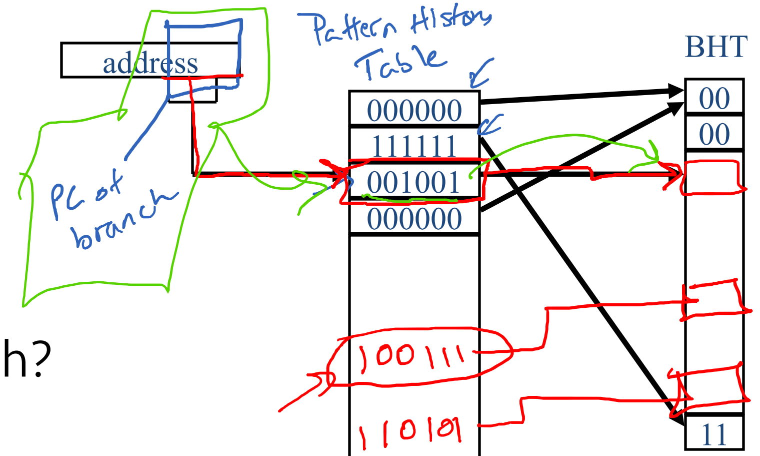
Can We Do Better?

PC: 8004 beq

- Can we get more information dynamically than just the recent bias of this branch?
- We can look at patterns (2-level *local* predictor) for a particular branch.
  - last eight branches 00100100, then it is a good guess that the next one is "1" (taken)



How many entries are in the BHT? 2^6

- even/odd branch?

Can We Do Better?

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Can We Do Better?

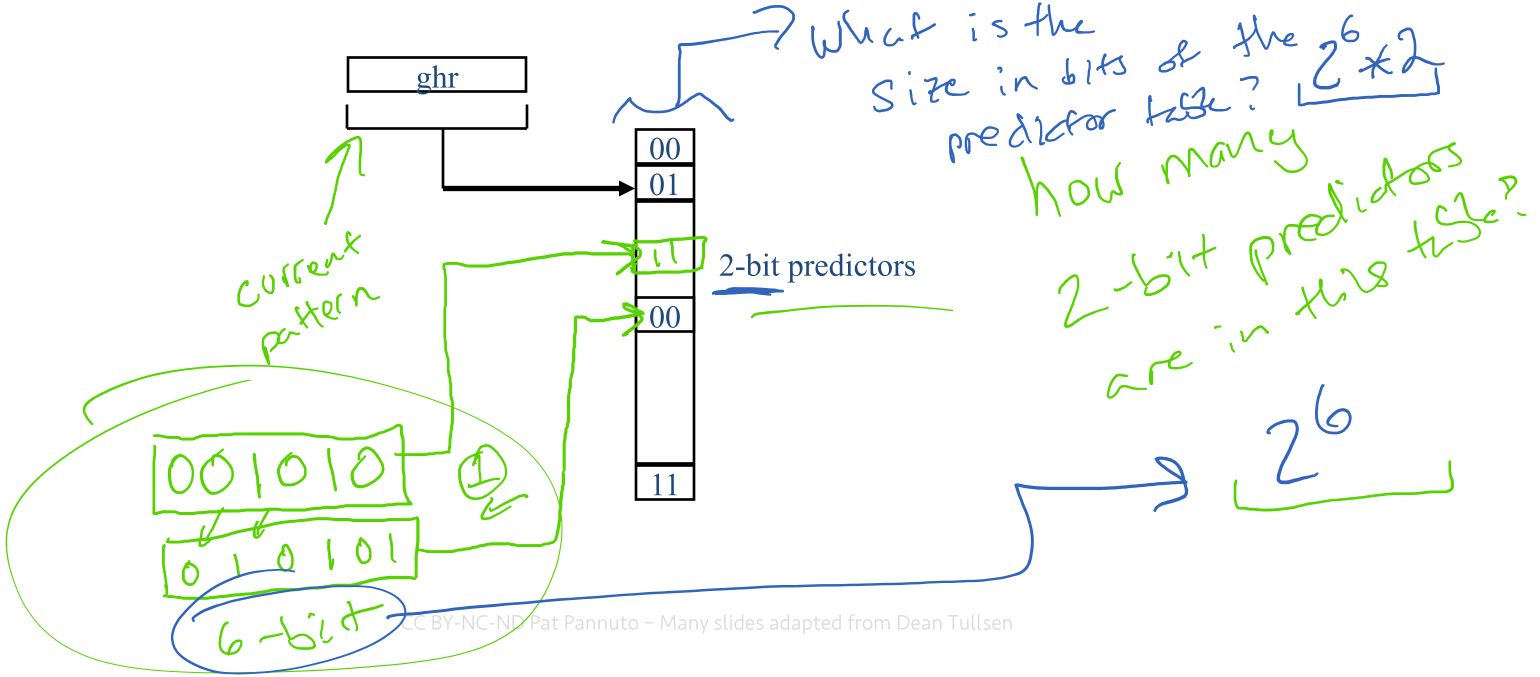
- *Correlating Branch Predictors* also look at other branches for clues

```
    if (i == 0)     NT
      ...
    if (i > 7)  NT  ?
      ...
```

- Typically use two indexes
  - Global history register --> history of last m branches (e.g., 0100011)
  - branch address

Correlating Branch Predictors

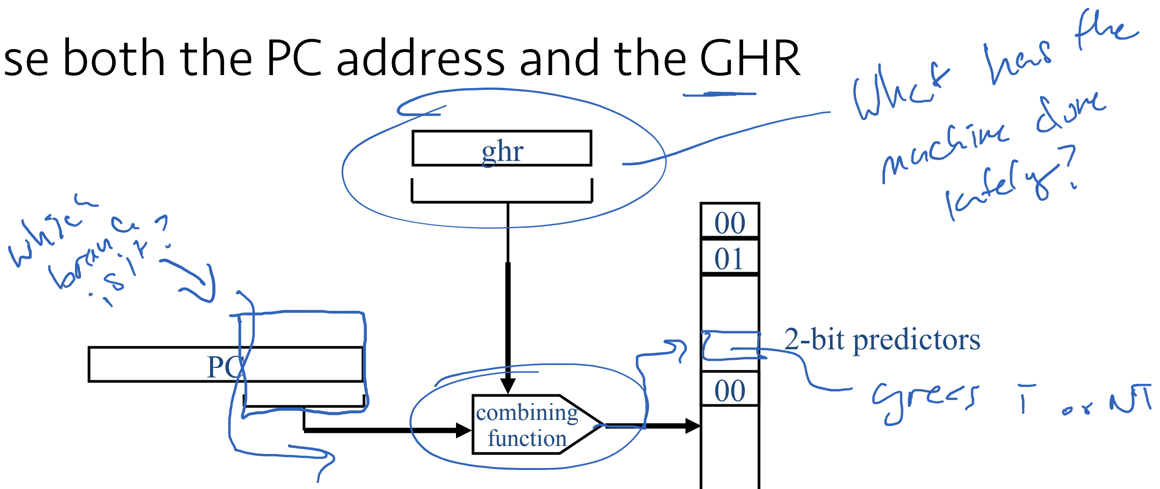
- The *global history register (ghr)* is a shift register that records the last *n* branches (of any address) encountered by the processor.


ghr
00
01
11
00
11
2-bit predictors
current pattern
001010
1
010101
6-bit
What is the size in bits of the predictor table?
$2^6 * 2$
how many 2-bit predictors are in this table?
2^6

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Two-level correlating branch predictors

- Can use both the PC address and the GHR

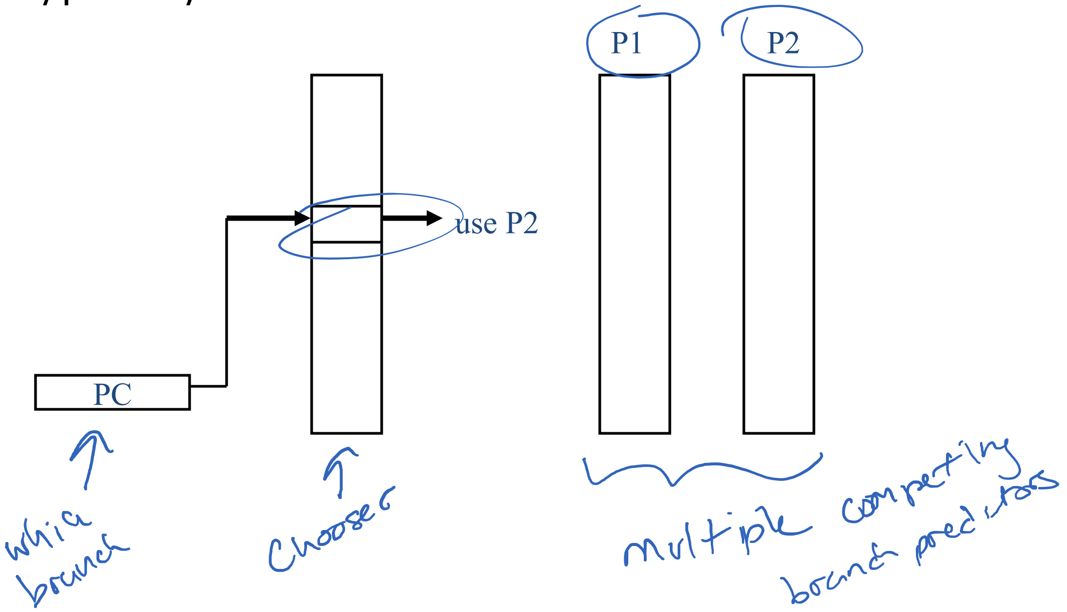


- Most common – *gshare*: use xor as the combining function.

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

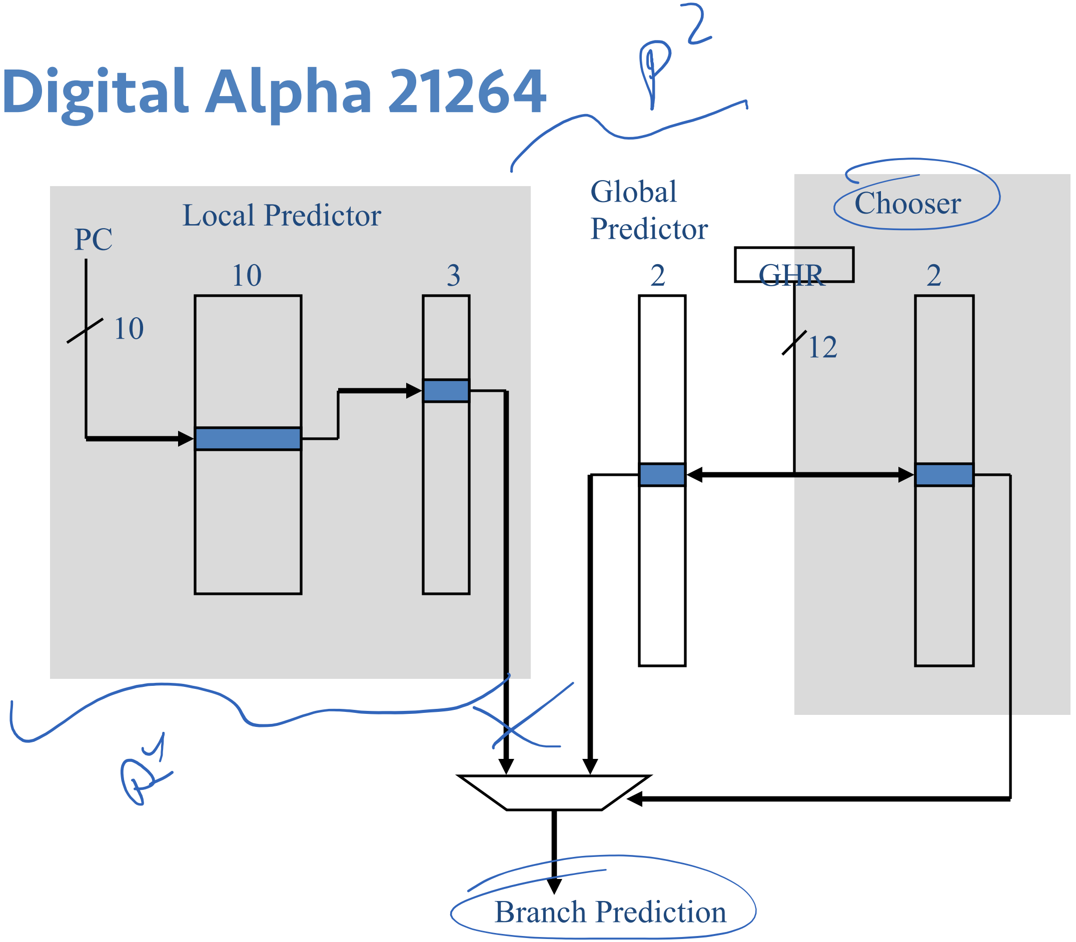
Are we happy yet????

- *Combining branch predictors* use multiple schemes and a voter to decide which one typically does better for *that* branch.



CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

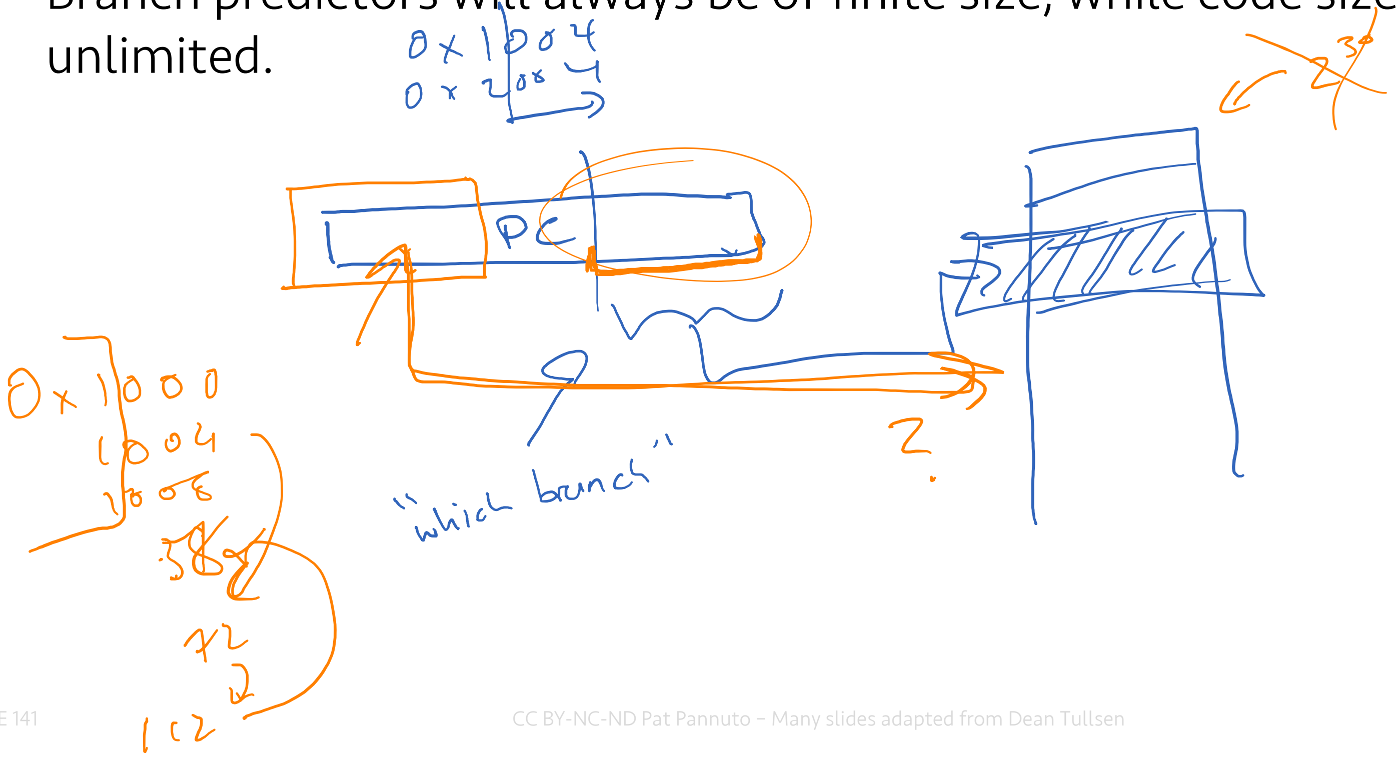
Compaq/Digital Alpha 21264



CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Aliasing in Branch Predictors

- Branch predictors will always be of finite size, while code size is relatively unlimited.



CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

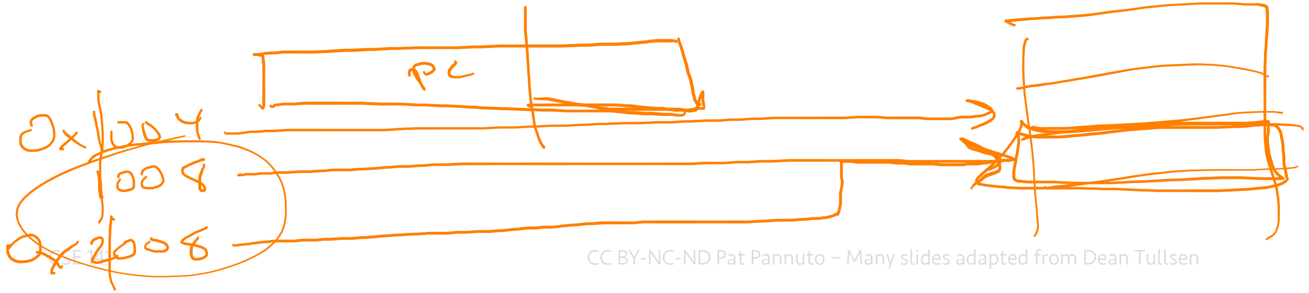
Aliasing in Branch Predictors

- Branch predictors will always be of finite size, while code size is relatively unlimited.
- What happens when (in the common case) there are more branches than entries in the branch predictor?

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

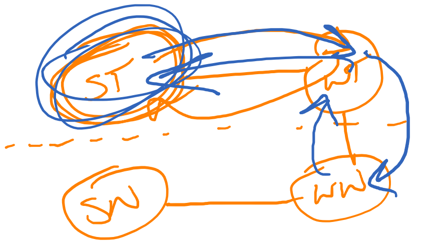
Aliasing in Branch Predictors

- Branch predictors will always be of finite size, while code size is relatively unlimited.
- What happens when (in the common case) there are more branches than entries in the branch predictor?
- We call these conflicts *aliasing*.
- We can have negative aliasing (when biases are different) or neutral aliasing (biases same). Positive aliasing is unlikely.



CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Bimodal aliasing



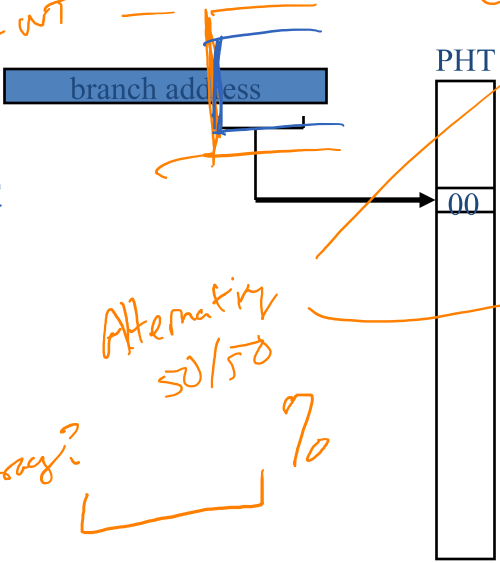
↑ predict T

↓ predict NT

History of the "004" branch?

$$\begin{array}{ccccccccccc} WT & ST & & & & & & & ST & WT & WN \\ T & T & T & T & T & T & T & T & T & N & N \\ X & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ & ✓ & X & X \end{array}$$

$$\frac{8}{11}$$



90% {

Branch at 0x1004

for i = ... 10

TTTTTTTTTNTT...

9 T's 1NT 5T's

Branch at 0x2004 ⇒ Never Taken

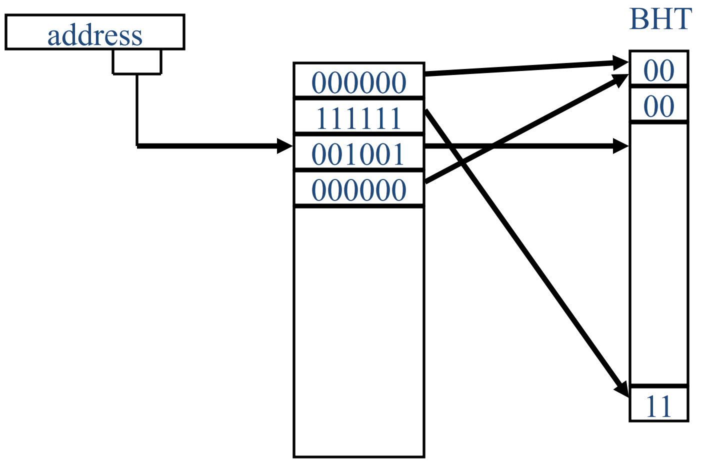
Alternating 50/50

Accuracy? ______ %

Alternates between the loop and the if

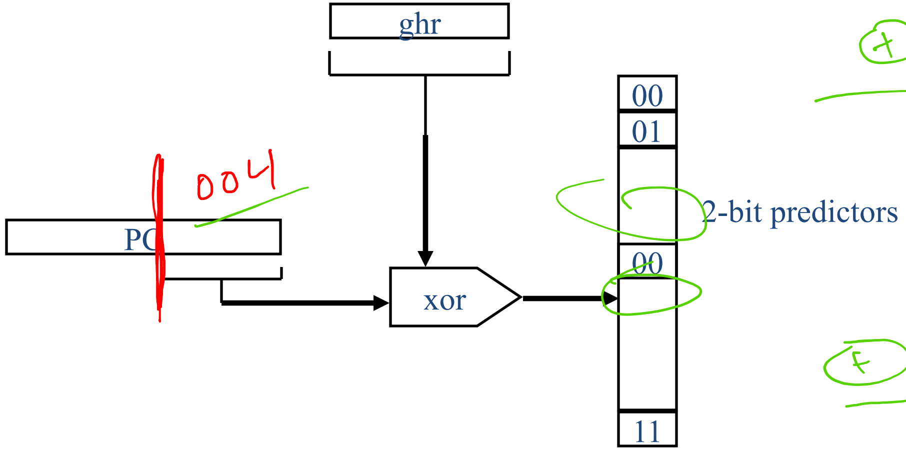
CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Local Predictor Aliasing



Gshare aliasing

T T T T T T T T T T N N

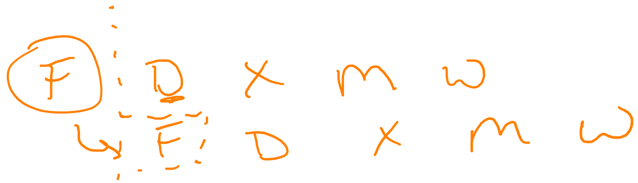


  1 1 1 1   0 0
⊕   1 0 0
―――――――――――――
  1 1 1 0 1 1

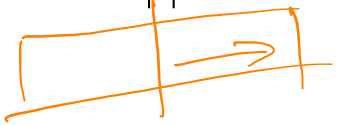
1 1 1 1 1 0
⊕   1 0 0
―――――――――――
1 1 1 0 1 1

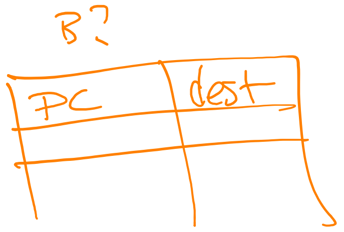
CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

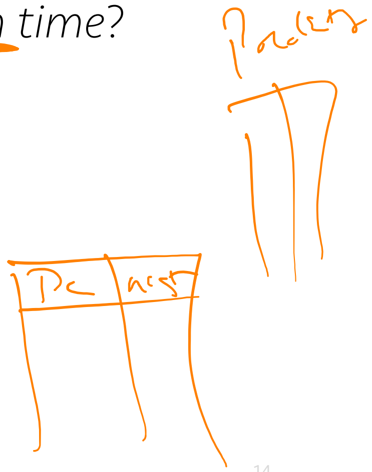
Branch Prediction



- Latest branch predictors significantly more sophisticated, using more advanced correlating techniques, larger structures, and soon possibly using AI techniques.
- Remember from earlier….
  - Presupposes what two pieces of information are available at *fetch time*?
    - Is it a branch?
    - Where is it going?
  - Branch Target Buffer supplies this information.







CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Pipeline performance
(And defining CSE141 “standard parameters”)

```
loop: lw $15, 1000($2)
      add $16, $15, $12
      lw $18, 1004($2)
      add $19, $18, $12
      beq $19, $0, loop
      nop
```

What is the steady-state CPI of this code?

Assume branch taken many times.
Assume 5-stage pipeline, forwarding, early branch resolution, branch delay slot

Always assume this architecture if not given the details

Can we improve this?

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Putting it all together.

For a given program on our 5-stage MIPS pipeline processor:

- 20% of insts are loads, 50% of instructions following a load are arithmetic instructions depending on the load
- 20% of instructions are branches.
- We manage to fill 80% of the branch delay slots with useful instructions.

- **What is the CPI of your program?**

	CPI
A	0.76
B	0.9
C	1.0
D	1.1
E	1.14

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Given our 5-stage MIPS pipeline…
What is the steady state CPI for the following code?

```
Loop:  lw r1, 0 (r2)
       add r2, r3, r4
       sub r5, r1, r2
       beq r5, $zero, Loop
       nop
```

Selection	CPI
A	1
B	1.25
C	1.5
D	1.75
E	None of the above

That was a lot.

- Seriously!
- Loosely, we just covered ~30 years of processor design in 4 weeks
  - (The good ideas are always more obvious in hindsight…)

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Pipelining Key Points

- ET = IC * CPI * CT
- Achieve high *throughput* without reducing instruction *latency*
- Pipelining exploits a special kind of parallelism (parallelism between functionality required in different cycles by different instructions).
- Pipelining uses combinational logic to generate (and registers to propagate) control signals.
- Pipelining creates potential hazards.

Data Hazard Key Points

- Pipelining provides high throughput, but does not handle data dependences easily.
- Data dependences cause *data hazards*.
- Data hazards can be solved by:
  - software (nops)
  - hardware stalling
  - hardware forwarding
- Our processor, and indeed all modern processors, use a combination of forwarding and stalling.
- ET = IC * CPI * CT

CC BY-NC-ND Pat Pannuto – Many slides adapted from Dean Tullsen

Control Hazard Key Points

- Control (branch) hazards arise because we must fetch the next instruction before we know:
  - if we are branching
  - where we are branching
- Control hazards are detected in hardware.
- We can reduce the impact of control hazards through:
  - early detection of branch address and condition
  - *branch prediction*
  - branch delay slots