Review: Getting CPI < 1: 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 1998?**

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**

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

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

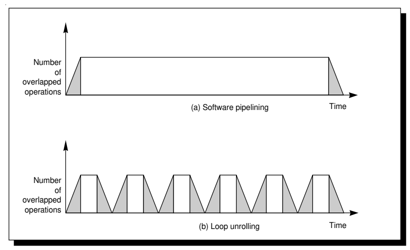
Software Pipelining Example

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,#16
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**
- **2 steps**
  1) ***Trace Selection*: find likely sequence of basic blocks (*trace*) of (predicted) long sequence of straight-line code**
  2) ***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 make hard for conditional operation**

HW support for More ILP

- ***Speculation***: allow instructio to isse 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*)
  - exeucte 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

4 Steps of Speculative Tomasolu 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.

Implementing Speculation

- Figure 5.13 (p. 331 beta) & 5.14 (p. 333 beta) show:
  - Instruciton comit in order
  - As a result, its easy to undo speculated instructions on mispredicted branches or on exceptions

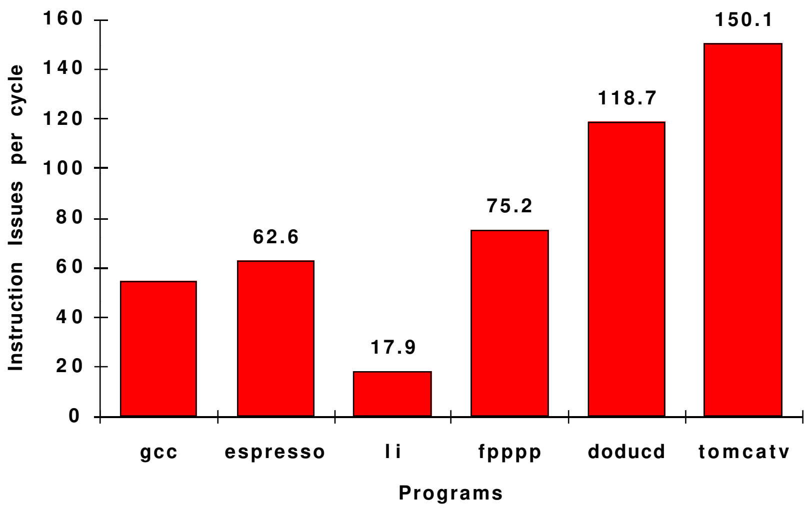
Limits to ILP

- Conflicting studies of amout 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 ≠

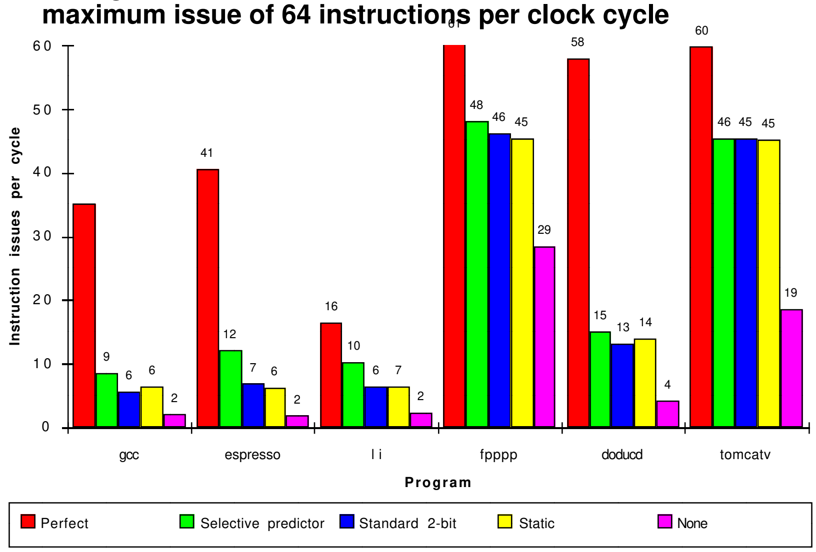
- 1 cycle latency for all instructions

Upper Limit to ILP *(Figure 5.34, p. 338 Beta)*



More Realistic HW: Branch Impact *(Figure 5.38, p. 345)*

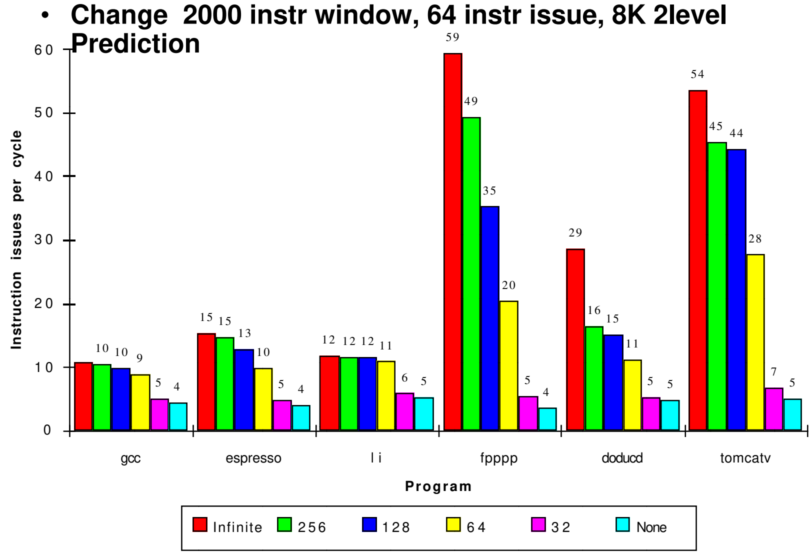
- **Change from Infinite window to examine to 2000 and maximum issue of 64 instructions per clock cycle**



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

More Realistic HW: Register Impact *(Figure 5.40, p. 348)*

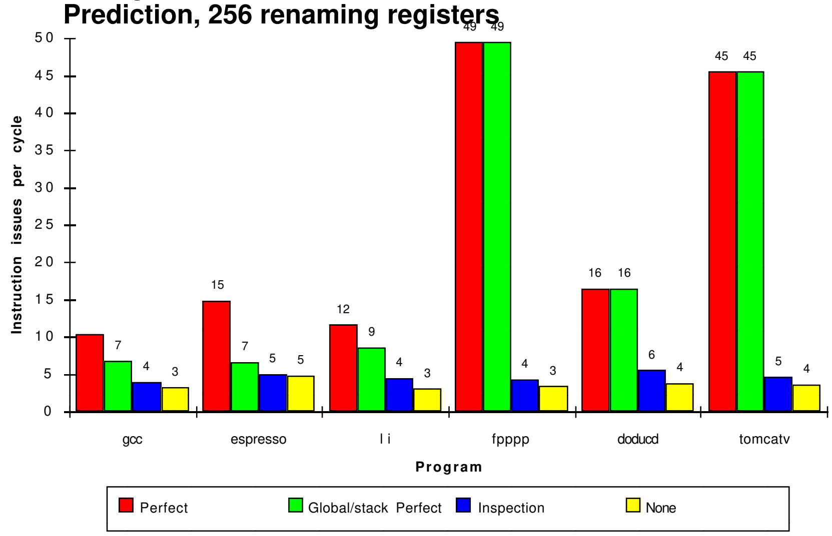
- **Change 2000 instr window, 64 instr issue, 8K 2level Prediction**



Infinite 256 128 64 32 None

More Realistic HW: Alias Impact *(Figure 5.42, p. 351)*

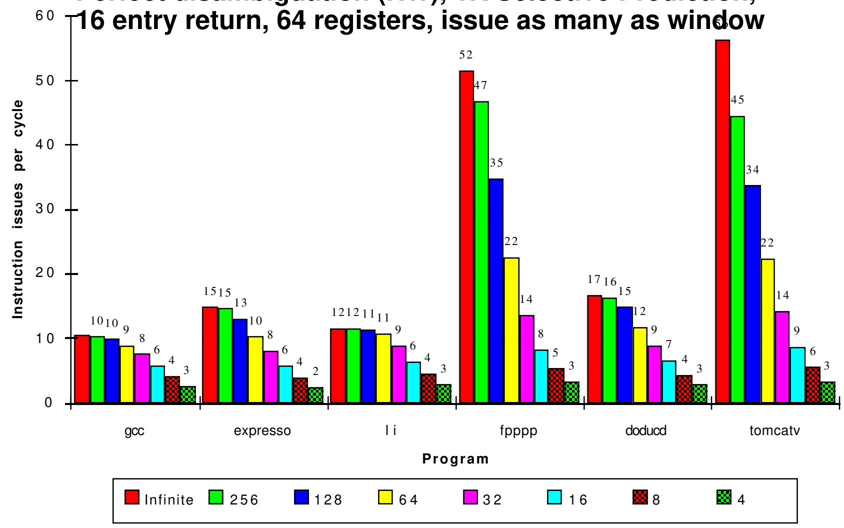
- **Change 2000 instr window, 64 instr issue, 8K 2level Prediction, 256 renaming registers**



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

Realistic HW for ‘9X: Window Impact *(Figure 5.44, p. 354)*

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



Infinite 256 128 64 32 16 8 4

Braniac vs. Speed Demon

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