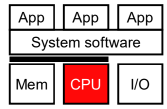
CSE 560
Computer Systems Architecture

Branch Prediction

1

1

This Unit: (Scalar In-Order) Pipelining



- Principles of pipelining
  - Effects of overhead and hazards
  - Pipeline diagrams
- Data hazards
  - Stalling and bypassing
- **Control hazards**
  - Branch prediction
  - Predication (later)

2

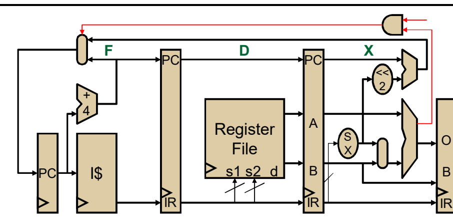
2

Control Dependences and Branch Prediction

3

3

What About Branches?



Control hazards options

- Could just stall to wait for branch outcome (two-cycle penalty)
- **Fetch past branch insns before branch outcome is known**
  - Default: assume "**not-taken**" (at fetch, can't tell it's a branch)

4

4

Big Idea: Speculative Execution

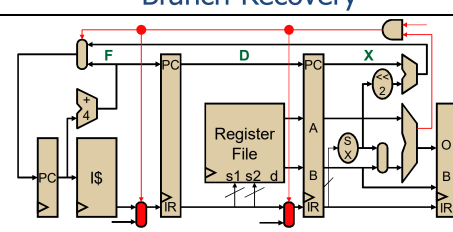
- Speculation: "risky transactions on chance of profit"
- **Speculative execution**
  - Execute before all parameters known with certainty
  - **Correct speculation**
    - + Avoid stall, improve performance
  - **Incorrect speculation (mis-speculation)**
    - – Must abort/flush/squash incorrect insns
    - – Must undo incorrect changes (recover pre-speculation state)

  the game: **[$\%_{correct}$ x gain] – [(1–$\%_{correct}$) x penalty]**
- **Control speculation**: speculation aimed at control hazards
  - *Are these the correct instructions to execute next?*

5

5

Branch Recovery



Branch recovery: what to do when branch is actually taken

- Insns that will be written into F/D and D/X are wrong
- **Flush them**, *i.e.*, replace them with `nops`

+ They haven't changed permanent state yet (regfile, DMem)

– 2-cycle penalty for taken branches

6

6

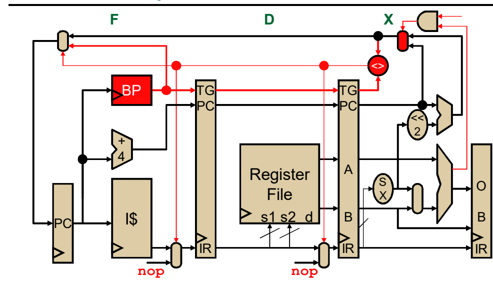
Branch Performance

- Back of the envelope calculation
  - **Branch: 20%**, load: 20%, store: 10%, other: 50%
  - Say, **75% of branches are taken**
- CPI = 1 + 20% x 75% x 2 =
  1 + **0.20** x **0.75** x 2 = 1.3
  - **Branches cause 30% slowdown**
    - Even worse with deeper pipelines

How do we reduce slowdown?

1. Reduce misprediction penalty (resolve branches sooner?)
2. Reduce misprediction frequency

Fewer Mispredictions: Branch Prediction

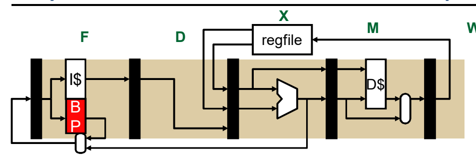


Dynamic branch prediction: hardware guesses outcome
- Start fetching from guessed address
- Flush on **mis-prediction**

Branch Prediction Performance

- Parameters
  - **Branch: 20%**, load: 20%, store: 10%, other: 50%
  - 75% of branches are taken
- Dynamic branch prediction
  - Branches predicted with 95% accuracy
  - Was:
    - CPI = 1 + 20% x **75%** x 2 = **1.3**
  - Now:
    - CPI = 1 + 20% x **5%** x 2 = **1.02**

Dynamic Branch Prediction Components



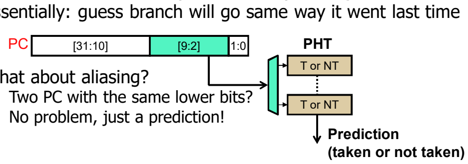
Step #1: is it a branch?
- Easy after decode...

Step #2: is the branch taken or not taken?
- **Direction predictor** (conditional branches only)
- Predicts taken/not-taken

Step #3: if the branch is taken, where does it go?
- Easy after decode…

Branch Direction Prediction

- **Learn from past, predict the future**
  - Record the past in a hardware structure
- **Direction predictor (DIRP)**
  - Map conditional-branch PC to taken/not-taken (T/N) decision
  - Individual conditional branches often biased or weakly biased
    - 90%+ one way or the other considered **"biased"**
    - Why? Loop back edges, checking for uncommon conditions
- **Pattern history table (PHT)**: simplest predictor
  - PC indexes table of bits (0 = N, 1 = T), no tags
  - Essentially: guess branch will go same way it went last time



- What about aliasing?
  - Two PC with the same lower bits?
  - No problem, just a prediction!

Pattern History Table (PHT)

Pattern history table (PHT): simplest direction predictor
- PC indexes table of bits (0 = N, 1 = T), no tags
- Essentially: branch will go same way it went last time
- Problem: consider **inner loop branch** below
  (* = mis-prediction)

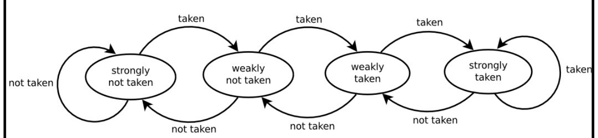
```
for (i=0;i<100;i++)
   for (j=0;j<3;j++)
      // whatever
```

State/prediction	N*	T	T	T*	N*	T	T	T*	N*	T	T	T*
Outcome	T	T	T	N	T	T	T	N	T	T	T	N

- Two "built-in" mis-predictions per inner loop iteration
- Branch predictor "changes its mind too quickly"

Two-Bit Saturating Counters (2bc)

Two-bit saturating counters (2bc) [Smith]
- Replace each single-bit prediction
  - (0,1,2,3) = (N,n,t,T)



- By Branch_prediction_2bit_saturating_counter.gif: Afogderivative work: ENORMATOR (talk) - Branch_prediction_2bit_saturating_counter.gif, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15955952

15

15

Two-Bit Saturating Counters (2bc)

Two-bit saturating counters (2bc) [Smith]
- Replace each single-bit prediction
  - (0,1,2,3) = (N,n,t,T)
- Adds "hysteresis"
  - Force predictor to mis-predict twice before "changing its mind"

State/prediction	**N***	**n***	**t**	**T***	**t**	**T**	**T**	**T***	**t**	**T**	**T**	**T***
Outcome	T	T	T	N	T	T	T	N	T	T	T	N

- One mispredict each loop execution (rather than two)
  + Fixes this pathology (not contrived, by the way)
  - Can we do even better?

16

16

Correlated Predictor

Correlated (two-level) predictor [Patt]
- Exploits observation that branch outcomes are correlated
- Maintains separate prediction per (PC, BHR)
  - **Branch history register (BHR)**: recent branch outcomes
- Simple working example: assume program has one branch
  - 2-bit history register (4 possible entries)

State/prediction	BHR=NN	**N***	T	T	T	T	T	T	T	T	T	T	T
"active pattern"	BHR=NT	N	**N***	T	T	T	T	T	T	T	T	T	T
	BHR=TN	N	N	N	N	**N***	T	T	T	T	T	T	T
	BHR=TT	N	N	**N***	**T***	N	N	**N***	**T***	N	N	**N***	**T***
Outcome	N N	T	T	T	N	T	T	T	N	T	T	T	N

– We didn't make anything better, what's the problem?

17

17

Correlated Predictor

- What happened?
  - BHR wasn't long enough to capture the pattern
  - Try again: 3-bit history register (8 possible entries)

State/prediction	BHR=NNN	**N***	T	T	T	T	T	T	T	T	T	T	T
	BHR=NNT	N	**N***	T	T	T	T	T	T	T	T	T	T
	BHR=NTN	N	N	N	N	N	N	N	N	N	N	N	N
"active pattern"	BHR=NTT	N	N	**N***	T	T	T	**T**	T	T	T	**T**	T
	BHR=TNN	N	N	N	N	N	N	N	N	N	N	N	N
	BHR=TNT	N	N	N	N	N	**N***	T	T	T	**T**	T	T
	BHR=TTN	N	N	N	N	**N***	T	T	T	**T**	T	T	T
	BHR=TTT	N	N	N	**N**	N	N	N	**N**	N	N	N	N
Outcome	N N N	T	T	T	N	T	T	T	N	T	T	T	N

+ No mis-predictions after predictor learns all relevant patterns

18

18

Correlated Predictor

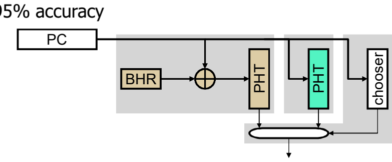
- Design choice I: one **global** BHR or one per PC (**local**)?
  - Each one captures different kinds of patterns
  - Global is better, captures local patterns for tight loop branches
- Design choice II: how many history bits (BHR size)?
  - Tricky one
  + Given unlimited resources, longer BHRs are better, but…
  – PHT utilization decreases
    – Many history patterns are never seen
    – Many branches are history independent (don't care)
    - PC xor BHR allows multiple PCs to dynamically share PHT
    - BHR length < $\log_2$(PHT size)
  – Predictor takes longer to train
  - Typical length: 8–12

19

19

Hybrid Predictor

- **Hybrid (tournament) predictor** [McFarling]
  - Attacks correlated predictor PHT capacity problem
  - Idea: combine two predictors
    - **Simple PHT** predicts history independent branches
    - **Correlated predictor** predicts only branches that need history
    - **Chooser** assigns branches to one predictor or the other
    - Branches start in simple PHT, move mis-prediction threshold
  + Correlated predictor can be **smaller**, handles fewer branches
  + 90–95% accuracy



20

20

When to Perform Branch Prediction?

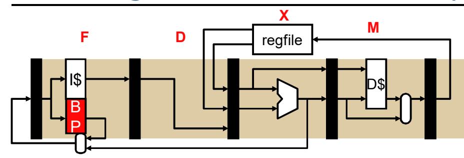
- During Decode
  - Look at insn opcode to determine branch instructions
  - Calculate next PC from insn (for PC-relative branches)
  – One cycle "mis-fetch" penalty **even if branch predictor is correct**

	1	2	3	4	5	6	7	8	9
`bnez r3,targ`	F	D	X	M	W				
`targ:add r4,r5,r4`			F	D	X	M	W		

- During Fetch?
  - How do we do that?

21

Revisiting Branch Prediction Components



Step #1: is it a branch?
- Easy after decode... during fetch: **predictor**

Step #2: is the branch taken or not taken?
- **Direction predictor** (as before)

Step #3: if the branch is taken, where does it go?
- **Branch target predictor (BTB)**
- Supplies target PC if branch is taken

22

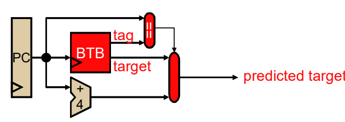
Branch Target Buffer (BTB)

- Record the past branch targets in a hardware structure
- **Branch target buffer (BTB)**:
  - "guess" the future PC based on past behavior
  - "Last time the branch X was taken, it went to address Y"
    "So, the next time address X is fetched, fetch address Y next"
- Operation
  - Like a cache: address = PC, data = target-PC
  - Access at Fetch *in parallel* with instruction memory
    - predicted-target = BTB[PC]
  - Updated at X whenever target != predicted-target
    - BTB[PC] = target
  - Aliasing? No problem; this is only a prediction.

23

Branch Target Buffer (continued)

- At Fetch, how does insn know it's a branch & should read BTB?
- Doesn't have to…all insns access BTB in parallel w/ I$ Fetch
- Key idea: **use BTB to predict which insn are branches**
  - Implement by "tagging" each entry with its corresponding PC
  - Update BTB on every taken branch insn, record target PC:
    - BTB[PC].tag = PC, BTB[PC].target = target of branch
  - All insns access at Fetch *in parallel* with I$
    - Check for tag match, signifies insn at that PC is a branch
    - Predicted PC = (BTB[PC].tag == PC) ? BTB[PC].target : PC+4



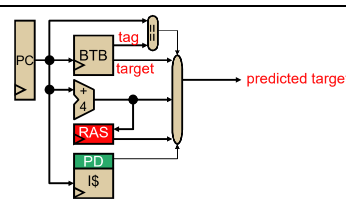
24

Why Does a BTB Work?

- Because most control insns use **direct targets**
  - Target encoded in insn itself → same "taken" target every time
- What about **indirect targets**?
  - Target held in a register → can be different each time
  - Indirect conditional jumps are not widely supported
  - Two indirect call idioms
    + Dynamically linked functions (DLLs): target always the same
    - Dynamically dispatched (virtual) functions: hard but uncommon
  - Also two indirect unconditional jump idioms
    - Switches: hard but uncommon
    – Function returns: hard and common but…

25

Return Address Stack (RAS)

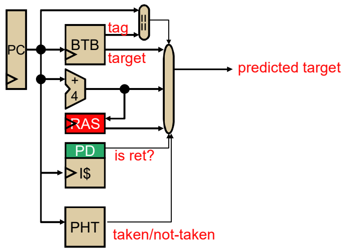


Return address stack (RAS)
- Call instruction? RAS[TOS++] = PC+4
- Return instruction? Predicted-target = RAS[--TOS]
- Q: how can you tell if an insn is a call/return before decoding it?
  - Accessing RAS on every insn BTB-style doesn't work
- Answer: **pre-decode bits** in I$, written when first executed
  - Can also be used to signify branches

26

Putting It All Together

- BTB & branch direction predictor during fetch



- If branch prediction correct → no taken branch penalty

A word about terminology

- Pattern History Table (PHT)
  - Sometimes called Branch History Table (BHT)
- Branch History Registers (BHR)
  - In book called "table of history registers (BHT)"

    ☹

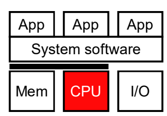
- I'm trying to avoid BHT everywhere now…
  - Please use context to help guide you

Branch Prediction Performance

- Dynamic branch prediction
  - 20% of instruction branches
  - Simple predictor: branches predicted with 75% accuracy
    - CPI = 1 + (20% x **25%** x 2) = **1.1**
  - More advanced predictor: 95% accuracy
    - CPI = 1 + (20% x **5%** x 2) = **1.02**

- Branch mis-predictions still a big problem though
  - Pipelines are long: typical penalty is 10+ cycles
  - Pipelines are superscalar (later)

Summary



- Principles of pipelining
  - Effects of overhead and hazards
  - Pipeline diagrams
- Data hazards
  - Stalling and bypassing
- Control hazards
  - Branch prediction
  - Predication (later)