mult		13 %
load	26 %	23 %
store	9 %	9 %
branch	16 % 6.25	8 % 12.5
other	10 %	12 %

SPECint92: *compress, eqntott, espresso, gcc , li*
SPECfp92: *doduc, ear, hydro2d, mdijdp2, su2cor*

What is the average *run length* between branches?

based on percentage: $\frac{100}{16}$, $\frac{100}{8}$


Cornell University

MIPS Branches and Jumps

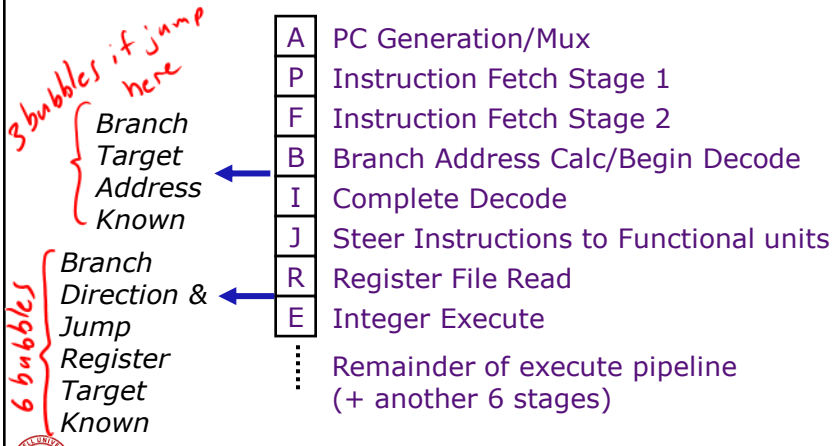
Each instruction fetch depends on one or two pieces of information from the preceding instruction:

1) Is it a jump or branch?
   ↳ is it a taken branch?

2) Target address (if taken branch or jump)

Branch Penalties in Modern Pipelines

UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)



Branch Prediction

- Motivation: branch penalties limit performance of deeply pipelined processors
  - it gets worse with issue width and pipeline depth
    ↑ # of simultaneous instructions
- Required hardware support:
  1) Prediction
     a) is it a branch?
     b) outcome?
     c) if taken, target?
  2) Recovery
     fix pipeline

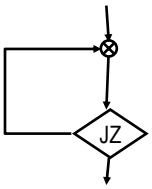
Cornell University

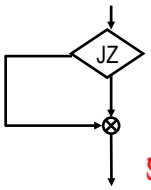
Static Branch Predication

Overall probability a branch is taken is ~60-70% but:

backwards branch ⇒ loop

90% chance to take it





forward branch ⇒ if/else

50% chance to take it

methods

1) SW ⇒ ISA hint
   bnez preferred taken
   beqz preferred not taken
2) HW determine based on forward/backwards

Cornell University

Dynamic Branch Prediction

Learning based on past behavior

Temporal correlation

The way a branch resolves may be a good predictor of the way it will resolve at the next execution

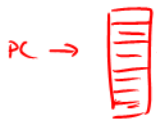
Spatial correlation

Several branches may resolve in a highly correlated manner *(a preferred path of execution)*

Cornell University

Branch History Table (BHT)

- Small memory to store the history of each branch
  - Sometimes called Pattern History Table (PHT)
    - use LSB's of branch PC to index
    - usually no tag ⇒ always hit
      - ↳ doesn't affect correctness, only performance
      - ↳ rather use the memory to store history



- Simplest: remember last outcome – one-bit BHT
  - Targets highly biased branches
  - Ex) loop branch taken 9/10 times; what is the misprediction rate?

2 mispredictions. if I assume always taken, get better performance how to fix?

	Branch Instance											
	1	2	3	4	5	6	7	8	9	10	11	12
Prediction	NT	T	T	T	T	T	T	T	T	T	NT	T
Outcome	T	T	T	T	T	T	T	T	T	NT	T	T

↑ end of loop

Cornell University

2-bit BHT

solution for earlier problem. Allows for gradient.

- Solution: two-bit prediction (*saturating counter, generalize to n-bits*)

On ¬taken ↓	On taken ↑			
		1	1	Strongly taken
		1	0	Weakly taken
		0	1	Weakly ¬taken
		0	0	Strongly ¬taken

From state 11, need 2 NT's to switch prediction

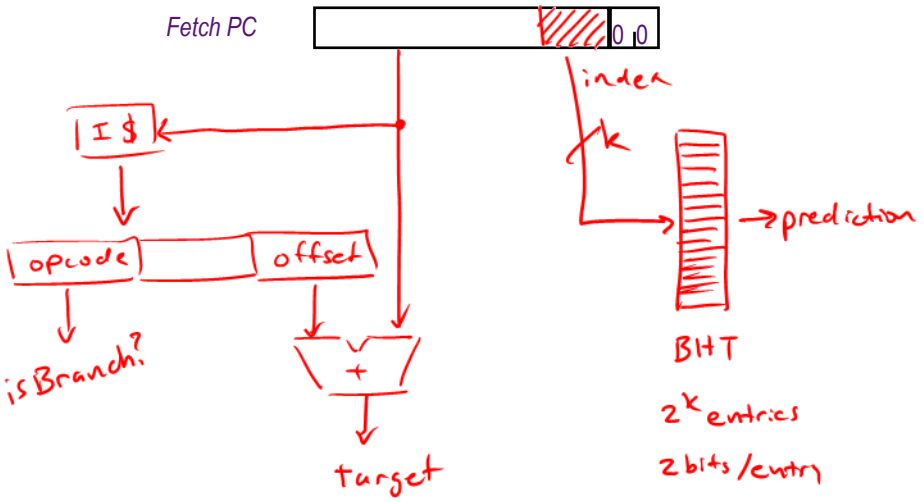
- (note: textbook & HW3 follows different encoding)

	Branch Instance											
	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
Prediction	01	10	11	11	11	11	11	11	11	11	10	11
Outcome	T	T	T	T	T	T	T	T	T	NT	T	T

Taken
Not Taken

Cornell University

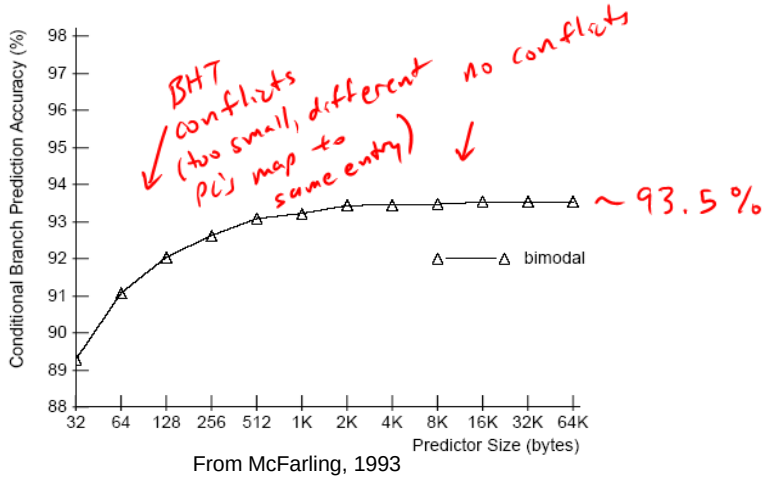
2-bit BHT Implementation



Cornell University

Accuracy of 2-bit BHT

- 4K vs. infinite # entries for SPEC89 (average)



From McFarling, 1993

Local History Predictor

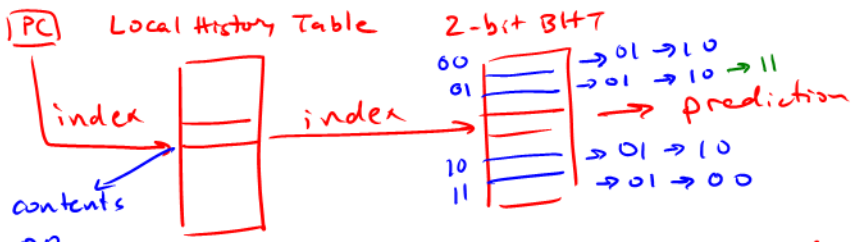
```
for (i=0; i<N; i++){
    for (j=0; j<3; j++)
    {
        ...
    }
}
```

- Exploit the history of a particular branch
  - The loop: 2 taken branches + 1 not taken branch

T T NT } use this to figure out branch behavior

- Select a prediction based on a branch's recent history
  - Local history table remembers the history of a particular branch
  - Use 2-bit BHT for a prediction

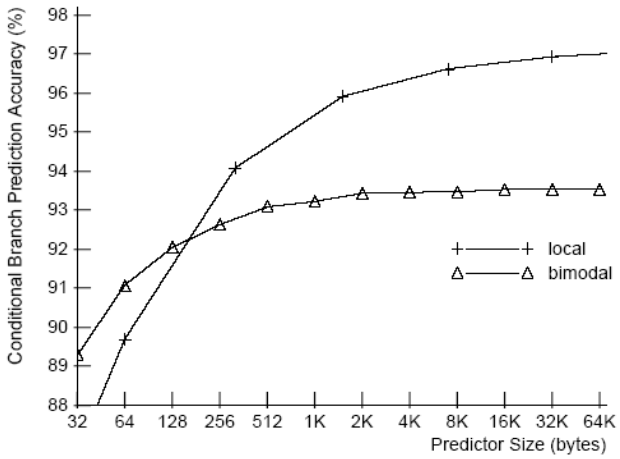
Local Predictor Example



	Branch Instance											
	1	2	3	4	5	6	7	8	9	10	11	12
Prediction	NT	NT	NT	NT	T	NT	T	T	NT	T	T	NT
Outcome	T	T	NT	T	T	NT	T	T	NT	T	T	NT

Cornell University

Local Predictor Accuracy



From McFarling, 1993

Cornell University

Exploiting Spatial Correlation

Yeh and Patt, 1992

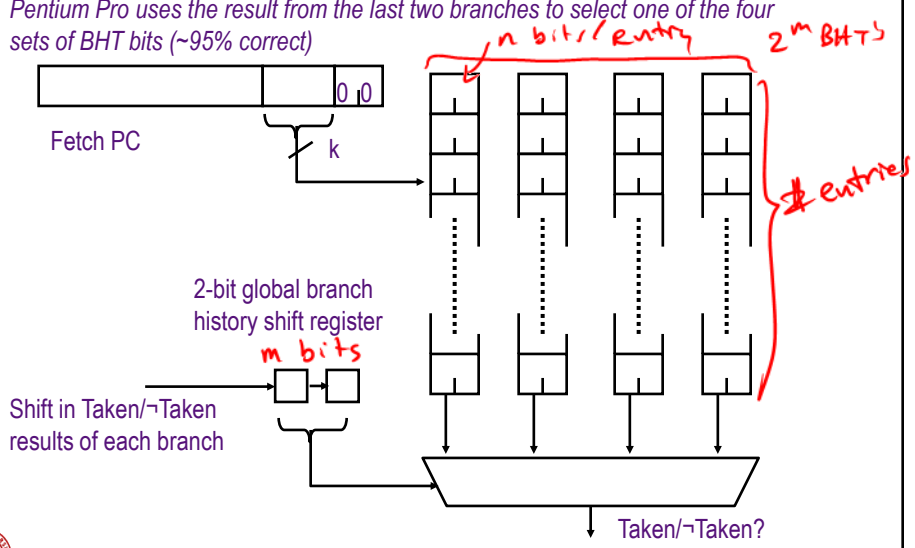
```
if (x[i] < 7) then
      y += 1;
if (x[i] < 5) then
      c -= 4;
```

- Look at behavior of recently executed branches
  - Ex) If first condition false, second condition also false
- Select a prediction based on the outcome of past branches (*global* history)

Cornell University

Global-Select Branch Predictor

Pentium Pro uses the result from the last two branches to select one of the four sets of BHT bits (~95% correct)



Cornell University

(m,n) Branch Predictors

- Generalized global-select BPs
  - m-bit GHR
  - n-bit BHTs
- m = 0?

no global history, just 1 BHT

- How many bits are required for (2,2) BP?

(m,n) BP?

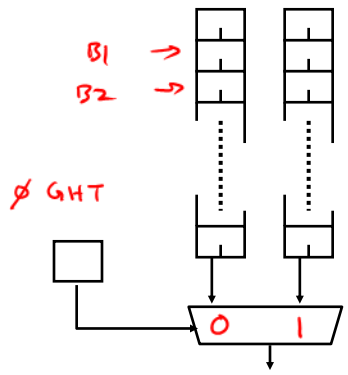
2^m (BHTs) × (# entries) × $\frac{n \text{ bits}}{\text{entry}}$

Cornell University

Global Predictor Example

2 bit BHT
01 default

```
B1: if (x[i] < 7) then
        y += 1;
B2: if (x[i] < 5) then
        c -= 4;
```



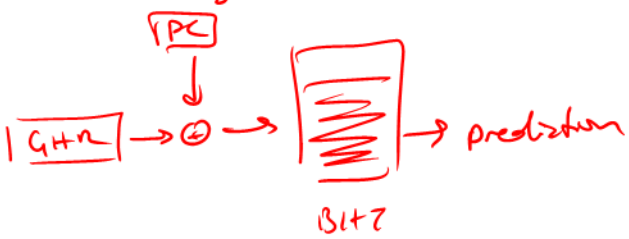
	Branch Instance											
	B1	B2	B1	B2	B1	B2	B1	B2	B1	B2	B1	B2
Prediction	NT	NT	NT									
Outcome	T	T	NT	NT	T	T	NT	NT	NT	NT	T	T

Cornell University

Global Predictor Variants

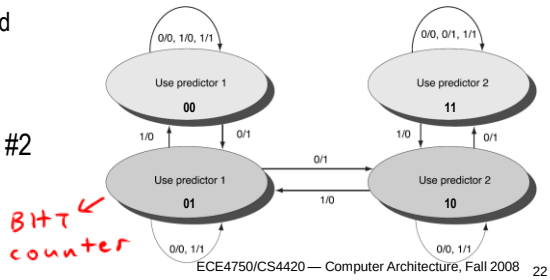
- Global

- Global-Share (aka g-share)

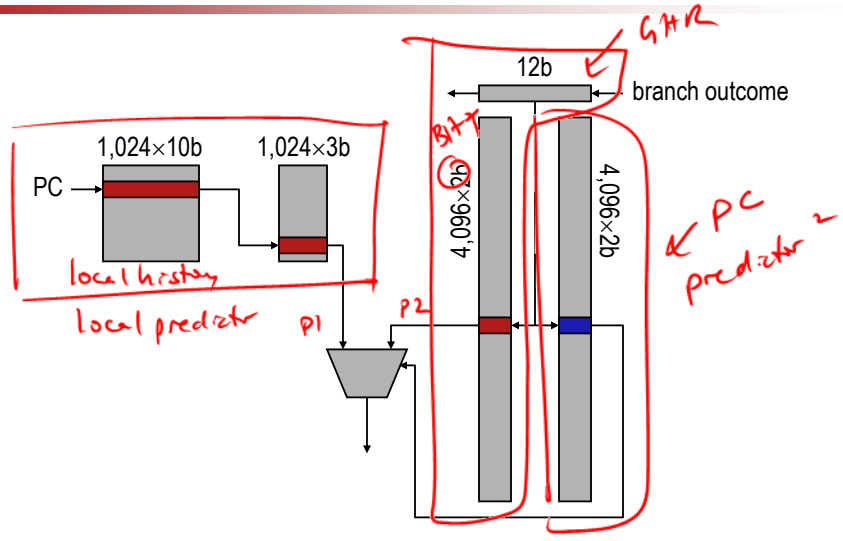


Tournament Predictors

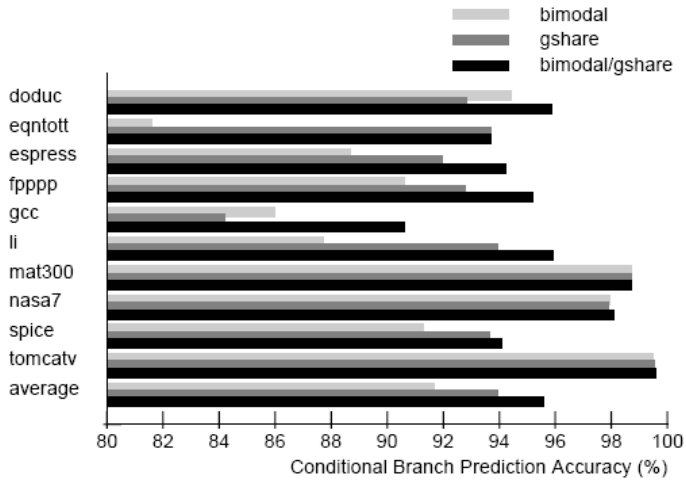
- Multilevel predictors – predicting predictors!
  - which predictor works best for this particular branch?
- Useful to pick between local, global
- All predictors cast vote
  - only predicted predictor's used
  - all predictions compared against branch outcome, then
    - each branch predictor updates its content
    - predictor history table updated
- Ex.: two-bit saturating counter, select between two predictors
  - 0X use pred. #1, 1X use pred. #2



Tournament Predictor: Alpha 21264

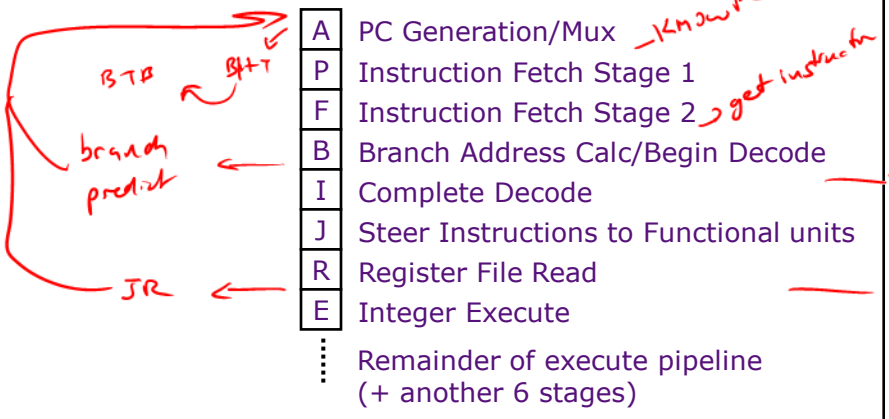


Combined Predictor Performance



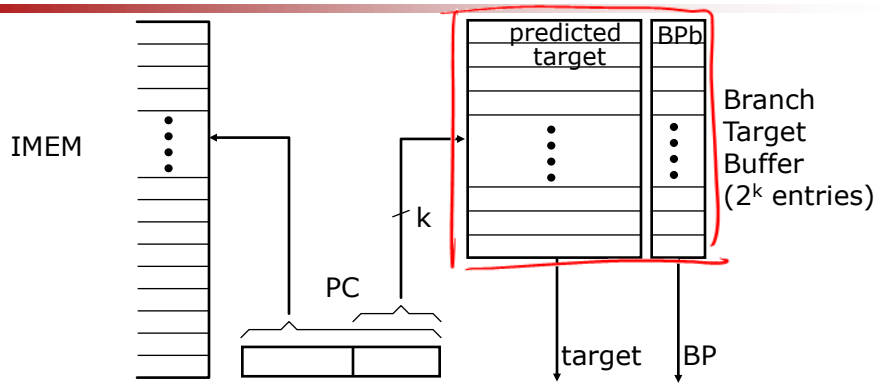
Limitations of BHTs

Only predicts branch direction. Therefore, cannot redirect fetch stream until after branch target is determined.



UltraSPARC-III fetch pipeline

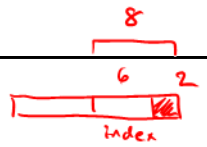
Branch Target Buffer (BTB)



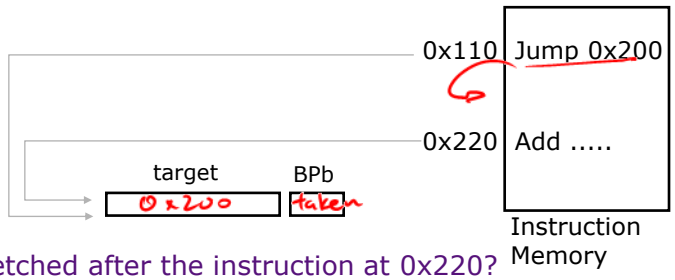
BP bits are stored with the predicted target address.

IF stage: *If (BP=taken) then nPC=target else nPC=PC+4*
later: *check prediction, if wrong then kill the instruction and update BTB & BPb else update BPb*

Address Collisions



Assume a 64-entry BTB



What will be fetched after the instruction at 0x220?

BTB prediction = 0x200

Correct target = 0x214 → PC + 4

not every instruction will jump

BTB is only for Control Instructions

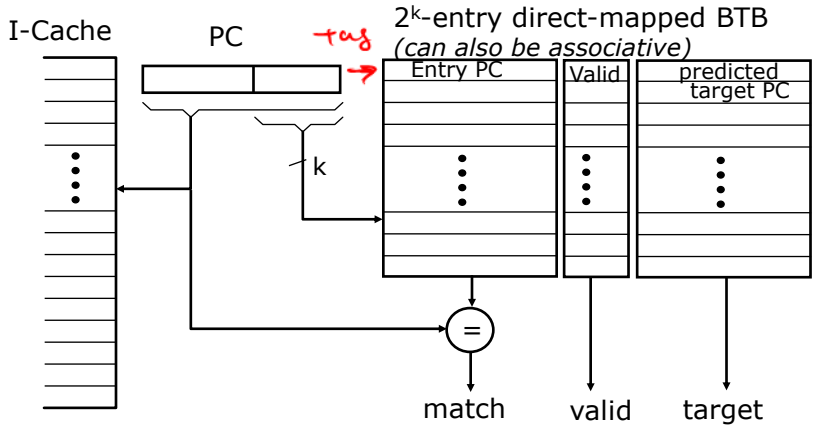
BTB contains useful information for branch and jump instructions only

→ do not BTB for other instructions

How to achieve this effect without decoding the instruction?

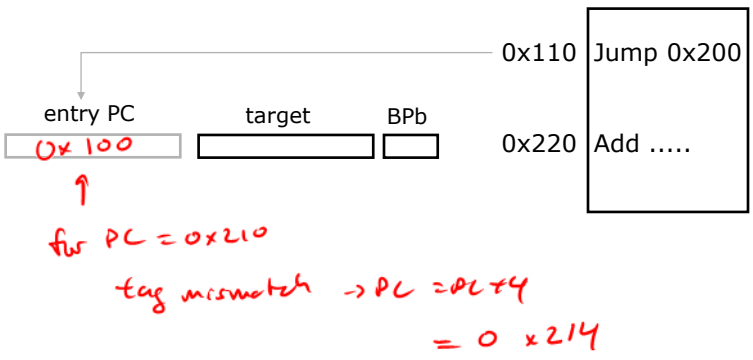
→ store tag!

Branch Target Buffer (BTB)



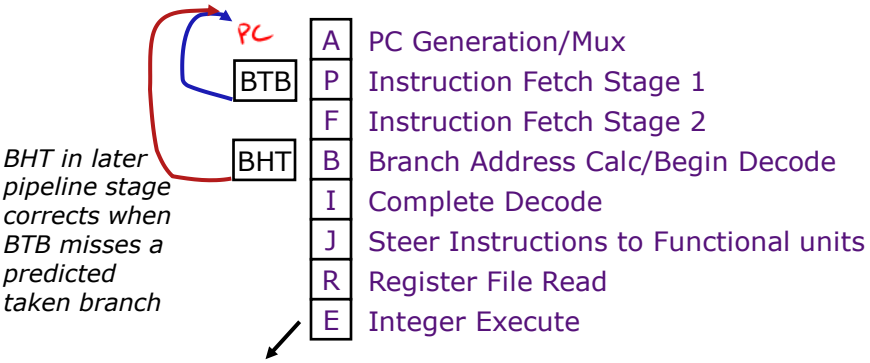
- Keep both the branch PC and target PC in the BTB
- PC+4 is fetched if match fails
- Only *taken* branches and jumps held in BTB
- Next PC determined *before* branch fetched and decoded

Consulting BTB Before Decoding



Combining BTB and BHT

- BTB entries are considerably more expensive than BHT, but can redirect fetches at earlier stage in pipeline and can accelerate indirect branches (JR)
- BHT can hold many more entries and is more accurate



BTB/BHT only updated after branch resolves in E stage

Uses of Jump Register (JR)

- Switch statements (jump to address of matching case)

BTB works well if you jump to same case

- Dynamic function call (jump to run-time function address)

BTB works well if jumping to same function

- Subroutine returns (jump to return address)

BTB works well if you return to the same place

How well does BTB work for each of these cases? ok, but case-dependent

Return Address Stack

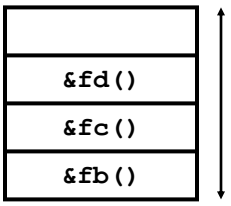
Small structure to accelerate JR for subroutine returns, typically much more accurate than BTBs.

```
fa() { fb(); }
fb() { fc(); }
fc() { fd(); }
```

return from
fd → fc
fc → fb
fb → fa

Push call address when function call executed

Pop return address when subroutine return decoded



k entries (typically k=8-16)