Digital Systems Architecture

EECE 343-01 EECE 292-02

Branch Prediction Reloaded

Dr. William H. Robinson
February 13, 2004

http://eecs.vanderbilt.edu/courses/eece343/

Topics

Why, oh why, didn't I take the ***blue*** pill?

Cypher
The Matrix

- Administrative stuff
  - Turn in Project 1
  - Discussion
- Branch prediction
  - Correlated prediction

Project 1 Description

- Write a MIPS assembly program to execute the following matrix multiplication in floating point double precision:

$$\mathbf{C} = \mathbf{AB}$$

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

- Use this program to evaluate performance on the WinMIPS64 simulator with various architectural parameters

---

Class Discussion

- Thoughts on the WinMIPS64 Simulator?
- Thoughts on the assembly programming?
- Thoughts on the design space exploration?
- What insight did you gain? (i.e. what did you learn?)

The Matrix

```
; matrix multiplication
; For c11
        MUL.D   F30, F0, F10
        ADD.D   F20, F20, F30
        MUL.D   F30, F1, F13
        ADD.D   F20, F20, F30
        MUL.D   F30, F2, F16
        ADD.D   F20, F20, F30
; For c12
        MUL.D   F30, F0, F11
        ADD.D   F21, F21, F30
        MUL.D   F30, F1, F14
        ADD.D   F21, F21, F30
        MUL.D   F30, F2, F17
        ADD.D   F21, F21, F30
; For c13
        MUL.D   F30, F0, F12
        ADD.D   F22, F22, F30
        MUL.D   F30, F1, F15
        ADD.D   F22, F22, F30
        MUL.D   F30, F2, F18
        ADD.D   F22, F22, F30
```

FP Add	**3**
FP Multiply	**7**
Forwarding	**Disabled**
CPI	**3.367**

FP Add	**3**
FP Multiply	**7**
Forwarding	**Enabled**
CPI	**2.064**

The Matrix Reloaded

```
; For c11, c12, and c13
        MUL.D   F20, F0, F10
        MUL.D   F21, F1, F13
        MUL.D   F23, F0, F11
        MUL.D   F24, F1, F14
        MUL.D   F26, F0, F12
        MUL.D   F27, F1, F15
        MUL.D   F22, F2, F16
        MUL.D   F25, F2, F17
        MUL.D   F28, F2, F18

        ADD.D   F20, F20, F21
        ADD.D   F23, F23, F24
        ADD.D   F26, F26, F27

        DADD R2, R0, R0             ; initialize R2 for storing elements

        ADD.D   F20, F20, F22
        ADD.D   F23, F23, F25
        ADD.D   F26, F26, F28
```

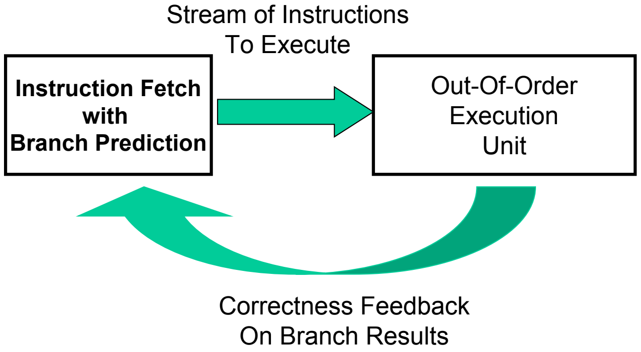
FP Add	**3**
FP Multiply	**7**
Forwarding	**Disabled**
CPI	**1.604**

FP Add	**3**
FP Multiply	**7**
Forwarding	**Enabled**
CPI	**1.242**

The Matrix Revolutions

- Using a loop for matrix calculation
  - Cost of loop overhead
  - Accessing array rows and colums
- Architectural parameters
  - Delay slot
  - Load delay
- If you used a loop, how does your CPI compare?

Problem: “Fetch” Unit



Stream of Instructions
To Execute

Instruction Fetch with Branch Prediction

Out-Of-Order Execution Unit

Correctness Feedback
On Branch Results

- Instruction fetch decoupled from execution
- Often issue logic (plus rename) included with Fetch

Adapted from John Kubiatowicz’s CS 252 lecture notes. Copyright © 2003 UCB.

Branches Must Be Resolved Quickly For Loop Overlap!

- In our loop-unrolling example, we relied on the fact that branches were under control of "fast" integer unit in order to get overlap!

```
Loop:   LD        F0   0    R1
        MULTD     F4   F0   F2
        SD        F4   0    R1
        SUBI      R1   R1   #8
        BNEZ      R1   Loop
```

- What happens if branch depends on result of multd??
  - We completely lose all of our advantages!
  - Need to be able to "predict" branch outcome.
  - If we were to predict that branch was taken, this would be right most of the time.
- Problem much worse for superscalar machines!

Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Prediction: Branches, Dependencies, Data

- Prediction has become essential to getting good performance from scalar instruction streams
- We will discuss predicting branches. However, architects are now predicting everything: *data dependencies, actual data,* and *results of groups of instructions*:
  - At what point does computation become a probabilistic operation plus verification?
  - We are pretty close with control hazards already…
- Why does prediction work?
  - Underlying algorithm has regularities
  - Data that is being operated on has regularities
  - Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems
- Prediction ⇒ Compressible information streams?

Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Static Branch Prediction

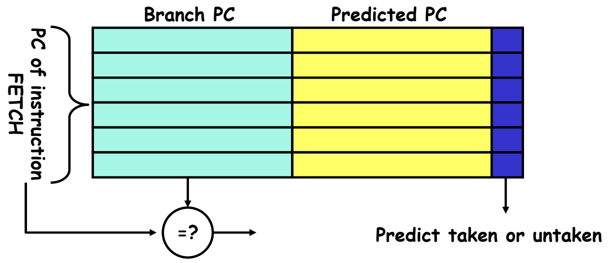
- Compiler uses branch instructions that specify the branch's normal outcome
  - Based upon the code to be compiled
  - Taken (**T**) or Not Taken (**NT**)
- Program is simulated to profile the branch behavior
  - Branch statistics are given to the compiler

Dynamic Branch Prediction

- **Branch History Table (BHT)**
  - CPU maintains a history of branch locations and their previous behavior
- **Branch Target Buffer (BTB)**
  - Stores the target location for branches predicted taken
- **Use 1 bit to store history**
  - Stores what the branch did the last time
  - With loops, you will always mispredict entering and exiting
- **Use 2 bits to store history**
  - Maintains a strong and weak prediction state
  - Must mispredict twice to change the prediction

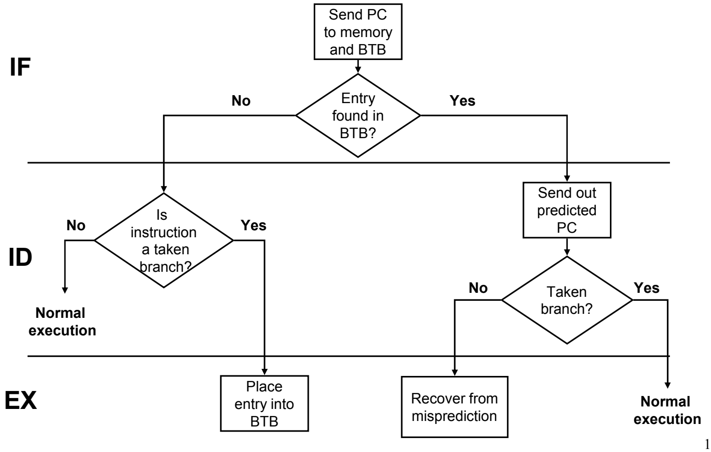
Need Address at Same Time as Prediction

- **Branch Target Buffer (BTB):** Address of branch index to get prediction AND branch address (if taken)
  - Note: must check for branch match now, since can't use wrong branch address (Figure 3.19, p. 210)

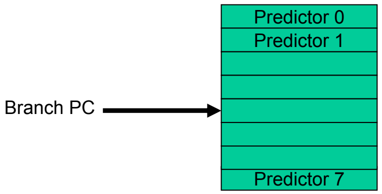


Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Flow Chart for BTB (Fig 3.20 p. 212)



Branch History Table



- BHT is a table of "Predictors"
  - Usually 2-bit, saturating counters
  - Indexed by PC address of Branch – without tags
- In Fetch state of branch:
  - BTB identifies branch
  - Predictor from BHT used to make prediction
- When branch completes
  - Update corresponding Predictor

Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Dynamic Branch Prediction (Standard Technologies)

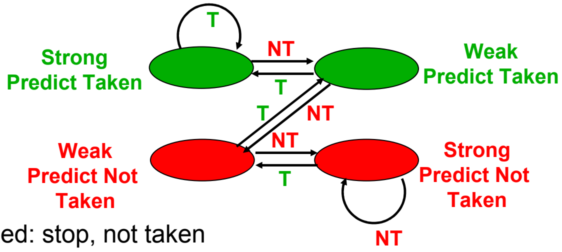
- **Combine Branch Target Buffer and History Tables**
  - Branch Target Buffer (BTB): identify branches and hold *taken addresses*
    - *Trick: identify branch before fetching instruction!*
    - *Must be careful not to misidentify branches or destinations*
  - Branch History Table makes prediction
    - Can be complex prediction mechanisms with long history
    - No address check: Can be good, can be bad (aliasing)

Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Dynamic Branch Prediction

(Jim Smith, 1981)

- Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 3.7, p. 198)



- Red: stop, not taken
- Green: go, taken
- Adds *hysteresis* to decision making process

BHT Accuracy

- Mispredict because either:
  - Wrong guess for that branch
  - Got branch history of wrong branch when index the table
- 4096 entry table programs vary from 1% misprediction (nasa7, tomcatv) to 18% (eqntott), with spice at 9% and gcc at 12%
- 4096 about as good as infinite table (in Alpha 211164)

Correlating Branches

- **Hypothesis:**
  - Recent branches are correlated
  - Behavior of recently executed branches affects prediction of current branch
- **Two possibilities; current branch depends on:**
  - Last *m* most recently executed branches anywhere in program Produces a "GA" (for "global adaptive") in the Yeh and Patt classification (e.g. GAg)
  - Last *m* most recent outcomes of same branch. Produces a "PA" (for "per-address adaptive") in same classification (e.g. PAg)
  - Little "g" means global pattern history table

Correlating Branches

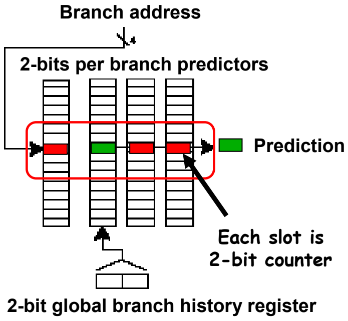
- **Idea:** record *m* most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table (BHT) entry
  - A single history table shared by all branches (appends a "g" at end), indexed by history value.
  - Address is used along with history to select table entry (appends a "p" at end of classification)
  - If only portion of address used, often appends an "s" to indicate "set-indexed" tables (i.e. GAs)

Correlating Branches

- For instance, consider global history, set-indexed BHT. That gives us a GAs history table.

(2,2) GAs predictor

- First 2 means that we keep two bits of history
- Second means that we have 2 bit counters in each slot.
- Then behavior of recent branches selects between, say, four predictions of next branch, updating just that prediction
- Note that the original two-bit counter solution would be a (0,2) GAs predictor
- Note also that aliasing is possible here...


Branch address
2-bits per branch predictors
Prediction
Each slot is 2-bit counter
2-bit global branch history register

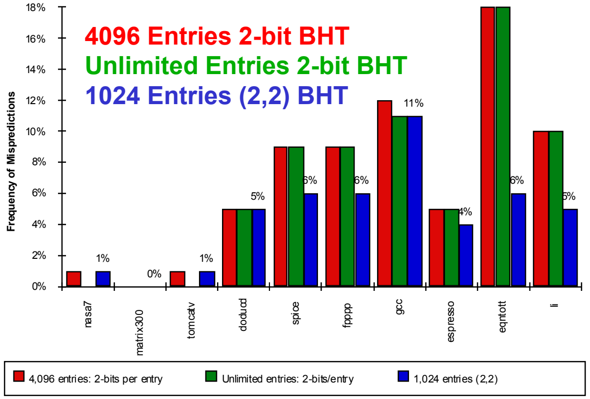
Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Calculating Number of State Bits

- **General case (*m*,*n*) predictor**
  - Uses behavior of last *m* branches
  - Selects among 2^m branch predictors
  - Each predictor is *n* bits

- **Number of bits in (*m*,*n*) predictor:**
  - $2^m \times n \times$ (Number of prediction entries selected by branch address)

Accuracy of Different Schemes



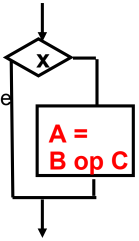
4096 Entries 2-bit BHT
Unlimited Entries 2-bit BHT
1024 Entries (2,2) BHT

Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Predicated Execution

- Avoid branch prediction by turning branches into conditionally executed instructions:

  if (x) then A = B op C else NOP
  - If false, then neither store result nor cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instruction
  - IA-64: 64 1-bit condition fields selected so conditional execution of any instruction
  - This transformation is called "if-conversion"



- Drawbacks to conditional instructions
  - Still takes a clock even if "annulled"
  - Stall if condition evaluated late
  - Complex conditions reduce effectiveness; condition becomes known late in pipeline

Adapted from John Kubiatowicz's CS 252 lecture notes. Copyright © 2003 UCB.

Summary

- Prediction becoming important part of scalar execution.
  - Prediction is exploiting “information compressibility” in execution
- Branch History Table: 2 bits for loop accuracy
- Correlation: Recently executed branches correlated with next branch.
  - Either different branches (GA)
  - Or different executions of same branches (PA).
- Branch Target Buffer: include branch address & prediction
- Predicated Execution can reduce number of branches, number of mispredicted branches

Things to Do

- **Study for Exam 1**
  - Covers material up to today’s lecture (branch prediction)
  - See sample problems on web site
- **Homework Assignment #2 due Monday**
  - Solutions posted after class on Monday