ILP, Core-Processors and Multithreading (pgs 223-247)

- There was significant progress in **Instruction-Level-Parallelism** (ILP) over last decade, with the Itanium, and multiple-issue dynamically-scheduled machines
- In 2000, after the Itanium demise, researchers predicted (within 10 years) dynamically-scheduled multiple-issue machines with speculation, with 6-8 instructions per clock cycle, with multiple 'processes' (pg 230)
- However, cache misses have large latencies and are hard to cover with ILP -> some researchers started the search for other forms of parallelism
- **Multithreading** allows multiple threads to **share one processing core** in an overlapping fashion, with rapid thread switches
- Many new processors include multiple cores, with multi-threading in each core
- Each thread maintains its own registers, program counter, and virtual-memory page table
- The core supports the ability to rapidly switch between threads, within 1 clock cycle
- A compiler generates the threads when it compiles the code

ILP, Core-Processors and Multithreading (pg. 229)

- **Fine-Grain Multithreading** : processor can switch between threads within 1 clock cycle
- threads usually activated in round-robin order, skipping stalled threads
- can hide very small and very large stalls, since thread switches are very fast
- slows down performance of any one thread, since all the threads share one processor
- **Course-Grain Multithreading** : only switch threads on costly stalls, i.e., level 2 or 3 cache misses
- Does not necessarily slow-down any one thread
- However, it cannot hide throughput losses from short stalls (i.e., level 1 cache)
- there is a thread-start-up stall effect, when the pipeline is loaded-up with a new thread and many clock cycles are effectively wasted,
- **Simultaneous Multithreading (SMT)**: when fine-grain multithreading is implemented in a multiple-issue dynamically-scheduled processor

ILP, Core-Processors and Multithreading (pg. 226)

- Next slide Fig. 3.28 shows several versions of multithreading

- 1) a regular multiple-issue with no multithreading : black boxes represent one instruction using a functional unit (FU) from one 'thread '; in each time-slot, many FUs are idle; there is a larger stall fir many clock cycles

- 2) a regular multiple-issue with **course-grain multithreading** : grey boxes represent one instruction using a FU from a second 'thread ' ; in each time-slot, many Fus are still idle; a 2nd thread has been activated in the large stall

- 3) a regular multiple-issue with **fine-grain multithreading** : light grey and dark grey boxes represent one instruction using a FU from a second and third 'thread ' ; in each time-slot, only one thread is active, and some FUs may be idle; threads share the processor in round-robin order

- 4) a regular multiple-issue **with simultaneous multithreading** : multiple threads are active using FUs in each time-slot; 'ready-to-execute' instruction are taken from each thread in round-robin order, to fill the Fus

- In all current processors, all issues from from one thread, but different threads can execute in the same clock cycle

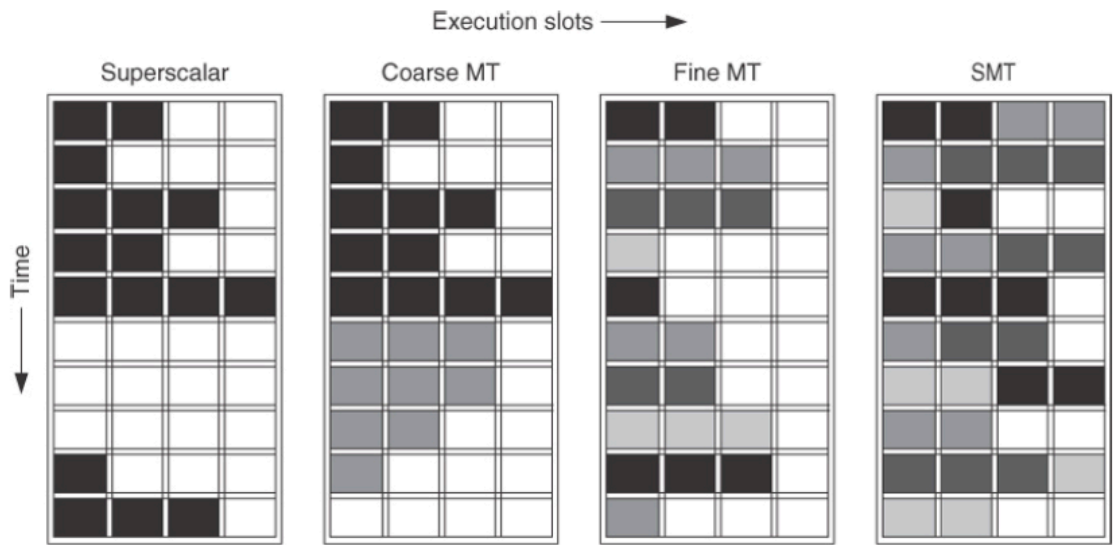

Figure 3.28 How four different approaches use the functional unit execution slots of a superscalar processor. The horizontal dimension represents the instruction execution capability in each clock cycle. The vertical dimension represents a sequence of clock cycles. An empty (white) box indicates that the corresponding execution slot is unused in that clock cycle. The shades of gray and black correspond to four different threads in the multithreading processors. Black is also used to indicate the occupied issue slots in the case of the superscalar without multithreading support. The Sun T1 and T2 (aka Niagara) processors are fine-grained multithreaded processors, while the Intel Core i7 and IBM Power7 processors use SMT. The T2 has eight threads, the Power7 has four, and the Intel i7 has two. In all existing SMTs, instructions issue from only one thread at a time. The difference in SMT is that the subsequent decision to execute an instruction is decoupled and could execute the operations coming from several different instructions in the same clock cycle.

Effectiveness on Sun T1 Processor (pg. 227)

- Sun introduced the T1 fine-grain multithreaded processor in 2005
- One of the first processors to de-emphasize ILP and favor Thread-Level-Parallelism (TLP)
- Chip returned to a basic simple 6-stage pipeline with support for multiple threads
- It did this while one of the most complex and aggressive ILP-based machines, the Itanium, was still around
- T1 processor: 8 cores, each with 4 threads: Each core is a simple 6 stage pipeline; like our basic 5 stage pipeline, plus one stage for thread-switching
- Fine-grain multithreading: switches to a new non-stalled thread in each clock-cycle
- Processor is idle only when all 4 threads are idle
- Loads / Stores incur 3 clock cycle delay that can be hidden with other threads
- All 8 cores share a single set of floating-point registers (the focus was on integer performance, for web-services)

Effectiveness on Sun T1 Processor (pg. 227)

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.

Figure 3.29 A summary of the T1 processor.

Effectiveness on Sun T1 Processor (pg. 228)

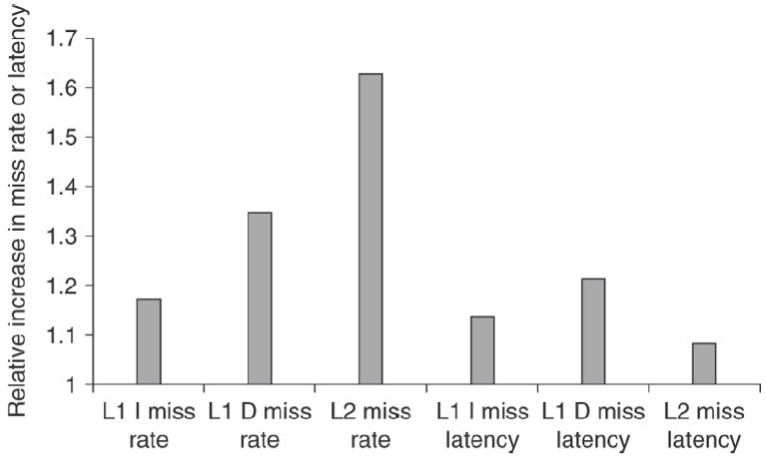

Figure 3.30 The relative change in the miss rates and miss latencies when executing with one thread per core versus four threads per core on the TPC-C benchmark. The latencies are the actual time to return the requested data after a miss. In the four-thread case, the execution of other threads could potentially hide much of this latency.

• when using 4-threads per core, L1 (I=Instruction and D=Data) and L2 (shared) miss rates increase by 18%, 35%, and 65%, due to increased contention for memory system

• unfortunately, we can't make larger and faster main memory; every new technology (ie magnetic bubble memory) has failed

Effectiveness on Sun T1 Processor (pg. 228)

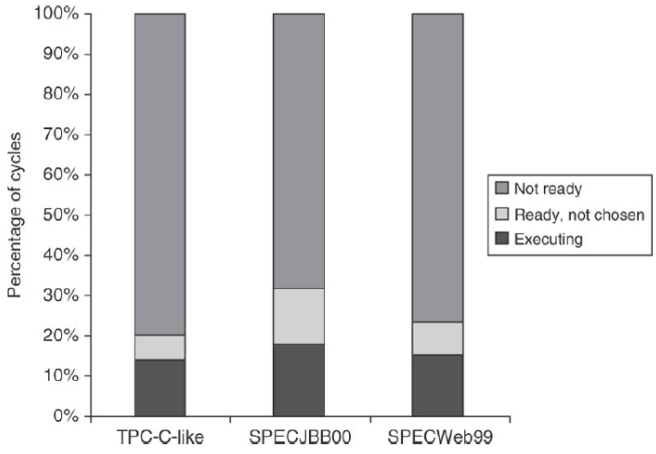

Figure 3.31 Breakdown of the status on an average thread. "Executing" indicates the thread issues an instruction in that cycle. "Ready but not chosen" means it could issue but another thread has been chosen, and "not ready" indicates that the thread is awaiting the completion of an event (a pipeline delay or cache miss, for example).

• On benchmarks, about 10-15% of instructions are executing, another 10-20% are ready to execute but not chosen, and about 70-80% are not ready to execute

• Most of the time, not-read-to-execute instruction are waiting for cache misses

Effectiveness on Sun T1 Processor (pg. 229)

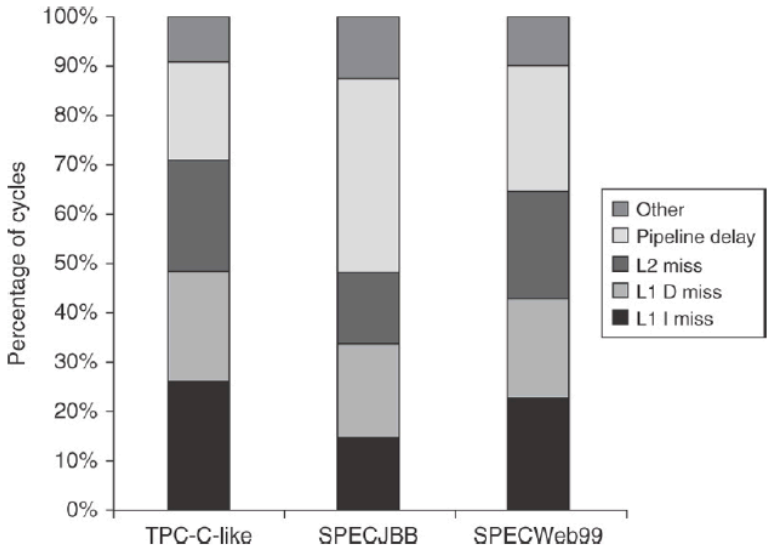

Figure 3.32 The breakdown of causes for a thread being not ready. The contribution to the "other" category varies. In PC-C, store buffer full is the largest contributor; in SPEC-JBB, atomic instructions are the largest contributor; and in SPECWeb99 both factors contribute.

Effectiveness on Sun T1 Processor (pg. 228)

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

Figure 3.33 The per-thread CPI, the per-core CPI, the effective eight-core CPI, and the effective IPC (inverse of CPI) for the eight-core T1 processor.

- ideal effective CPI per thread = 4 clock cycles per instruction (1 instruction executed every 4 clock cycles for each thread): a smaller CPI is better
- ideal CPI per core = 1 (1 instruction executed every clock cycle from some thread)
- The T1 processor runs at > 50% effectiveness, after considering all cache misses, which is pretty good
- In 2005, the SUN T1 processor with 8 cores had the best reported integer performance, beating out the Itanium and some traditional and more complex dynamically-scheduled multiple-issue machines

Energy Efficiency and Multithreading (pg. 232)

- Intel i7 core supports SMT (with 3 threads per core)
- Next slide explores energy-efficiency ratio and SMT performance in the i7 core
- energy-efficiency ratio = inverse energy-consumption, so that higher number is better
- performance can be expressed as effective instructions per clock cycle when all cores are in use, after considering all cache misses
- in next figure first reported in 2011, on average SMT improves performance and energy-efficiency-ratio
- In 2011, the industry moved towards more simpler cores (multiple-issue with 3-4 instructions per clock cycle, dynamically-scheduled) with SMT, rather than fewer more complex cores with ILP (pg 231)
- higher energy-efficiency-ratio reported for Intel i5 core and the Intel Atom machine

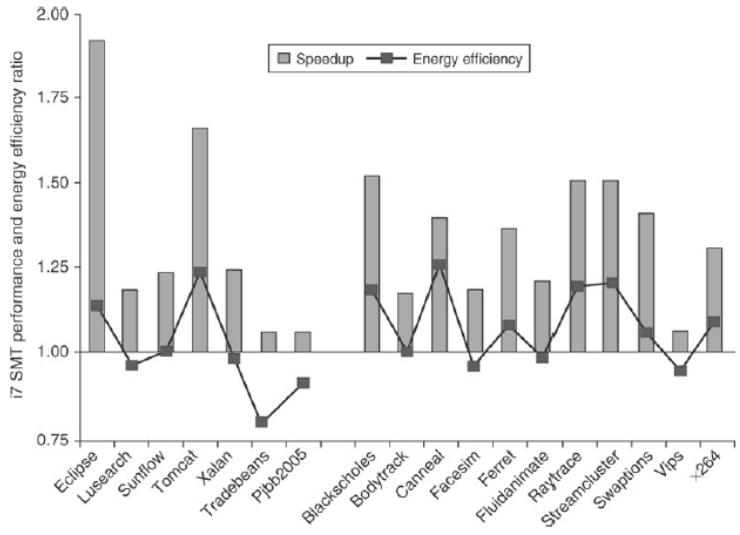

Figure 3.35 The speedup from using multithreading on one core on an i7 processor averages 1.28 for the Java benchmarks and 1.31 for the PARSEC benchmarks (using an unweighted harmonic mean, which implies a workload where the total time spent executing each benchmark in the single-threaded base set was the same). The energy efficiency averages 0.99 and 1.07, respectively (using the harmonic mean). Recall that anything above 1.0 for energy efficiency indicates that the feature reduces execution time by more than it increases average power. Two of the Java benchmarks experience little speedup and have significant negative energy efficiency because of this. Turbo Boost is off in all cases. These data were collected and analyzed by Esmaeilzadeh et al. [2011] using the Oracle (Sun) HotSpot build 16.3-b01 Java 1.6.0 Virtual Machine and the gcc v4.4.1 native compiler.

The ARM Cortex A8 Processor (pg. 233)

- ARM A8 processor is the basis for the Apple A9 processor used in the iPAD
- dual-issue statically-scheduled machine with a 13-stage pipeline
- it uses dynamic branch prediction with a 512-entry 2-way set-associative Branch Target Buffer (BTB) and a 4K-entry global branch history buffer, to minimize the branch stalls
- it uses an 8-entry branch return stack
- a branch misprediction results in flushing the pipeline and incurring a 13 clock cycle penalty
- up to 2 instructions can issue per clock cycle
- if the instructions are data-dependent and if data-forwarding can resolve the dependence, then they can issue together
- otherwise, the dependent instruction (and all instructions following it) are stalled until the data-hazard clears
- ideal CPI = 0.5 (since 2 instructions can issue per clock cycle)

The ARM Cortex A8 Pipeline (pg. 232)

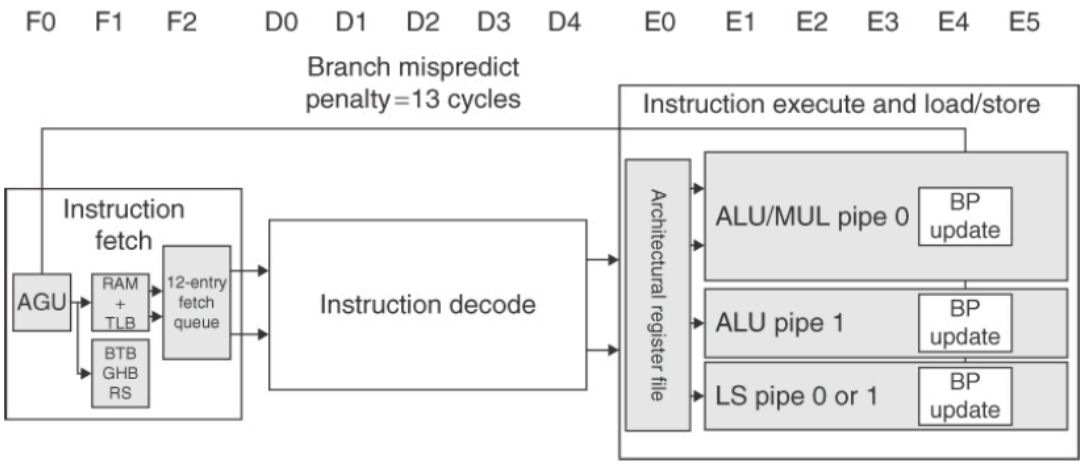

Figure 3.36 The basic structure of the A8 pipeline is 13 stages. Three cycles are used for instruction fetch and four for instruction decode, in addition to a five-cycle integer pipeline. This yields a 13-cycle branch misprediction penalty. The instruction fetch unit tries to keep the 12-entry instruction queue filled.

The ARM Cortex A8 Instruction-Decode Pipeline (pg. 232)

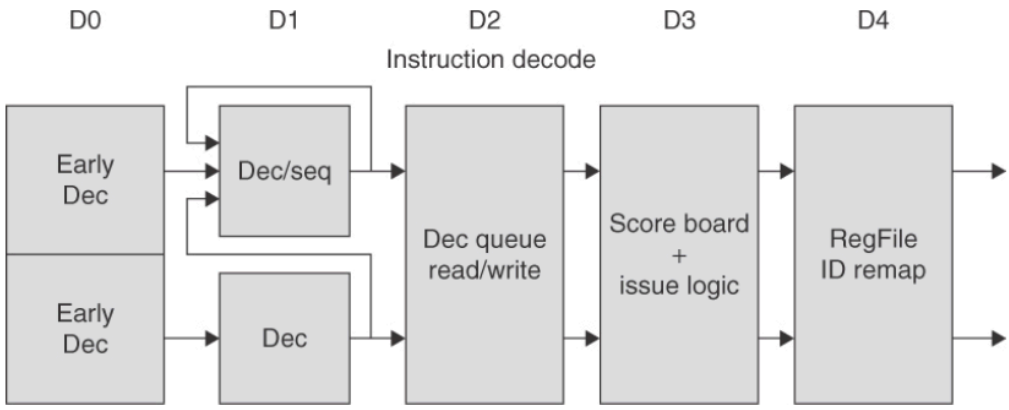

Figure 3.37 The five-stage instruction decode of the A8. In the first stage, a PC produced by the fetch unit (either from the branch target buffer or the PC incrementer) is used to retrieve an 8-byte block from the cache. Up to two instructions are decoded and placed into the decode queue; if neither instruction is a branch, the PC is incremented for the next fetch. Once in the decode queue, the scoreboard logic decides when the instructions can issue. In the issue, the register operands are read; recall that in a simple scoreboard, the operands always come from the registers. The register operands and opcode are sent to the instruction execution portion of the pipeline.

The ARM Cortex A8 Execution Pipeline (pg. 234)

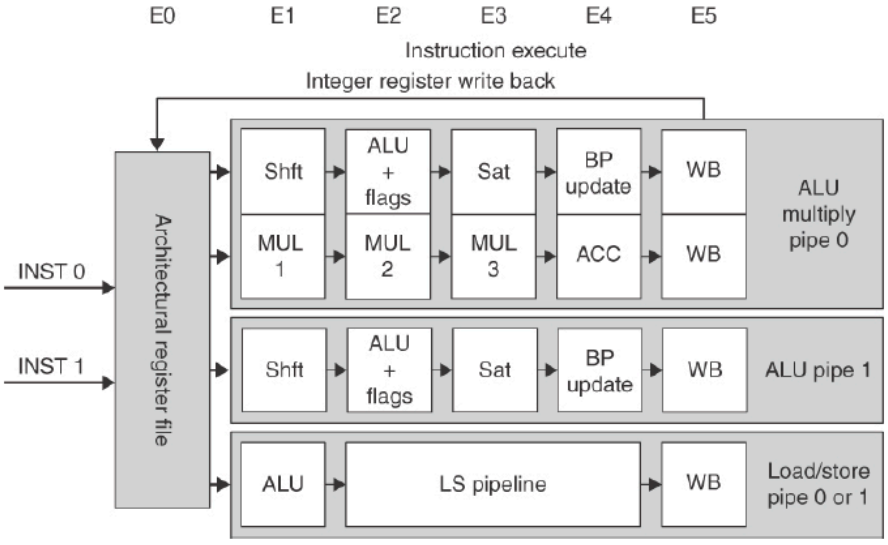

Figure 3.38 The five-stage instruction decode of the A8. Multiply operations are always performed in ALU pipeline 0.

The ARM Cortex A8 Processor (pg. 236)

- The next slide shows the performance, based on the CPI, for several programs
- The CPI per program is further broken down into components that cause the CPI
- Overall, L2 stalls contribute about < 0.5 CPI
- Overall, L1 stalls contribute about< 0.2 CPI
- Overall, pipeline stalls contribute about 1 CPI
- As a result of this analysis, Apple made the Apple A9 processor dynamically-scheduled
- A9 is a dual-issue, dynamically-scheduled machine with speculation
- up to 4 instructions (2 ALU, 1 Load/Store, and 1 branch) can begin execution per clock cycle
- A9 uses a better branch predictor, to lower effect of branch stalls
- Overall, A9 outperforms A9 by about 30 % on average

ARM Cortex A8 CPI and Pipeline Stalls (pg. 235)

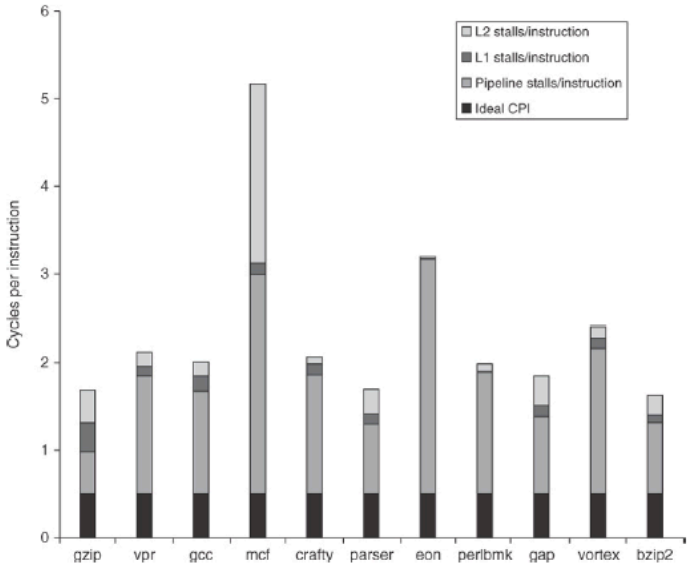

Figure 3.39 The estimated composition of the CPI on the ARM A8 shows that pipeline stalls are the primary addition to the base CPI. eon deserves some special mention, as it does integer-based graphics calculations (ray tracing) and has very few cache misses. It is computationally intensive with heavy use of multiples, and the single multiply pipeline becomes a major bottleneck. This estimate is obtained by using the L1 and L2 miss rates and penalties to compute the L1 and L2 generated stalls per instruction. These are subtracted from the CPI measured by a detailed simulator to obtain the pipeline stalls. Pipeline stalls include all three hazards plus minor effects such as way misprediction.

ARM Cortex A8 and Apple A9 Performance (pg. 236)

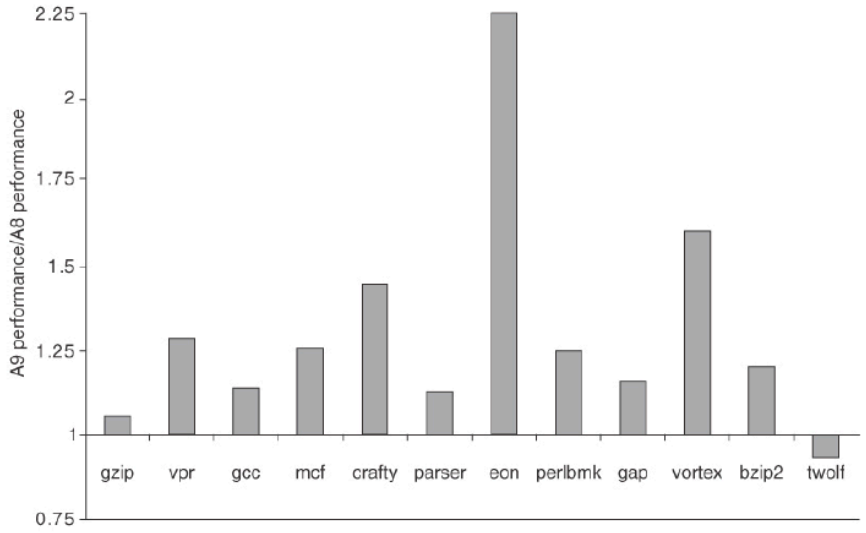

Figure 3.40 The performance ratio for the A9 compared to the A8, both using a 1 GHz clock and the same size caches for L1 and L2, shows that the A9 is about 1.28 times faster. Both runs use a 32 KB primary cache and a 1 MB secondary cache, which is 8-way set associative for the A8 and 16-way for the A9. The block sizes in the caches are 64 bytes for the A8 and 32 bytes for the A9. As mentioned in the caption of Figure 3.39, eon makes intensive use of integer multiply, and the combination of dynamic scheduling and a faster multiply pipeline significantly improves performance on the A9. twolf experiences a small slowdown, likely due to the fact that its cache behavior is worse with the smaller L1 block size of the A9.

The Intel Core i7 (pg. 236)

- The Intel Core i7 is an aggressive multiple-issue dynamically-scheduled machine with a reasonably deep pipeline: up to 6 issues per clock cycle
- Goal: high instruction throughput combining multiple issue with high clock rates
- Main Steps shown in Fig. 3.41:
- 1) Instruction Fetch: IF unit fetches 16 bytes from cache per clock cycle: it uses a multilevel branch prediction buffer and branch return address stack, to lower branch stalls; a branch mis-prediction causes a 15 clock cycle penalty
- 2) 16 bytes placed in predecode instruction buffer: breaks 16 bytes into individual CISC **80x86** instructions. One **80x86** instruction can vary in length from 1 up to 17 bytes (remember that CISC instructions are difficult to pipeline).
- Also performs Macro-op fusion, where 2 instructions (ie compare & branch) may be fused into 1 instruction
- Individual instructions placed in an 18-entry instruction queue
- 3) Micro-OP Decode**: 80x86** instructions are translated into pipelinable RISC micro-ops. There is a 28-entry micro-op buffer.

The Intel Core i7 (pg. 236)

- 4) Loop stream detection and microfusion: if a loop of <= 28 instructions is detected in the microop buffer, the loop will issue directly from the microop buffer, bypassing the IF and ID stages. Microfusion combines pairs of instructions (ie load and ALU) and issues them to one shared reservation station, where they can execute independently. (Since then, it has been shown that microfusion has little benefits)
- 5) Basic Instruction Issue: look up operands in registers, perform register renaming, generate an entry in the reorder buffer if necessary, send instruction(s) to reservation station
- 6) A Centralized reservation-station with 36 entries is shared by 6 functional units. Up to 6 microops can start execution per clock cycle
- 7) Microops complete execution, results are sent back to ID stage and reorder buffer and to any waiting reservation station
- 8) when 1 or more instructions at the head of the reorder buffer are labelled complete, the results are written into the register file and the entries in the reorder buffer are cleared

The Intel Core i7 (pg. 236)

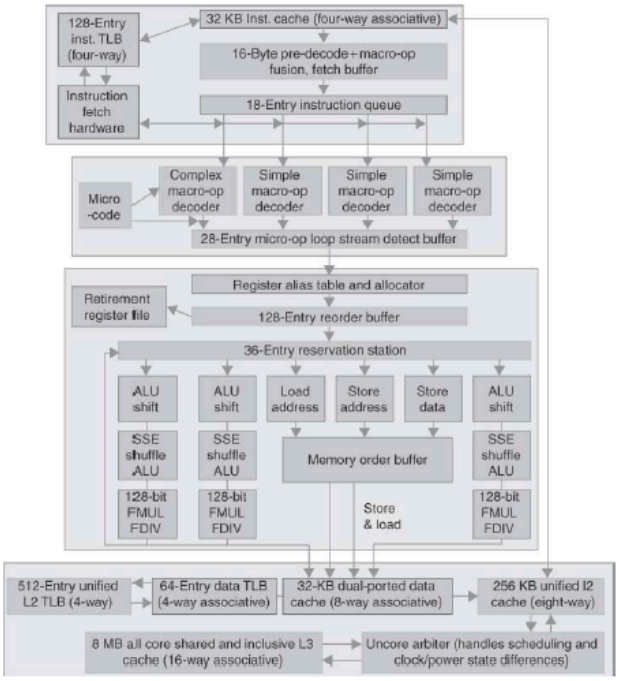

Figure 3.41 The Intel Core i7 pipeline structure shown with the memory system components. The total pipeline depth is 14 stages, with branch mispredictions costing 17 cycles. There are 48 load and 32 store buffers. The six independent functional units can each begin execution of a ready micro-op in the same cycle.

The Intel Core i7 Performance (pg. 239)

- Textbook examines single-thread performance
- next slide shows 'wasted work' when no instructions can issue
- about 3% of LOADs are delayed due to structural hazard on Reservation-Station
- most losses come from branch prediction error (we flush the pipeline), or cache missses
- cost of one branch misprediction = 15 clock cycles * (6 issues per cc) = 90 instructions
- cost of one L1 cache miss is 10 clock cycles * (6 issues per cc) = 60 instructions
- cost of one L2 cache miss is 30 clock cycles * (6 issues per cc) = 180 instructions
- cost of one L3 cache miss is 130 clock cycles * (6 issues per cc) = 780 instructions
- multi-threading allows the processor to switch threads, but the reservation-stations have finite depth and they will fill-up, eventually causing structural hazards
- Fig 3.42 shows fraction of instructions whose results are killed due to pipeline flushing

Core i7 Performance – Fraction of Wasted Issues (pg. 239)

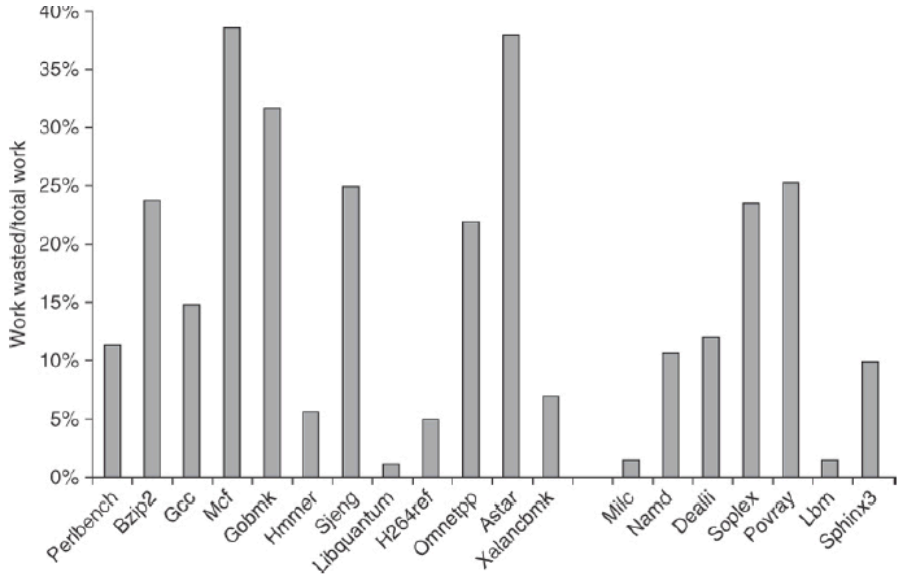

Figure 3.42 The amount of "wasted work" is plotted by taking the ratio of dispatched micro-ops that do not graduate to all dispatched micro-ops. For example, the ratio is 25% for sjeng, meaning that 25% of the dispatched and executed micro-ops are thrown away. The data in this section were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University.

The Intel Core i7 Performance (pg. 239)

- Fig 3.43 shows the overall CPI for the 19 benchmark programs
- ideal instructions-per-clock-cycle = 6 instructions per clock cycle
- so the ideal clock-cycles-per-instruction is 1/6 clock-cycles
- the ideal CPU ignores branch mis-predictions and cache misses
- a real machine like the CORE i7 has very large penalties (90 instructions for branch mis-prediction, and 60, 180, 780 instructions for L1, L2 and L3 cache misses)
- the overall CPI is about 0.5, which is not bad considering the large penalties

Core i7 Performance CPI (pg. 239)

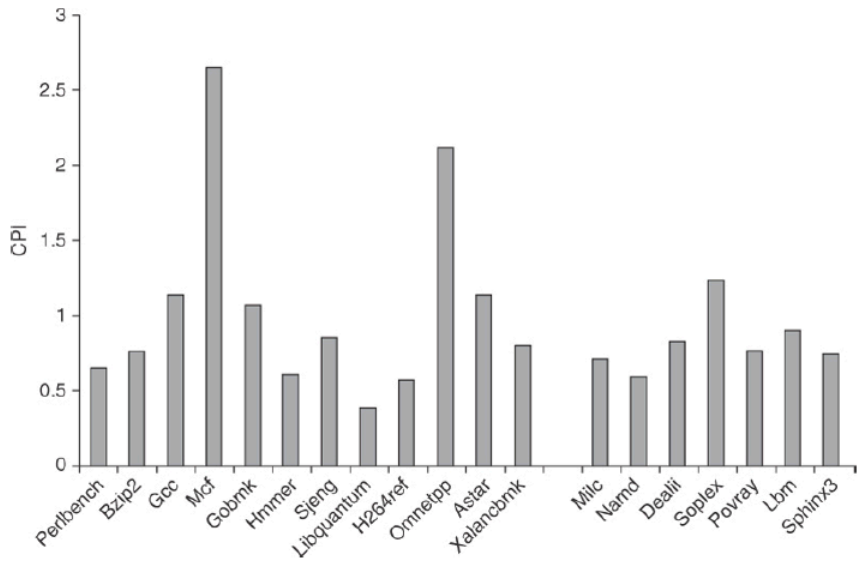

Figure 3.43 The CPI for the 19 SPECCPU2006 benchmarks shows an average CPI for 0.83 for both the FP and integer benchmarks, although the behavior is quite different. In the integer case, the CPI values range from 0.44 to 2.66 with a standard deviation of 0.77, while the variation in the FP case is from 0.62 to 1.38 with a standard deviation of 0.25. The data in this section were collected by Professor Lu Peng and Ph.D. student Ying Zhang, both of Louisiana State University.

Area	Specific characteristic	Intel i7 920 Four cores, each with FP	ARM A8 One core, no FP	Intel Atom 230 One core, with FP
Physical chip properties	Clock rate	2.66 GHz	1 GHz	1.66 GHz
	Thermal design power	130 W	2 W	4 W
	Package	1366-pin BGA	522-pin BGA	437-pin BGA
Memory system	TLB	Two-level All four-way set associative 128 I/64 D 512 L2	One-level fully associative 32 I/32 D	Two-level All four-way set associative 16 I/16 D 64 L2
	Caches	Three-level 32 KB/32 KB 256 KB 2–8 MB	Two-level 16/16 or 32/32 KB 128 KB–1MB	Two-level 32/24 KB 512 KB
	Peak memory BW	17 GB/sec	12 GB/sec	8 GB/sec
Pipeline structure	Peak issue rate	4 ops/clock with fusion	2 ops/clock	2 ops/clock
	Pipeline scheduling	Speculating out of order	In-order dynamic issue	In-order dynamic issue
	Branch prediction	Two-level	Two-level 512-entry BTB 4K global history 8-entry return stack	Two-level

Figure 3.44 An overview of the four-core Intel i7 920, an example of a typical Arm A8 processor chip (with a 256 MB L2, 32K L1s, and no floating point), and the Intel ARM 230 clearly showing the difference in design philosophy between a processor intended for the PMD (in the case of ARM) or netbook space (in the case of Atom) and a processor for use in servers and high-end desktops. Remember, the i7 includes four cores, each of which is several times higher in performance than the one-core A8 or Atom. All these processors are implemented in a comparable 45 nm technology.

Core i7 Performance and Energy-Efficiency (pg. 239)

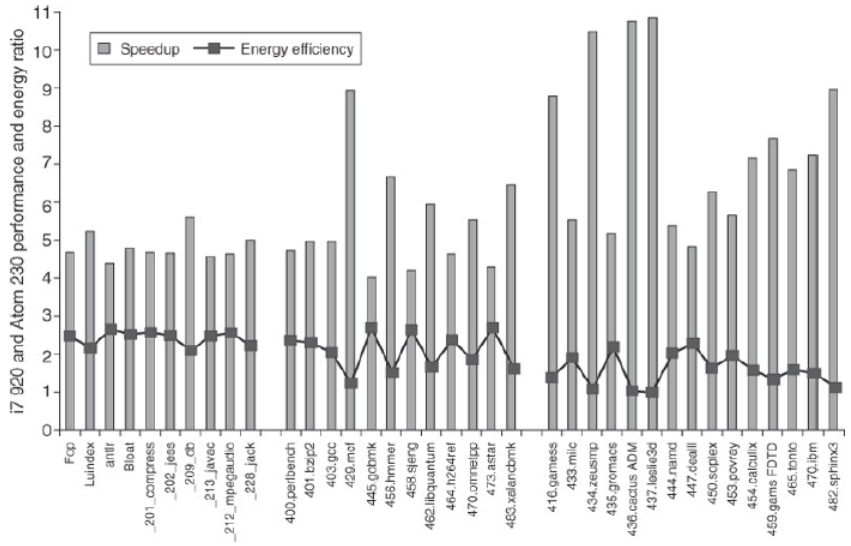

Figure 3.45 The relative performance and energy efficiency for a set of single-threaded benchmarks shows the i7 920 is 4 to over 10 times faster than the Atom 230 but that it is about 2 times *less* power efficient on average! Performance is shown in the columns as i7 relative to Atom, which is execution time (i7)/execution time (Atom). Energy is shown with the line as Energy (Atom)/Energy (i7). The i7 never beats the Atom in energy efficiency, although it is essentially as good on four benchmarks, three of which are floating point. The data shown here were collected by Esmaeilzadeh et al. [2011]. The SPEC benchmarks were compiled with optimization on using the standard Intel compiler, while the Java benchmarks use the Sun (Oracle) Hotspot Java VM. Only one core is active on the i7, and the rest are in deep power saving mode. Turbo Boost is used on the i7, which increases its performance advantage but slightly decreases its relative energy efficiency.

Intel Itanium and CORE i7 (pg. 244)

Processor	Clock rate	SPECCInt2006 base	SPECCFP2006 baseline
Intel Pentium 4 670	3.8 GHz	11.5	12.2
Intel Itanium -2	1.66 GHz	14.5	17.3
Intel i7	3.3 GHz	35.5	38.4

Figure 3.46 Three different Intel processors vary widely. Although the Itanium processor has two cores and the i7 four, only one core is used in the benchmarks.

IBM Power Cores (pg. 247)

	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.

ILP, Core-Processors and Multithreading (pg. 229)

- There was significant progress in **Instruction-Level-Parallelism** (ILP) over last decade, with the Itanium, and multiple-issue dynamically-scheduled machines
- However, ILP cannot hide the real penalties associated with branch mis-predictions and L1, L2 and L3 cache misses
- **Multithreading** allows multiple threads to **share one processing core** in an overlapping fashion, with rapid thread switches
- When one thread encounters a large penalty (ie 2 clock cycles up to 100s iof clock cycles), then the treads can be quickly switched by the core
- Many new processors include multiple cores with multi-threading in each core
- Each thread maintains its own registers, program counter, and virtual-memory page table
- The core supports the ability to rapidly switch between threads, typically within 1 clock cycle
- It looks like multiple cores with multithreading are here to stay