

# **Lecture 13: Trace Scheduling, Conditional Execution, Speculation, Limits of ILP**

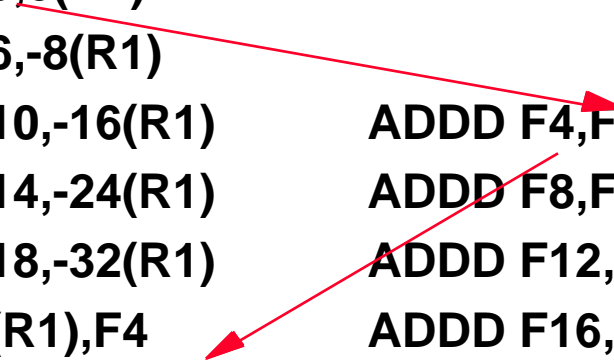
**Professor Randy H. Katz  
Computer Science 252  
Spring 1996**

# Review: Getting CPI < 1 Multiple Instructions/Cycle

- **Two variations:**
  - **Superscalar**: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
    - » IBM PowerPC, Sun SuperSparc, DEC Alpha, HP 7100
  - **Very Long Instruction Words (VLIW)**: fixed number of instructions (16) scheduled by the compiler
    - » Joint HP/Intel agreement in 1998?

# Loop Unrolling in SuperScalar

	<i>Integer instruction</i>	<i>FP instruction</i>	<i>Clock cycle</i>
Loop:	LD F0,0(R1)		1
	LD F6,-8(R1)		2
	LD F10,-16(R1)	ADDD F4,F0,F2	3
	LD F14,-24(R1)	ADDD F8,F6,F2	4
	LD F18,-32(R1)	ADDD F12,F10,F2	5
	SD 0(R1),F4	ADDD F16,F14,F2	6
	SD -8(R1),F8	ADDD F20,F18,F2	7
	SD -16(R1),F12		8
	SD -24(R1),F16		9
	SUBI R1,R1,#40		10
	BNEZ R1,LOOP		11
	SD -32(R1),F20		12



**Unrolled 5 times to avoid delays (+1 due to SS)**

**12 clocks, or 2.4 clocks per iteration**

# Loop Unrolling in VLIW

<i>Memory reference 1</i>	<i>Memory reference 2</i>	<i>FP operation 1</i>	<i>FP op. 2</i>	<i>Int. op/branch</i>	<i>Clock</i>
LD F0,0(R1)	LD F6,-8(R1)				1
LD F10,-16(R1)	LD F14,-24(R1)				2
LD F18,-32(R1)	LD F22,-40(R1)	ADDD F4,F0,F2	ADDD F8,F6,F2		3
LD F26,-48(R1)		ADDD F12,F10,F2	ADDD F16,F14,F2		4
		ADDD F20,F18,F2	ADDD F24,F22,F2		5
SD 0(R1),F4	SD -8(R1),F8	ADDD F28,F26,F2			6
SD -16(R1),F12	SD -24(R1),F16				7
SD -32(R1),F20	SD -40(R1),F24			SUBI R1,R1,#48	8
SD -0(R1),F28				BNEZ R1,LOOP	9

**Unrolled 7 times to avoid delays**

**7 results in 9 clocks, or 1.3 clocks per iteration**

**Need more registers in VLIW**

# Limits to Multi-Issue Machines

- **Inherent limitations of ILP**
  - 1 branch in 5: How to keep a 5-way VLIW busy?
  - Latencies of units: many operations must be scheduled
  - Need about Pipeline Depth x No. Functional Units of independent operations to keep machines busy
- **Difficulties in building HW**
  - Duplicate FUs to get parallel execution
  - Increase ports to Register File
    - » VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Increase ports to memory
  - Decoding SS and impact on clock rate, pipeline depth

# Limits to Multi-Issue Machines

- **Limitations specific to either SS or VLIW implementation**
  - Decode issue in SS
  - VLIW code size: unroll loops + wasted fields in VLIW
  - VLIW lock step => 1 hazard & all instructions stall
  - VLIW & binary compatibility is practical weakness

# Software Pipelining Example

Before: Unrolled 3 times

```

1  LD    F0,0(R1)
2  ADDD  F4,F0,F2
3  SD    0(R1),F4
4  LD    F6,-8(R1)
5  ADDD  F8,F6,F2
6  SD    -8(R1),F8
7  LD    F10,-16(R1)
8  ADDD  F12,F10,F2
9  SD    -16(R1),F12
10 SUBI  R1,R1,#24
11 BNEZ  R1,LOOP
    
```

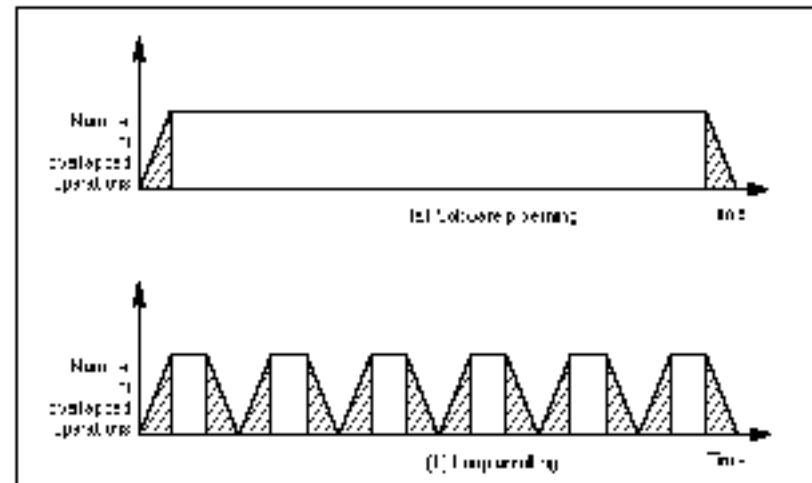
After: Software Pipelined

```

1  SD    0(R1),F4 ; Stores M[i]
2  ADDD  F4,F0,F2 ; Adds to M[i-1]
3  LD    F0,-16(R1); Loads M[i-2]
4  SUBI  R1,R1,#8
5  BNEZ  R1,LOOP
    
```

- **Symbolic Loop Unrolling**

- Less code space
- Fill & drain pipe only once  
vs. each iteration in loop unrolling



# Review: Summary

- **Branch Prediction**
  - Branch History Table: 2 bits for loop accuracy
  - Correlation: Recently executed branches correlated with next branch
  - Branch Target Buffer: include branch address & prediction
- **SuperScalar and VLIW**
  - $CPI < 1$
  - Dynamic issue vs. Static issue
  - More instructions issue at same time, larger the penalty of hazards
- **SW Pipelining**
  - Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead



# Trace Scheduling

- **Parallelism across IF branches vs. LOOP branches**
- **Two steps:**
  - ***Trace Selection***
    - » Find likely sequence of basic blocks (*trace*) of (statically predicted) long sequence of straight-line code
  - ***Trace Compaction***
    - » Squeeze trace into few VLIW instructions
    - » Need bookkeeping code in case prediction is wrong

# HW support for More ILP

- **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 or cause exception
  - Expanded ISA of Alpha, MIPS, PowerPC, SPARC have conditional move; PA-RISC can annul any following instr.
- **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

# HW support for More ILP

- ***Speculation***: allow an instruction to issue that is dependent on branch predicted to be taken *without* any consequences (including exceptions) if branch is not actually taken (“HW undo”)
- Often try to combine with dynamic scheduling
- Tomasulo: separate ***speculative*** bypassing of results from real bypassing of results
  - When instruction no longer speculative, write results (***instruction commit***)
  - execute out-of-order but commit in order

# HW support for More ILP

- **Need HW buffer for results of uncommitted instructions:**  
***reorder buffer***
  - Reorder buffer can be operand source
  - Once operand commits, result is found in register
  - 3 fields: instr. type, destination, value
  - Use reorder buffer number instead of reservation station
  - Instructions commit in order
  - As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions

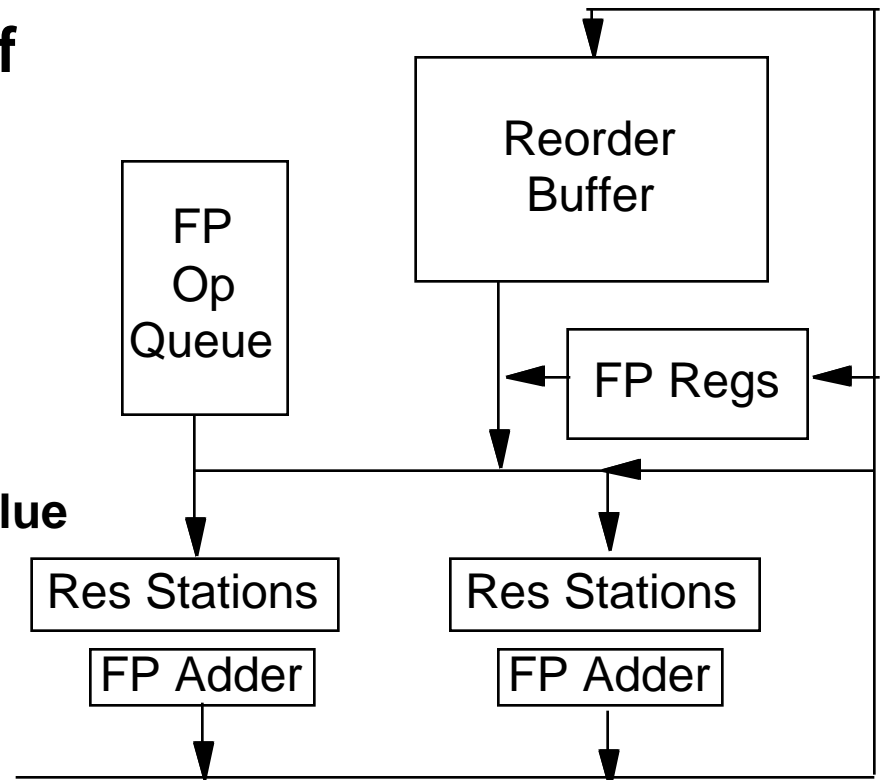


Figure 4.34, page 311

# Four Steps of Speculative Tomasulo Algorithm

## 1. Issue—get instruction from FP Op Queue

If reservation station or reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination.

## 2. Execution—operate on operands (EX)

When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute

## 3. Write result—finish execution (WB)

Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

## 4. Commit—update register with reorder result

When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer.

# Limits to ILP

- **Conflicting studies of amount of parallelism available in late 1980s and early 1990s. Different assumptions about:**
  - **Benchmarks (vectorized Fortran FP vs. integer C programs)**
  - **Hardware sophistication**
  - **Compiler sophistication**

# Limits to ILP

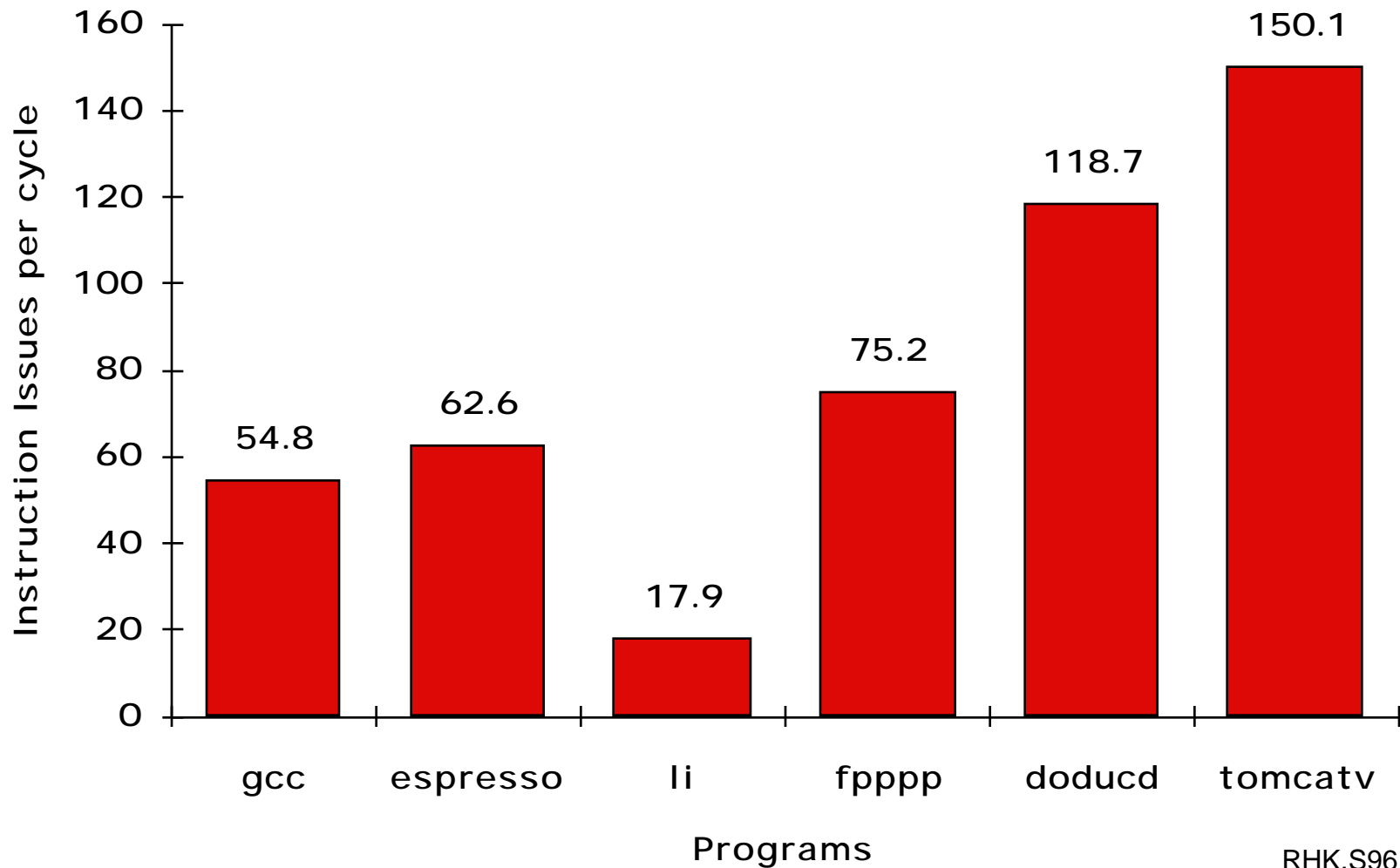
Initial HW Model here; MIPS compilers

1. ***Register renaming***—infinite virtual registers and all WAW & WAR hazards are avoided
2. ***Branch prediction***—perfect; no mispredictions
3. ***Jump prediction***—all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. ***Memory-address alias analysis***—addresses are known & a store can be moved before a load provided addresses not equal

1 cycle latency for all instructions

# Upper Limit to ILP

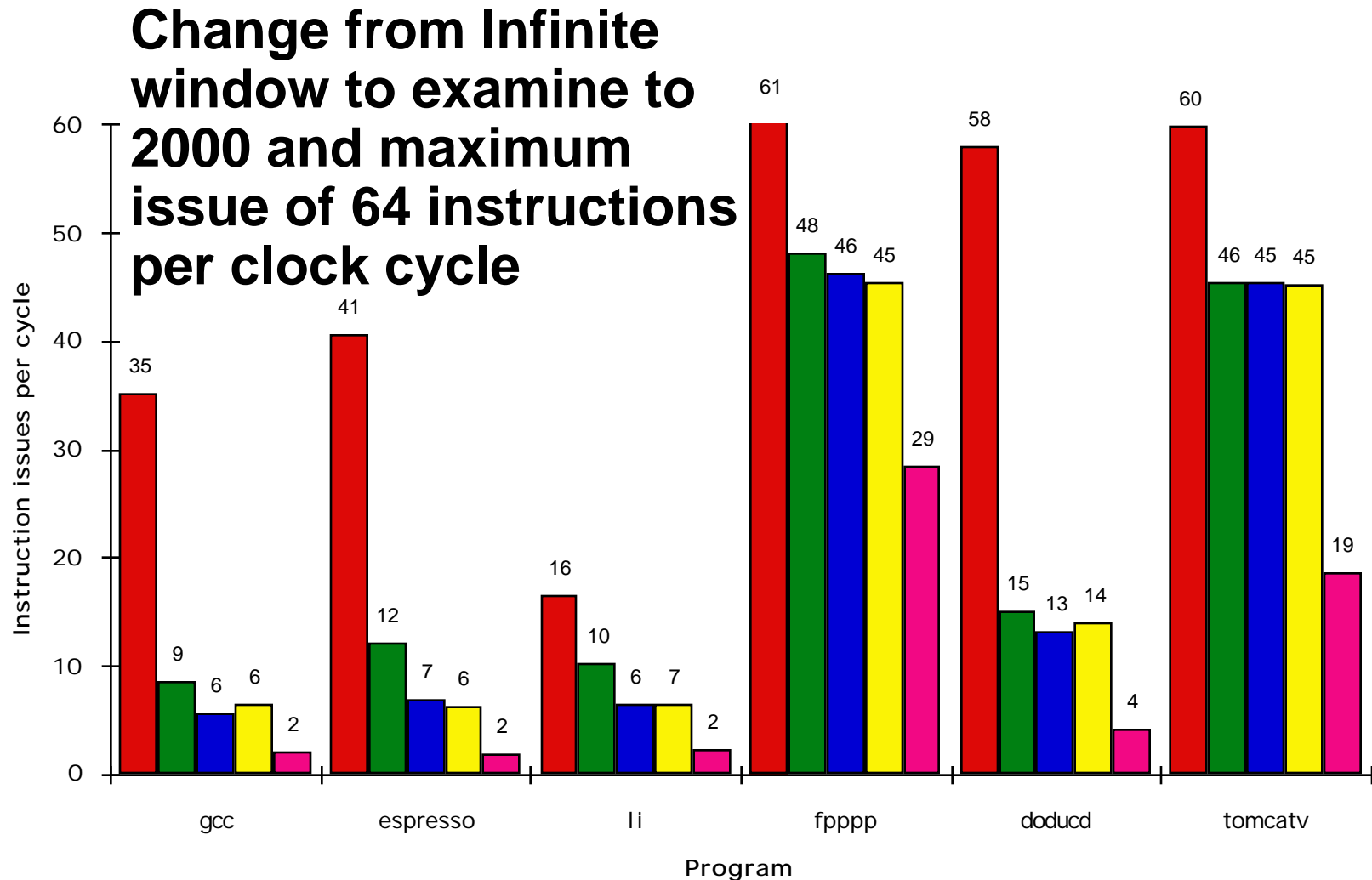
(Figure 4.38, page 319)





# More Realistic HW: Branch Impact

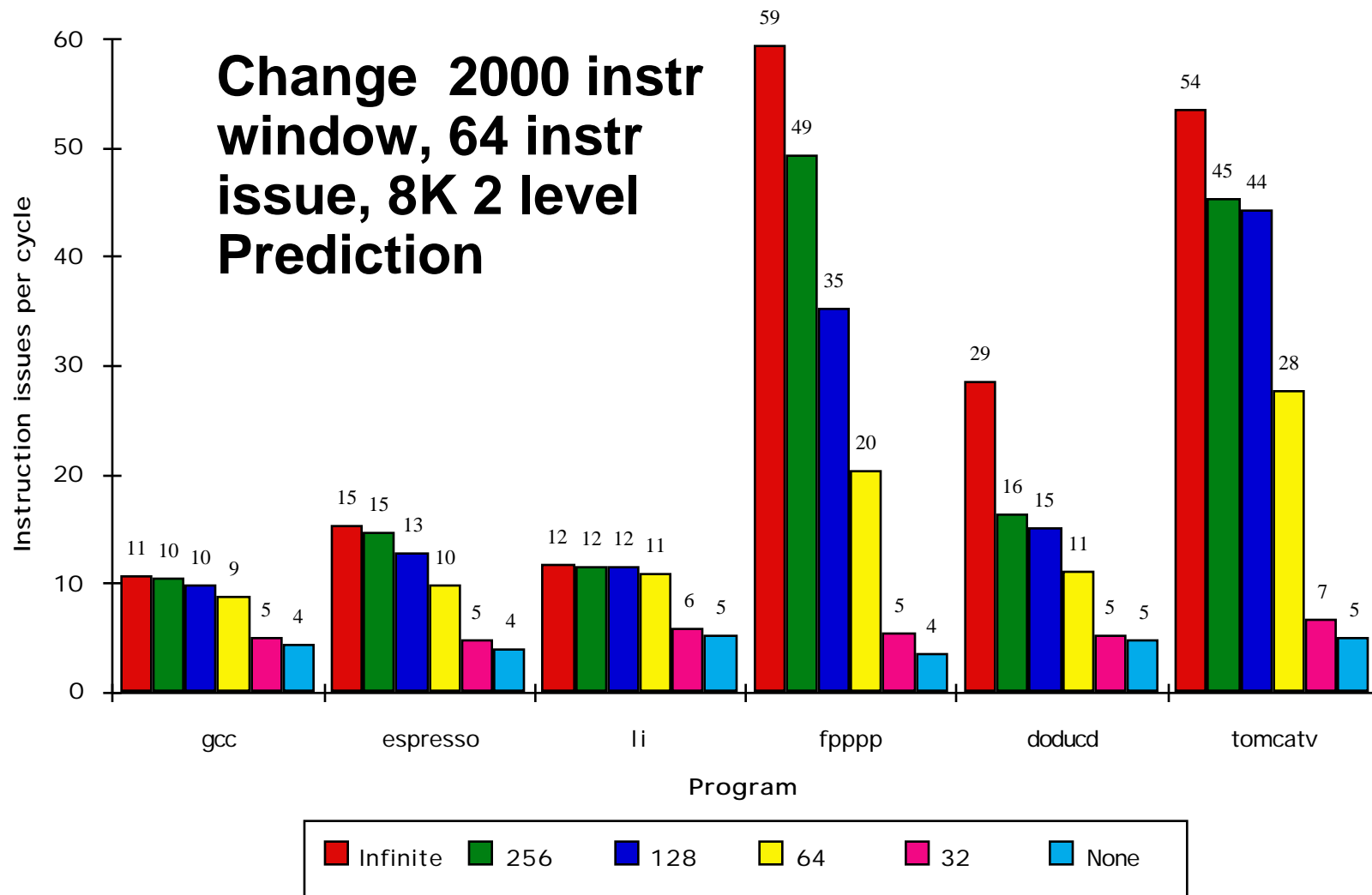
Figure 4.40, Page 323



**Perfect** **Pick Cor. or BHT** **BHT (512)** **Profile**

# More Realistic HW: Register Impact

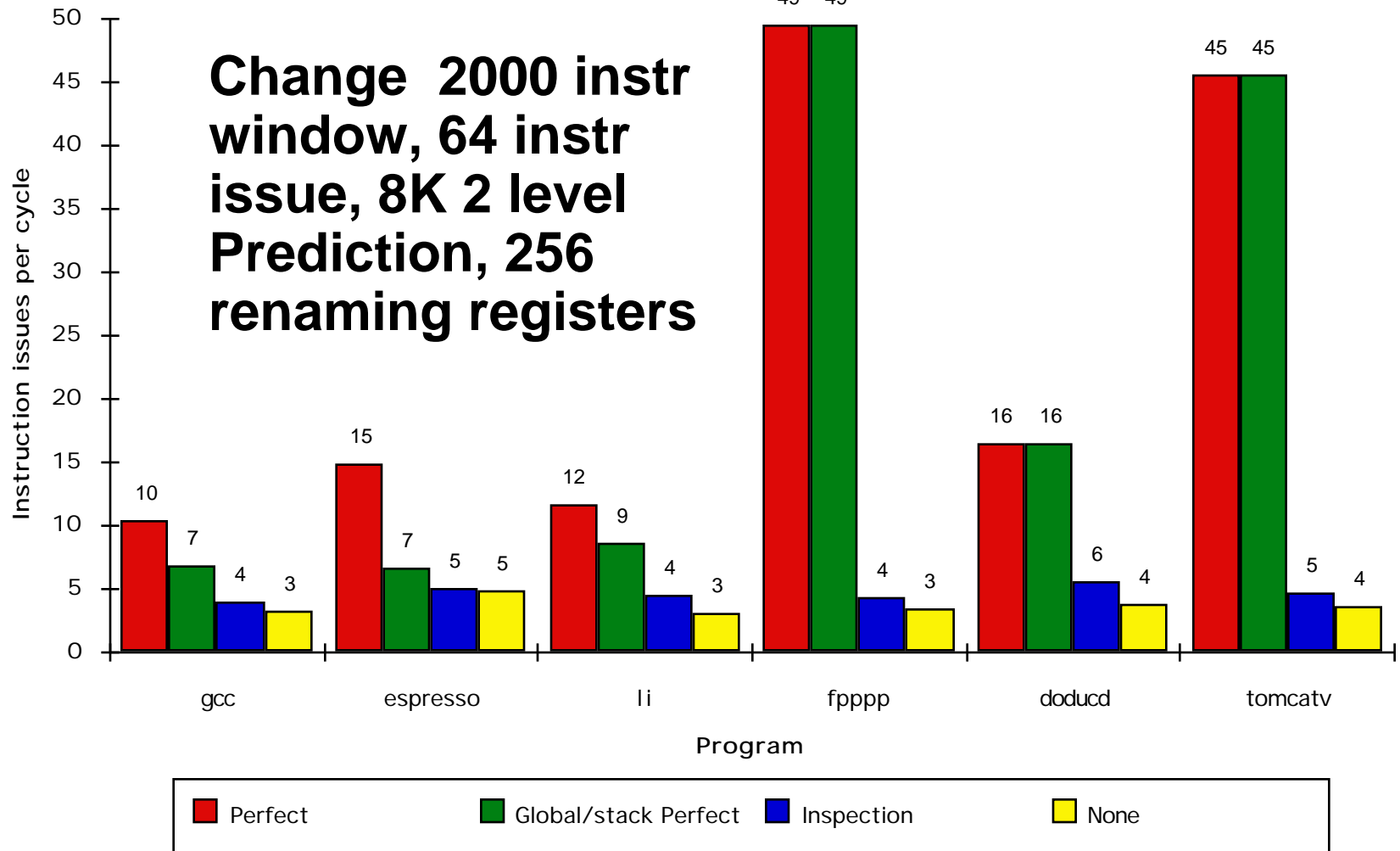
Figure 4.42, Page 325



**Infinite 256 128 64 32 None**

# More Realistic HW: Alias Impact

Figure 4.44, Page 328



**Perfect**

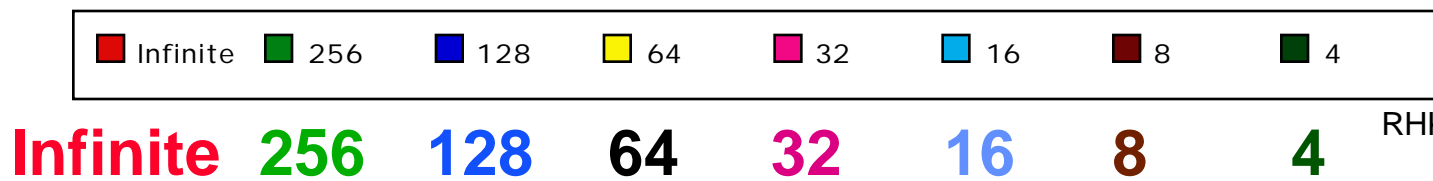
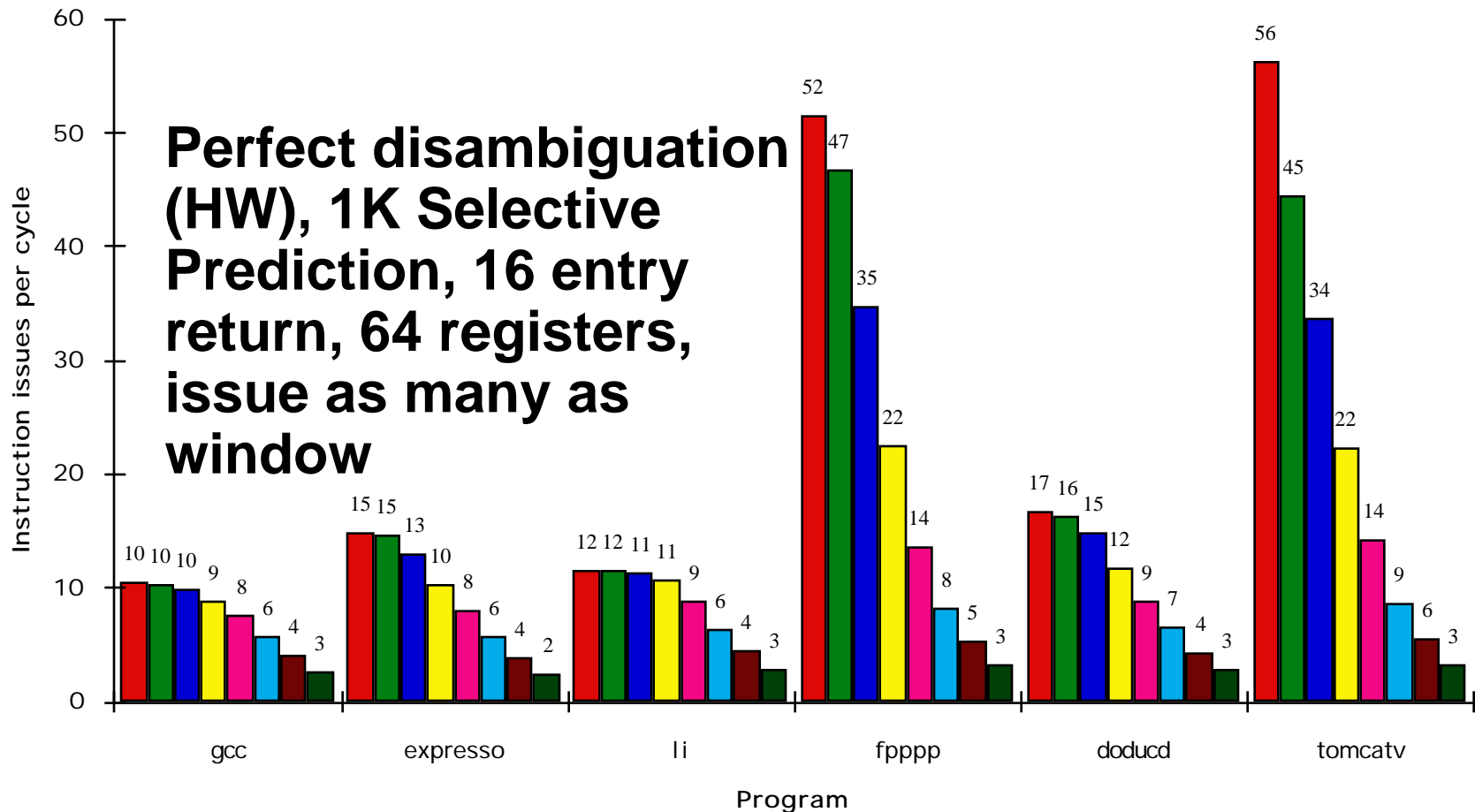
**Global/Stack perf;  
heap conflicts**

**Inspection  
Assem.**

**None**

# Realistic HW for '9X: Window Impact

(Figure 4.48, Page 332)



# Braniac vs. Speed Demon

- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe)  
vs. 2-scalar Alpha @ 200 MHz (7 stage pipe)

