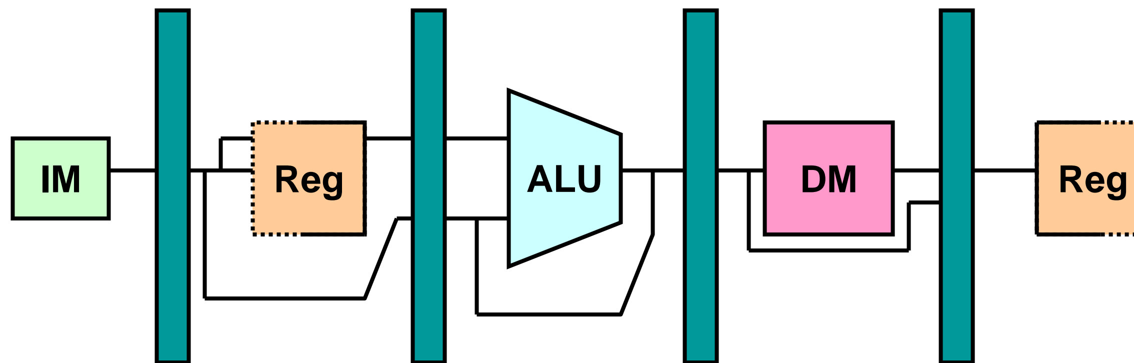
Branch Prediction

Introduction

Pipelined design:

Prevalent in today's processor implementations



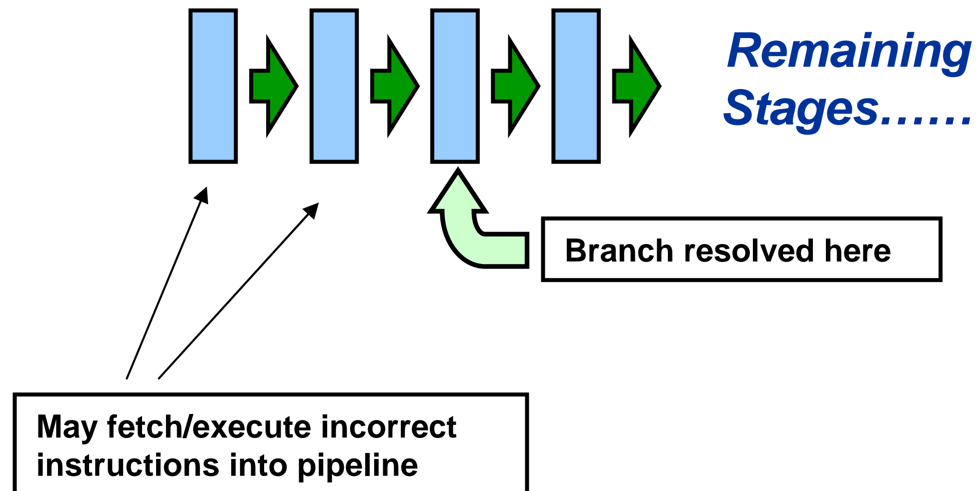
Overlaps the execution of consecutive instructions

Results in more efficient utilization of hardware

Introduction (cont.)

However:

Pipelined structure causes *delay of change*,
when branch instructions *reconfigures* the pipeline operation



Wrong instructions in pipeline need to be flushed (thrown away)

Introduction (cont.)

Ways to deal with branch delay

Let it be:

(1) Stall until branch resolved

(2) Or, expose branch delay as part of the architecture

(read: be sloppy and let the compiler do the work ☺)

Simple Policy:

Always proceed as if branches are taken or not-taken
E.g. predict taken for all backward branches (loops), etc.

We can also attempt to *predict* the outcome of branches:

- **Dynamic prediction:**
  processor does branch prediction on-the-fly
- **Static prediction:**
  compiler predicts the behavior of branch operations in the program; can possibly do some optimizations for better performance

Branch Prediction: *Dynamic and Static*

Dynamic Branch Prediction

Processors predict the outcome of branches dynamically, during program runtime

Static Branch Prediction

Compiler statically predicts the properties of branches in the program

Branch Prediction

Dynamic Branch Prediction

The *processor* computes a Predict function dynamically, during runtime:

Predict: *BranchInstruction* → *Boolean* [local predictor]

A true/false value indicates predicted taken/not-taken outcome of the branch

Provides an estimate for the processor fetch stages: what to do next upon a branch?

Allows more efficient pipeline operation

Exact branch resolving may be deep in pipeline, branch prediction can make more of the *speculative* fetch behavior be *useful work*

Branch Prediction

Dynamic Branch Prediction

Can be grouped by what predictions are based on:

Local Predictors: ***BranchInstruction* $\rightarrow$ *Boolean***

Path Predictors: ***PastBranchHistory* $\rightarrow$ *Boolean***

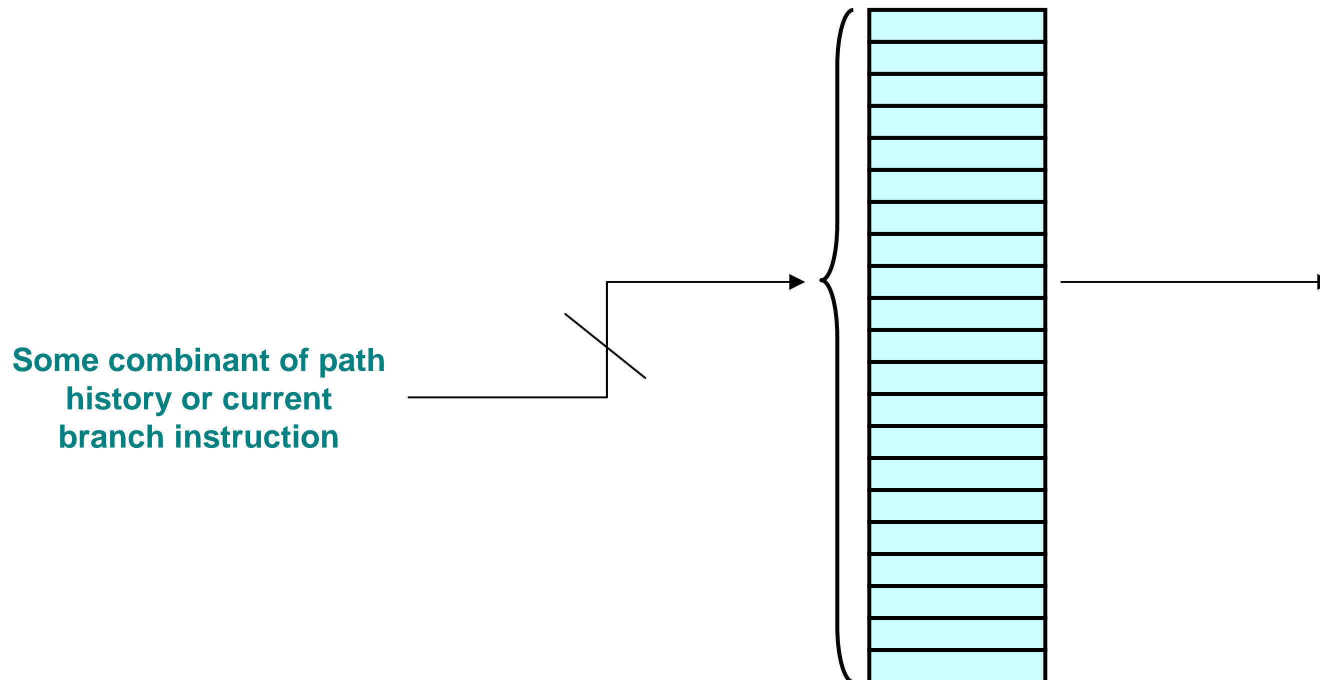
Two-level Predictors: ***BranchInstruction* $\times$ *PastBranchHistory* $\rightarrow$ *Boolean***

Branch Prediction

Dynamic Branch Prediction

Pattern History Tables (PHT)

Dominant approach for dynamic branch prediction



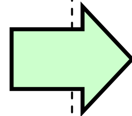
Branch Prediction

Dynamic Branch Prediction

General Taxonomy

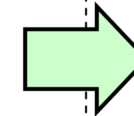
Current Branch/Past branch history

Combinations of:
(1) Current Branch Instruction Address
(2) A number of past taken/not-taken history bits

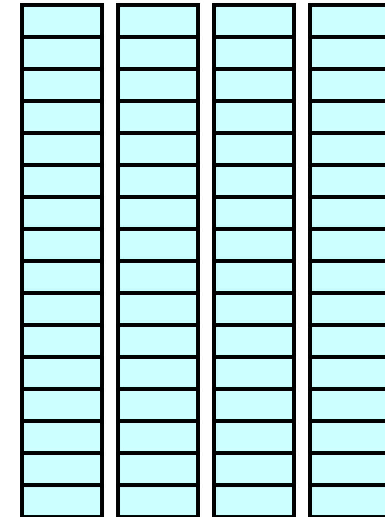


Hash function

Some well known hashes:
(1) Concatenation
(2) XOR (the "gshare" predictor)



PHT Lookup



Branch Prediction

Dynamic Branch Prediction

Today's State-of-the-art processors

(1) Higher clock frequencies through *deeper pipelines*
(2) Multiple issue:
Increases the performance hit for every lost cycle

Such designs calls for even MORE accurate branch predictors

However, the issues are not so simple……

Branch Prediction

Dynamic Branch Prediction

Deep pipelines and multiple issue require better branch prediction:

Large prediction resources:
More hardware budget going to PHTs

Single predictors are not enough!!

Combined Tournament predictors
(multiple prediction schemes working against each other)

Branch Prediction

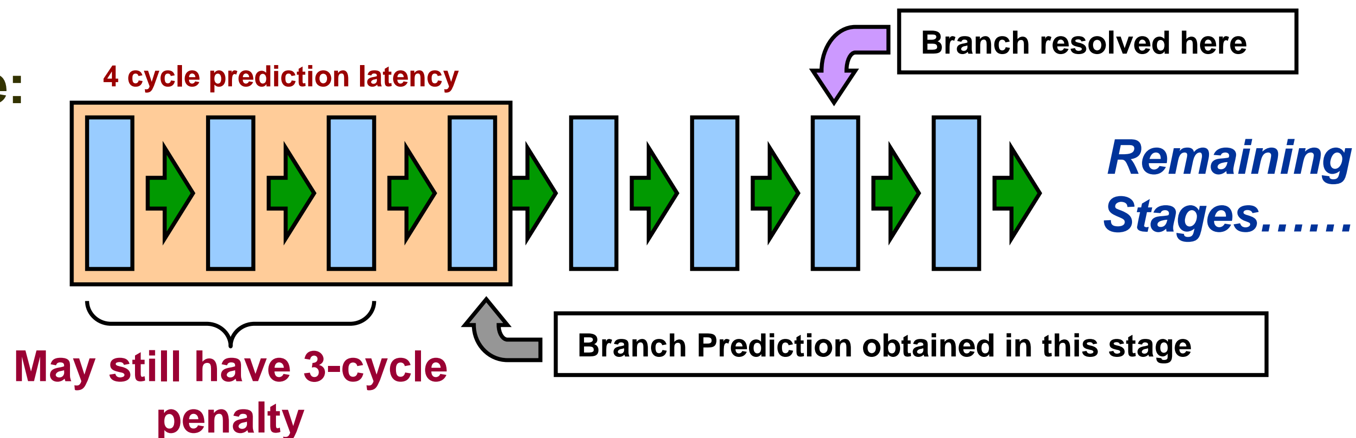
Dynamic Branch Prediction

More hardware budget going to PHTs (larger area)

→ Longer latencies for them to work

Branch Predictors get more accurate, but predicts SLOWER
Eventually, they start needing multiple cycles
Meaning: correct prediction may still incur a branch penalty
(though smaller than mis-prediction)

Example:



Recent study reveals: ***Such trends performing slower overall***
Sacrificing speed for precision does not pay off here……

Refer to: ***"Reconsidering Complex Branch Predictors"*** **Daniel A. Jiménez. HPCA-9 (2003)**

Branch Prediction

Dynamic Branch Prediction

A Relatively Recent Approach:
Neural Branch Prediction

Neural Methods for Dynamic Branch Prediction.
Daniel A. Jiménez and Calvin Lin.
ACM Transactions on Computer Systems, Vol. 20, No. 4, November 2002

Fast Path-Based Neural Branch Prediction.
Daniel A. Jiménez. MICRO-36 (2003)

Branch Prediction

Static Branch Prediction

The compiler at work:

Branch prediction analysis done by the compiler at static compile time

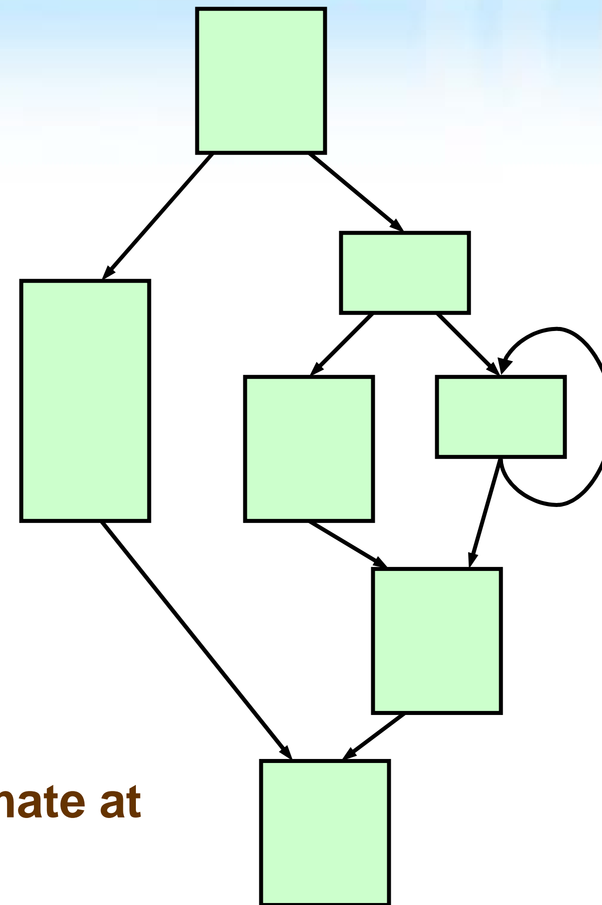
※ Note that *dynamically translating JIT compilers* are exceptions, since logically they form a part of the [virtual] machine, and is actually dynamic by nature.

Two main issues:

(1) How do we obtain the static branch prediction information?
(2) Where can we apply such information?
(read: what compiler transformations can we do?)

Branch Prediction

Static Branch Prediction



Given a **Control Flow Graph (CFG)** of a program:

The ***directed graph edges*** represent control flow transfer among basic blocks

Static Branch Prediction attempts to estimate at compile time,
probabilistic information of each branch

Static Branch Prediction

Branch Analysis: *obtaining* the information

Known approaches:

- **Profile based**

  The use of *profiling* to obtain empirical program behavior statistics, including branch probability information
  (Not really analysis "by" the compiler)

- **Heuristics based**

  Through profile runs, we can derive *patterns of program behavior*
  (e.g. loop branches are usually taken, etc.)
  Such patterns can be formulated into compiler heuristics

- **Machine learning**

  Instead of manually forming heuristics,
  we can *generate* static branch predictors using training data obtained by profile runs, through the use of *Machine Learning*
  This can provide a basis for analysis tests *way better* than human engineered heuristics

Static Branch Prediction

Profile based approach

Available data from profile runs (related to control flow):

- **Branch probability**
  - How often the branch is taken/not-taken
- **Block frequency**
  - How much time the program spends in each basic block
- **Path/Trace statistics**
  - How control flow behaves over the entire program

Can obtain quite accurate information

But requires an additional "profile run" during program development

Static Branch Prediction

Heuristics based prediction

Profiling can be cumbersome to use

How can we predict branches without going through a whole profile?

Observe from given profile data, general heuristics for statically predicting branches

Paper on this topic:

Branch Prediction for Free. Thomas Ball, James R. Larus. PLDI 1993

Static Branch Prediction

Prediction by Machine Learning

Next step after heuristics based prediction

We have the profile data, instead trying to engineer heuristics by human observation, leave it to machine learning

Much more scalable to not easily observed rules, more detailed attributes of prediction, etc.

Papers on this topic:

Corpus-Based Static Branch Prediction
Brad Calder, Dirk Grunwald, Donald Lindsay, James Martin, Michael Mozer, Ben Zorn
ACM SIGPLAN Notices, Volume 30, Issue 6 (June 1995)

Evidence-Based Static Branch Prediction Using Machine Learning
Brad Calder, Dirk Grunwald, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, Ben Zorn
ACM Transactions on Programming Languages and Systems (TOPLAS) Volume 19, Issue 1 (Jan. 1997)

Static Branch Prediction

Transformations: *using* the information

- Transformations with Speculative effects:
  - Global instruction scheduling
- Structural region forming
  - Superblock/Hyperblock formation
  - Region-based compilation
- Code layout
  - Basic block reordering
  - Software trace cache
- Branch hinting
  - Some instruction sets have branch hinting capability usable by the compiler (e.g. Itanium)
- etc.

Particularly important in processors with less dynamic features (e.g. Itanium, some embedded RISCs, VLIW, DSP, etc.)

References

Effective Compiler Support for Predicated Execution Using the Hyperblock
Scott A. Mahlke, David C. Lin, William Y. Chen, Richard E. Hank, Roger A. Bringmann, MICRO25 (1992)

The Superblock: An Effective Technique for VLIW and Superscalar Compilation
Wen-mei W. Hwu, Scott A. Mahlke, William Y. Chen, Pohua P. Chang, Nancy J. Warter, Roger A. Bringmann, Roland G. Ouellette, Richard E. Hank, Tokuzo Kiyohara, Grant E. Haab, John G. Holm, and Daniel M. Lavery
The Journal of Supercomputing, Kluwer Academic Publishers, 1993, pp. 229-248

Profile Guided Code Positioning Karl Pettis, Robert C. Hansen. PLDI 1990

Global Instruction Scheduling for Superscalar Machines. David Bernstein, Michael Rodeh. PLDI 1991

Region-Based Compilation. Richard E. Hank.
PhD thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana IL, May 1996

Region-Based Compilation: An Introduction and Motivation.
Richard E. Hank, Wen-mei W. Hwu, B. Ramakrishna Rau. MICRO28 (1995)

Software Trace Cache. Alex Ramírez, Josep-L. Larriba-Pey, Carlos Navarro, Josep Torrellas, Mateo Valero.
International Conference on Supercomputing 1999

Neural Methods for Dynamic Branch Prediction.
Daniel A. Jiménez and Calvin Lin. ACM Transactions on Computer Systems, Vol. 20, No. 4, November 2002

Fast Path-Based Neural Branch Prediction. Daniel A. Jiménez. MICRO-36 (2003)

Reconsidering Complex Branch Predictors. Daniel A. Jiménez. HPCA-9 (2003)

Combining Branch Predictors. Scott McFarling.
Compaq Western Research Lab Technical Note TN-36, June 1993