EECS 470

RS/ROB examples
True Physical Registers?
Project

Lecture 8 – Winter 2024

Slides developed in part by Profs. Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, and Wenisch of Carnegie Mellon University, Purdue University, University of Michigan, University of Pennsylvania, and University of Wisconsin.

Today

- RS/ROB
  - A bit more detail
- True physical registers: Removing the ARF
  - How and why
  - Probably will only get started on this, we’ll see.
- Project discussion
  - Help me to stop the above at 1pm.

Reminder

- In the original Tomasulo’s algorithm you should only update the ARF if you overwrite the data in the RAT.

- In the P6 scheme you always write to the ARF.

Tomasulo Data Structures

(Timing Free Example, “P6 scheme”)

CDB	
T	V

Map Table	
Reg	Tag
r0	
r1	
r2	
r3	
r4	

Reservation Stations (RS)								
T	FU	busy	op	RoB	T1	T2	V1	V2
1								
2								
3								
4								
5								

ARF	
Reg	V
r0	
r1	
r2	
r3	
r4	

Instruction
r0=r1*r2
r1=r2*r3
Branch if r1=0
r0=r1+r1
r2=r2+1

Reorder Buffer (RoB)							
RoB Number	0	1	2	3	4	5	6
Dest. Reg.							
Value							

Review Questions

- Could we make this work without the RS?
  - If so, why do we do that?
- Why is it important to retire in order?
- Why must branches wait until retirement before they announce their mispredict?
  - Any other ways to do this?

More review questions

1. What is the purpose of the RoB?
2. Why do we have both a RoB and a RS?
   - Yes, that was pretty much on the last page…
3. Misprediction
   a) When to we resolve a mis-prediction?
   b) What happens to the main structures (RS, RoB, ARF, Rename Table) when we mispredict?
4. What is the whole purpose of OoO execution?

And yet more review questions!

1. What is the purpose of the RoB?
2. Why do we have both a RoB and a RS?
3. Misprediction
   a) When to we resolve a mis-prediction?
   b) What happens to the main structures (RS, RoB, ARF, Rename Table) when we mispredict?
4. What is the whole purpose of OoO execution?

When an instruction is *dispatched* how does it impact each major structure?

- Rename table?
- ARF?
- RoB?
- RS?

When an instruction *completes execution* how does it impact each major structure?

- Rename table?

- ARF?

- RoB?

- RS?

When an instruction *retires* how does it impact each major structure?

- Rename table?
- ARF?
- RoB?
- RS?

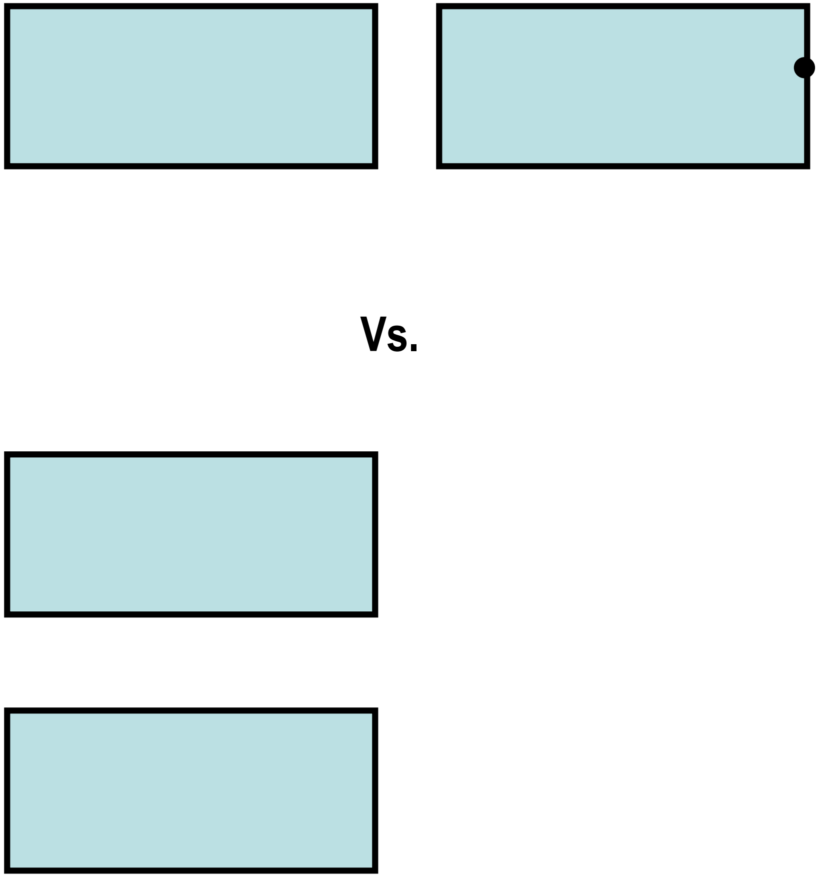
Topic change

- Why on earth are we doing this?
  - Why do we think it helps?
- Homework 2 problems 5 and 6 made the argument.
  - Only need to obey true data dependencies.
    - Huge speedup ***potential***.

Optimizing CPU Performance

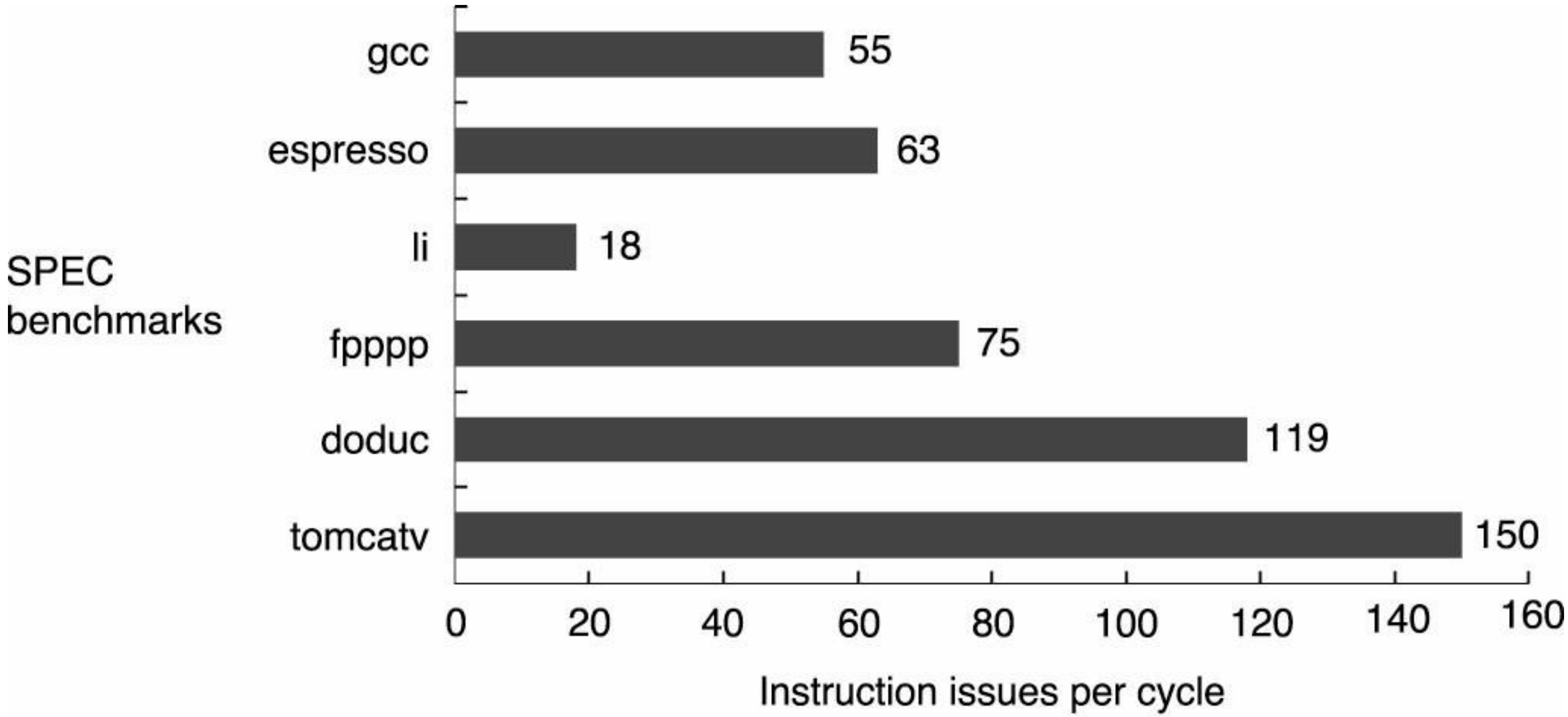
- Golden Rule: $t_{CPU} = N_{inst}*CPI*t_{CLK}$
- Given this, what are our options
  - Reduce the number of instructions executed
  - Reduce the cycles to execute an instruction
  - Reduce the clock period
- Our first focus: Reducing CPI
  - Approach: *Instruction Level Parallelism* (ILP)

Why ILP?

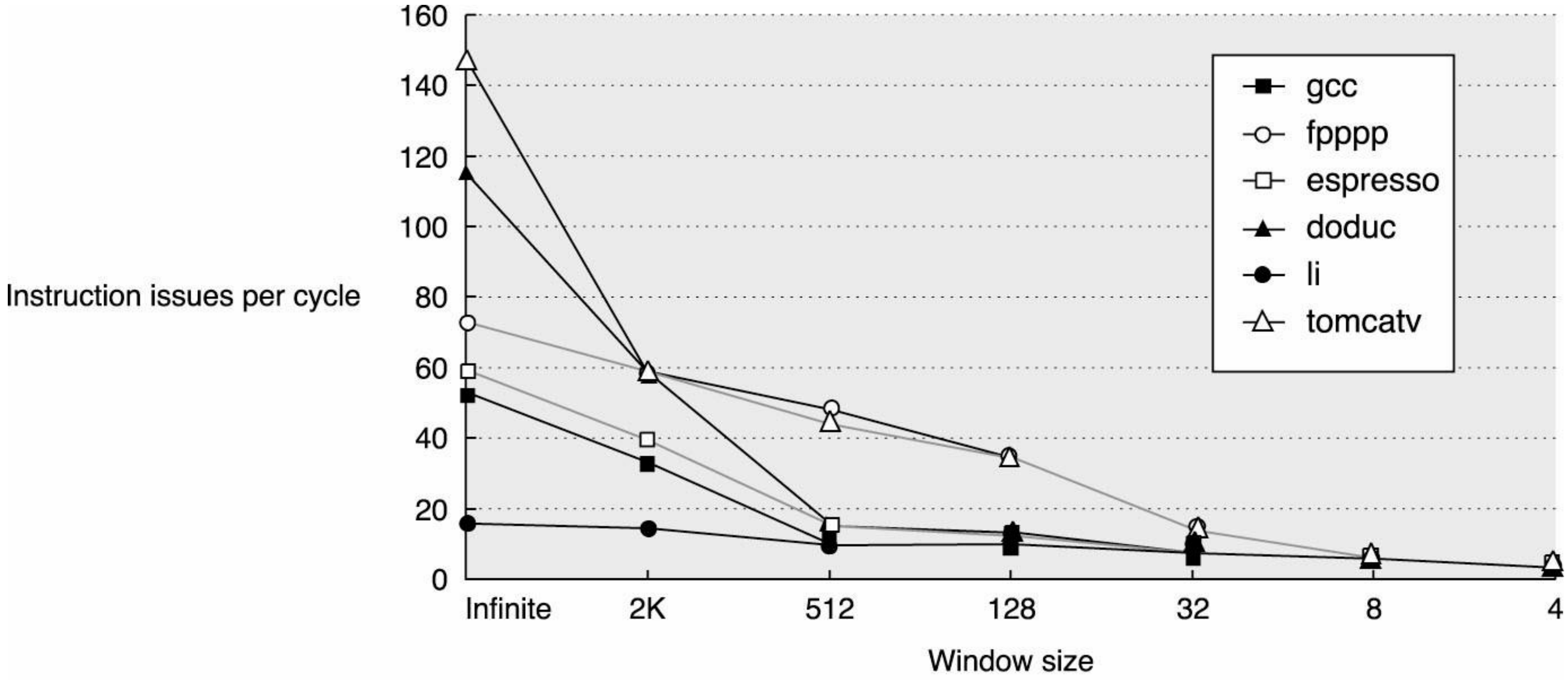


- Requirements
  - – Parallelism
  - – Large window
  - – Limited control deps
  - – Eliminate “false” deps
  - – Find run-time deps

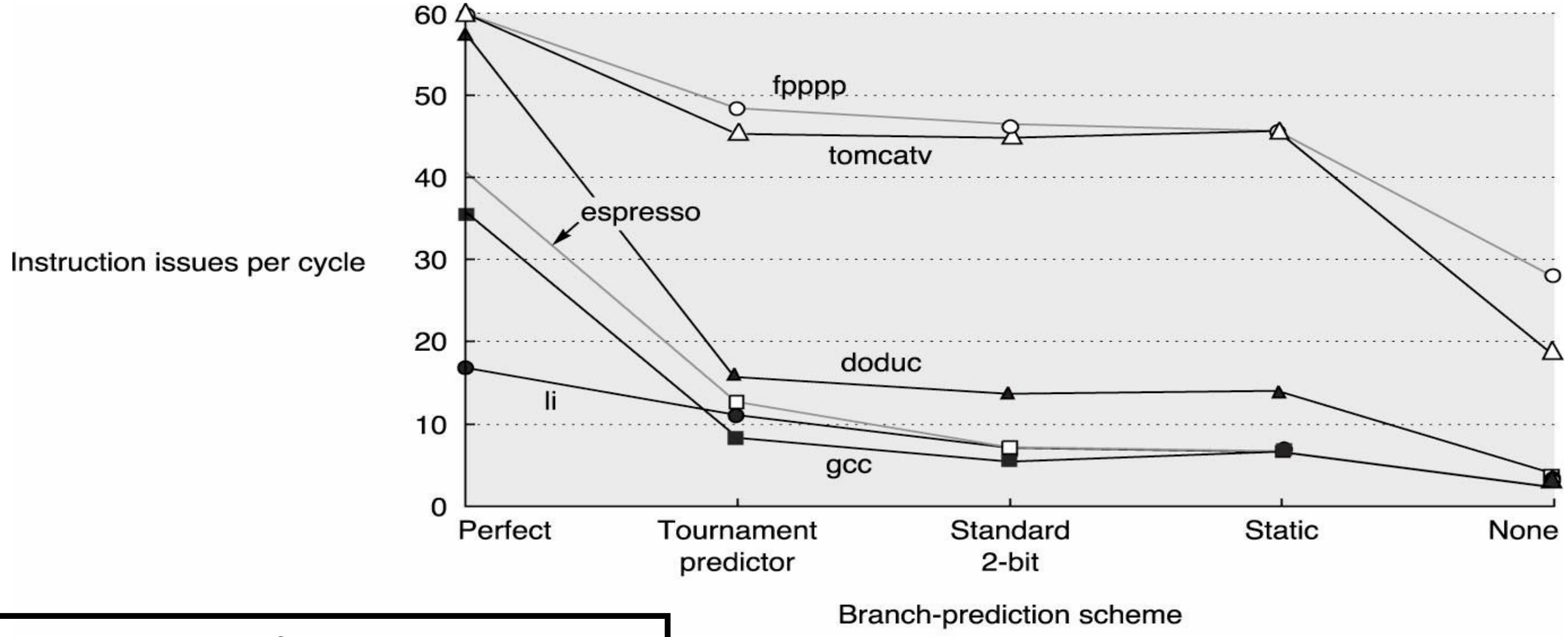
How Much ILP is There? (Chapter 3.10)



How Large Must the “Window” Be?



ALU Operation *GOOD*, Branch *BAD*

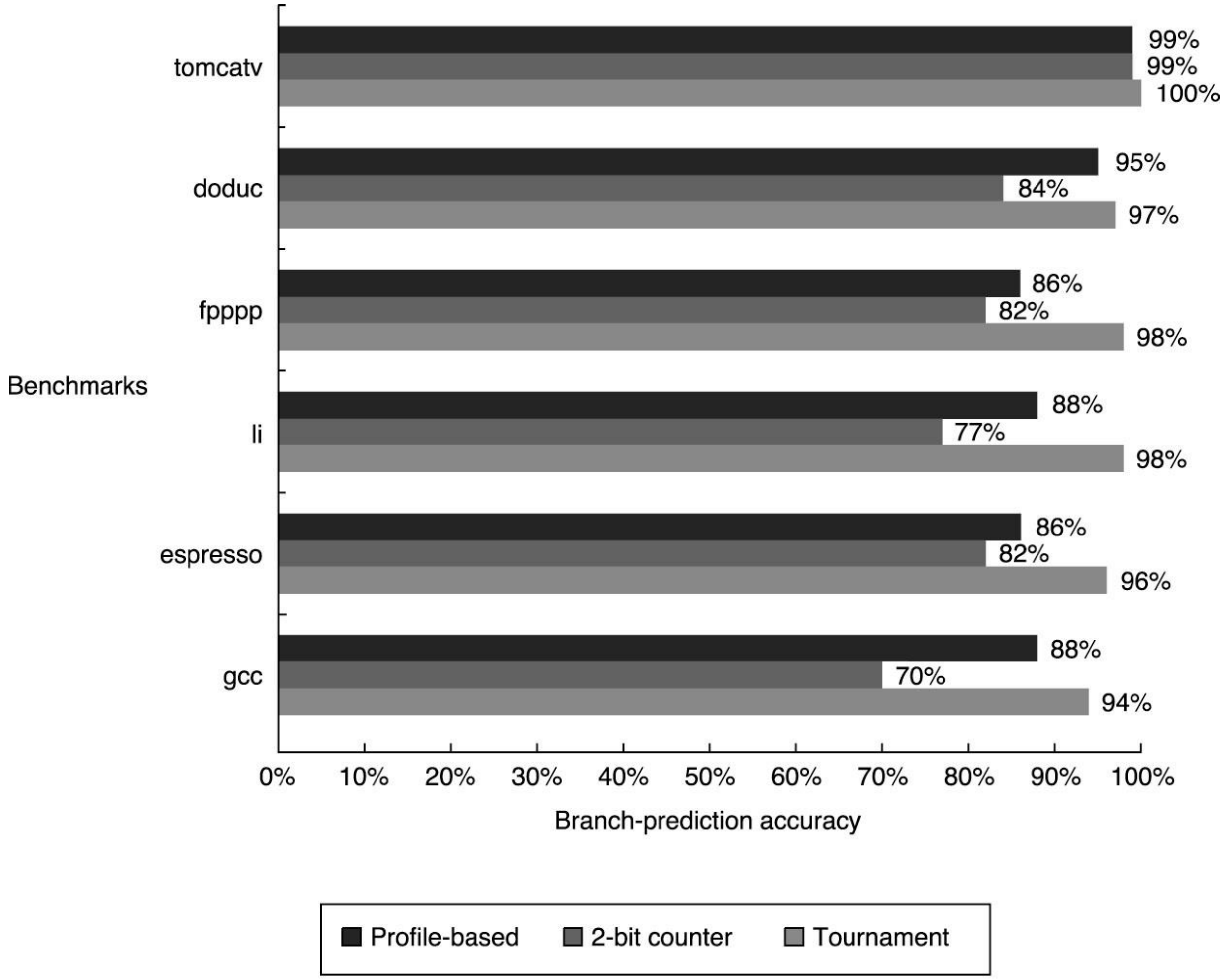


Expected Number of Branches Between Mispredicts

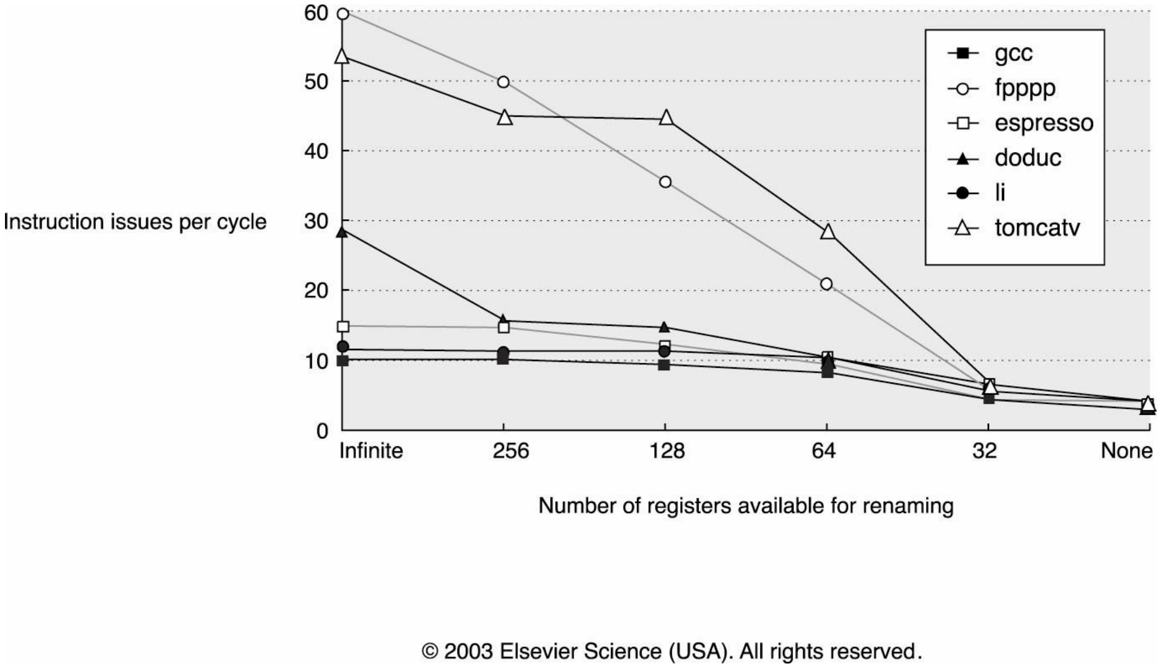
E(X) ~ 1/(1-p)

E.g., p = 95%, E(X) ~ 20 brs, 100-ish insts

How Accurate are Branch Predictors?



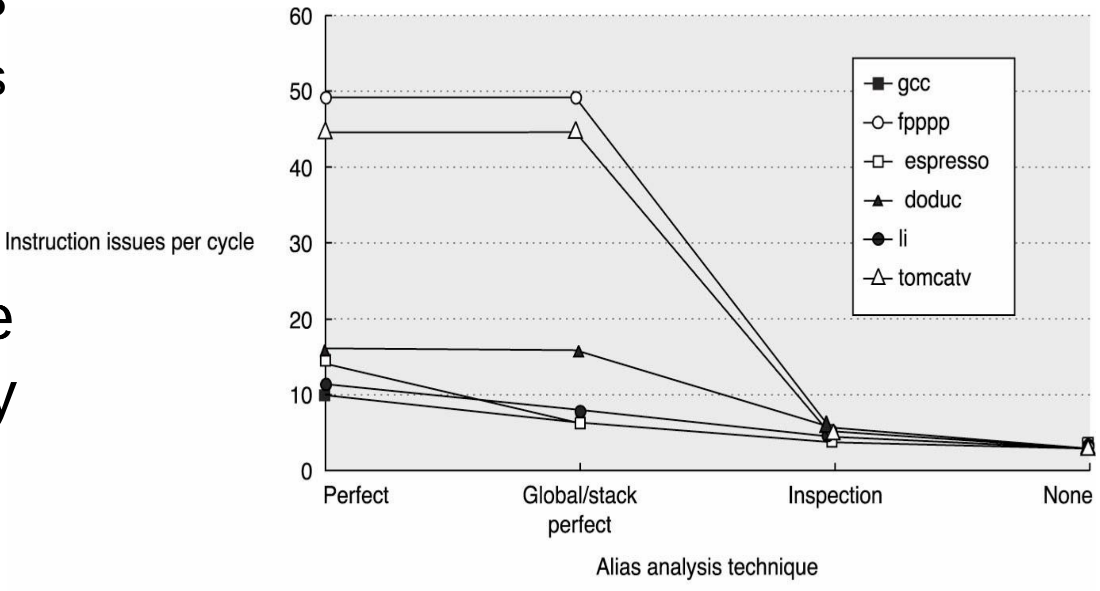
Impact of Physical Storage Limitations



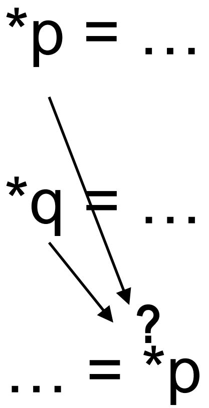
- Each instruction “in flight” must have storage for its result
  - – Really worse than this because of mispeculation…

Registers *GOOD*, Memory *BAD*

- Benefits of registers
  - Well described deps
  - Fast access
  - Finite resource
- Memory loses these benefits for flexibility



© 2003 Elsevier Science (USA). All rights reserved.



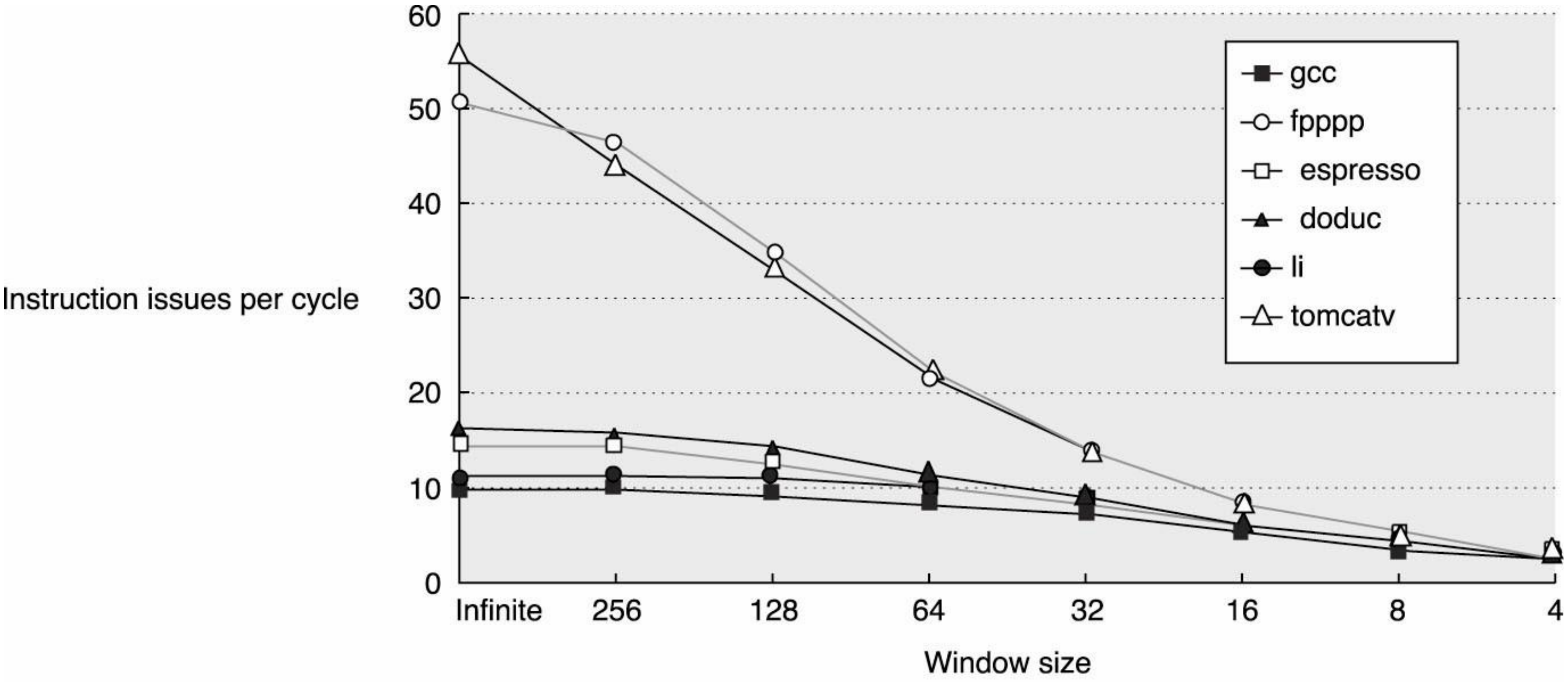
*p = …

*q = …

?

… = *p

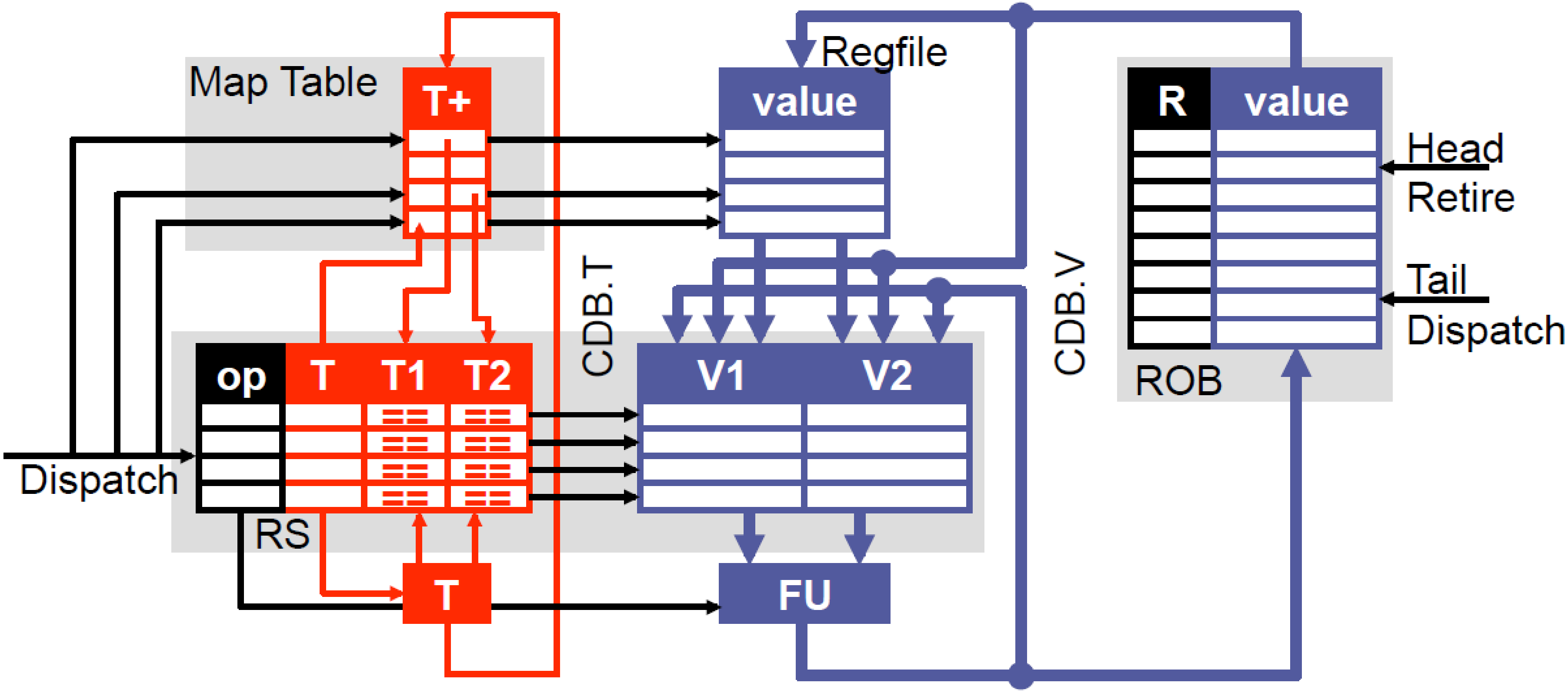
“Bottom Line” for an Ambitious Design



P6 reviewed

- Steps are:
  - Dispatch to the OoO system
  - Issue to functional units
    - Wakeup
    - Select
  - Complete Execute
  - Retire

We added a Reorder Buffer



Let’s lose the ARF! (R10K scheme)

- Why?
  - Currently have two structures that may hold values (ROB and ARF)
  - Need to write back to the ARF after every instruction!
- Other motivations?
  - ROB currently holds result (which needs to be accessible to all) as well as other data (PC, etc.) which does not.
    - So probably two separate structures anyways
  - Many ROB entry *result fields* are unused (stores, branches)

Physical Register file
Version 1.0

- Keep a “Physical register file”
  - If you want to get the ARF back you need to use the RAT.
- But the RAT has speculative information in it!
  - We need to be able to undo the speculative work!
    - How?

How?

- Remove
  - The value field of the ROB
  - The whole ARF
- Add
  - A “retirement RAT” (RRAT)
  - A “Physical Register File” (PRF)
- Actions:
  - When you finish executing, send data to the PRF
  - When you retire, update the RRAT as if you were dispatching and updating the RAT.
  - (Other stuff we need to think about goes here.)
  - On a mis-predict, update the RAT with the RRAT when squashing.

RAT/RRAT Example

RAT

AR	PR
0	1
1	2
2	3
3	4
4	10

Assembly
R1=R2*R3
R3=R1+R3

RRAT

AR	PR
0	1
1	2
2	3
3	4
4	10

RAT/RRAT Example

RAT

AR	PR
0	1
1	0
2	3
3	5
4	10

In-flight

Assembly
R1=R2*R3
R3=R1+R3

Renamed
P0=P3*P4
P5=P0+P4

RRAT

AR	PR
0	1
1	2
2	3
3	4
4	10

This seems sorta okay but…

- There seem to be some problems
  - When can I free a physical register?
  - If I’m writing to the physical register file at *execute* doesn’t that mean I committing at that point?
  - How do I squash instructions?
  - How do I recover architected state in the event of an exception?

Freedom

- Freeing the PRF
  - How long must we keep each PRF entry?
    - Until we are *sure* no one else will read it before the corresponding Architected Register is again written.
    - Once the instruction overwriting the Architected Register commits we are certain safe.
  - So free the PR when the instruction which overwrites it commits.
    - In other words: when an instruction commits, it frees the PR it overwrites in the RRAT.

- We could do better (?)
  - The value is dead once it is no longer needed.
    - Right now waiting until the AR is overwritten…
  - Freeing earlier would reduce the number of PRs needed.
  - But unclear how to do given speculation and everything else.

Sidebar

- One thing that must happen with the PRF is that a “free list” must exist letting the processor know which physical registers are available.
  - Maintaining these free lists can be a pain!

RAT

AR	Target
0	4
1	2
2	7
3	1

A: R1=MEM[R2+0]
B: R2=R3/R1
C: R3=R2+R0
D: Branch (if R1!=0)
E: R3=R1+R3
F: R3=R3+R0
G: R3=R3+19
H: R1=R7+R6

RRAT

AR	Target
0	
1	
2	
3	

RoB

0	1	2	3	4	5	6	7	8	9
3	2	44	56	3	66	7	11	8	20

PRF

Resolving Branches Early: A variation

- Keep a RAT copy for each branch *in a RS!*
  - – If mis-predict, can recover RAT quickly.
  - – Free lists also

Project Overview

- Grade breakdown
  - 22 points: Basics
    - Out-of-order and *something* works
  - 20 points: Correctness
    - Measured by how many tests you pass.
  - 17 points: Advanced features
  - 20 points: Performance
    - Measured against your peers and previous semesters.
  - 10 points: Analysis
    - Measuring something interesting. Ideally the impact of an advanced feature.
  - 6 points: Documentation
    - You'll do this at the end, don't worry about it now.
  - 3 points: Milestone 1
    - You'll turn in some *self-testing* code. We'll see if it does a good job.
  - 2 points: Peer feedback
    - Do it on time & take it seriously you'll get these points.

Advanced features

- 17 points of advanced feature stuff.
  - We suggest you consider one big thing in the core and a few small things outside of the core.
    - Superscalar execution (3-way*, arbitrary **)
    - Simultaneous Multi-threading (SMT) ***
    - Multi-core with a shared, coherent and consistent write-back L2 cache. ***
    - Exception handling$^{?}$
    - Early branch resolution (before the branch hits the head of the RoB)
    - Multi-path execution on low-confidence branches (this may not help performance much…)

Non-core features

- Much of this we haven’t covered yet.
- Better caches
  - Associative, longer cache lines, etc.
  - Non-blocking caches
    - Harder than it looks
- Better predictors
  - Gshare, tournament, etc.
- Prefetching

Psuedo-core features

- Adding instructions
  - Say cmov
    - This probably involves rewriting at least one benchmark.
- Checkers
  - Tricky.

Wacky features

- Think of something interesting and run with it.
  - We’ve had weird schedulers for EX units and other things..

Performance

- Simple measure of how long it takes to finish a program.
  - – Doesn’t include flushing caches etc.
  - – Only get credit for right answers.
    - If you don’t synthisize, we can’t know your clock period, so few if any points here.
- You’d like to pick your features so you double-dip.
  - – Hint: Prefetching instructions is good.

Analysis

- Think about what you want to measure.
  - – Impact of a better cache?
  - – How full your RoB is?
  - – How much your early branch resolution helps.
- Do a good job grabbing the data.
  - – *Be sure you can distinguish testbenches that are good for measuring **performance** vs. those that are good for **correctness!***

Report

- Only thing to think about now is that we like things which show us how a feature works.
  - So having your debug data be readable could be handy.

Forming teams

- We’ll post a signup sheet for teams tomorrow.
- Teams are groups of 5
  - We may have one group of 6?
- We’ll finish about 5 minutes early on Thursday to finish team formation.