Lecture 5: VLIW, Software Pipelining, and Limits to ILP

Professor David A. Patterson

Computer Science 252

Spring 1998

Review: Tomasulo

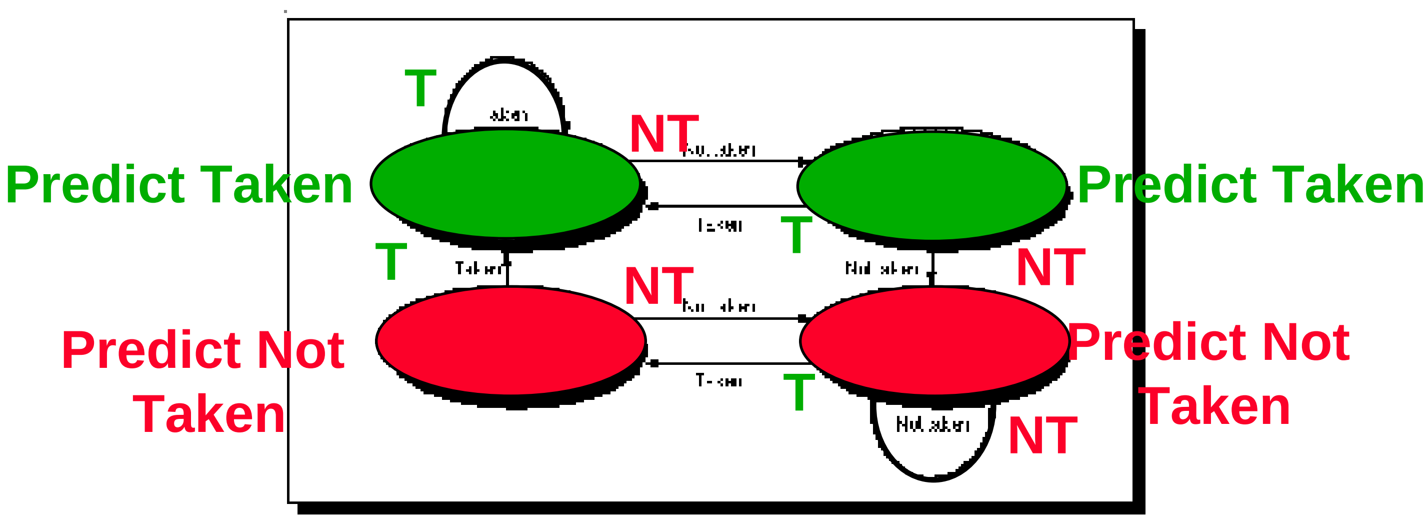
- **Prevents Register as bottleneck**
- **Avoids WAR, WAW hazards of Scoreboard**
- **Allows loop unrolling in HW**
- **Not limited to basic blocks (provided branch prediction)**
- **Lasting Contributions**
  - **Dynamic scheduling**
  - **Register renaming**
  - **Load/store disambiguation**
- **360/91 descendants are PowerPC 604, 620; MIPS R10000; HP-PA 8000; Intel Pentium Pro**

Dynamic Branch Prediction

- **Performance = *f*(accuracy, cost of misprediction)**
- **Branch History Table is simplest**
  - **Lower bits of PC address index table of 1-bit values**
  - **Says whether or not branch taken last time**
  - **No address check**
- **Problem: in a loop, 1-bit BHT will cause two mispredictions (avg is 9 iteratios before exit):**
  - **End of loop case, when it exits instead of looping as before**
  - **First time through loop on *next* time through code, when it predicts exit instead of looping**

Dynamic Branch Prediction

- **Solution: 2-bit scheme where change prediction only if get misprediction *twice:* (Figure 4.13, p. 264)**



- **Red: stop, not taken**
- **Green: go, taken**

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)**

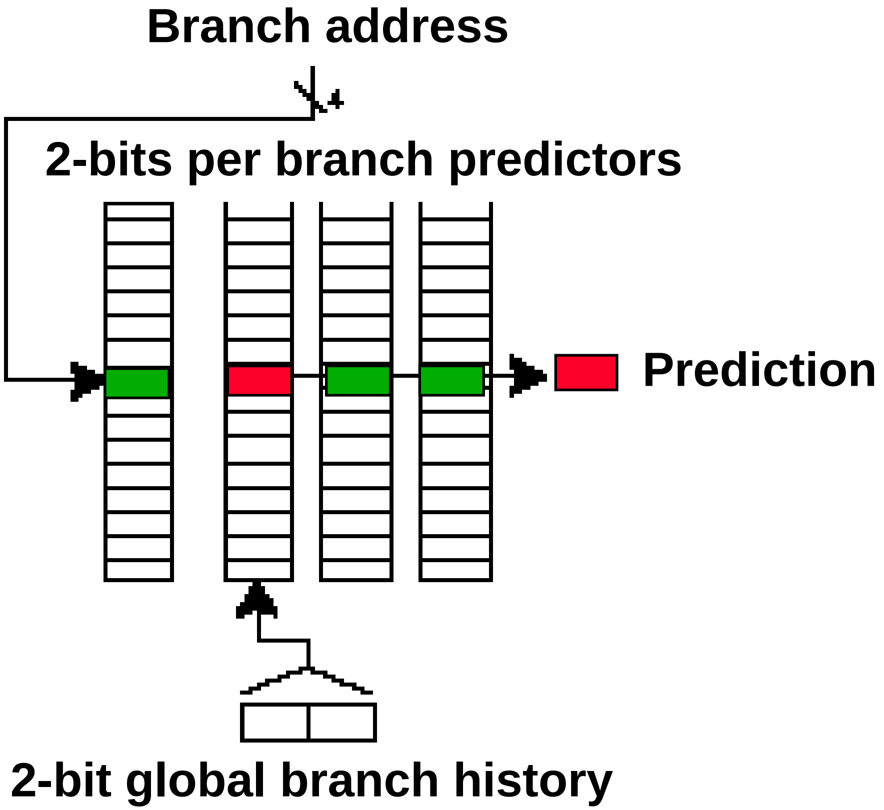
Correlating Branches

- **Hypothesis: recent branches are correlated; that is, behavior of recently executed branches affects prediction of current branch**
- **Idea: record m most recently executed branches as taken or not taken, and use that pattern to select the proper branch history table**
- **In general, (m,n) predictor means record last m branches to select between 2^m history talbes each with n-bit counters**
  - **Old 2-bit BHT is then a (0,2) predictor**

Correlating Branches

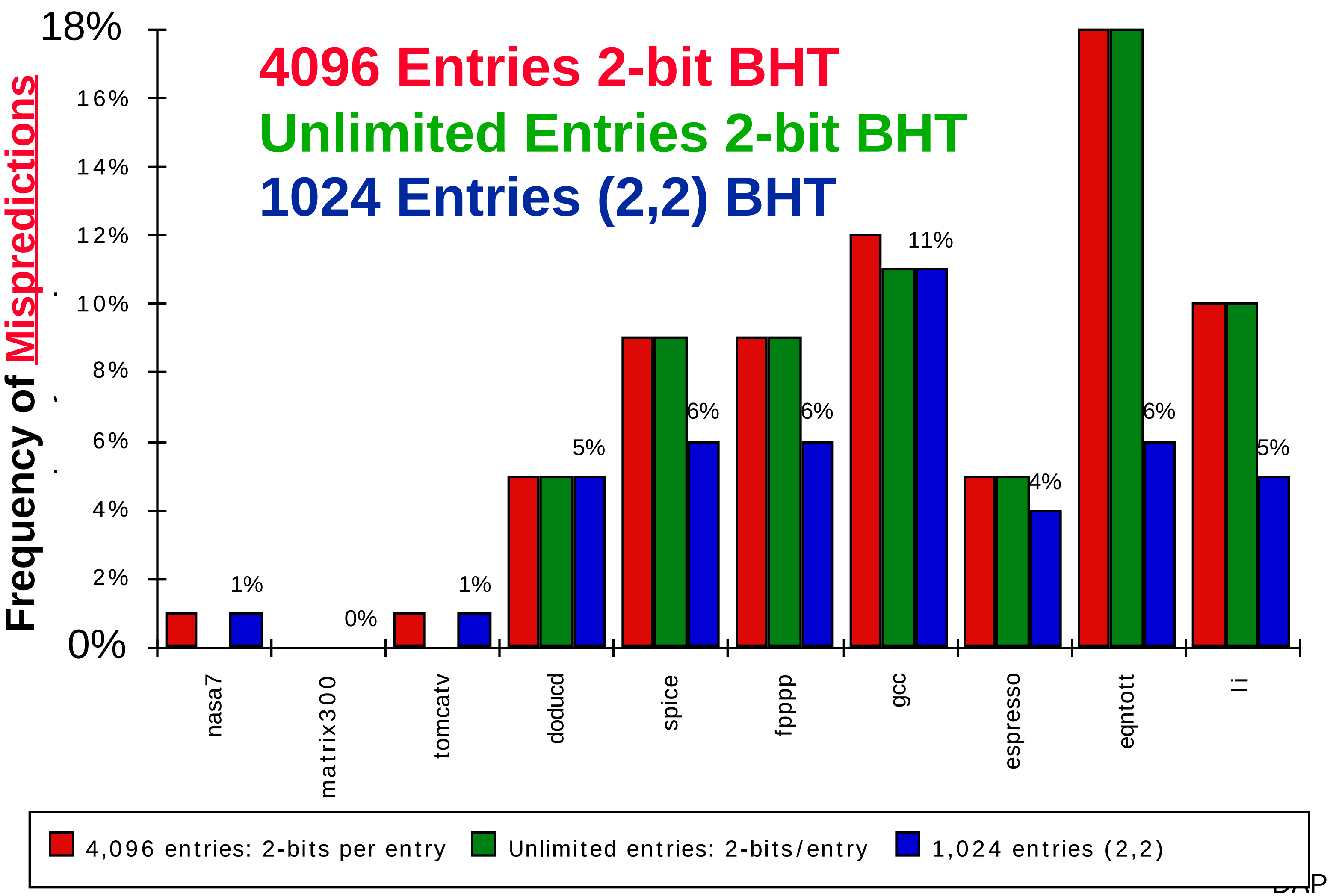
(2,2) predictor

- Then behavior of recent branches selects between, say, four predictions of next branch, updating just that prediction



Accuracy of Different Schemes

(Figure 4.21, p. 272)



Re-evaluating Correlation

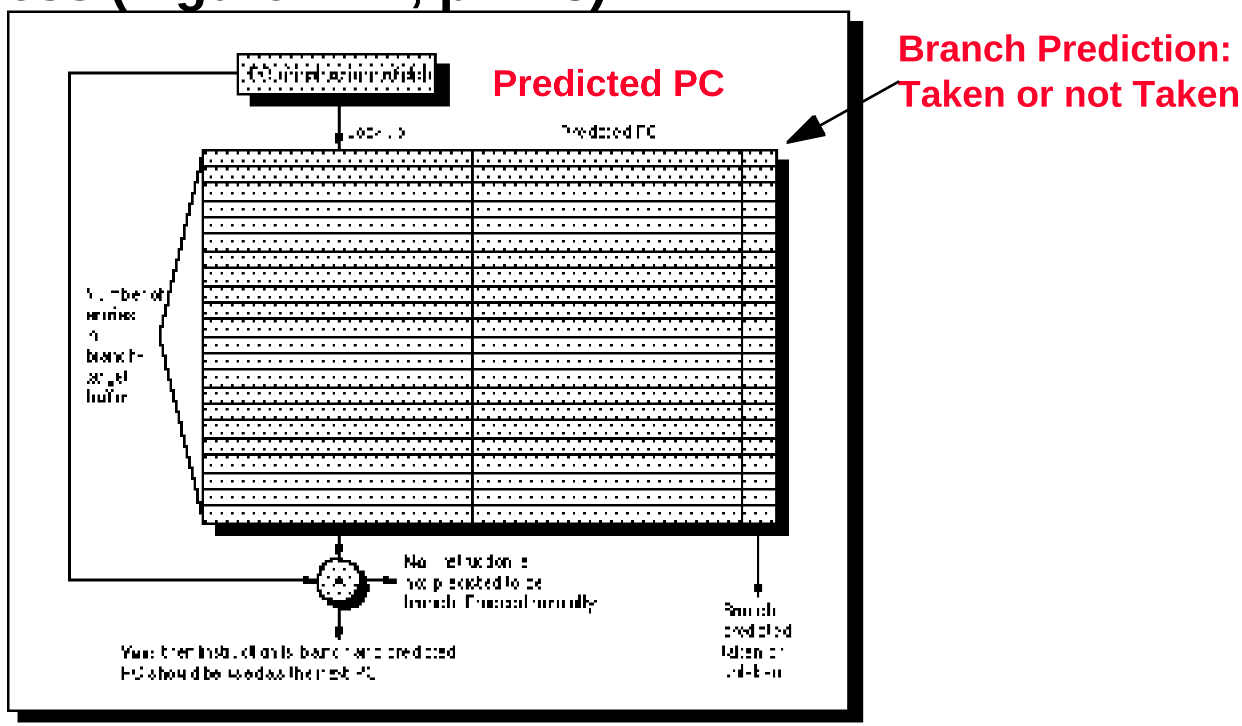
- **Several of the SPEC benchmarks have less than a dozen branches responsible for 90% of taken branches:**

program	branch %	static	# = 90%
compress	14%	236	13
eqntott	25%	494	5
gcc	15%	9531	2020
mpeg	10%	5598	532
real gcc	13%	17361	3214

- **Real programs + OS more like gcc**
- **Small benefits beyond benchmarks for correlation? problems with branch aliases?**

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 4.22, p. 273)**



- **Return instruction addresses predicted with stack**

HW support for More ILP

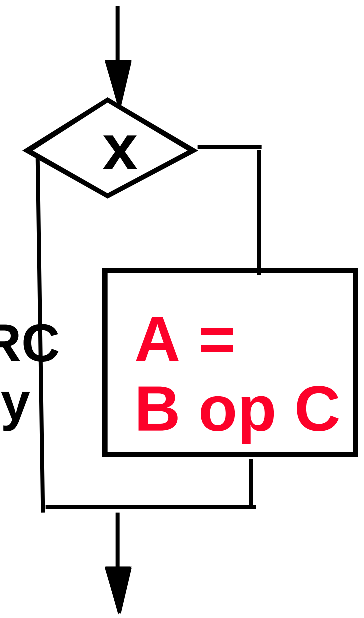
- **Avoid branch prediction by turning branches into conditionally executed instructions:**

  if (x) then A = B op C else NOP

  – **If false, then neither store result nor cause exception**

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

  – **IA-64: 64 1-bit condition fields selected so conditional execution of any instruction**



- **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**

Dynamic Branch Prediction Summary

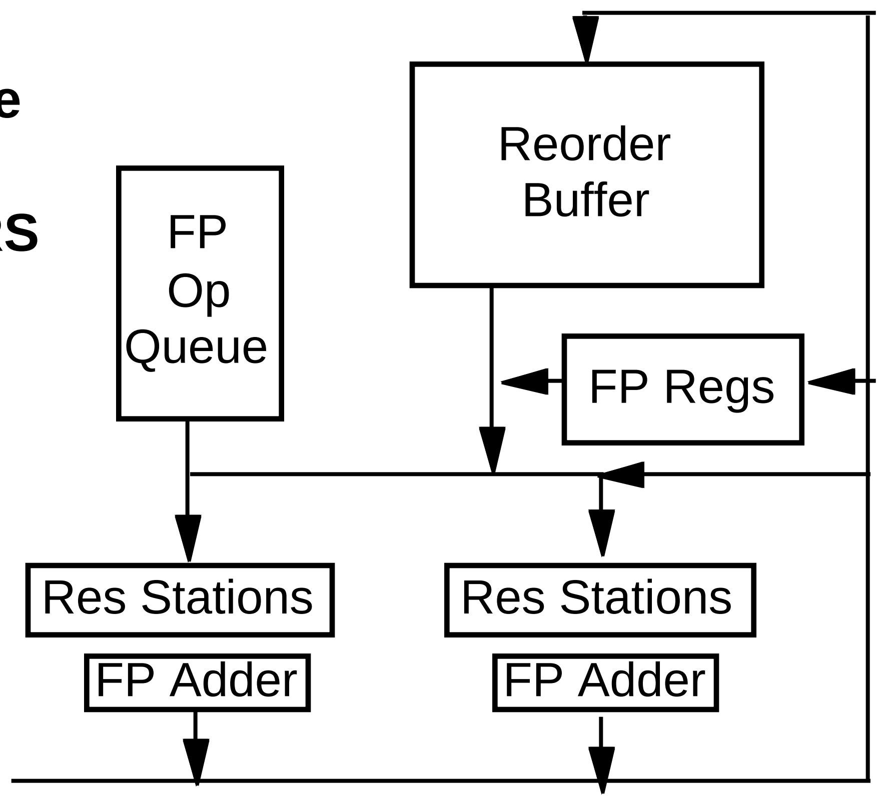
- **Branch History Table: 2 bits for loop accuracy**
- **Correlation: Recently executed branches correlated with next branch**
- **Branch Target Buffer: include branch address & prediction**
- **Predicated Execution can reduce number of branches, number of mispredicted branches**

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"); called "**boosting**"
- Combine branch prediction with dynamic scheduling to execute before branches resolved
- Separate *speculative* bypassing of results from real bypassing of results
  - When instruction no longer speculative, write boosted results (*instruction commit*) or discard boosted results
  - execute out-of-order but commit in-order to prevent irrevocable action (update state or exception) until instruction commits

HW support for More ILP

- **Need HW buffer for results of uncommitted instructions:** ***reorder buffer***
  - **3 fields: instr, destination, value**
  - **Reorder buffer can be operand source => more registers like RS**
  - **Use reorder buffer number instead of reservation station when execution completes**
  - **Supplies operands between execution complete & commit**
  - **Once operand commits, result is put into register**
  - **Instructionscommit in order**
  - **As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions**



Four Steps of Speculative Tomasulo Algorithm

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

   If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination (this stage sometimes called "dispatch")

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; checks RAW (sometimes called "issue")

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. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

Renaming Registers

- **Common variation of speculative design**
- **Reorder buffer keeps instruction information but <u>not</u> the result**
- **Extend register file with extra <u>renaming registers</u> to hold speculative results**
- **Rename register allocated at issue; result into rename register on execution complete; rename register into real register on commit**
- **Operands read either from register file (real or speculative) or via Common Data Bus**
- **Advantage: operands are always from single source (extended register file)**

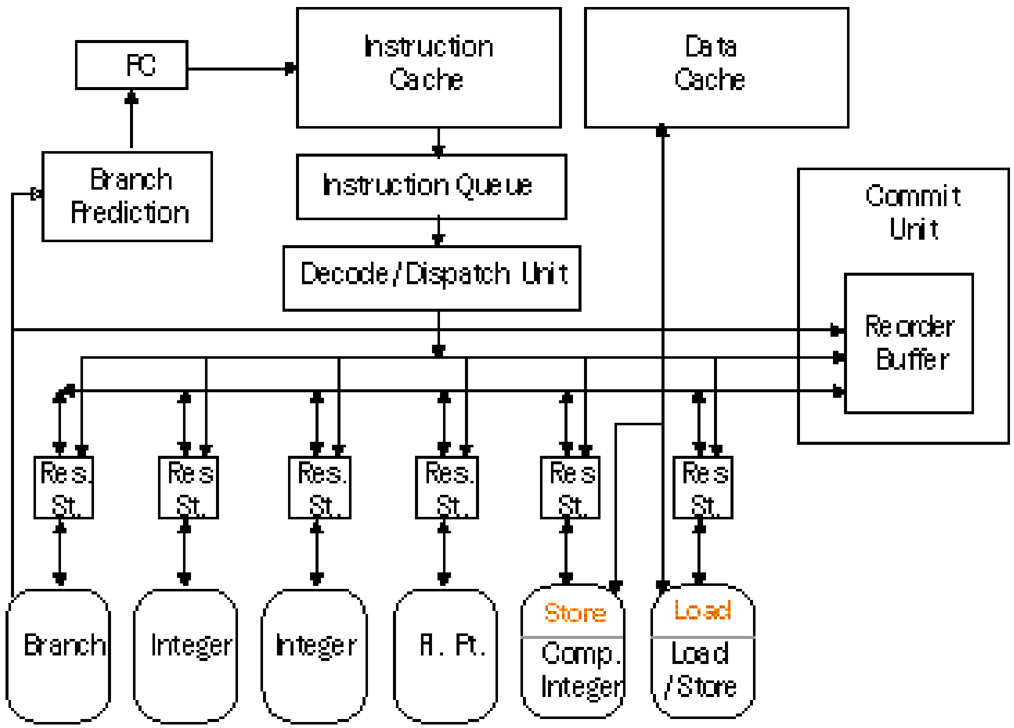
CS 252 Administrivia

- **Get your photo taken by Joe Gebis! (or give URL)**
- **Video in 201 McLaughlin, starting day of lecture**

  Mon 9-11AM, 2 - 5 PM; Tue 9 AM - 5 PM;
  Wed 9-11AM, 2 - 10 PM; Thu 9 AM - 6 PM;
  Fri 9 - 5PM, 6 - 10 PM;
- **Reading Assignments for Lectures 3 to 7**
  - **Computer Architecture: AQA, Chapter 4, Appendix B**
- **Exercises for Lectures 3 to 7**
  - **Due Thursday Febuary 12 at 5PM homework box in 283 Soda (building is locked at 6:45 PM)**
  - **4.2, 4.10, 4.19, B.2**
  - **4.14 parts c) and d) only**
  - **Done in pairs, but both need to understand whole assignment**
  - **Study groups encouraged, but pairs do own work**

Dynamic Scheduling in PowerPC 604 and Pentium Pro

- **Both In-order Issue, Out-of-order execution, In-order Commit**



Pentium Pro more like a scoreboard since central control vs. distributed

Dynamic Scheduling in PowerPC 604 and Pentium Pro

Parameter	PPC	PPro
Max. instructions issued/clock	4	3
Max. instr. complete exec./clock	6	5
Max. instr. commited/clock	6	3
Window (Instrs in reorder buffer)	16	40
Number of reservations stations	12	20
Number of rename registers	8int/12FP	40
No. integer functional units (FUs)	2	2
No. floating point FUs	1	1
No. branch FUs	1	1
No. complex integer FUs	1	0
No. memory FUs	1	1 load +1 store

Q: How pipeline 1 to 17 byte x86 instructions?

Dynamic Scheduling in Pentium Pro

- PPro doesn’t pipeline 80x86 instructions
- PPro decode unit translates the Intel instructions into 72-bit micro-operations ($\approx$ DLX)
- Sends micro-operations to reorder buffer & reservation stations
- Takes 1 clock cycle to determine length of 80x86 instructions + 2 more to create the micro-operations
- 12-14 clocks in total pipeline ($\approx$ 3 state machines)
- Many instructions translate to 1 to 4 micro-operations
- Complex 80x86 instructions are executed by a conventional microprogram (8K x 72 bits) that issues long sequences of micro-operations

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- Two variations
- Superscalar: varying no. instructions/cycle (1 to 8), scheduled by compiler or by HW (Tomasulo)
  - IBM PowerPC, Sun UltraSparc, DEC Alpha, HP 8000
- (Very) Long Instruction Words (V)LIW: fixed number of instructions (4-16) scheduled by the compiler; put ops into wide templates
  - Joint HP/Intel agreement in 1999/2000?
  - Intel Architecture-64 (IA-64) 64-bit address
  - Style: "Explicitly Parallel Instruction Computer (EPIC)"
- Anticipated success lead to use of Instructions Per Clock cycle (IPC) vs. CPI

Getting CPI < 1: Issuing Multiple Instructions/Cycle

- **Superscalar DLX: 2 instructions, 1 FP & 1 anything else**
  - **Fetch 64-bits/clock cycle; Int on left, FP on right**
  - **Can only issue 2nd instruction if 1st instruction issues**
  - **More ports for FP registers to do FP load & FP op in a pair**

Type	***PipeStages***						
Int. instruction	**IF**	**ID**	**EX**	**MEM**	**WB**		
FP instruction	**IF**	**ID**	**EX**	**MEM**	**WB**		
Int. instruction		**IF**	**ID**	**EX**	**MEM**	**WB**	
FP instruction		**IF**	**ID**	**EX**	**MEM**	**WB**	
Int. instruction			**IF**	**ID**	**EX**	**MEM**	**WB**
FP instruction			**IF**	**ID**	**EX**	**MEM**	**WB**

- **1 cycle load delay expands to 3 instructions in SS**
  - **instruction in right half can’t use it, nor instructions in next slot**

Review: Unrolled Loop that Minimizes Stalls for Scalar

LD to ADDD: 1 Cycle
ADDD to SD: 2 Cycles

```
1 Loop: LD     F0,0(R1)
2       LD     F6,-8(R1)
3       LD     F10,-16(R1)
4       LD     F14,-24(R1)
5       ADDD   F4,F0,F2
6       ADDD   F8,F6,F2
7       ADDD   F12,F10,F2
8       ADDD   F16,F14,F2
9       SD     0(R1),F4
10      SD     -8(R1),F8
11      SD     -16(R1),F12
12      SUBI   R1,R1,#32
13      BNEZ   R1,LOOP
14      SD     8(R1),F16     ; 8-32 = -24
```

14 clock cycles, or 3.5 per iteration

Loop Unrolling in Superscalar

	Integer instruction	*FP instruction*	*Clock cycle*
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 (1.5X)**

Multiple Issue Challenges

- **While Integer/FP split is simple for the HW, get CPI of 0.5 only for programs with:**
  - **Exactly 50% FP operations**
  - **No hazards**
- **If more instructions issue at same time, greater difficulty of decode and issue**
  - **Even 2-scalar => examine 2 opcodes, 6 register specifiers, & decide if 1 or 2 instructions can issue**
- **VLIW: tradeoff instruction space for simple decoding**
  - **The long instruction word has room for many operations**
  - **By definition, all the operations the compiler puts in the long instruction word are independent => execute in parallel**
  - **E.g., 2 integer operations, 2 FP ops, 2 Memory refs, 1 branch**
    - » **16 to 24 bits per field => 7*16 or 112 bits to 7*24 or 168 bits wide**
  - **Need compiling technique that schedules across several branches**

Loop Unrolling in VLIW

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP op. 2*	*Int. op/ branch*	*Clock*
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

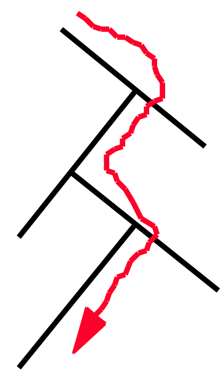
7 results in 9 clocks, or 1.3 clocks per iteration (1.8X)

Average: 2.5 ops per clock, 50% efficiency

Note: Need more registers in VLIW (15 vs. 6 in SS)

Trace Scheduling

- **Parallelism across IF branches vs. LOOP branches**
- **Two steps:**
  - *Trace Selection*
    - » **Find likely sequence of basic blocks (*trace*) of (statically predicted or profile predicted) long sequence of straight-line code**



  - *Trace Compaction*
    - » **Squeeze trace into few VLIW instructions**
    - » **Need bookkeeping code in case prediction is wrong**
- **Compiler undoes bad guess (discards values in registers)**
- **Subtle compiler bugs mean wrong answer vs. pooer performance; no hardware interlocks**

Advantages of HW (Tomasulo) vs. SW (VLIW) Speculation

- **HW determines address conflicts**
- **HW better branch prediction**
- **HW maintains precise exception model**
- **HW does not execute bookkeeping instructions**
- **Works across multiple implementations**
- **SW speculation is much easier for HW design**

Superscalar v. VLIW

- Smaller code size
- Binary compatability across generations of hardware

- Simplified Hardware for decoding, issuing instructions
- No Interlock Hardware (compiler checks?)
- More registers, but simplified Hardware for Register Ports (multiple independent register files?)

Intel/HP "Explicitly Parallel Instruction Computer (EPIC)"

- **3 Instructions in 128 bit "groups"; field determines if instructions dependent or independent**
  - **Smaller code size than old VLIW, larger than x86/RISC**
  - **Groups can be linked to show independence > 3 instr**
- **64 integer registers + 64 floating point registers**
  - **Not separate filesper funcitonal unit as in old VLIW**
- **Hardware checks dependencies (interlocks => binary compatibility over time)**
- **Predicated execution (select 1 out of 64 1-bit flags) => 40% fewer mispredictions?**
- **IA-64 : name of instruction set architecture; EPIC is type**
- **Merced is name of first implementation (1999/2000?)**
- **LIW = EPIC?**

Dynamic Scheduling in Superscalar

- **Dependencies stop instruction issue**
- **Code compiler for old version will run poorly on newest version**
  - **May want code to vary depending on how superscalar**

Dynamic Scheduling in Superscalar

- **How to issue two instructions and keep in-order instruction issue for Tomasulo?**
  - **Assume 1 integer + 1 floating point**
  - **1 Tomasulo control for integer, 1 for floating point**
- **Issue 2X Clock Rate, so that issue remains in order**
- **Only FP loads might cause dependency between integer and FP issue:**
  - **Replace load reservation station with a load queue; operands must be read in the order they are fetched**
  - **Load checks addresses in Store Queue to avoid RAW violation**
  - **Store checks addresses in Load Queue to avoid WAR,WAW**
  - **Called "decoupled architecture"**

Performance of Dynamic SS

Iteration no.	*Instructions*	*Issues*	*Executes*	*Writes result*
		clock-cycle number		
1	LD F0,0(R1)	1	2	4
1	ADDD F4,F0,F2	1	5	8
1	SD 0(R1),F4	2	9	
1	SUBI R1,R1,#8	3	4	5
1	BNEZ R1,LOOP	4	5	
2	LD F0,0(R1)	5	6	8
2	ADDD F4,F0,F2	5	9	12
2	SD 0(R1),F4	6	13	
2	SUBI R1,R1,#8	7	8	9
2	BNEZ R1,LOOP	8	9	

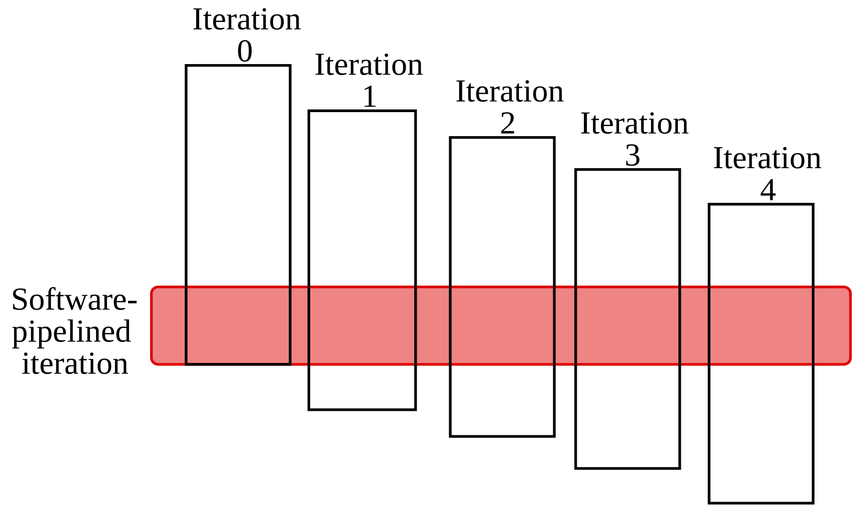
≈ 4 clocks per iteration; only 1 FP instr/iteration

Branches, Decrements issues still take 1 clock cycle

How get more performance?

Software Pipelining

- **Observation: if iterations from loops are independent, then can get more ILP by taking instructions from <u>different</u> iterations**
- **Software pipelining: reorganizes loops so that each iteration is made from instructions chosen from different iterations of the original loop (≈ Tomasulo in SW)**



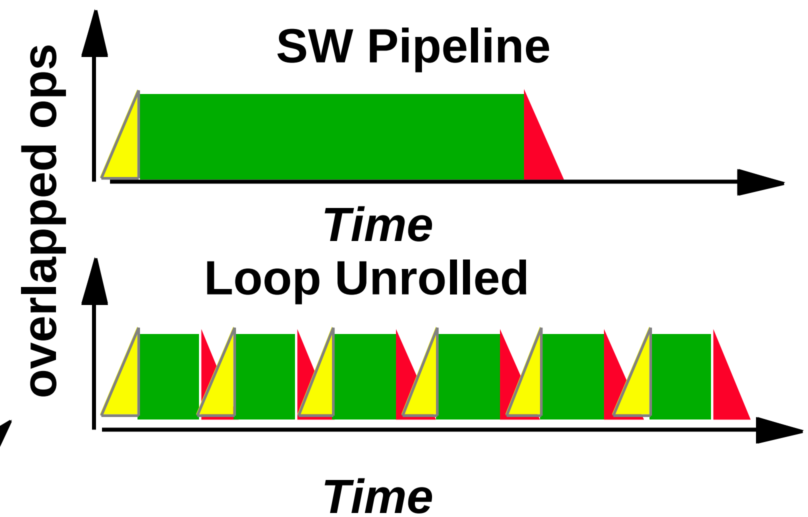
Software Pipelining Example

Before: Unrolled 3 times

```
LD      F0,0(R1)
ADDD    F4,F0,F2
SD      0(R1),F4
LD      F6,-8(R1)
ADDD    F8,F6,F2
SD      -8(R1),F8
LD      F10,-16(R1)
ADDD    F12,F10,F2
SD      -16(R1),F12
SUBI    R1,R1,#24
BNEZ    R1,LOOP
```

After: Software Pipelined

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



- **Symbolic Loop Unrolling**
  - Maximize result-use distance
  - Less code space than unrolling
  - Fill & drain pipe only once per loop vs. once per each unrolled iteration in loop unrolling

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, e.g. 5 x 4 = 15–20 independent instructions?
- Difficulties in building HW
  - Easy: More instruction bandwidth
  - Easy: Duplicate FUs to get parallel execution
  - Hard: Increase ports to Register File (bandwidth)
    - » VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Harder: Increase ports to memory (bandwidth)
  - Decoding Superscalar and impact on clock rate, pipeline depth?

Limits to Multi-Issue Machines

- Limitations specific to either Superscalar or VLIW implementation
  - Decode issue in Superscalar: how wide practical?
  - VLIW code size: unroll loops + wasted fields in VLIW
    - IA-64 compresses dependent instructions, but still larger
  - VLIW lock step => 1 hazard & all instructions stall
    - IA-64 not lock step? Dynamic pipeline?
  - VLIW & binary compatibility is practical weakness as vary number FU and latencies over time
    - IA-64 promises binary compatibility

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**
- **How much ILP is available using existing mechanims with increasing HW budgets?**
- **Do we need to invent new HW/SW mechanisms to keep on processor performance curve?**

Limits to ILP

Initial HW Model here; MIPS compilers.

Assumptions for ideal/perfect machine to start:

1. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided

2. *Branch prediction*–perfect; no mispredictions

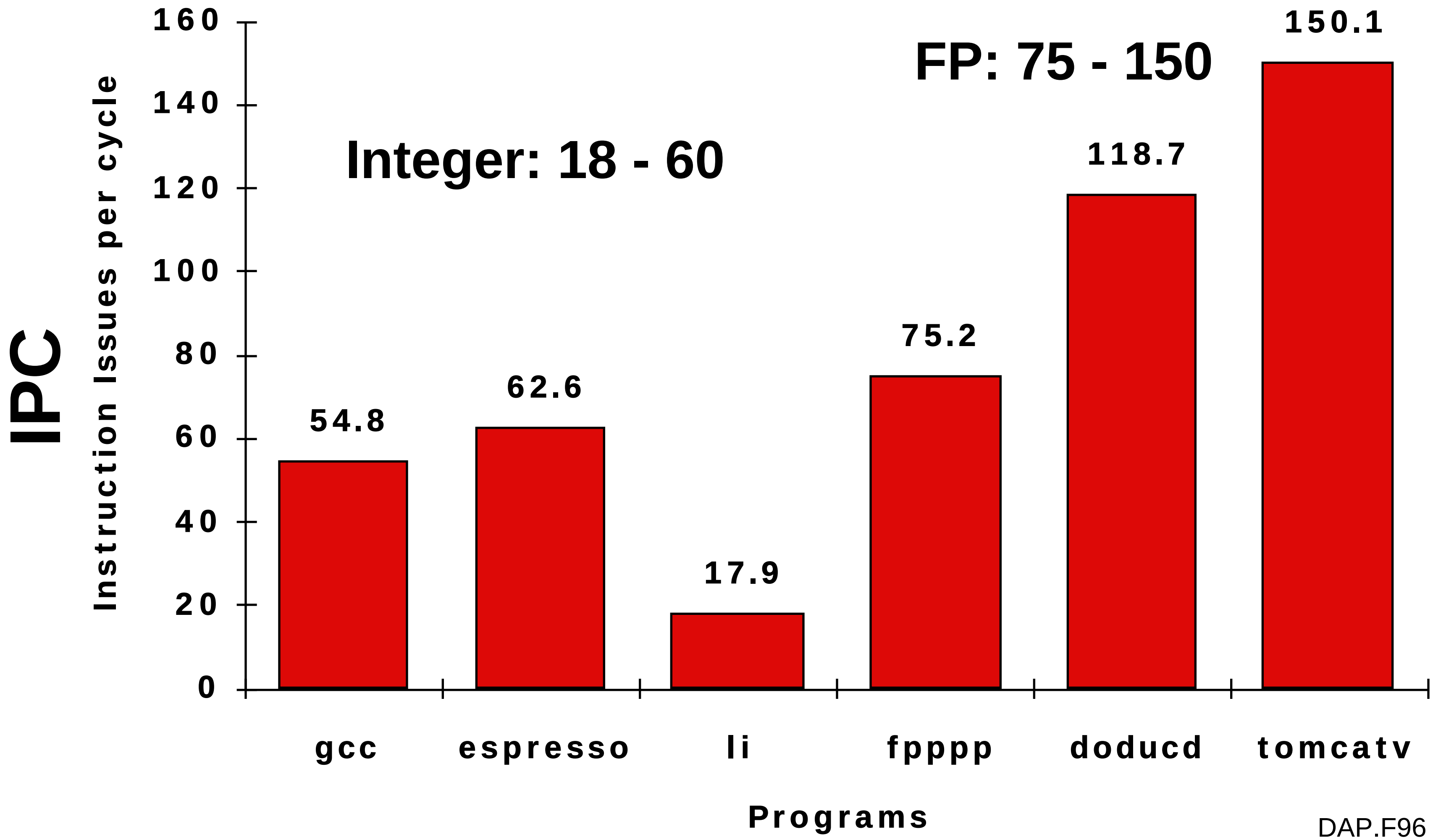
3. *Jump prediction*–all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available

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; unlimited number of instructions issued per clock cycle

Upper Limit to ILP: Ideal Machine

(Figure 4.38, page 319)



More Realistic HW: Branch Impact

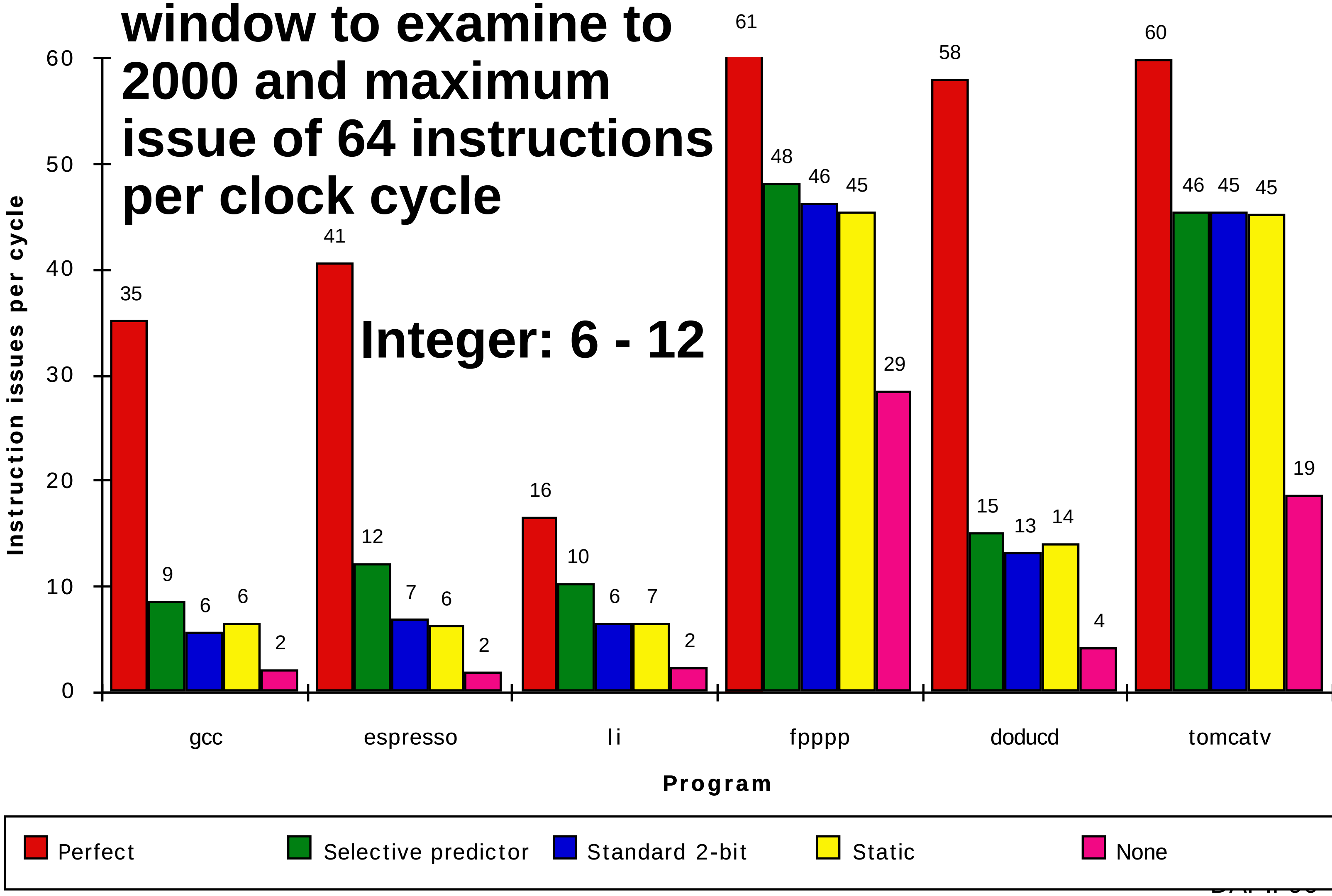
Figure 4.40, Page 323

Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle

FP: 15 - 45

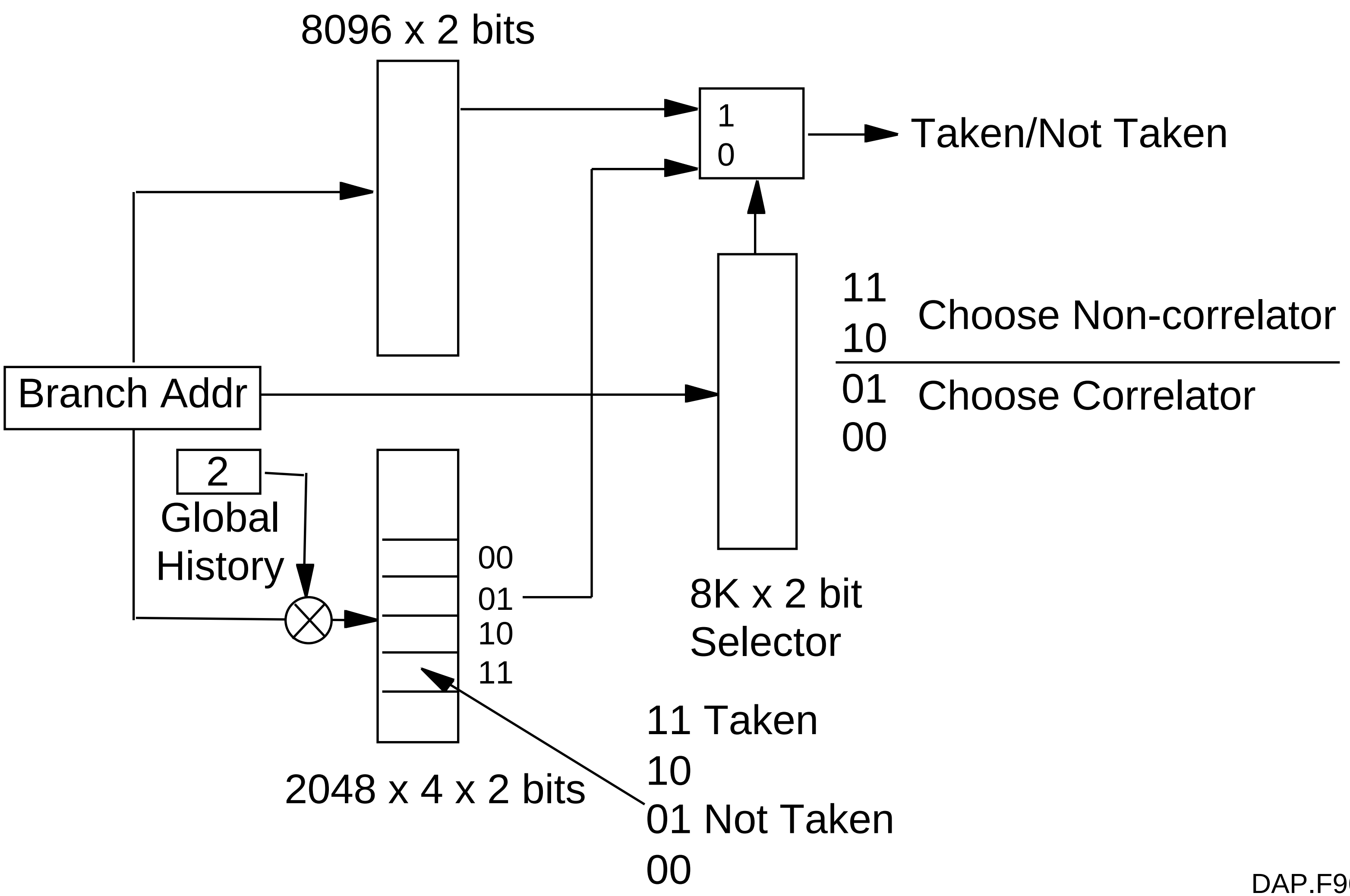
Integer: 6 - 12

IPC



Perfect **Pick Cor. or BHT** **BHT (512)** **Profile** **No prediction**

Selective History Predictor



More Realistic HW: Register Impact

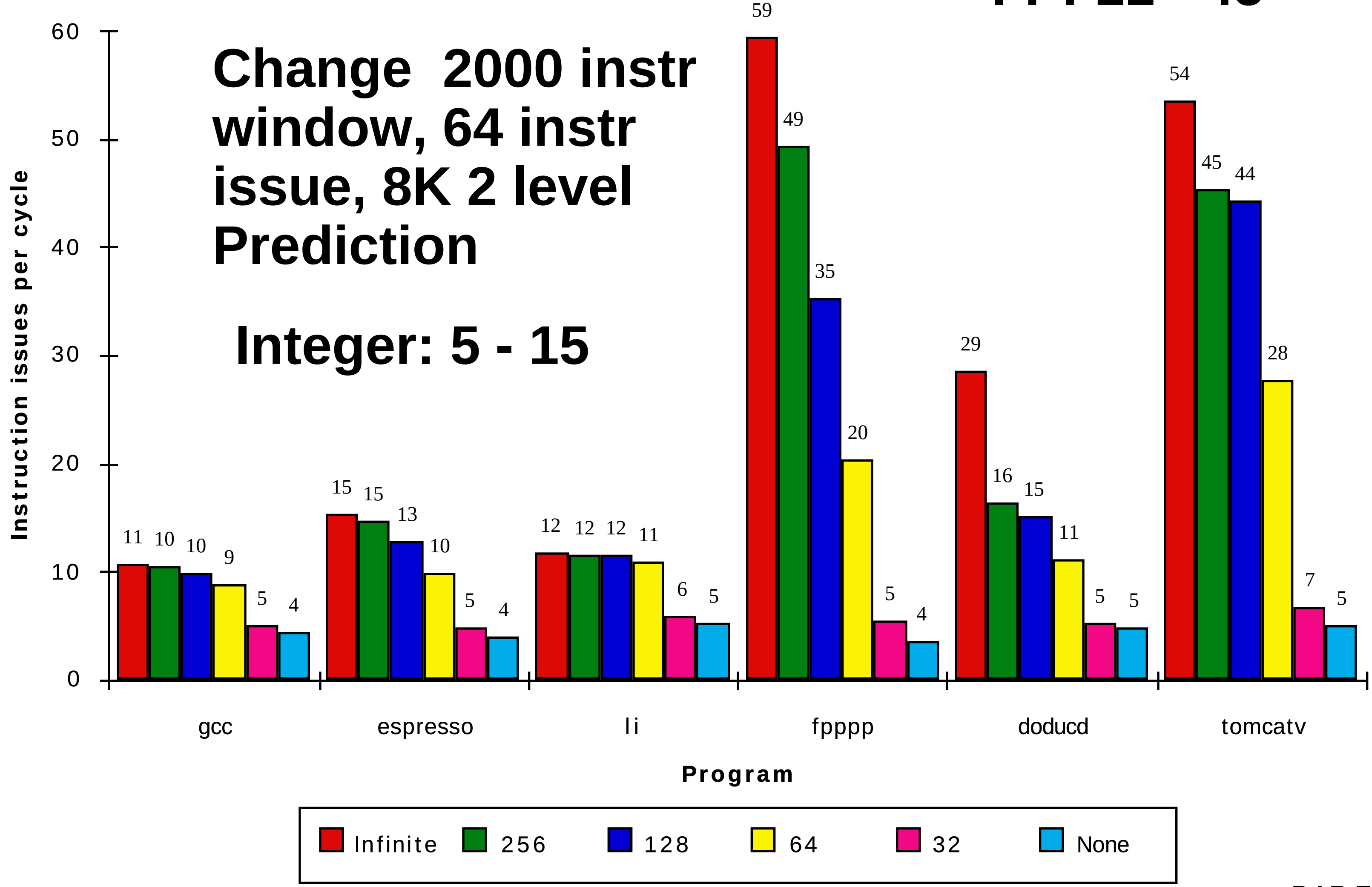
Figure 4.44, Page 328

FP: 11 - 45

Change 2000 instr window, 64 instr issue, 8K 2 level Prediction

Integer: 5 - 15

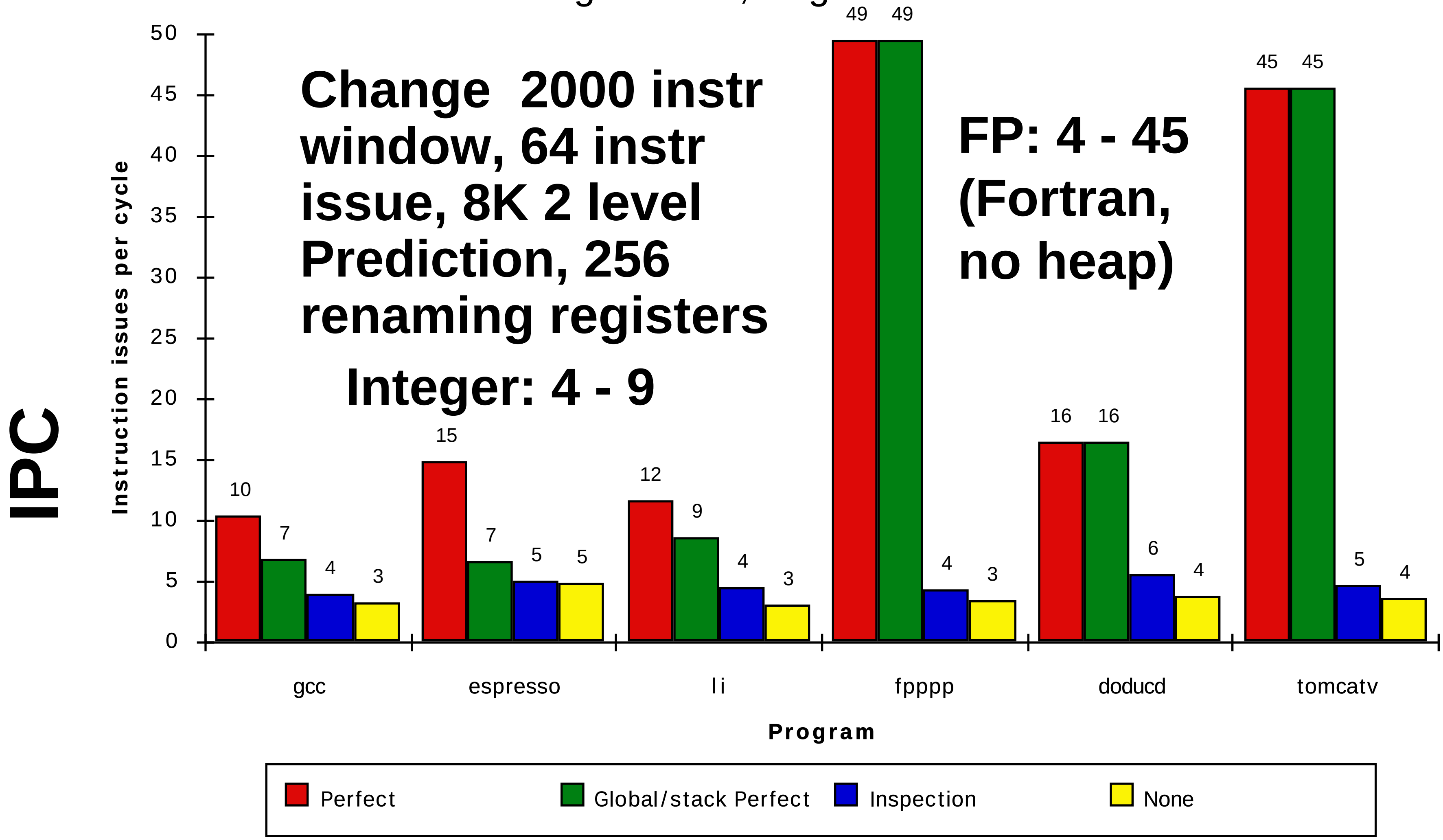
IPC



Infinite 256 128 64 32 None

More Realistic HW: Alias Impact

Figure 4.46, Page 330



Perfect

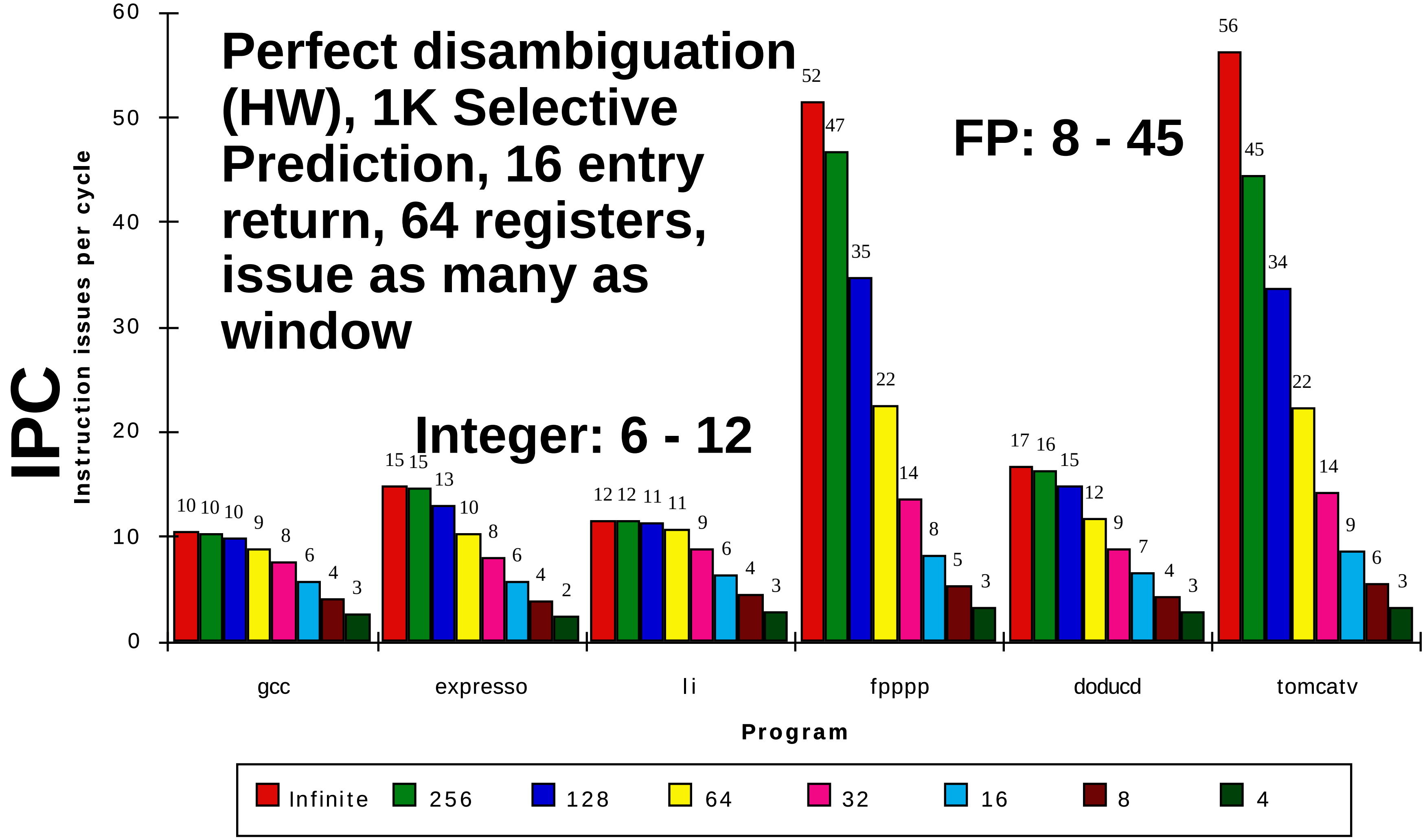
Global/Stack perf; heap conflicts

Inspec. Assem.

None

Realistic HW for '9X: Window Impact

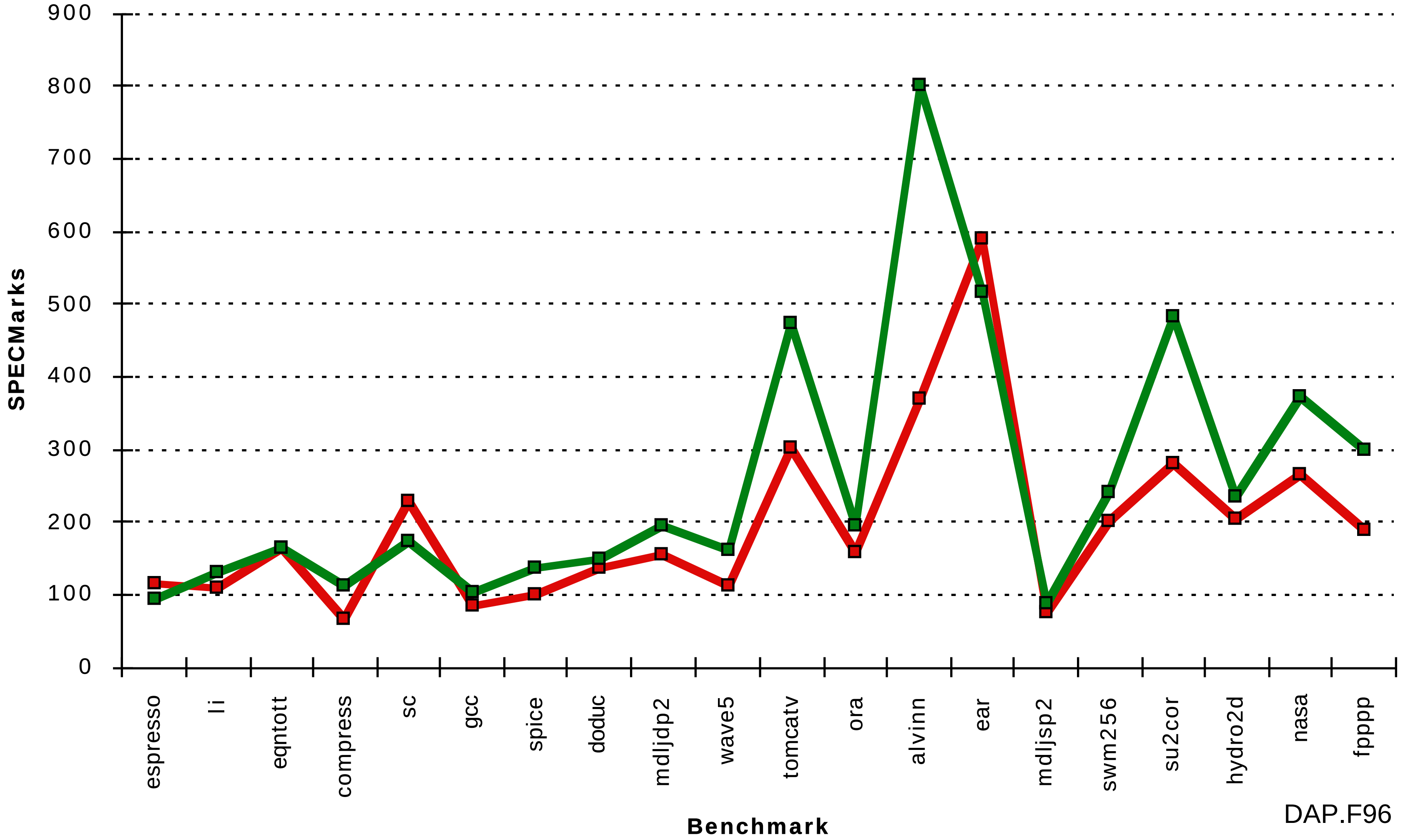
(Figure 4.48, Page 332)



Infinite 256 128 64 32 16 8 4

Braniac vs. Speed Demon(1993)

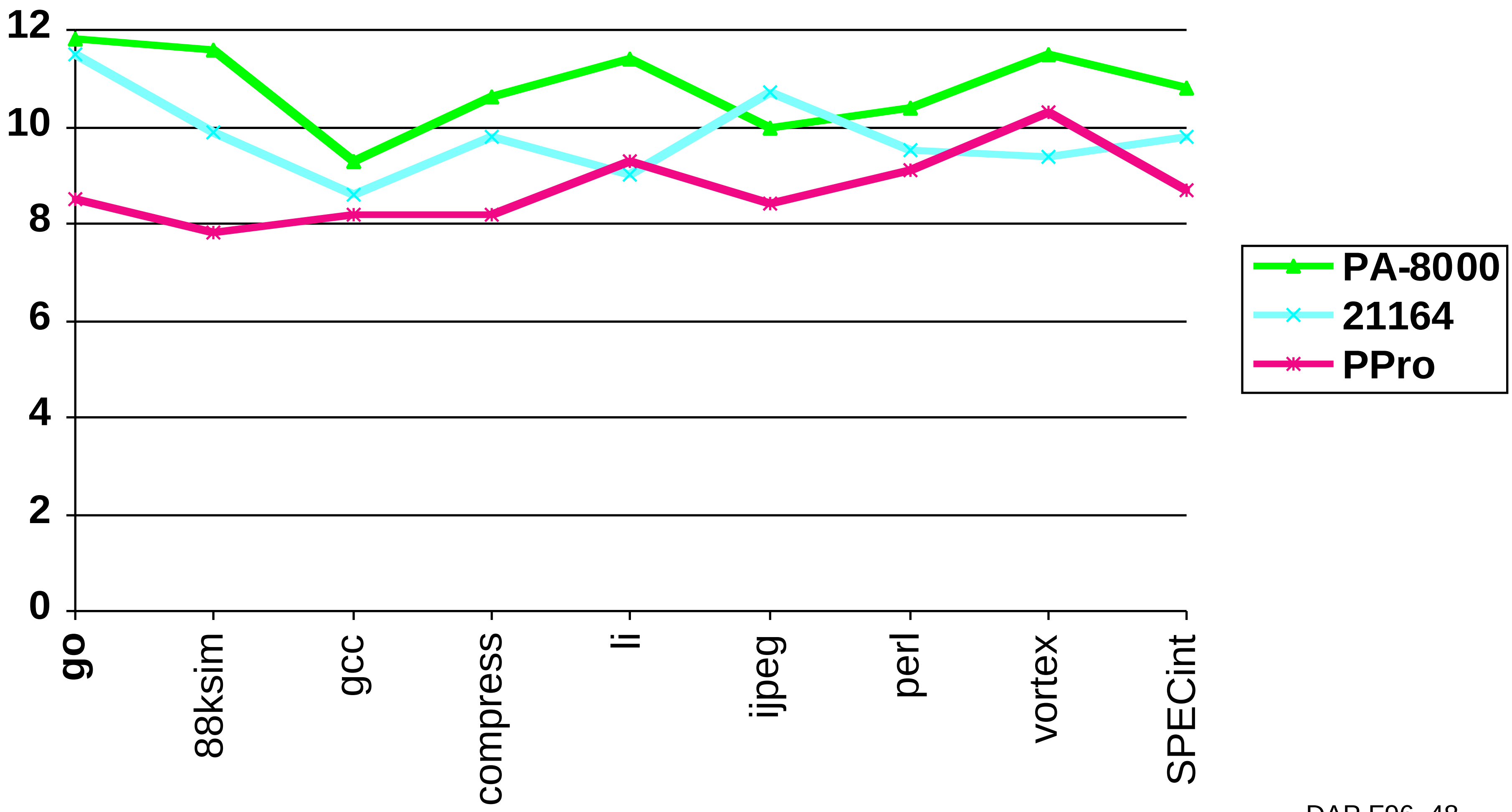
- 8-scalar IBM Power-2 @ 71.5 MHz (5 stage pipe) vs. 2-scalar Alpha @ 200 MHz (7 stage pipe)



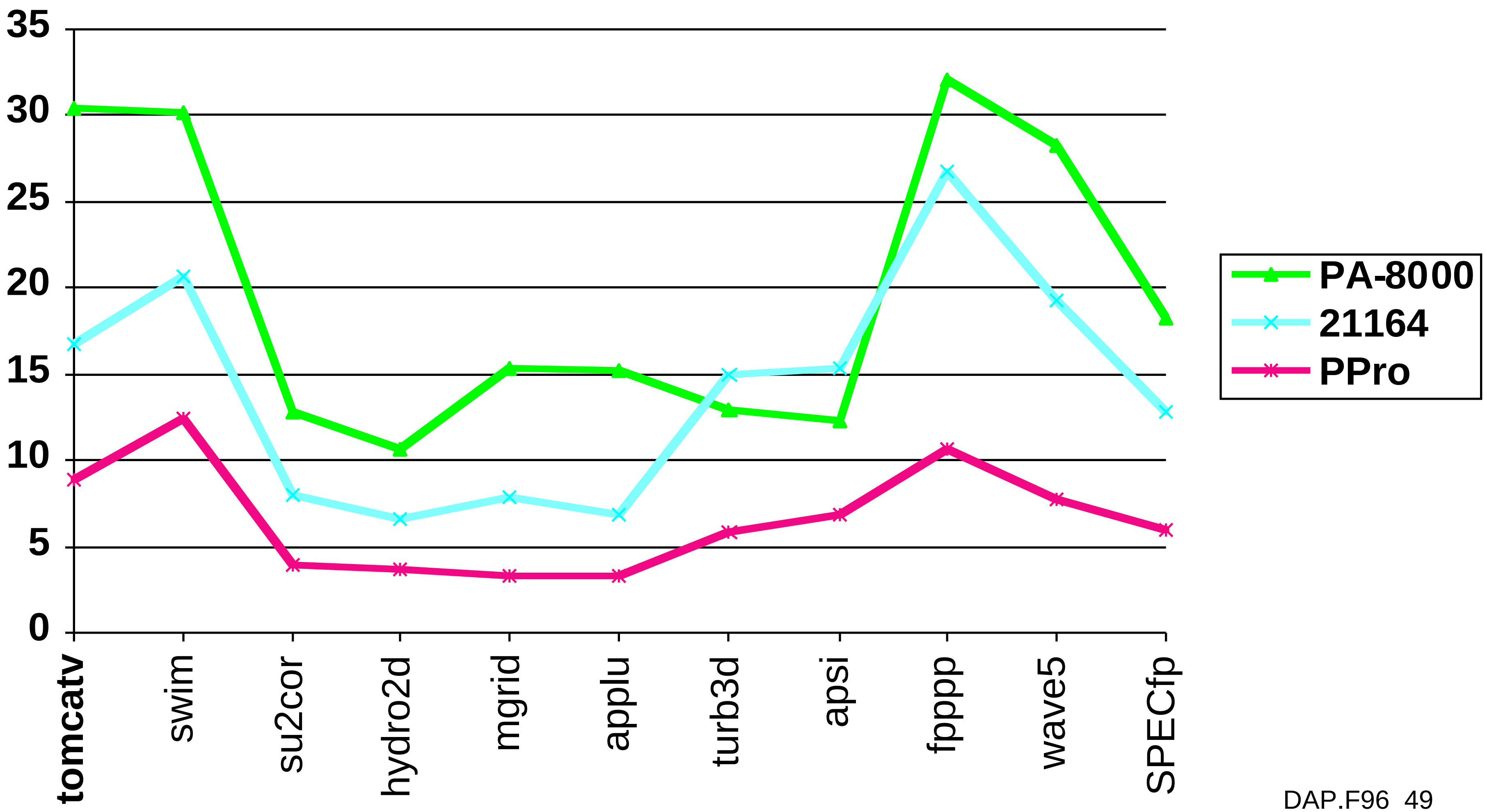
3 1996 Era Machines

	Alpha 21164	PPro	HP PA-8000
Year	1995	1995	1996
Clock	400 MHz	200 MHz	180 MHz
Cache	8K/8K/96K/2M	8K/8K/0.5M	0/0/2M
Issue rate	2int+2FP	3 instr (x86)	4 instr
Pipe stages	7-9	12-14	7-9
Out-of-Order	6 loads	40 instr (µop)	56 instr
Rename regs	none	40	56

SPECint95base Performance (July 1996)



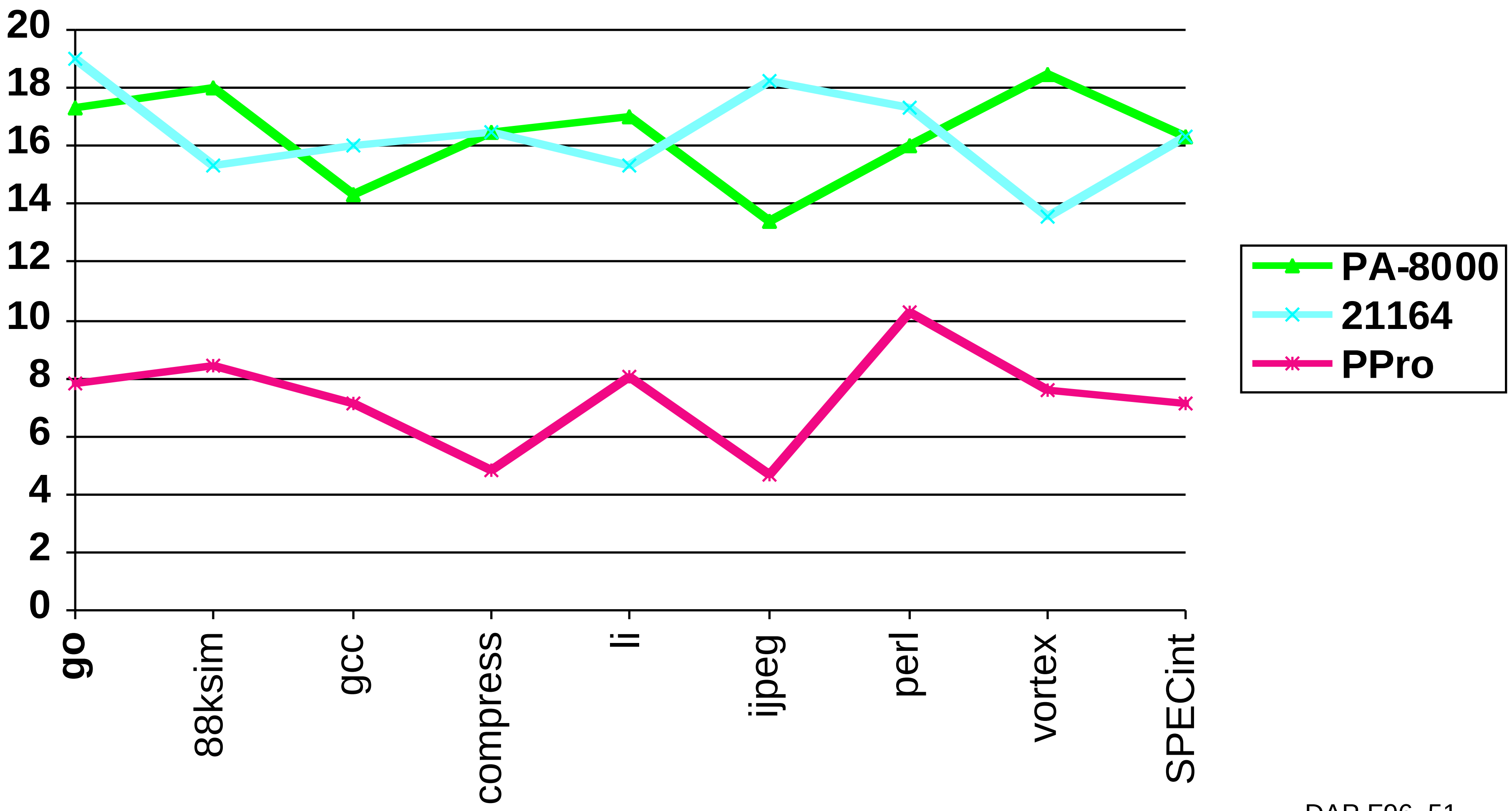
SPECfp95base Performance (July 1996)



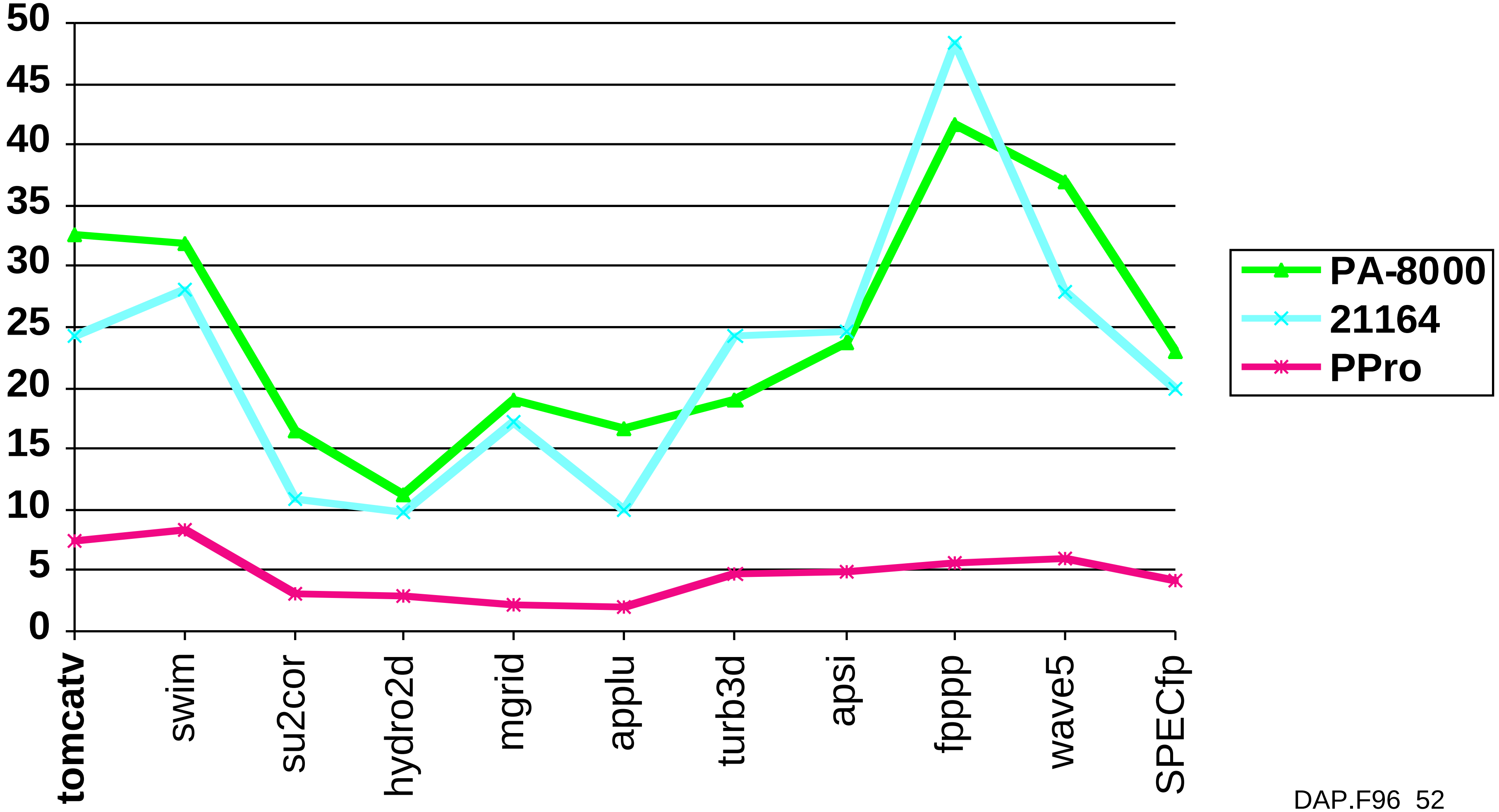
3 1997 Era Machines

	Alpha 21164	Pentium II	HP PA-8000
Year	1995	1996	1996
Clock	600 MHz ('97)	300 MHz ('97)	236 MHz ('97)
Cache	8K/8K/96K/2M	16K/16K/0.5M	0/0/4M
Issue rate	2int+2FP	3 instr (x86)	4 instr
Pipe stages	7-9	12-14	7-9
Out-of-Order	6 loads	40 instr (µop)	56 instr
Rename regs	none	40	56

SPECint95base Performance (Oct. 1997)



SPECfp95base Performance (Oct. 1997)



Summary

- **Branch Prediction**
  - Branch History Table: 2 bits for loop accuracy
  - Recently executed branches correlated with next branch?
  - Branch Target Buffer: include branch address & prediction
  - Predicated Execution can reduce number of branches, number of mispredicted branches
- **Speculation: Out-of-order execution, In-order commit (reorder buffer)**
- **SW Pipelining**
  - Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead
- **Superscalar and VLIW: CPI < 1 (IPC > 1)**
  - Dynamic issue vs. Static issue
  - More instructions issue at same time => larger hazard penalty