CS 152
Computer Architecture and Engineering
Lecture 22: Instruction Level Parallelism

April 14, 1995

Dave Patterson (patterson@cs) and
Shing Kong (shing.kong@eng.sun.com)

Slides available on http://http.cs.berkeley.edu/~patterson

© DAP & SIK 1995

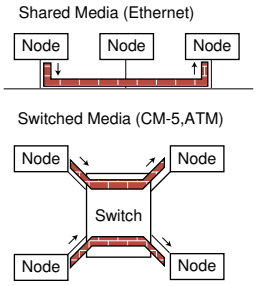
Recap: Interconnection Network Implementation Issues

Interconnect	*MPP*	*LAN*	*WAN*
Example	CM-5	Ethernet	ATM
Maximum length between nodes	25 m	500 m; ≤5 repeaters	copper: 100 m optical: 1000 m
Number data lines	4	1	1
Clock Rate	40 MHz	10 MHz	≥ 155.5 MHz
Shared vs. Switch	Switch	Shared	Switch
Maximum number	2048	254	> 10,000 of nodes
Media Material	Copper	Twisted pair copper wire or Coaxial cable	Twisted pair copper wire or optical fiber

© DAP & SIK 1995

Recap: Implementation Issues

- Advantages of Serial vs. Parallel lines:
  - No synchronizing signals
  - Higher clock rate and longer distance than parallel lines. (e.g., 60 MHz x 256 bits x 0.5 m vs. 155 MHz x 1 bit x 100 m)
    - Imperfections in the copper wires or integrated circuit pad drivers can cause skew in the arrival of signals, limiting the clock rate, and the length and number of the parallel lines.
- Switched vs. Shared Media: pairs communicate at same time: "point-to-point" connections



© DAP & SIK 1995

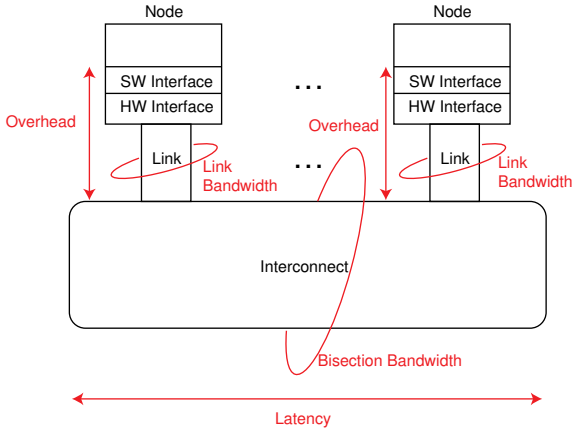
Recap: Other Interconnection Network Issues

Interconnect	*MPP*	*LAN*	*WAN*
Example	CM-5	Ethernet	ATM
Topology	"Fat" tree	Line	Variable, constructed from multistage switches
Connection based?	No	No	Yes
Data Transfer Size	Variable: 4 to 20B	Variable: 0 to 1500B	Fixed: 48B

© DAP & SIK 1995

Recap: Network Performance Measures

- **Overhead: latency of interface vs. Latency: network**



© DAP & SIK 1995

Recap: Interconnection Network Summary

- **Communication between computers**
- **Packets for standards, protocols to cover normal and abnormal events**
- **Implementation issues: length, width, media**
- **Performance issues: overhead, latency, bisection BW**
- **Topologies: many to chose from, but (SW) overheads make them look the alike; cost issues in topologies**

© DAP & SIK 1995

Outline of Today's Lecture

- Recap (5 minutes)
- Introduction to Instruction Level Parallelism (15 minutes)
- Superpipeline, superscalar, VLIW
- Questions and Administrative Matters (5 minutes)
- Register renaming (5 minutes)
- Out-of-order execution(5 minutes)
- Branch Prediction (5 minutes)
- Break (5 minutes)
- Limits to ILP (15 minutes)
- Summary (5 minutes)

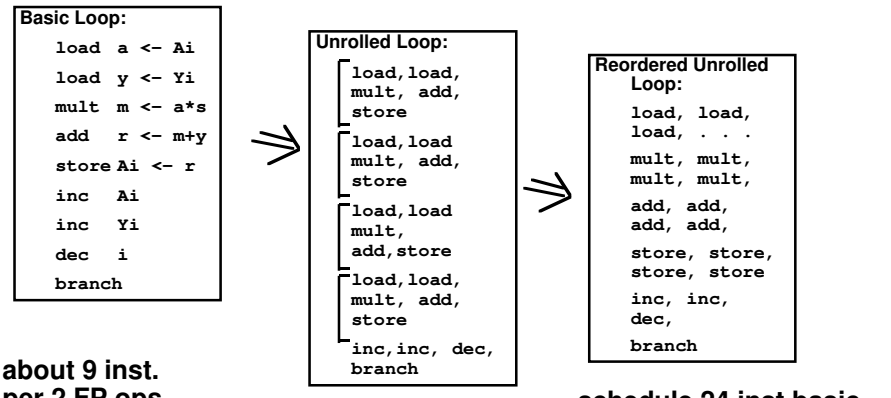
© DAP & SIK 1995

Advanced Pipelining and Instruction Level Parallelism

- gcc 17% control transfer => 5 instructions + 1 branch
  => beyond single block to get more instruction level parallelism
- Loop level parallelism one opportunity, SW and HW

© DAP & SIK 1995

What's going on in the loop



about 9 inst. per 2 FP ops

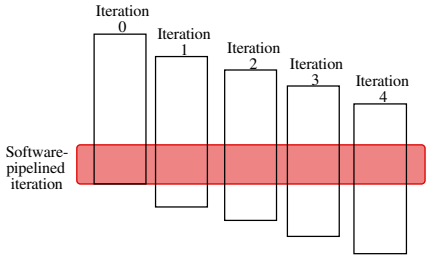
about 6 inst. per 2 FP ops dependencies between instructions remain.

schedule 24 inst basic block relative to pipeline
- delay slots
- function unit stalls
- multiple function units
- pipeline depth

© DAP & SIK 1995

Software Pipelining

- Observation: if iterations from loops are independent, then can get ILP by taking instructions from different iterations
- Software pipelining: reorganizs loops such that each iteration is made from instructions chosen from different iterations of the original loop (≈ Tomasulo in SW)



© DAP & SIK 1995

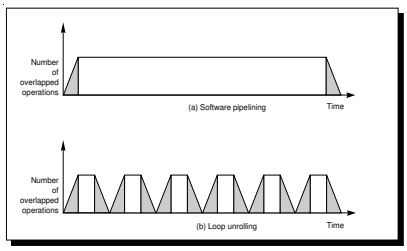
SW Pipelining Example

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

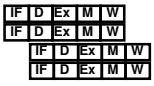
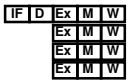
```
1   SD    0(R1),F4 ; Stores M[i]
2   ADDD  F4,F0,F2 ; Adds to M[i-1]
3   LD    F10,-16(R1); loads M[i-2]
4   SUBI  R1,R1,#16
5   BNEZ  R1,LOOP
```



- Symbolic Loop Unrolling
- Less code space
- Overhead paid only once vs. each iteration in loop unrolling

© DAP & SIK 1995

How can the machine exploit available ILP?

Technique		Limitation
° Pipelining		Issue rate, FU stalls, FU depth
° Super-pipeline - Issue 1 instr. / (fast) cycle - IF takes multiple cycles		Clock skew, FU stalls, FU depth
° Super-scalar - Issue multiple scalar instructions per cycle		Hazard resolution
° VLIW - Each instruction specifies multiple scalar operations		Packing

© DAP & SIK 1995

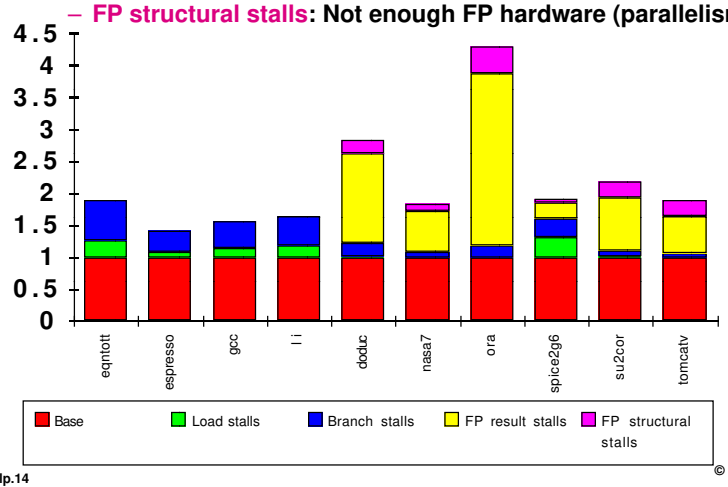
Case Study: MIPS R4000 (100 MHz to 200 MHz)

- 8 stage pipeline:
  - IF–first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS–second half of access to instruction cache.
  - RF–instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX–execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF–data fetch, first half of access to data cache.
  - DS–second half of access to data cache.
  - TC–tag check, determine whether the data cache access hit.
  - WB–write back for loads and register-register operations.
- 8 stages & impact on Load delay? Branch delay? Why?

© DAP & SIK 1995

R4000 Performance

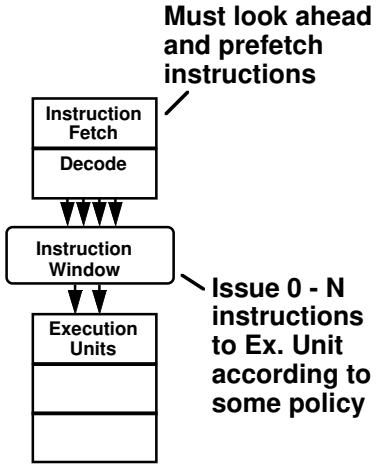
- Not ideal CPI of 1:
  - Load stalls (1 or 2 clock cycles)
  - Branch stalls (2 cycles + unfilled slots)
  - FP result stalls: RAW data hazard (latency)
  - FP structural stalls: Not enough FP hardware (parallelism)



© DAP & SIK 1995

Issues raised by Superscalar execution

- Available parallelism
- Resources and available bandwidth
- Branch prediction
- Hazard detection and (aggressive) resolution
  - out-of-order issue => WAR and WAW
  - register renaming to avoid false dependies
  - out-of-order completion
- Exception handling



© DAP & SIK 1995

Questions and Administrative Matters (5 Minutes)

- Remaining schedule (Guests live in 306 Soda)

12-Apr	Wed	Networks and MP
14-Apr	Fri	Superscalar
19-Apr	Wed	Midterm II 5 PM in Bechtel/Pizza
21-Apr	Fri	SGI T5 Microarchitecture
26-Apr	Wed	DEC Alpha Microarchitecture
28-Apr	Fri	UltraSPARC Microarchitecture
3-May	Wed	Project Demo Meetings
5-May	Fri	Class Summary
8-May	Mon	Projects Due 4PM in 634

© DAP & SIK 1995

Hardware Schemes for Instruction Parallelism

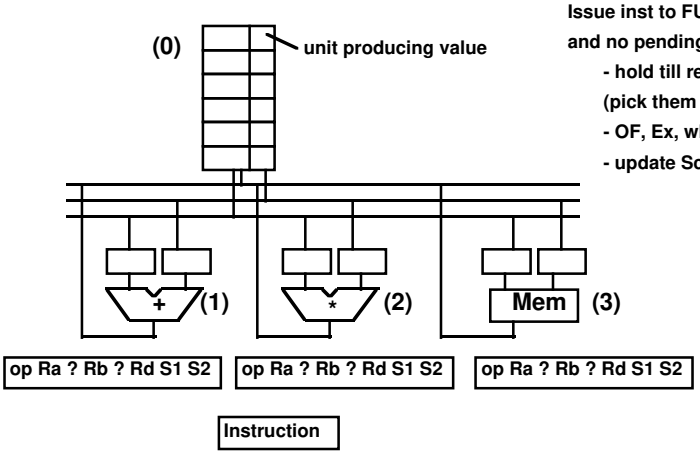
- Why in HW at run time?
  - Works when can't know dependence at run time
  - compiler simpler
  - code for one machine runs well on another
- Key idea: Allow instructions behind stall to proceed

```
DIVD   F0,F2,F4
ADDD   F10,F0,F8
SUBD   F8,F8,F14
```

- enables out-of-order execution => out-of-order completion
- ID stage checked both for structural & data dependencies
- Out-of-order execution divides ID stage:
  1. Issue—decode instructions, check for structural hazards
  2. Read operands—wait until no data hazards, then read operands
- Scoreboards allow instruction to execute whenever 1 & 2 hold, not waiting for prior instructions

© DAP & SIK 1995

Scoreboard (CDC 6600)



Issue inst to FU when free
and no pending updates in dest.
- hold till registers available
(pick them up while waiting)
- OF, Ex, when ready
- update Scoreboard on WB

r1 <- M[r1 + r2]
r2 <- r2 * r3
r4 <- r2 + r5
r2 <- r0

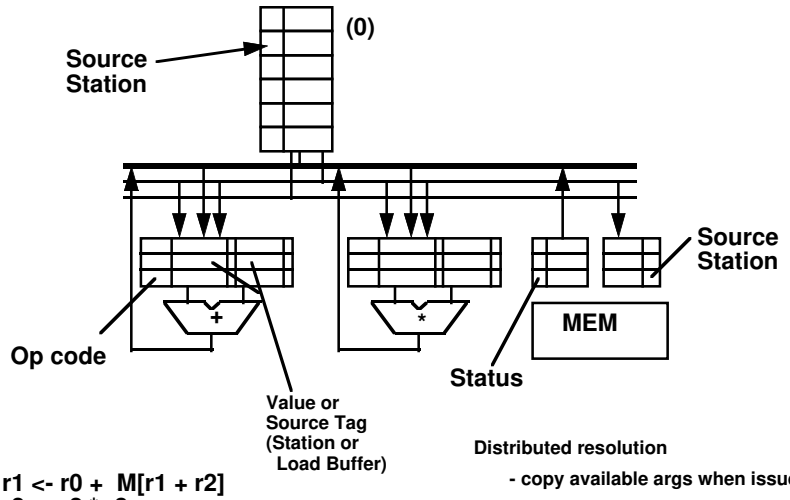
© DAP & SIK 1995

Scoreboard implications

- Out-of-order completion => WAR, WAW hazards?
- Solutions for WAR
  - queue both the operation and copies of its operands
  - read registers only during Read Operands stage
- For WAW, must detect hazard: stall until other completes
- Need to have multiple instructions in execution phase => multiple execution units or pipelined execution units
- Scoreboard keeps track of dependencies, state or operations
- Scoreboard replaces ID, EX, WB with 4 stages: Issue/ID, Read Operands, EX, WB

© DAP & SIK 1995

Tomosulo




```
r1 <- r0 +  M[r1 + r2]
r2 <- r2 * r3
r4 <- r2 + r5
r2 <- r0
```

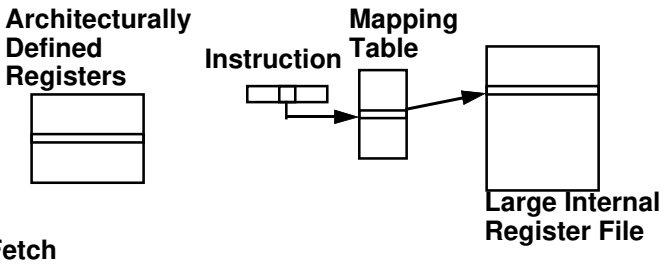
Distributed resolution

- copy available args when issued
- forward pending ops directly from FUs

© DAP & SIK 1995

Register Renaming

With a large register set, compiler can rename to eliminate WAR
- sometimes requires moves
- HW can do it on the fly (but it can't look at the rest of the program)



Operand Fetch
All source registers renamed through the map

On issue:
Assign new pseudo register for the destination
Update the map
- applies to all following instructions unti the next store

© DAP & SIK 1995

Exceptions and Out-of-order Completion

OOC important when FU (including memory) takes many cycles
- allow independent instructions to flow through other FUs

```
L1:   r1 <- (r2 + A)
      r3 <- (r2 + B)
      r4 <- r1 +F r3
      r2 <- r2 + 8
      r5 <- r5 - 1
      (r2 + C) < r4
      BNZ r5, l1
```

MIPS solution:
- 3 independent destinations: Int Reg, HI/LO, FP reg
- Check for possible exceptions before any following inst. modify state (at WB)
  Stall if exception is possible
- Moves from one register space explicit

© DAP & SIK 1995

HW support for More ILP

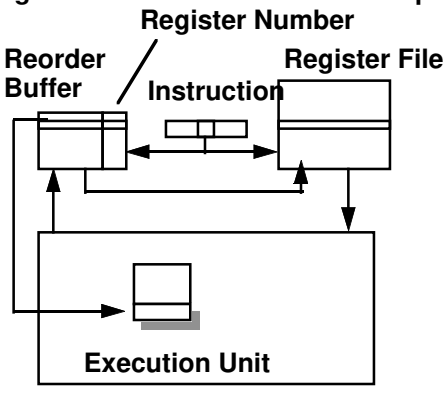
- *Speculation*: allow instruction to issue that is dependent on branch being taken without any consequences (including exceptions) if branch is not taken ("HW undo")
- Often try to combine with dynamic scheduling
- Tomasulo: separate *speculative* bypassing of results from real bypassing of results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

© DAP & SIK 1995

Reorder Buffers

Keep track of pending updates to register
- in parallel with register file access,
  do (prioritized) associative lookup in reorder buffer
- hit says register file is old,
  - reorder buffer provides new value
  - RB gives FU that new value should be bypassed from.

Updates go to reorder buffer
- retired to register file when instruction completes (e.g., in order)



© DAP & SIK 1995

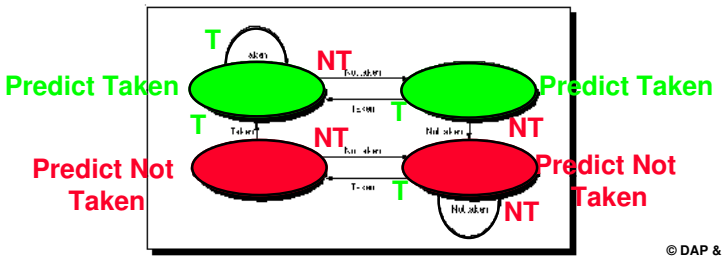
Review:Tomasulo Summary

- Registers not the bottleneck
- Avoids the WAR, WAW hazards of Scoreboard
- Not limited to basic blocks (provided branch prediction)
- Allows loop unrolling in HW
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- Next stop: More branch prediction

© DAP & SIK 1995

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
- Problem: in a loop, 1-bit BHT will cause 2 mispredictions:
  1) end of loop case, when it exits instead of looping as before
  2) first time through loop on ***next*** time through code, when it predicts exit instead of looping
- Solution: 2-bit scheme where change prediction only if get misprediction ***twice:*** (Figure 5.13, p. 284)



© DAP & SIK 1995

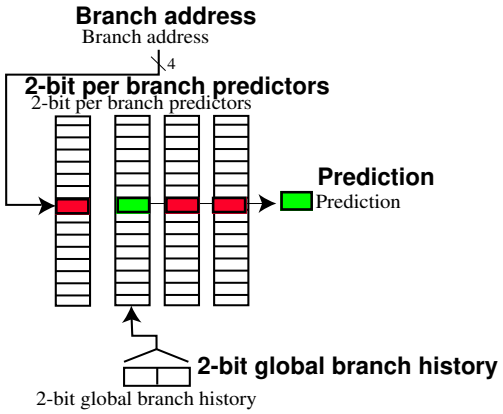
BHT Accuracy

- 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, but 4096 is a lot of HW

© DAP & SIK 1995

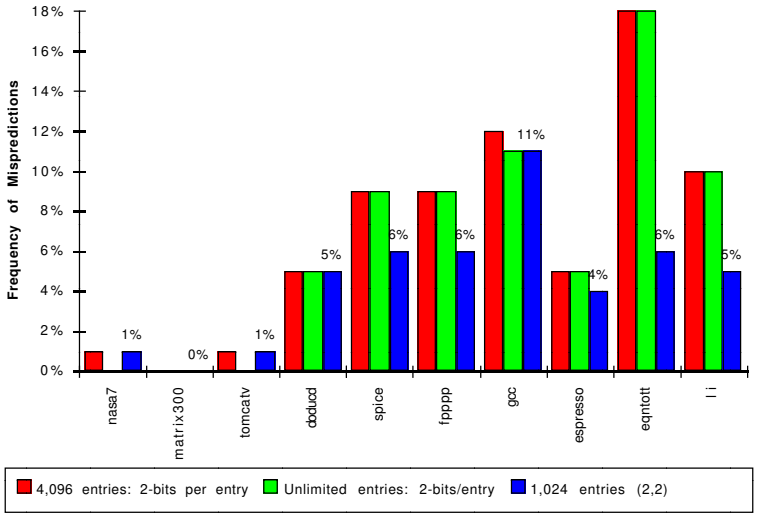
Correlating Branches

- Idea: taken/not taken of a recently executed branches is related to behavior of next branch (as well as the history of that branch behavior)
  - Then behavior of recent branches selects between, say, 4 predictions of next branch, updating just that prediction



© DAP & SIK 1995

Accuracy of Different Schemes: Mispredictions



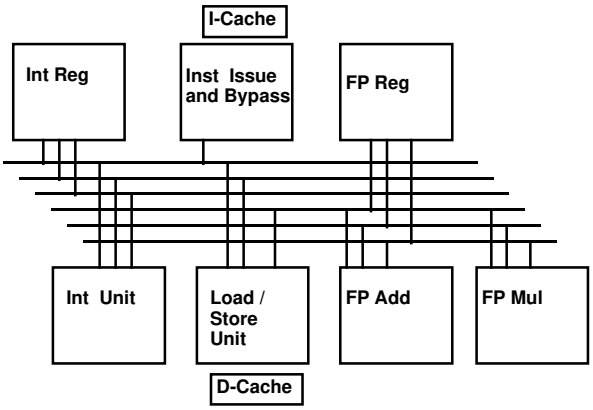
© DAP & SIK 1995

Getting CPI < 1: Issuing Multiple Instr/Cycle

- **2 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 1997 (P86?)?**

© DAP & SIK 1995

Easy Superscalar



Issue integer and FP operations in parallel !

- potential hazards?

- expected speedup?

- what combinations of instructions make sense?

© DAP & SIK 1995

Getting CPI < 1: Issuing Multiple Instr/Cycle

- Superscalar: 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 instruction in SS
  - instruction in right half can't use it, nor instructions in next slot

© DAP & SIK 1995

Unrolled Loop that minimizes stalls for scalar

```
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

© DAP & SIK 1995

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

© DAP & SIK 1995

Break (5 Minutes)

© DAP & SIK 1995

Limits of SuperScalar

- 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 is 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 can 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

© DAP & SIK 1995

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
- Need more registers inVLIW
- What happens with next generation? Will old code work?

© DAP & SIK 1995

Limits to Multi-Issue Machines

- Inherent limitations of ILP
  - 1 branch in => 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
- 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

© DAP & SIK 1995

Exploring Limits to ILP

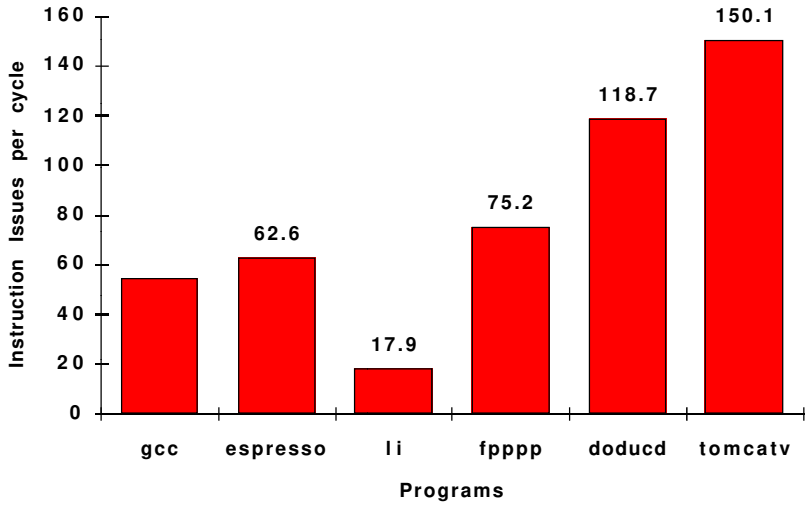
- Conflicting studies of amount of parallelism available in late 1980s and early 1990s. Different assumptions about:
  - Benchmarks (vectorized Fortran FP vs. integer C programs)
  - Hardware sophistication
  - Compiler sophistication
- Initial HW Model here; MIPS compilers

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 ≠

- 1 cycle latency for all instructions

© DAP & SIK 1995

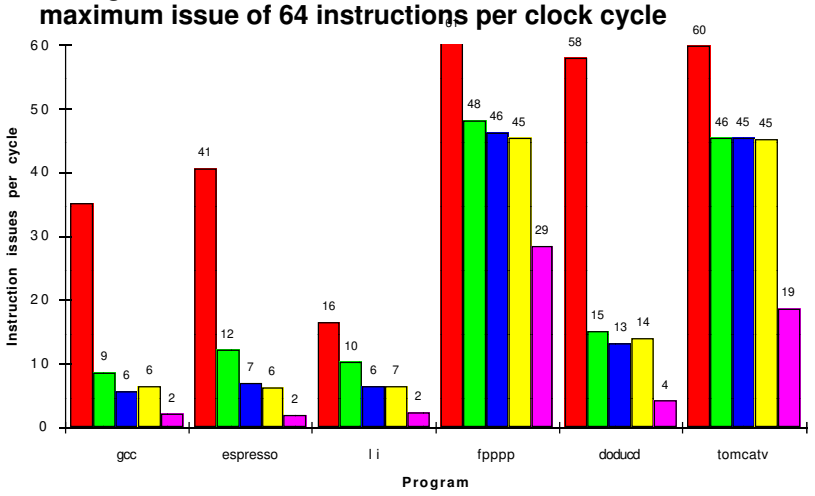
Upper Limit to ILP



© DAP & SIK 1995

More Realistic HW: Branch Impact

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

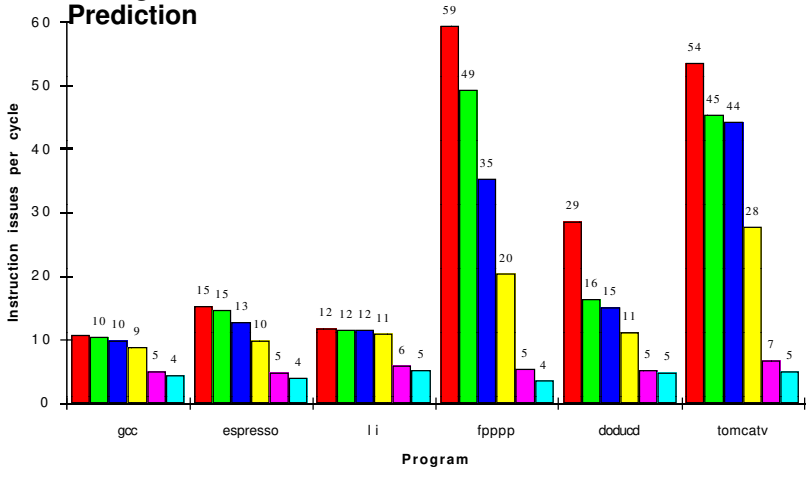


Perfect **Pick Cor. or BHT** **BHT (512)** **Profile**

© DAP & SIK 1995

More Realistic HW: Register Impact

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

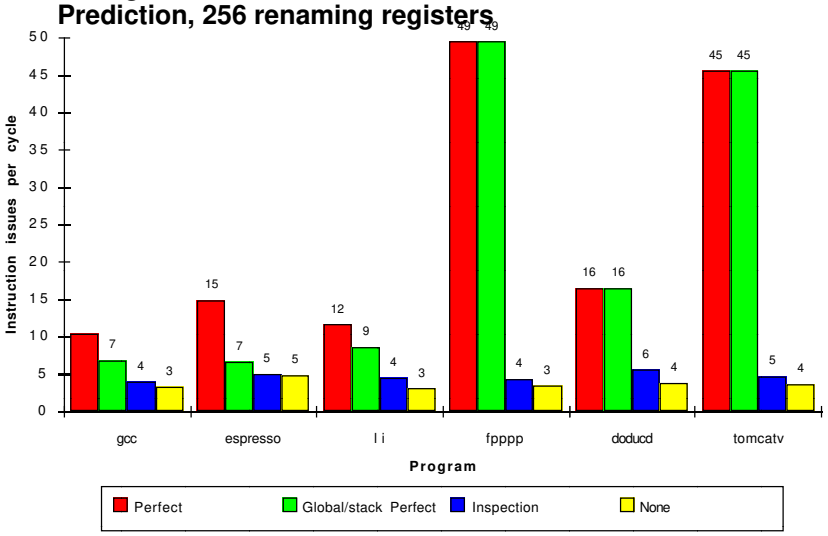


Infinite **256** **128** **64** **32** **None**

© DAP & SIK 1995

More Realistic HW: Alias Impact

- Change 2000 instr window, 64 instr issue, 8K 2level Prediction, 256 renaming registers

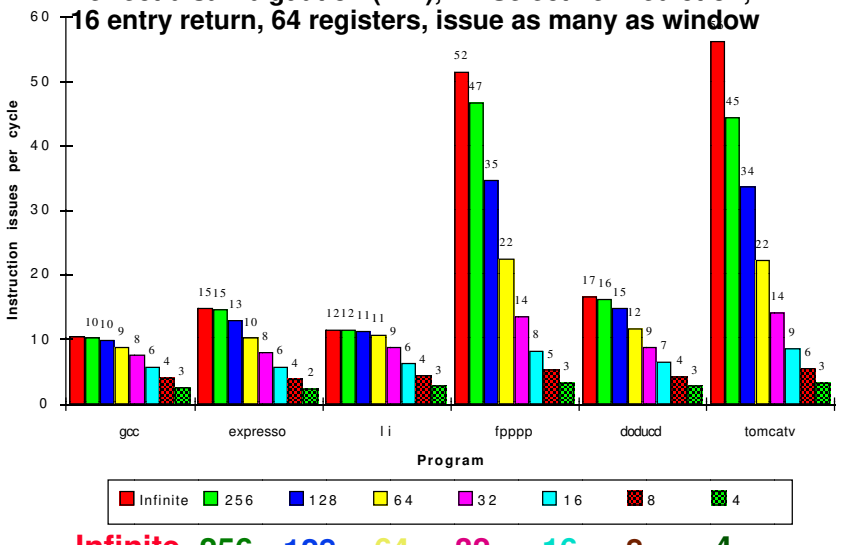


Perfect Global/Stack perf; heap conflicts Inspec. Assem. None

© DAP & SIK 1995

Realistic HW for '9X: Issue Window Impact

- Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window

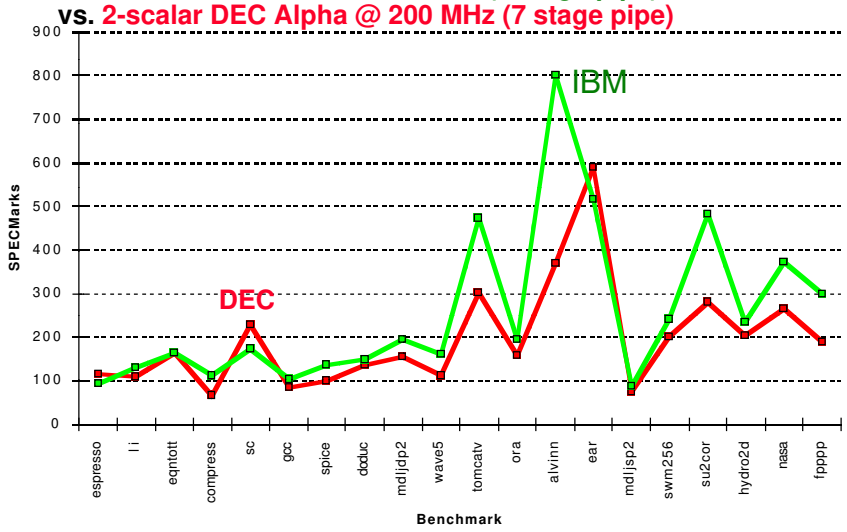


Infinite 256 128 64 32 16 8 4

© DAP & SIK 1995

Braniac vs. Speed Demon (1994)

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



© DAP & SIK 1995

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 make hard for conditional operation

© DAP & SIK 1995

Summary

- Instruction Level Parallelism in SW or HW
- Loop level parallelism is easiest to see
- SW dependencies/Compiler sophistication determine if compiler can unroll loops
- SW Pipelining
  - Symbolic Loop Unrolling to get most from pipeline with little code expansion, little overhead
- HW “unrolling”
  - Scoreboard & Tomasulo=> Register renaming, reorder
- Branch Prediction
  - Branch History Table: 2 bits for loop accuracy
  - Correlation: Recently executed branches correlated with next branch
- SuperScalar and VLIW
  - CPI < 1
  - Dynamic issue vs. Static issue
  - More instructions issue at same time, larger the penalty of hazards
- Future? Stay tuned…

© DAP & SIK 1995

To probe further

- links to corperate home-pages and press releases.

http://infopad.eecs.berkeley.edu/CIC/

© DAP & SIK 1995