ECE 3057: Architecture, Concurrency and Energy in Computation
Summer 2019

http://tusharkrishna.ece.gatech.edu/teaching/uca_f18/

Lecture 08: Speculation

David Anderson

School of Electrical and Computer Engineering
Georgia Institute of Technology

Acknowledgment: Lecture slides adapted from GT ECE 3056 (T. Krishna and S. Yalamanchilli), MIT 6.823 (Arvind and J. Emer)

Resolving Data/Control Hazards

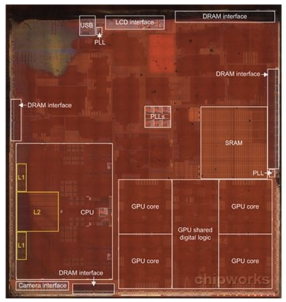
- Strategy 0: remove dependence
  - Detect and eliminate the dependence at the software level
- Strategy 1: stall
  - *Wait for the result to be available by freezing earlier pipeline stages*
- Strategy 2: bypass/forward
  - *Route data as soon as possible after it is calculated to the earlier pipeline stage*
- Strategy 3: speculate
  - *Guess the result and proceed*
    - ***Guessed correctly*** → *no special action required*
    - ***Guessed incorrectly*** → *kill and restart*
- Strategy 4: do something else
  - *Switch to some other task/thread while dependence resolves*

Speculation in Practice

Apple found in infringement of University of Wisconsin CPU patent, faces $862M in damages

By Mikey Campbell
Tuesday, October 13, 2015, 02:24 pm PT (05:24 pm ET)

A Wisconsin jury on Tuesday found Apple in infringement of computer processor patent owned by the University of Wisconsin Alumni Research Foundation, a ruling that could potentially lead to a damages payout of $862.4 million.



Inside look at Apple's A7. | Source: Chipworks

U.S. Patent No. 5,781,752 for a **"Table based data speculation circuit for parallel processing computer"**

This "table" is used by Apple A7 and A8 → iPhone 5S, iPad Air, iPad Mini, iPhone 6, iPhone 6 Plus …

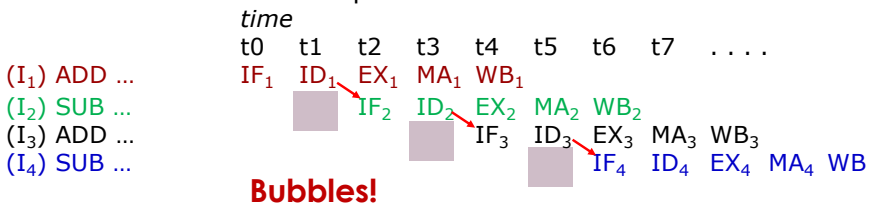
aka ***address speculation***

When can we speculate?

- Data Hazards
  - An instruction depends on the value produced by a previous instruction
  - Can we speculate?
    - Maybe – if we can guess the output value of an ALU or LW instruction
- Control Hazards
  - Next instruction's PC depends on the previous instruction
    - For which instructions is this true?
      - Branches and Jumps
  - Can we speculate?
    - Yes – this lecture

PC Calculation Bubbles

- In which stage do we know the opcode of the current instruction?
  - Decode
- Suppose we wait to do PC calculation till we know what the opcode is

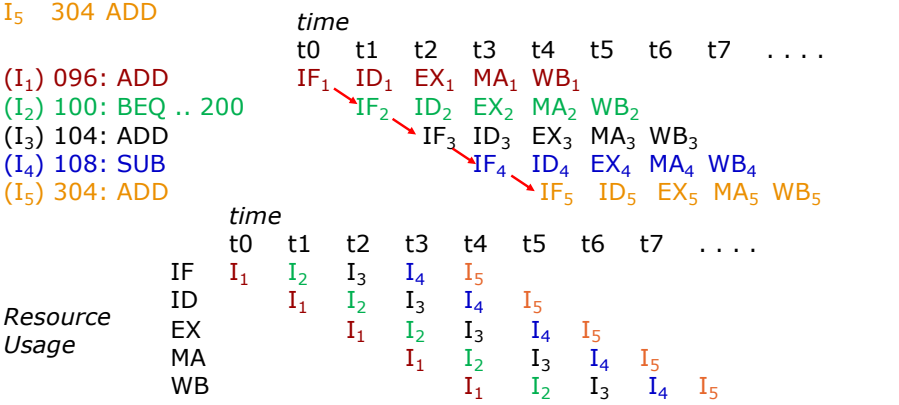


- What is a good guess for next PC?
  - PC+4

Example of correct speculation

I1 096 ADD
I2 100 BEQ .. 200
I3 104 ADD
I4 108 SUB
I5 304 ADD

Suppose Branch **Not Taken**



What if we mis-speculate?

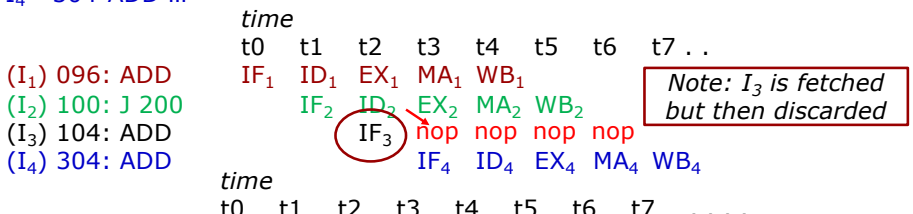
- When do we know we have mis-speculated?
  - Jump Instruction
    - At Decode stage
  - Branch Instruction
    - At Execute stage
- What do we do upon mis-speculation?
  - Start fetching instructions from the correct PC
  - What happens to the instructions we have already fetched?
    - Will be "flushed"
      - How?
        - Insert NOP in place of speculated instruction
        - Shows up as bubble in the pipeline

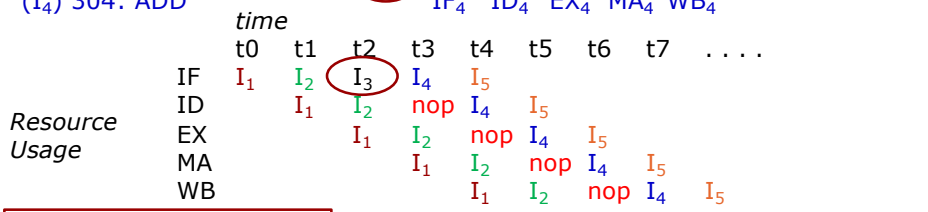
Example of Mis-speculation: Jump

I1 096 ADD ...
I2 100 J 200
I3 ~~104 ADD ...~~ *kill*
I4 304 ADD ...

Jump is always **Taken**

Jump Instruction: Resolved in Decode stage





One Pipeline Bubble (nop)

nop ⇒ *pipeline bubble*

Example of Mis-speculation: Branch

I_1 096 ADD
I_2 100 BEQ .. 200
I_3 ~~104 ADD~~ *kill*
I_4 ~~108 ...~~ *kill*
I_5 304 ADD

Suppose Branch ***Taken***

Branch Target: Resolved in Execute stage

	time t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	WB_1				
(I_2) 100: BEQ .. 200		IF_2	ID_2	EX_2	MA_2	WB_2			
(I_3) 104: ADD			IF_3	ID_3	nop	nop	nop		
(I_4) 108:				IF_4	nop	nop	nop	nop	
(I_5) 304: ADD					IF_5	ID_5	EX_5	MA_5	WB_5

Resource Usage

	time t0	t1	t2	t3	t4	t5	t6	t7	. . . .
IF	I_1	I_2	I_3	I_4	I_5				
ID		I_1	I_2	I_3	nop	I_5			
EX			I_1	I_2	nop	nop	I_5		
MA				I_1	I_2	nop	nop	I_5	
WB					I_1	I_2	nop	nop	I_5

nop ⇒ *pipeline bubble*

Two Pipeline Bubbles (nop)

Why are branches important?

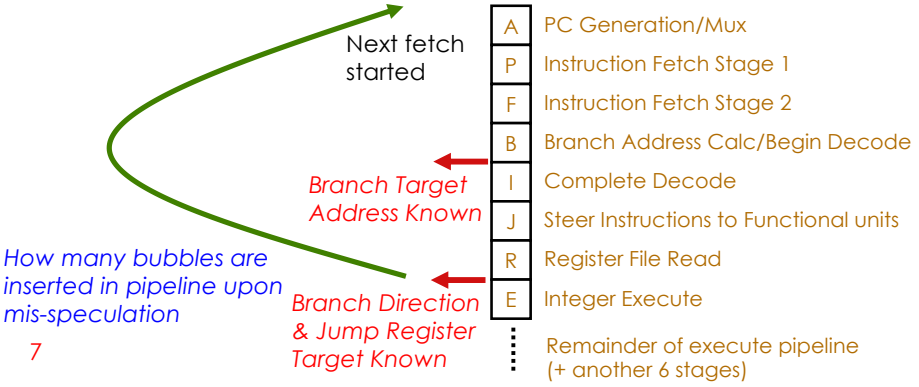
- Branches are very frequent
  - Approx. 20% of all instructions, i.e., 1 in every 5
  - Suppose 75% of them are Taken
    - CPI = 1 + 0.2*.75*2 = 1.3 → 30% slowdown
      - Worse penalty with deeper pipelines
- If we stall till we resolve the branch
  - Branch outcome is known after B cycles
    - B bubbles in the pipeline
      - B very large for long pipelines

Mis-prediction penalty

- 5-stage Pipeline
  - Jumps: 1-cycle
  - Branches: 2-cycle
- What about deeper pipelines?

Branch Mis-prediction Penalty in Modern Pipelines

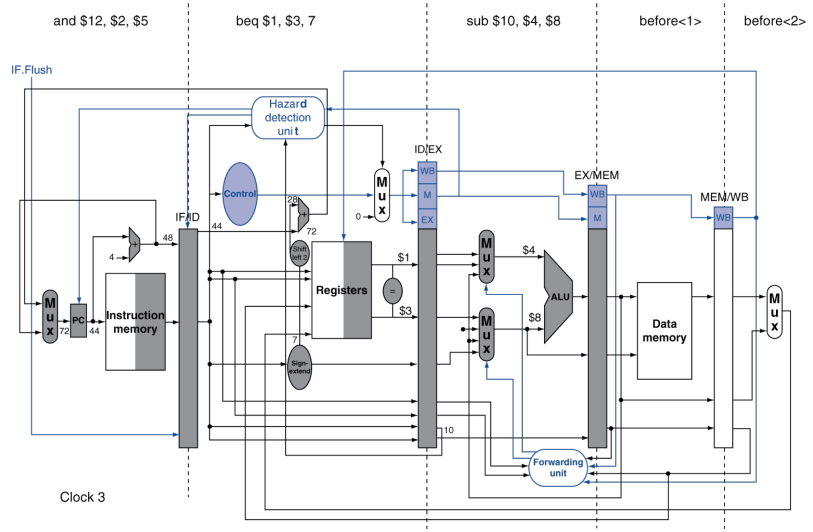
UltraSPARC-III instruction fetch pipeline stages
(in-order issue, 4-way superscalar, 750MHz, 2000)



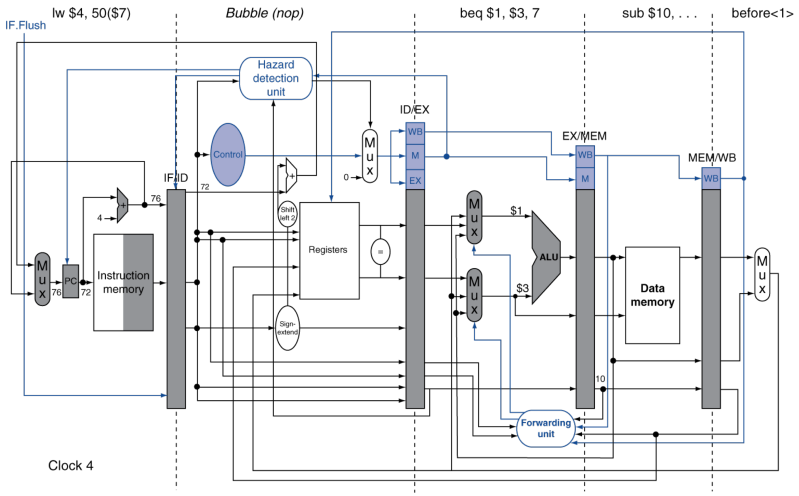
Reducing Branch Mis-prediction Penalty in 5-stage Pipeline

- Register Comparator
  - Add hardware to determine branch outcome in ID stage
    - Compare values of the two registers read
  - Now branches are identical to jumps → 1-cycle penalty
- Delay Slots (in MIPS)
  - Guarantee that the instruction after a Branch is always executed without being flushed
    - How?
      - Compiler adds a useful instruction or a NOP in the delay slot
    - Challenge?
      - Too specific to the microarchitecture

Example: Branch Taken



Example: Branch Taken



Improving Branch Prediction Accuracy

- What does nextPC = PC + 4 mean for branch instructions?
  - Predicting that the branch is Not Taken
- How true is that for real applications?
  - Consider the following piece of code

```
for (i=0; i < 100000; i++)
{
    C[i] = A[i] + B[i]
}
```

```
// Suppose $R0 = 0, and $R10 is the iteration value
0xDC08    LW
...
0xDC48    SUBI R10, R10, 1
0xDC4C    BNE R10, R0, -0x44   ← Branch Instruction
```

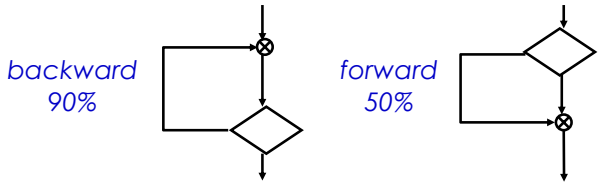
What will be the most common outcome of this branch? **Taken**

Branch Prediction

- Each instruction fetch depends on one or two pieces of information from the preceding instruction:
  - Is the preceding instruction a **taken branch**?
  - If so, what is the **target address**?

Static Branch Prediction

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



ISA can attach preferred direction semantics to branches, e.g., Motorola MC88110

bne0 *(preferred taken)* beq0 *(not taken)*

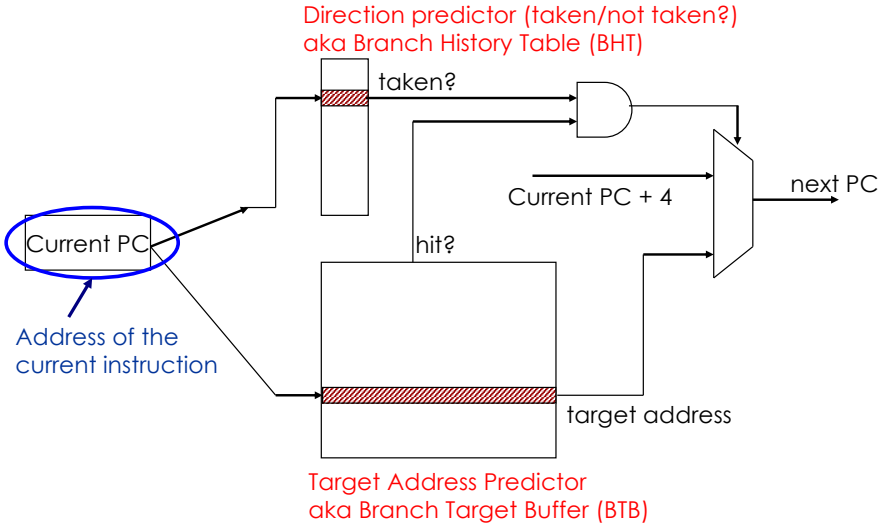
ISA can allow arbitrary choice of statically predicted direction, e.g., HP PA-RISC, Intel IA-64

typically reported as ~80% accurate

Dynamic Branch Prediction

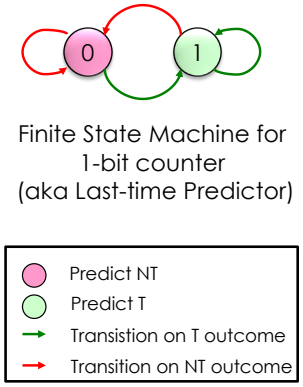
- ***Temporal correlation***
  - The way a branch resolves may be a good predictor of the way it will resolve at the next execution
    - *Example: loops*
- ***Spatial correlation***
  - Several branches may resolve in a highly correlated manner
    - *Example: a preferred path of execution*

Branch Predictor



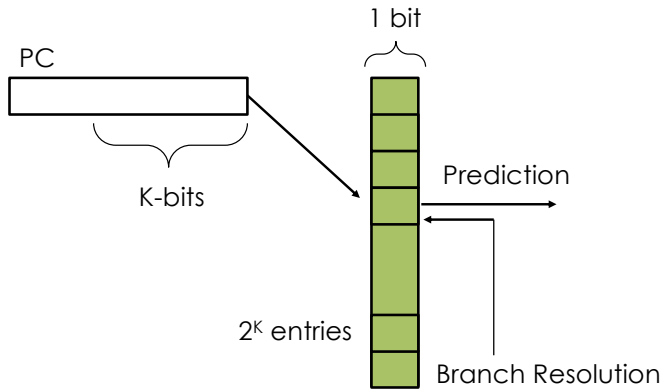
One-Bit Direction Predictor

For each branch, remember its last outcome (T/NT)



Finite State Machine for
1-bit counter
(aka Last-time Predictor)

One-Bit Predictor Implementation



How well does this perform on loop branches?

One-bit predictor performance

Suppose the following loop is called multiple times

```
0xDC08:     for (i=0; i < 100000; i++)
            {

            }
```

Misprediction Rate

2/100000 = 0.002 %

i	0	1	2	…	99998	99999	0	1	…	99998	99999	0	1	…	99998	99999	0	1
Predict	N	T	T	…	T	T	N	T	…	T	T	N	T	…	T	T	N	T
Actual	T	T	T	…	T	N	T	T	…	T	N	T	T	…	T	N	T	T

How well does this perform on loop branches?

Mispredicts **twice** for every use of loop.

One-bit predictor performance

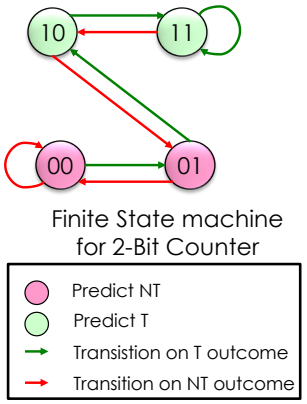
Misprediction Rate

```
0xDC08:     for (i=0; i < 100000; i++)                    2/100000 = 0.002 %
            {
0xDC44:         if ((i % 100) == 0)
                    CountHundred( );                      2000/100000 = 2 %

0xDC50:         if ((i & 1) == 1)
                    CountOddNums( );                      100000/100000 = 100 %

            }
```

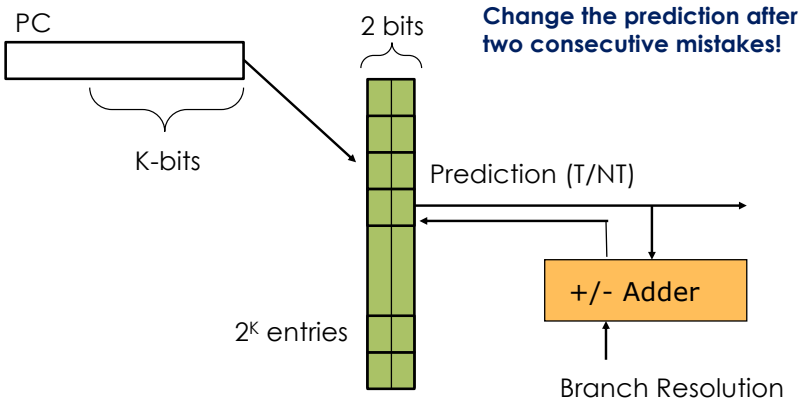
What happens on loop branches?

Mispredicts **twice** for every use of loop.

Two-bit Direction Predictor



Two-bit Predictor Implementation



Two-bit Predictor Performance

Misprediction Rate

```
0xDC08:   for (i=0; i < 100000; i++)
          {


          }
```

1/100000 = 0.001 %

i	0	1	2	…	99998	99999	0	1	…	99998	99999	0	1	…	99998	99999	0	1
State	00	01	10	11	11	11	10	11	11	11	11	10	11	11	11	11	10	11
Predict	N	N	T	…	T	T	T	T	…	T	T	T	T	…	T	T	T	T
Actual	T	T	T	…	T	N	T	T	…	T	N	T	T	…	T	N	T	T

What happens on loop branches?

Mispredicts **one** for every use of loop.

Two-bit Predictor Performance

Misprediction Rate

```
0xDC08:   for (i=0; i < 100000; i++)
          {
0xDC44:       if ((i % 100) == 0)
                 CountHundred( );

0xDC50:       if ((i & 1) == 1)
                 CountOddNums( );
          }
```

1/100000 = 0.001 %

1000/100000 = 1 %

?

0xDC50 Predictor Starting State:

00 → 00 → 01 → 00 → 01 → 00 … 50000/100000 = 50 %

01 → 00 → 01 → 00 → 01 … 50000/100000 = 50 %

10 → 01 → 10 → 01 → 10 … 100000/100000 = 100 %

11 → 10 → 11 → 10 → 11 … 50000/100000 = 50 %

Advanced Predictors

- Leverage Correlation

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

If first condition false, second condition also false

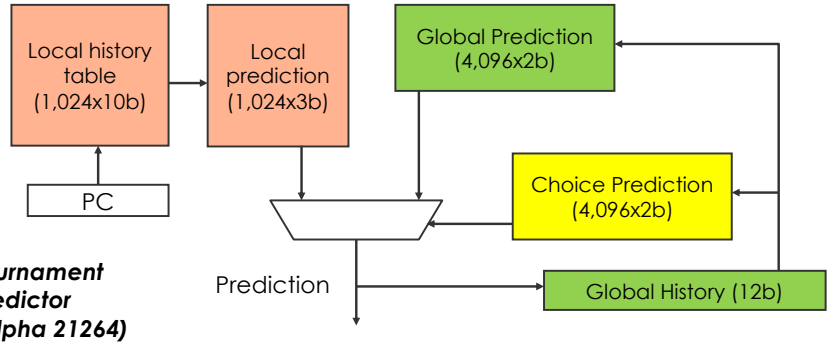
```
if (aa == 2) then
   aa = 0;
if (bb == 2) then
   bb = 0;
if (aa != bb) {

}
```

Outcome of third branch depends on previous two branches

- Learn Optimal Prediction
  - Perceptron-based Neural Network

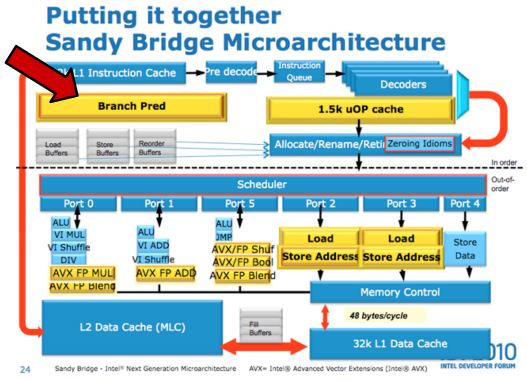
Branch Predictors in Modern Processors

- No predictor is clearly the best
  - Design a predictor to predict which predictor will predict better! ☺

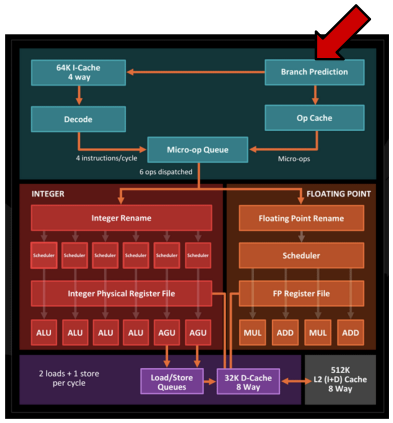


Tournament Predictor (Alpha 21264)

Branch Predictors in Modern Processors



Intel Sandy Bridge



AMD Ryzen