

Lecture 7

ILP, ILP Limits, and TLP

(Ch.3)

Cristinel Ababei

Dept. of Electrical and Computer Engineering



BE THE DIFFERENCE.

Credits: Slides adapted from presentations of Sudeep Pasricha and others: Kubiawicz, Patterson, Mutlu, Elsevier

1

1

Outline

- **Hardware Based Speculation**
- Multiple Issue Processors
- Limits of ILP
- TLP - Multithreading

2

2

Hardware Based Speculation

- Combines three key ideas:
 1. Dynamic branch prediction to choose which instructions to execute
 2. Speculation to allow execution of instructions before control dependences are resolved
 - + ability to undo effects of incorrectly speculated sequence
 3. Dynamic scheduling to deal with scheduling of different combinations of basic blocks

3

Extending Tomasulo's Algorithm to Support Speculation

- Must separate execution from allowing instruction to finish or "commit"
- This additional step called **instruction commit**
- When an instruction is no longer speculative, allow it to update the register file or memory
- Requires **additional set of buffers** to hold results of instructions that have finished execution but have not committed
- This **Reorder Buffer (ROB)** is also used to pass results among instructions that may be speculated

4

Reorder Buffer (ROB)

- In Tomasulo's algorithm, once an instruction writes its result, any subsequently issued instructions will find result in the register file
- With speculation, the register file is not updated until the instruction commits
 - (we know definitively that the instruction should execute)
- Thus, the ROB supplies operands in **interval between completion of instruction execution and instruction commit**
 - ROB is a source of operands for instructions, just as reservation stations (RS) provide operands in Tomasulo's algorithm
 - ROB extends architecture registers like Reservation Stations (RS) do

5

ROB Details: Fields of Reorder Buffer Entry

Each entry in the ROB contains four fields:

1. Instruction type

- a branch (has no destination result), a store (has a memory address destination), or a register operation (ALU operation or load, which has register destinations)

2. Destination

- Register number (for loads and ALU operations) or memory address (for stores) where the instruction result should be written

3. Value

- Value of instruction result until the instruction commits

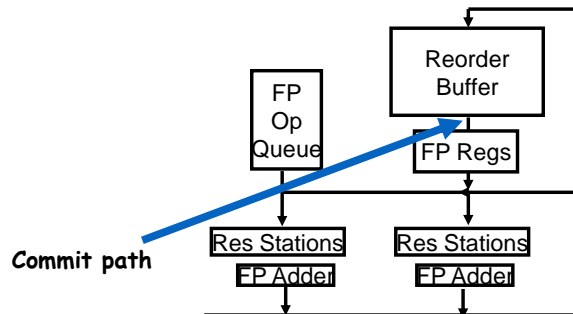
4. Ready

- Indicates that instruction has completed execution, and the value is ready

6

Reorder Buffer Operation

- Holds instructions in FIFO order, exactly as issued
- When instructions complete, results placed into ROB
 - Supplies operands to other instruction between execution complete & commit \Rightarrow more registers like RS
 - Tag results with ROB buffer number instead of reservation station
- When instructions commit \Rightarrow values at head of ROB placed in registers
- As a result, easy to undo speculated instructions on mispredicted branches [or on exceptions](#) – basically, FLUSH the ROB!



7

4 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 (this stage sometimes called “issue”)

3. Write result—finish execution (WB)

Write on Common Data Bus to all awaiting FUs & [reorder buffer](#); mark reservation station available.

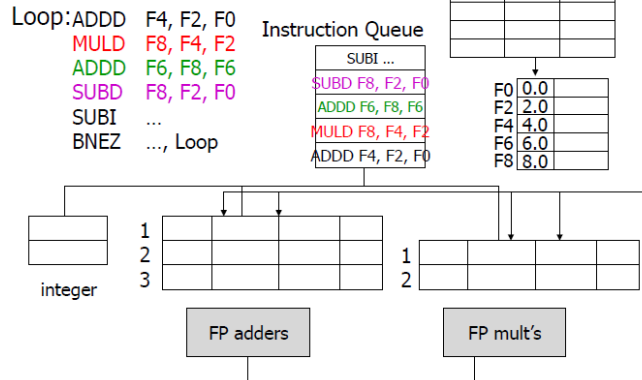
4. Commit—update register with reorder result

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 at head of ROB flushes ROB. (this state sometimes called “[graduation](#)”)

8

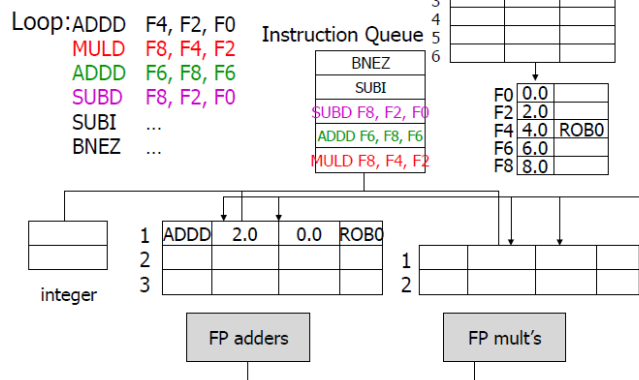
Example: ROB/Tomasulo

ROB/Tomasulo – cycle 0



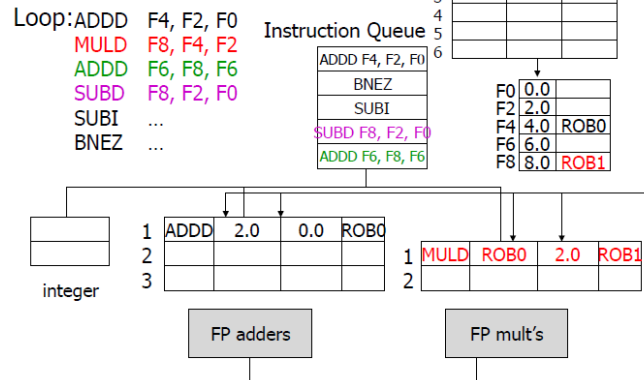
9

ROB/Tomasulo – cycle 1



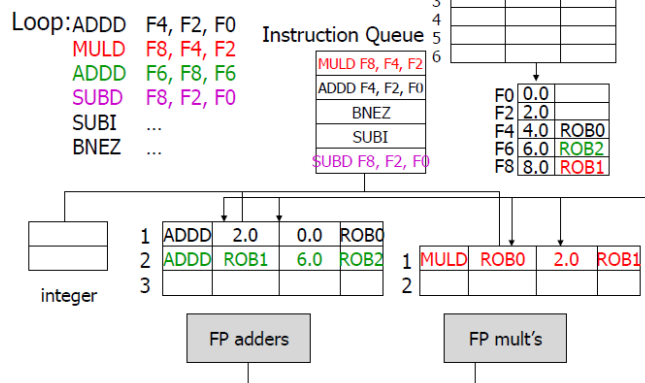
10

ROB/Tomasulo – cycle 2

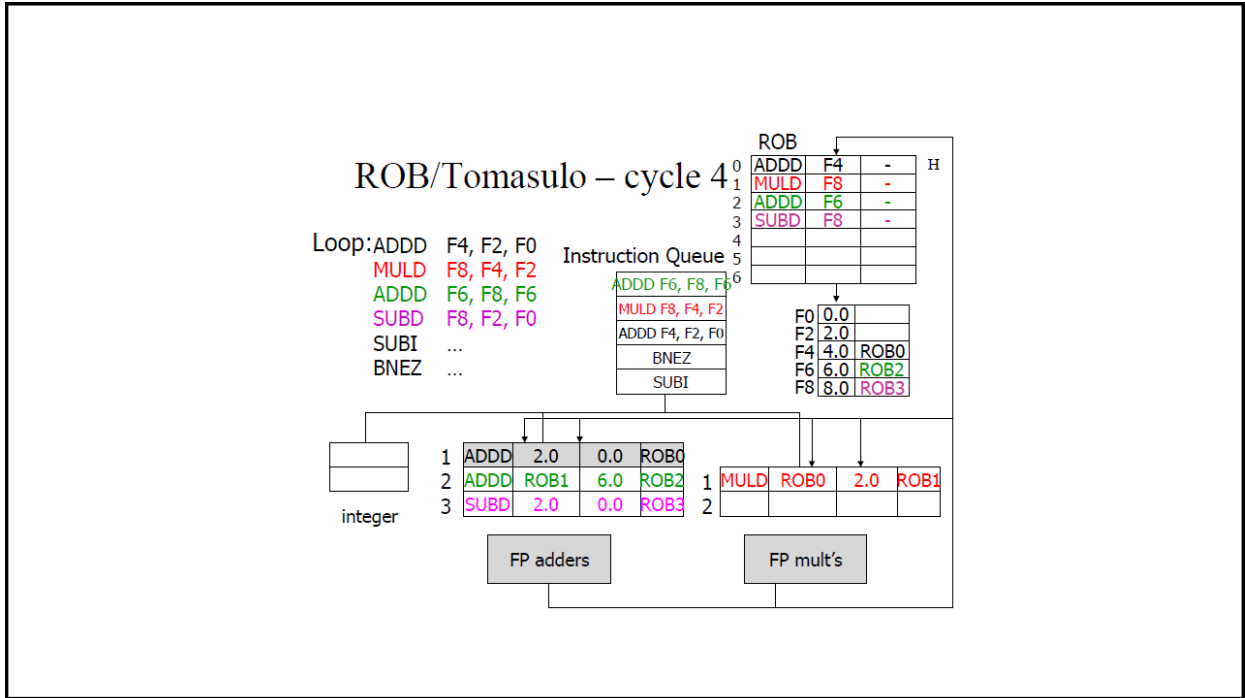


11

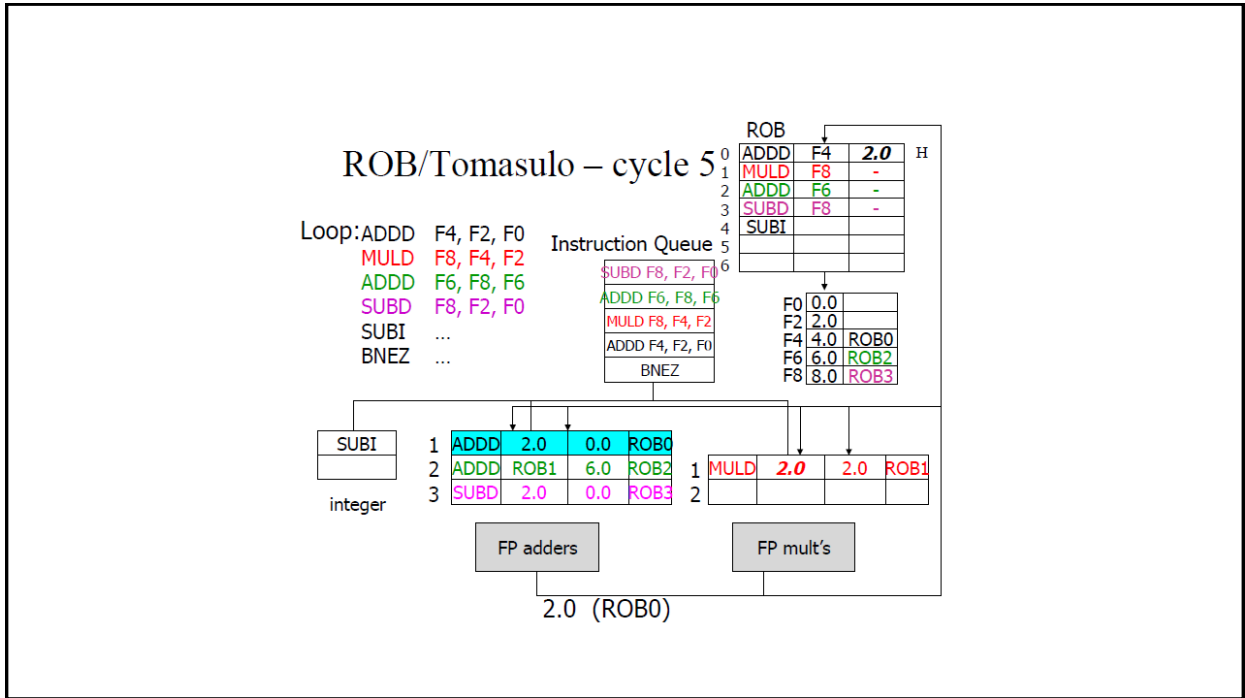
ROB/Tomasulo – cycle 3



12

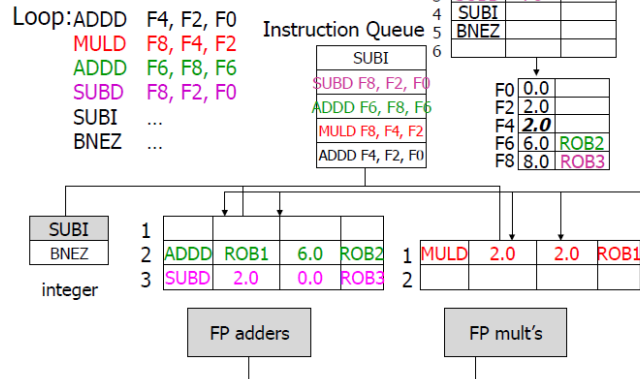


13



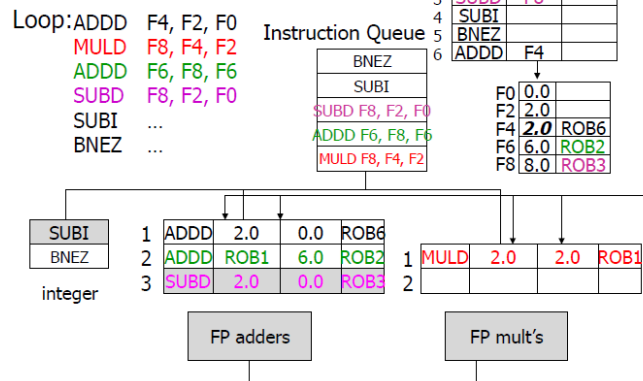
14

ROB/Tomasulo – cycle 6



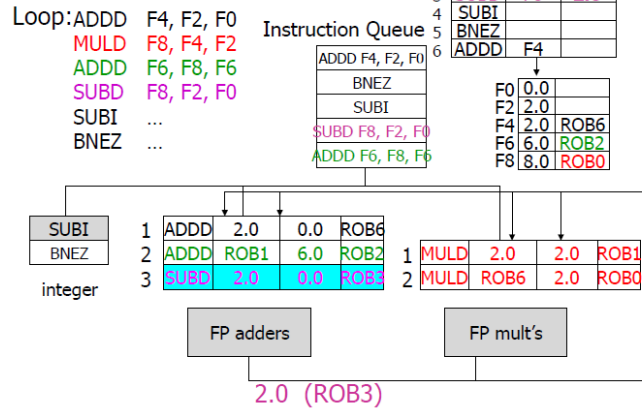
15

ROB/Tomasulo – cycle 7



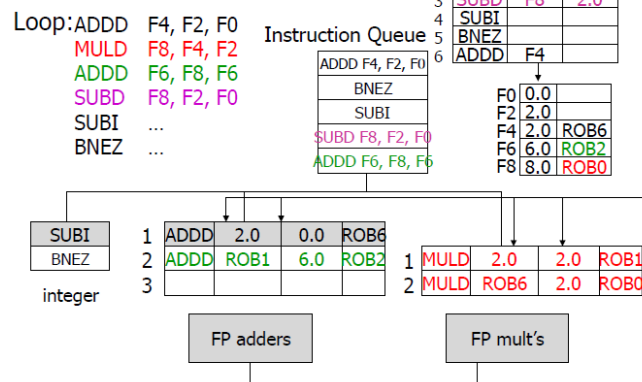
16

ROB/Tomasulo – cycle 8



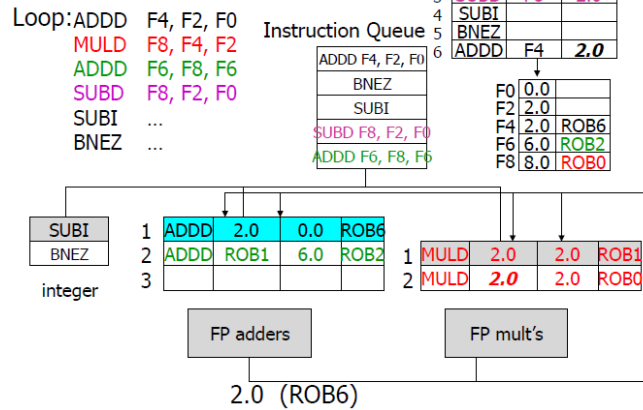
17

ROB/Tomasulo – cycle 9



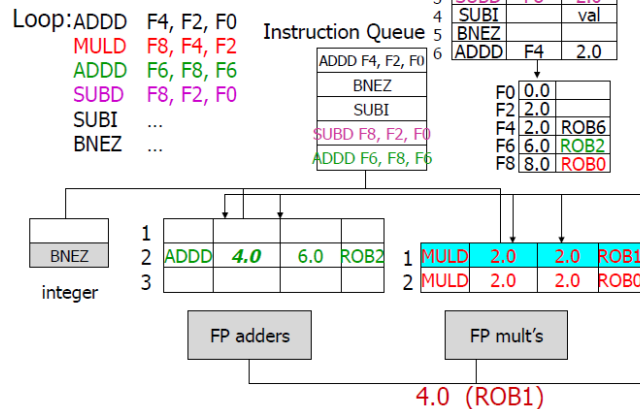
18

ROB/Tomasulo – cycle 11

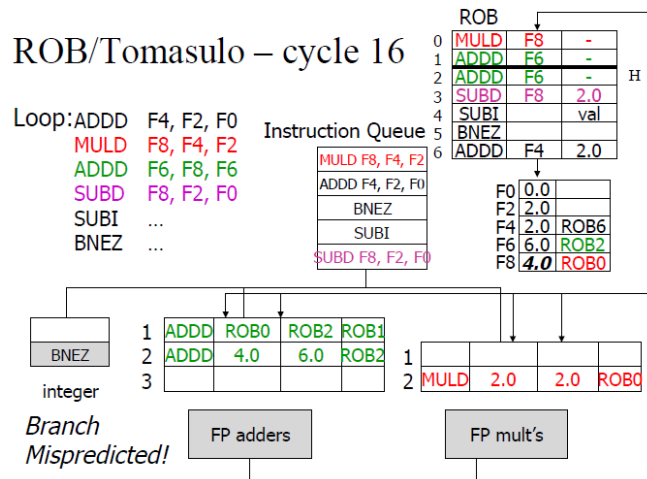


19

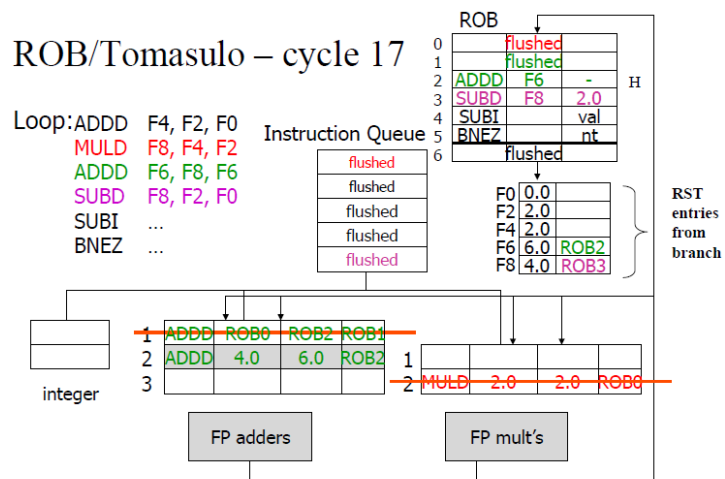
ROB/Tomasulo – cycle 15



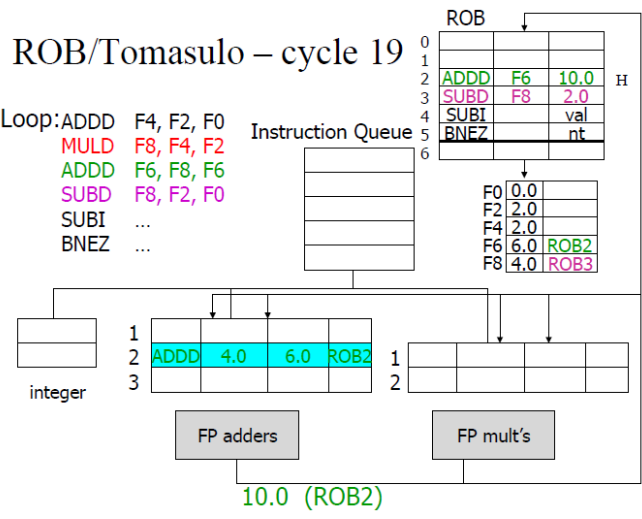
20



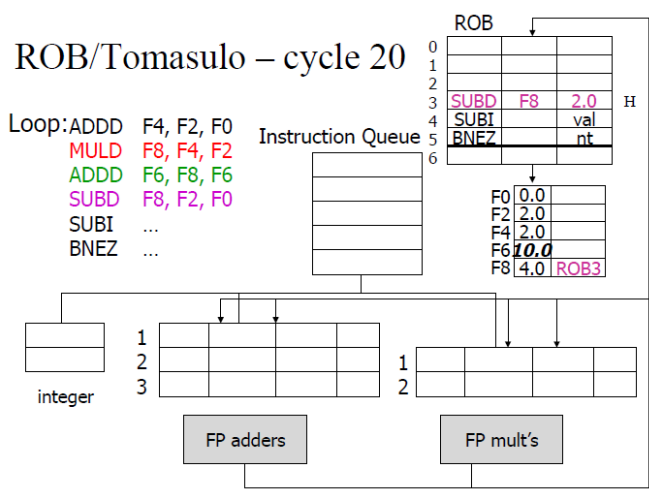
21



22

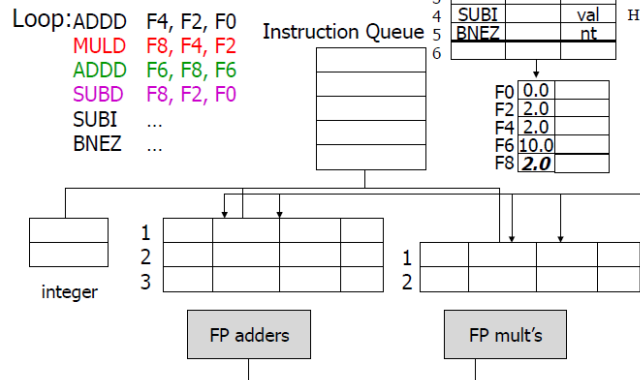


23



24

ROB/Tomasulo – cycle 21



25

Outline

- Hardware Based Speculation
- Multiple Issue Processors
- Limits of ILP
- TLP - Multithreading

26

26

Getting CPI less than 1

Superscalar and VLIW processors

- Trade-off between static and dynamic instruction scheduling
 - Static scheduling places burden on software
 - Dynamic scheduling places burden on hardware
- **Superscalar processors** issue a variable number of instructions per cycle, up to a maximum, using static (compiler) or dynamic (hardware) scheduling
- **VLIW processors** issue a fixed number of instructions per cycle, seen as a packet (potentially with empty slots), and created and scheduled statically by the compiler

27

Superscalar Execution

- A **superscalar architecture** is one in which several instructions can be initiated simultaneously and executed independently.
- Pipelining allows several instructions to be executed at the same time, but they have to be in different pipeline stages at a given moment.
- Superscalar architectures include all features of pipelining but, in addition, there can be several instructions executing simultaneously in the same pipeline stage.

28

Superscalar execution illustrated

- ▶ Higher instruction fetch bandwidth
- ▶ INT instructions, loads and stores occupy one slot
- ▶ FP instructions occupy second slot
 - ▶ Need pipelined FP datapath, otherwise FP datapath becomes bottleneck

Instruction type	Pipe stages					
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

29

Example: Loop unrolling and instruction scheduling

```
for (i=1; i<=1000; i=i+1;)
    x[i] = x[i] + s;
```

Assume latencies

Producer	Consumer	Latency
FP ALU op	Another FP ALU op	3
FP ALU op	store double	2
Load double	FP ALU op	1
Load double	store double	0

30

Loop unrolling and instruction scheduling

Example in MIPS assembly

	Clock cycle issued
Loop: LD F0, 0(R1)	1
ADDD F4, F0, F2	3
SD F4, 0(R1)	6
SUBI R1, R1, 8	7
BNEZ R1, Loop	8

- 9 cycles per loop iteration due to stalls

31

Loop unrolling in single-issue processor

Loop unrolling; 4 iterations

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

Improved instruction scheduling

- LD's up eliminate LD-ADDD
1-cycle stall
- ADDD's up eliminate ADDD-SD
2-cycle stalls
- SD down fills branch delay slot
- Need adjustment of SD indices
- 14 cycles for 4 iterations, or 3.5
cycles per iteration

32

Loop unrolling in superscalar processor

Unroll 3 iterations with dual-issue instruction scheduling

	INT	FP	Cycle
Loop:	LD F0, 0(R1)		1
	LD F6, -8(R1)		2
	LD F10, -16(R1)	ADDD F4, F0, F2	3
		ADDD F8, F6, F2	4
		ADDD F12, F10, F2	5
	SD 0(R1), F4		6
	SD -8(R1), F8		7
	SUBI R1, R1, 24		8
	BNEZ R1, Loop		9
	SD 8(R1), F12		10

- 10 cycles per 3 iterations, 3.3 cycles per iteration

33

Loop unrolling and static scheduling in superscalar

Unroll 5 iterations with dual-issue instruction scheduling

	INT	FP	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		9
	BNEZ R1, Loop		11
	SD 8(R1), F20		12

- 12 cycles per 5 iterations, 2.4 cycles per iteration

34

Superscalar processor with dynamic scheduling

- Extend Tomasulo's algorithm to handle multiple issue
- Instructions issued in program order
- Cannot issue multiple dependent instructions in the same cycle.

35

Dual-issue pipeline with Tomasulo

Assume usual loop example

Iteration number	Instructions	Issues at	Starts execute	Memory access at	Writes CDB at	Comment
1	LD F0, 0(R1)	1	2	3	4	First issue
1	ADDD F4, F0, F2	1	5		8	Wait for LD
1	SD F4, 0(R1)	2	3	9		Wait for ADDD
1	SUBI R1, R1, 8	3	4		5	Wait for INT FU
1	BNEZ R1, Loop	4	6			Wait for SUBI
2	LD F0, 0(R1)	5	7	8	9	Wait for BNE
2	ADDD F4, F0, F2	5	10		13	Wait for LD
2	SD F4, 0(R1)	6	8	14		Wait for ADDD
2	SUBI R1, R1, 8	7	9		10	Wait for INT FU
2	BNEZ R1, Loop	8	11			Wait for SUBI
3	LD F0, 0(R1)	9	12	13	14	Wait for BNE
3	ADDD F4, F0, F2	9	15		18	Wait for LD
3	SD F4, 0(R1)	10	13	19		Wait for ADDD
3	SUBI R1, R1, 8	11	14		15	Wait for INT FU
3	BNEZ R1, Loop	12	16			Wait for SUBI

► 19 cycles per 4 iterations, or 4.8 cycles per iteration

36

Limitations of multiple-issue processors

Software and hardware implications

- ▶ Inherent limitations of ILP in programs (control and data dependencies)
 - ▶ Need as many independent instructions as pipeline depth
 - ▶ Deep unrolling, register pressure
- ▶ Hardware complexity increasing rapidly with instructions issued per cycle
 - ▶ Adding FUs scales complexity linearly
 - ▶ Higher memory and register-file bandwidth, more ports
 - ▶ Memory system implications, interleaving
 - ▶ Dynamic scheduling expensive
 - ▶ Issue logic of dynamic scheduled processors expensive
- ▶ Middle ground: combination of static (compiler) and dynamic scheduling

37

Outline

- Hardware Based Speculation
- Multiple Issue Processors
- Limits of ILP (Not included in Ed.6, 2019)
- TLP - Multithreading

38

38

Extracting more ILP

- Speculation

- Look for parallelism beyond branches
- Need easy roll-back – ROB
- Combined with multiple-issue regains performance loss

- Multiple instructions issued per cycle

- Natural parallelism in loops
- Increased hardware complexity
- More ports to register files, memory, other resources
- Branch predictors, potentially for multiple branches simultaneously
- Complex issue logic and control logic

39

Limits to ILP

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

40

40

Limits to ILP: Quantitative Analysis

Assumptions for ideal/perfect machine (all constraints on ILP are removed):

1. *Register renaming* – infinite virtual registers
=> all register WAW & WAR hazards are avoided
2. *Branch prediction* – perfect; no mispredictions
3. *Jump prediction* – all jumps perfectly predicted (returns, case statements)
- 2 & 3 => no control dependencies; perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis* – addresses known & a load can be moved before a store provided that the addresses are not equal; 1&4 eliminates all but RAW

Also: *perfect caches*; 1 cycle latency for all instructions (FP *,/); unlimited instructions issued/clock cycle;

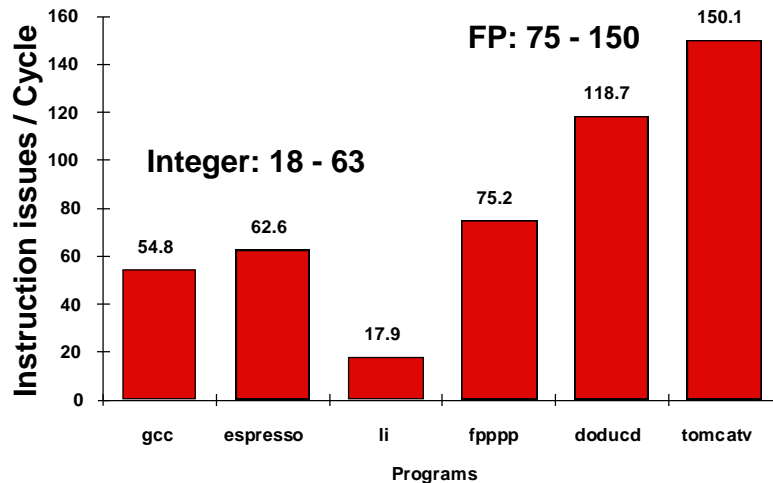
41

Limits to ILP HW Model comparison

	Model	Power 5
Instructions Issued per clock	Infinite	4
Instruction Window Size	Infinite	200
Renaming Registers	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias Analysis	Perfect	??

42

Upper Limit to ILP: Ideal Machine



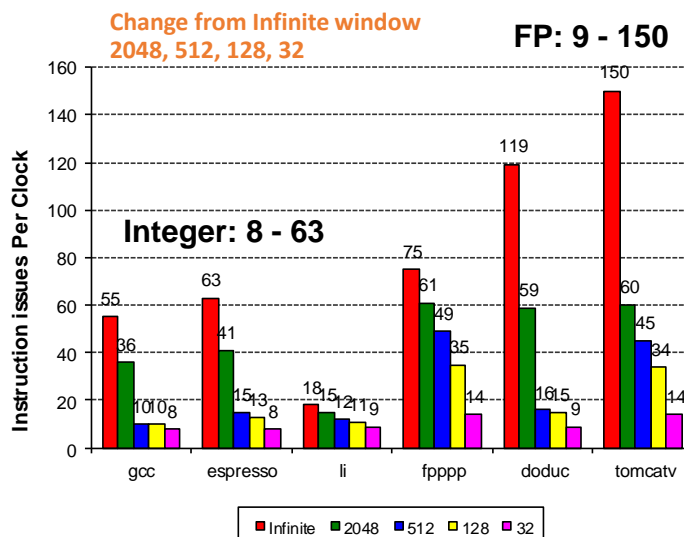
43

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	Infinite	Infinite	4
Instruction Window Size	Infinite, 2K, 512, 128, 32	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	??

44

More Realistic HW: Window Impact



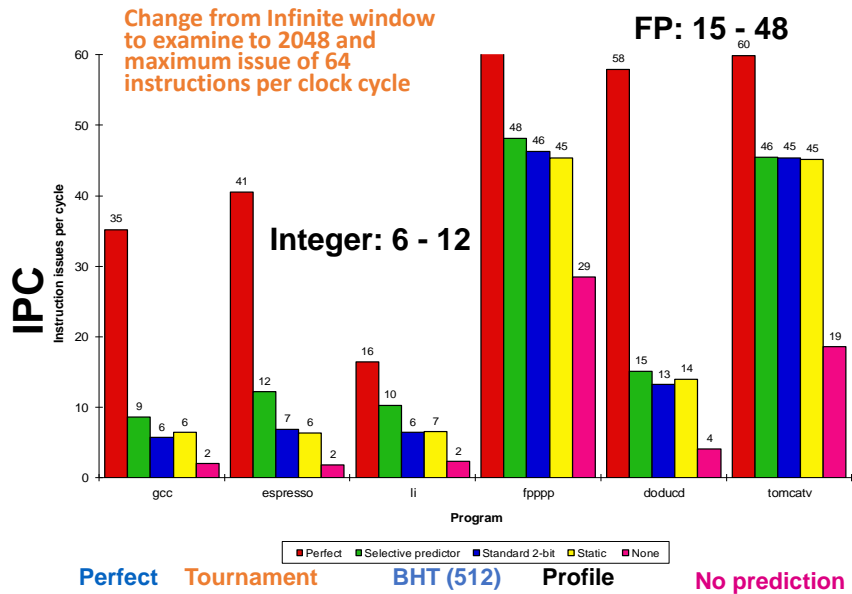
45

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect vs. 8K Tournament vs. 512 2-bit vs. profile vs. none	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	??

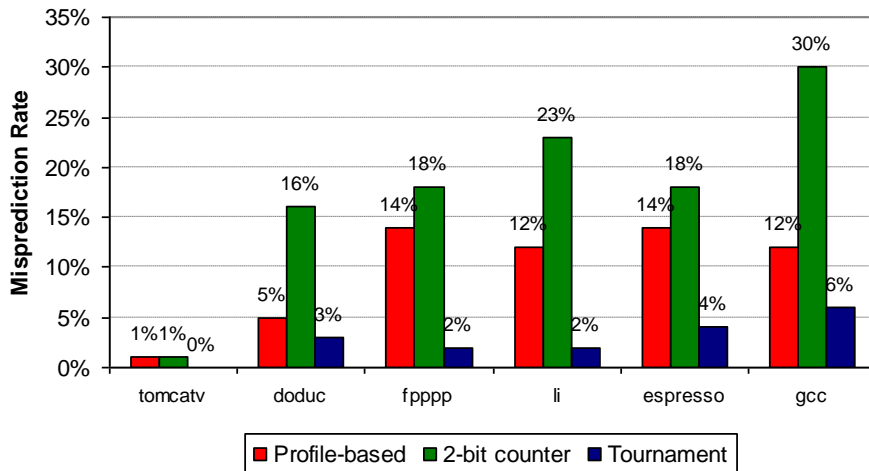
46

More Realistic HW: Branch Impact



47

Misprediction Rates



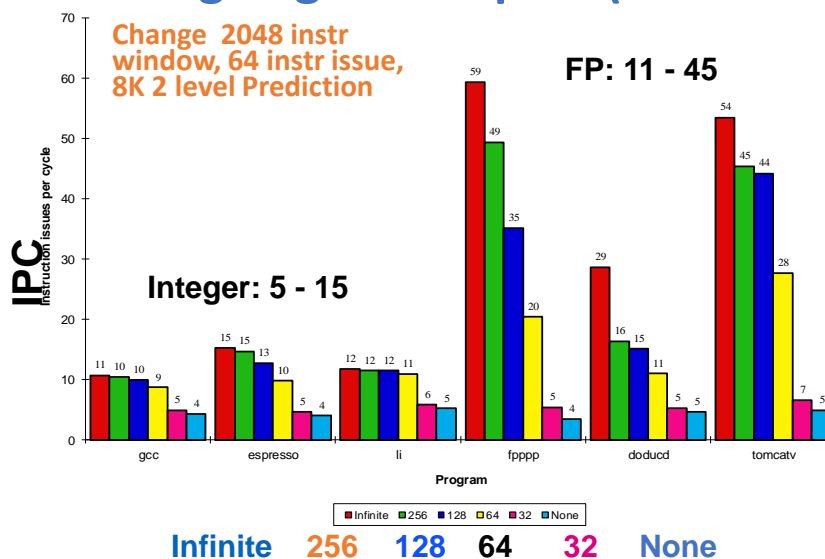
48

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	Infinite v. 256, 128, 64, 32, none	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	8K 2-level	Perfect	Tournament Branch Predictor
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	??

49

More Realistic HW: Renaming Register Impact (N int + N fp)



50

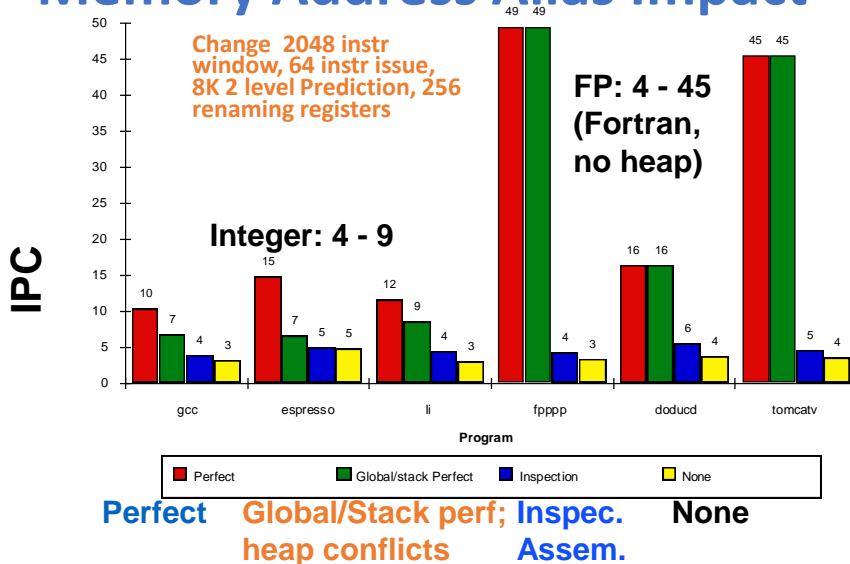
Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	256 Int + 256 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	8K 2-level	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect v. Stack v. Inspect v. none	Perfect	??

51

51

More Realistic HW: Memory Address Alias Impact



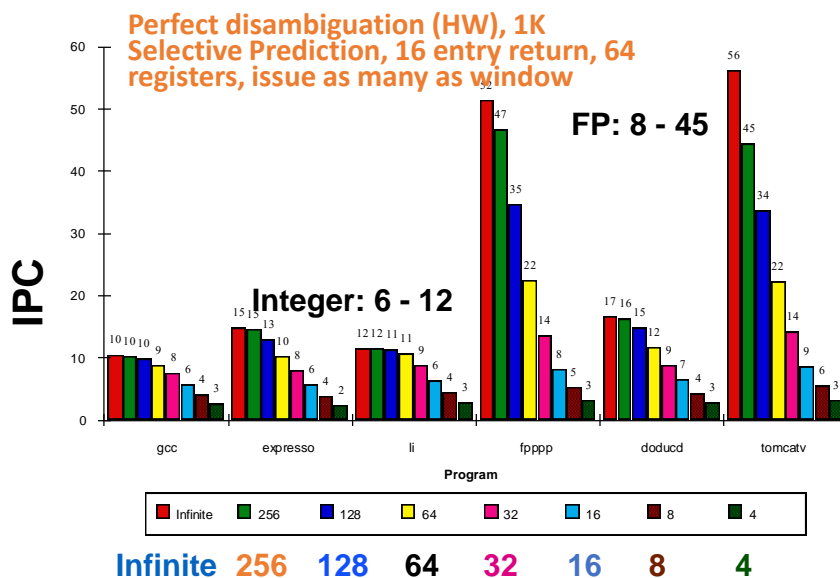
52

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64 (no restrictions)	Infinite	4
Instruction Window Size	Infinite vs. 256, 128, 64, 32	Infinite	200
Renaming Registers	64 Int + 64 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	1K 2-level	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	HW disambiguation	Perfect	Perfect

53

Realistic HW: Window Impact



54

Limits to ILP

- Advances in compiler technology + significantly new and different hardware techniques **may** be able to overcome limitations assumed in studies
- However, unlikely such advances when coupled **with realistic hardware** will overcome these limits in near future

55

Outline

- Hardware Based Speculation
- Multiple Issue Processors
- Limits of ILP
- **TLP - Multithreading**

56

56

Performance Beyond Traditional ILP

- There can be much higher natural parallelism in some applications (e.g., Database or Scientific codes)
- Explicit
 - Thread Level Parallelism (TLP) or
 - Data Level Parallelism (DLP)
- **Thread**: light-weight process with own instructions and data
 - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
- **Data Level Parallelism**: Perform identical operations on data, and lots of data

57

Thread Level Parallelism (TLP)

- Goal: Improve **Uniprocessor** Throughput
- ILP exploits implicit parallel operations within a loop or straight-line code segment
- TLP explicitly represented by the use of multiple threads of execution that are inherently parallel
- Goal: Use **multiple instruction streams** to improve
 1. Throughput of computers that run many programs
 2. Execution time of multithreaded programs
- TLP could be more cost-effective to exploit than ILP

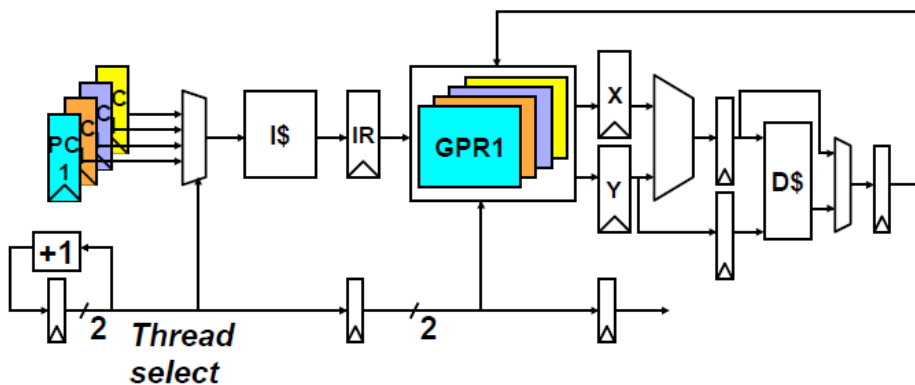
58

Multithreaded Execution

- Multithreading: multiple threads to share the functional units of 1 processor via overlapping
 - Processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
 - Memory shared through the virtual memory mechanisms, which already support multiple processes
 - HW for fast thread switch; much faster than full process switch (which can take 100s to 1000s of clocks)
 - Single process might contain multiple threads; all threads within a process share the same memory space, and can communicate with each other directly, because they share the same variables
- When to switch between threads?
 1. Alternate instruction per thread (**fine grain**)
 2. When a thread is stalled, perhaps for a cache miss, another thread can be executed (**coarse grain**)

59

HW: Simple Multithreaded Pipeline



- Have to carry thread select down pipeline to ensure correct state bits read/written at each pipe stage
- Appears to software (including OS) as multiple, albeit slower, CPUs

60

1. Fine-Grained Multithreading

- Switches between threads on each cycle, causing the execution of multiples threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads every clock
- Advantage is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used on Sun's T1 (2005) and T2 (Niagara, 2007)

61

Fine-Grained Multithreading on the Sun T1

- Circa 2005; first major processor to focus on TLP rather than ILP

Characteristic	Sun T1
Multiprocessor and multithreading support	Eight cores per chip; four threads per core. Fine-grained thread scheduling. One shared floating-point unit for eight cores. Supports only on-chip multiprocessing.
Pipeline structure	Simple, in-order, six-deep pipeline with three-cycle delays for loads and branches.
L1 caches	16 KB instructions; 8 KB data. 64-byte block size. Miss to L2 is 23 cycles, assuming no contention.
L2 caches	Four separate L2 caches, each 750 KB and associated with a memory bank. 64-byte block size. Miss to main memory is 110 clock cycles assuming no contention.
Initial implementation	90 nm process; maximum clock rate of 1.2 GHz; power 79 W; 300 M transistors; 379 mm ² die.

62

CPI on Sun T1

Benchmark	Per-thread CPI	Per-core CPI
TPC-C	7.2	1.80
SPECJBB	5.6	1.40
SPECWeb99	6.6	1.65

- T1 has 8 cores; with 4 threads/core
- Ideal effective CPI per thread?
- Ideal per-core CPI?
- In 2005 when it was introduced, the Sun T1 processor had the best performance on integer applications with extensive TLP and demanding memory performance, such as SPECJBB and transaction processing workloads

63

2. Course-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses
- Advantages
 - Relieves need to have very fast thread-switching
 - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
 - Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
 - New thread must fill pipeline before instructions can complete
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill time \ll stall time
- Used in IBM AS/400, but not in modern processors

64

3. Simultaneous MultiThreading (SMT) for OoO superscalars

- Techniques presented so far have all been “vertical” multithreading where each pipeline stage works on one thread at a time
- SMT uses fine-grain control already present inside an OoO superscalar to allow instructions from multiple threads to enter execution on same clock cycle. Gives better utilization of machine resources.

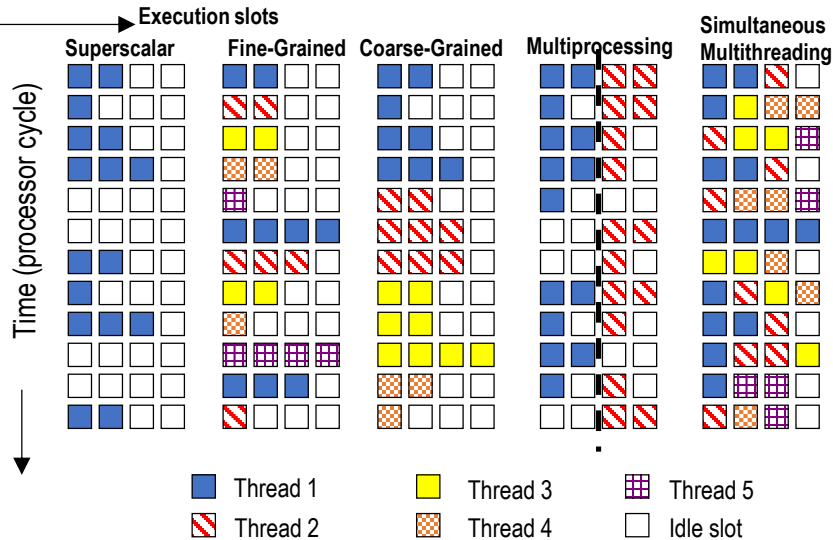
65

Simultaneous Multithreading (SMT)

- Simultaneous multithreading (SMT): a variation of fine-grained multithreading implemented on top of a multiple-issue, dynamically scheduled processor
- Add multiple contexts and fetch engines and allow instructions fetched from different threads to issue simultaneously
- Utilize wide out-of-order superscalar processor issue queue to find instructions to issue from multiple threads
- OoO instruction window already has most of the circuitry required to schedule from multiple threads
- Any single thread can utilize whole machine
- Used in Core i7 (2008) and IBM Power 7 (2010)

66

Illustration of Multithreading Categories



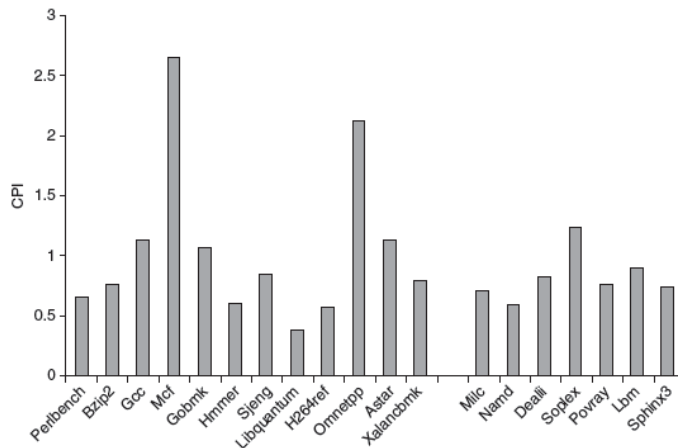
67

Example 1: Intel Core i7

- Aggressive out-of-order speculative microarchitecture
- Total pipeline depth is 14 stages; branch mispredictions cost 17 cycles
- 48 load and 32 store buffers
- Six independent functional units can each begin execution of a ready micro-op in the same cycle
- Uses a 36-entry centralized reservation station shared by six functional units, with ROB
 - Up to six micro-ops may be dispatched to functional units every cycle

68

Intel Core i7 Performance: CPI



The CPI for the 19 SPEC CPU2006 benchmarks shows an average CPI of 0.83 for both the FP and integer benchmarks. SMT further improves performance by 15-30% (according to Intel)

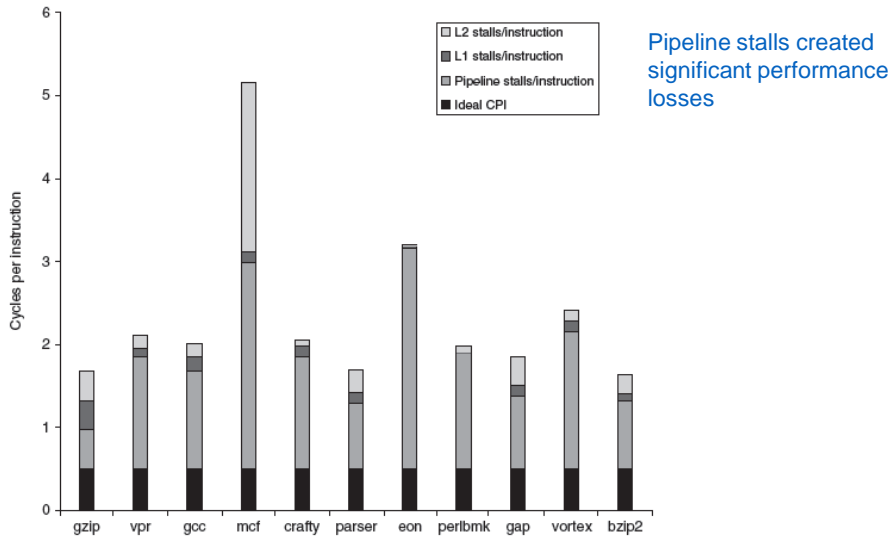
69

Example 2: ARM Cortex A8

- 13-stage pipeline, dual-issue, statically scheduled superscalar with dynamic issue detection, which allows the processor to issue one or two instructions per clock
- Dynamic branch predictor with a 512-entry two-way set associative branch target buffer and a 4K-entry global history buffer, indexed by branch history and current PC
 - incorrect prediction results in a 13-cycle penalty as pipeline is flushed
 - eight-entry return stack is kept to track return addresses
- Has an ideal CPI of 0.5 due to its dual-issue structure
- Hazards:
 - Functional and data (compiler must handle; otherwise issue 1 instr/cycle)
 - Control (arise only when branches are mispredicted)

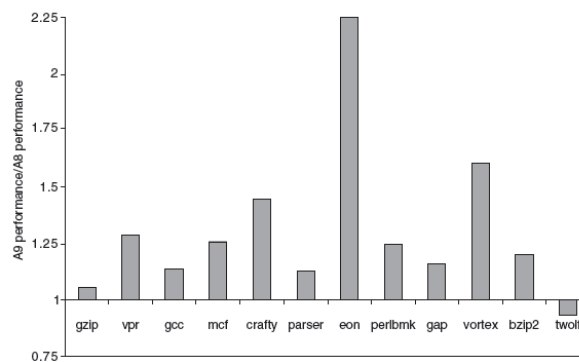
70

ARM Cortex A8 Performance: CPI



71

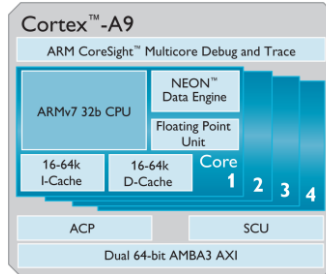
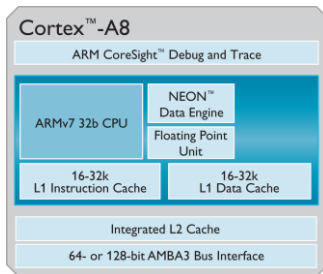
ARM Cortex A9 vs. A8



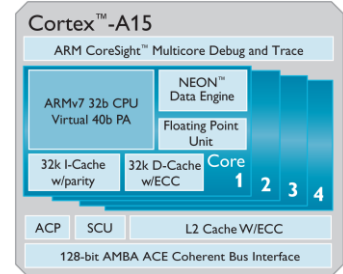
- The insight that the pipeline stalls created significant performance losses probably played a key role in the decision to make the ARM Cortex-A9 a dynamically scheduled superscalar w/ speculation
- A9 uses a more powerful branch predictor, OO execution, instruction cache prefetch, faster multiply pipeline and a nonblocking L1 data cache

72

ARM A Series Processors



~50% faster than A8



~40% faster than A9
up to 8 cores

73

The Future?

● Moving away from ILP enhancements

	Power4	Power5	Power6	Power7
Introduced	2001	2004	2007	2010
Initial clock rate (GHz)	1.3	1.9	4.7	3.6
Transistor count (M)	174	276	790	1200
Issues per clock	5	5	7	6
Functional units	8	8	9	12
Cores/chip	2	2	2	8
SMT threads	0	2	2	4
Total on-chip cache (MB)	1.5	2	4.1	32.3

Figure 3.47 Characteristics of four IBM Power processors. All except the Power6 were dynamically scheduled, which is static, and in-order, and all the processors support two load/store pipelines. The Power6 has the same functional units as the Power5 except for a decimal unit. Power7 uses DRAM for the L3 cache.

74