4.11 Real stuff: The ARM Cortex-A53 and Intel Core i7 pipelines

> This section has been set as optional by your instructor.

The figure below describes the two microprocessors we examine in this section, whose targets are the two endpoints of the post-PC Era.

Figure 4.11.1: Specification of the ARM Cortex-A53 and the Intel Core i7 920 (COD Figure 4.72).

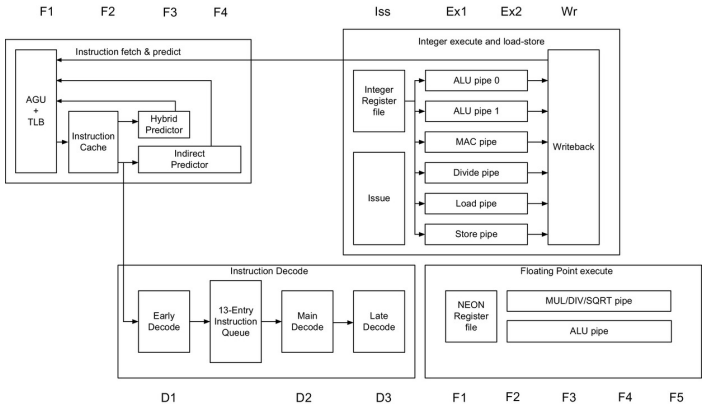
Processor	ARM A53	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal design power	100 milliWatts (1 core @ 1 GHz)	130 Watts
Clock rate	1.5 GHz	2.66 GHz
Cores/Chip	4 (configurable)	4
Floating point?	Yes	Yes
Multiple Issue?	Dynamic	Dynamic
Peak instructions/clock cycle	2	4
Pipeline Stages	8	14
Pipeline schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch prediction	Hybrid	2-level
1st level caches/core	16-64 KiB I, 16-64 KiB D	32 KiB I, 32 KiB D
2nd level cache/core	128–2048 KiB (shared)	256 KiB (per core)
3rd level cache (shared)	(platform dependent)	2–8 MiB

The ARM Cortex-A53

The ARM Cortex-A53 runs at 1.5 GHz with an eight-stage pipeline and executes the ARMv8 instruction set. It uses dynamic multiple issue, with two instructions per clock cycle. It is a static in-order pipeline, in that instructions issue, execute, and commit in order. The pipeline consists of three sections for instruction fetch, instruction decode, and execute. The figure below shows the overall pipeline.

Figure 4.11.2: The Cortex-A53 pipeline (COD Figure 4.73).

The first three stages fetch instructions into a 13-entry instruction queue. The *Address Generation Unit* (AGU) uses a *Hybrid Predictor*, *Indirect Predictor*, and a *Return Stack* to predict branches to try to keep the instruction queue full. Instruction decode is three stages and instruction execution is three stages. With two additional stages for floating point and SIMD operations.



The first three stages fetch two instructions at a time and try to keep a 13-entry instruction queue full. It uses a 6k-bit hybrid conditional branch predictor, a 256-entry indirect branch predictor, and an 8-entry return address stack to predict future function returns. The prediction of indirect branches takes an additional pipeline stage. This design choice will incur extra latency if the instruction queue cannot decouple the decode and execute stages from the fetch stage, primarily in the case of a branch misprediction or an instruction cache miss. When the branch prediction is wrong, it empties the pipeline, resulting in an eight-clock cycle misprediction penalty.

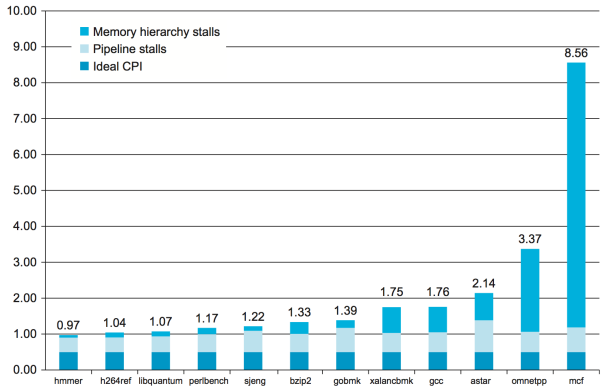
The decode stages of the pipeline determine if there are dependences between a pair of instructions, which would force sequential execution, and in which pipeline of the execution stages to send the instructions.

The instruction execution section primarily occupies three pipeline stages and provides one pipeline for load instructions, one pipeline for store instructions, two pipelines for integer arithmetic operations, and separate pipelines for integer multiply and divide operations. Either instruction from the pair can be issued to the load or store pipelines. The execution stages have full forwarding between the pipelines.

Floating-point and SIMD operations add two more pipeline stages to the instruction execution section and feature one pipeline for multiply/divide/square root operations and one pipeline for other arithmetic operations.

The figure below shows the CPI of the Cortex-A53 using the SPEC2006 benchmarks. While the ideal CPI is 0.5, the best case achieved is 1.0, the median case is 1.3, and the worst case is 8.6. For the median case, 60% of the stalls are due to the pipelining hazards and 40% are stalls due to the memory hierarchy. Pipeline stalls are caused by branch mispredictions, structural hazards, and data dependencies between pairs of instructions. Given the static pipeline of the Cortex-A53, it is up to the compiler to try to avoid structural hazards and data dependences.

Figure 4.11.3: CPI on ARM Cortex-A53 for the SPEC2006 integer benchmarks. (COD Figure 4.74).



Elaboration

The Cortex-A53 is a configurable core that supports the ARMv8 instruction set architecture. It is delivered as an IP (Intellectual Property) core. IP cores are the dominant form of technology delivery in the embedded, personal mobile device, and related markets; billions of ARM and MIPS processors have been created from these IP cores.

Note that IP cores are different than the cores in the Intel i7 multicore computers. An IP core (which may itself be a multicore) is designed to be incorporated with other logic (hence it is the "core" of a chip), including application-specific processors (such as an encoder or decoder for video), I/O interfaces, and memory interfaces, and then fabricated to yield a processor optimized for a particular application. Although the processor core is almost identical logically, the resultant chips have many differences. One parameter is the size of the L2 cache, which can vary by a factor of 16.

PARTICIPATION ACTIVITY 4.11.1: ARM Cortex-A53 pipeline design.

1) The instruction execution section has _____ stages.
   - 3
   - 8

2) The instruction fetch section collects _____ instructions at a time to enter the pipeline.
   - 3
   - 2
   - 13

3) The AGU uses a branch predictor comprised of a Hybrid Predictor, an Indirect Predictor, and a _____.
   - Load and Store
   - Return Stack

4) An incorrect branch prediction results in a _____-clock cycle misprediction penalty.
   - 256
   - 13
   - 8

5) The instruction execution section provides _____ pipeline(s) for integer arithmetic operations.
   - 1
   - 2
   - 3

The Intel Core i7 920

x86 microprocessors employ sophisticated pipelining approaches, using both dynamic multiple issue and dynamic pipeline scheduling with out-of-order execution and speculation for their pipelines. These processors, however, are still faced with the challenge of implementing the complex x86 instruction set, described in COD Chapter 2 (Instructions: Language of the Computer). Intel fetches x86 instructions and translates them into internal ARMv8-like instructions, which Intel calls *micro-operations*. The micro-operations are then executed by a sophisticated, dynamically scheduled, speculative pipeline capable of sustaining an execution rate of up to six micro-operations per clock cycle. This section focuses on that micro-operation pipeline.

When we consider the design of such processors, the design of the functional units, the cache and register file, instruction issue, and overall pipeline control become intermingled, making it difficult to separate the datapath from the pipeline. Because of this, many engineers and

researchers have adopted the term *microarchitecture* to refer to the detailed internal architecture of a processor.

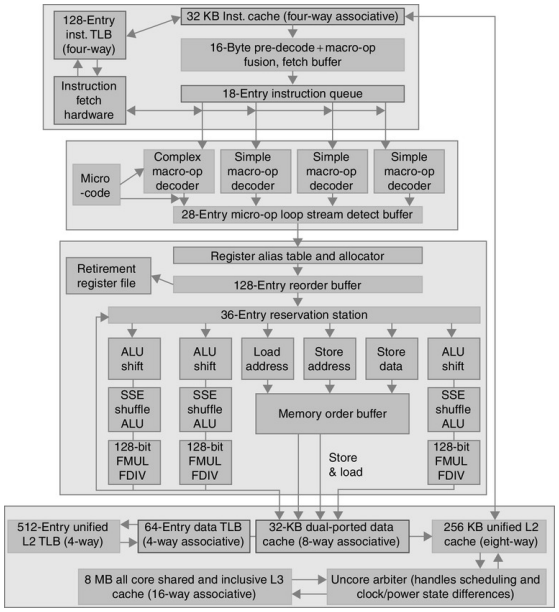
Microarchitecture: The organization of the processor, including the major functional units, their interconnection, and control.

The Intel Core i7 uses a scheme for resolving antidependences and incorrect speculation that uses a reorder buffer together with register renaming. Register renaming explicitly renames the *architectural registers* in a processor (16 in the case of the 64-bit version of the x86 architecture) to a larger set of physical registers. The Core i7 uses register renaming to remove antidependences. Register renaming requires the processor to maintain a map between the architectural registers and the physical registers, indicating which physical register is the most current copy of an architectural register. By keeping track of the renamings that have occurred, register renaming offers another approach to recovery in the event of incorrect speculation: simply undo the mappings that have occurred since the first incorrectly speculated instruction. This undo will cause the state of the processor to return to the last correctly executed instruction, keeping the correct mapping between the architectural and physical registers.

Architectural registers: The instruction set of visible registers of a processor; for example, in LEGv8, these are the 32 integer and 32 floating-point registers.

Figure 4.11.4: The Core i7 pipeline with memory components (COD Figure 4.75).

The total pipeline depth is 14 stages, with branch mispredictions costing 17 clock cycles. This design can buffer 48 loads and 32 stores. The six independent units can begin execution of a ready micro-operation each clock cycle.



The figure above shows the overall organization and pipeline of the Core i7. Below are the eight steps an x86 instruction goes through for execution.

1. Instruction fetch—The processor uses a multilevel branch target buffer to achieve a balance between speed and prediction accuracy. There is also a return address stack to speed up function return. Mispredictions cause a penalty of about 15 cycles. Using the predicted address, the instruction fetch unit fetches 16 bytes from the instruction cache.
2. The 16 bytes are placed in the predecode instruction buffer—The predecode stage transforms the 16 bytes into individual x86 instructions. This predecode is nontrivial since the length of an x86 instruction can be from 1 to 15 bytes and the predecoder must look through a number of bytes before it knows the instruction length. Individual x86 instructions are placed into the 18-entry instruction queue.
3. Micro-op decode—Individual x86 instructions are translated into micro-operations (micro-ops). Three of the decoders handle x86 instructions that translate directly into one micro-op. For x86 instructions that have more complex semantics, there is a microcode engine that is used to produce the micro-op sequence; it can produce up to four micro-ops every cycle and continues until the necessary micro-op sequence has been generated. The micro-ops are placed according to the order of the x86 instructions in the 28-entry micro-op buffer.
4. The micro-op buffer performs *loop stream detection*—If there is a small sequence of instructions (less than 28 instructions or 256 bytes in length) that comprises a loop, the loop stream detector will find the loop and directly issue the micro-ops from the buffer, eliminating the need for the instruction fetch and instruction decode stages to be activated.
5. Perform the basic instruction issue—Looking up the register location in the register tables, renaming the registers, allocating a reorder buffer entry, and fetching any results from the registers or reorder buffer before sending the micro-ops to the reservation stations.
6. The i7 uses a 36-entry centralized reservation station shared by six functional units. Up to six micro-ops may be dispatched to the functional units every clock cycle.
7. The individual function units execute micro-ops and then results are sent back to any waiting reservation station as well as to the register retirement unit, where they will update the register state, once it is known that the instruction is no longer speculative. The entry corresponding to the instruction in the reorder buffer is marked as complete.
8. When one or more instructions at the head of the reorder buffer have been marked as complete, the pending writes in the register retirement unit are executed, and the instructions are removed from the reorder buffer.

Elaboration

Hardware in the second and fourth steps can combine or fuse operations together to reduce the number of operations that must be performed. Macro-op fusion in the second step takes x86 instruction combinations, such as compare followed by a branch, and fuses them into a single operation. Microfusion in the fourth step combines micro-operation pairs such as

load/ALU operation and ALU operation/store and issues them to a single reservation station (where they can still issue independently), thus increasing the usage of the buffer. In a study of the Intel Core architecture, which also incorporated microfusion and macrofusion, Bird et al. [2007] discovered that microfusion had little impact on performance, while macrofusion appears to have a modest positive impact on integer performance and little impact on floating-point performance.

PARTICIPATION ACTIVITY 4.11.2: Intel Core i7 pipeline design.

1) Register renaming renames physical registers to a set of architectural registers.
   - O True
   - O False

2) The instruction fetch unit fetches 16 bytes from the instruction cache.
   - O True
   - O False

3) There are three micro-operation decoders that convert x86 instructions to ARMv8-like instructions.
   - O True
   - O False

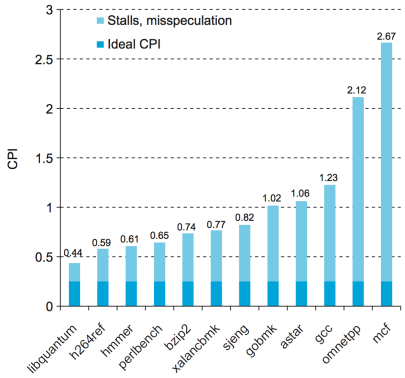
4) The loop stream detection buffer identifies a loop in a sequence of x86 instructions and directly issues micro-operations from the buffer that correspond to the loop.
   - O True
   - O False

5) Micro-operations are executed at a rate of 36 micro-operations per clock cycle.
   - O True
   - O False

Performance of the Intel Core i7 920

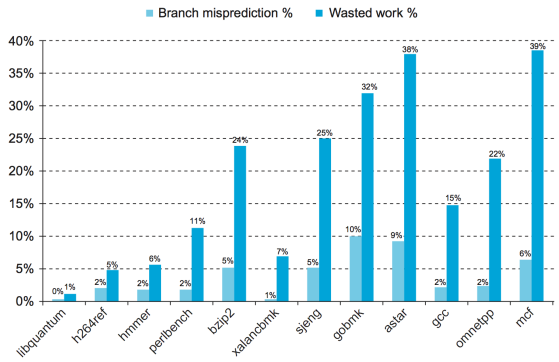
The figure below shows the CPI of the Intel Core i7 for each of the SPEC2006 benchmarks. While the ideal CPI is 0.25, the best case achieved is 0.44, the median case is 0.79, and the worst case is 2.67.

Figure 4.11.5: CPI of Intel Core i7 920 running SPEC2006 integer benchmarks (COD Figure 4.76).



Although it is difficult to differentiate between pipeline stalls and memory stalls in a dynamic out-of-order execution pipeline, we can show the effectiveness of branch prediction and speculation. The figure below shows the percentage of branches mispredicted and the percentage of the work (measured by the numbers of micro-ops dispatched into the pipeline) that does not retire (that is, their results are annulled) relative to all micro-op dispatches. The min, median, and max of branch mispredictions are 0%, 2%, and 10%. For wasted work, they are 1%, 18%, and 39%.

Figure 4.11.6: Percentage of branch mispredictions and wasted work due to unfruitful speculation of Intel Core i7 920 running SPEC2006 integer benchmarks (COD Figure 4.77).



The wasted work in some cases closely matches the branch misprediction rates, such as for gobmk and astar. In several instances, such as mcf, the wasted work seems relatively larger than the misprediction rate. This divergence is likely due to the memory behavior. With very high data cache miss rates, mcf will dispatch many instructions during an incorrect speculation as long as sufficient reservation stations are available for the stalled memory references. When a branch among the many speculated instructions is finally mispredicted, the micro-ops corresponding to all these instructions will be flushed.

Understanding program performance

The Intel Core i7 combines a 14-stage pipeline and aggressive multiple issue to achieve high performance. By keeping the latencies for back-to-back operations low, the impact of data dependences is reduced. What are the most serious potential performance bottlenecks for programs running on this processor? The following list includes some possible performance problems, the last three of which can apply in some form to any high-performance pipelined processor.

- The use of x86 instructions that do not map to a few simple micro-operations
- Branches that are difficult to predict, causing misprediction stalls and restarts when speculation fails
- Long dependences—typically caused by long-running instructions or the **memory hierarchy**—that lead to stalls
- Performance delays arising in accessing memory (see COD Chapter 5 (Large and Fast: Exploiting Memory Hierarchy)) that cause the processor to stall

PARTICIPATION ACTIVITY 4.11.3: ARM Cortex-A53 vs. Intel Core i7.

Match the term to the processor type.

1) Micro-operations
   - ARM Cortex-A53
   - Intel Core i7
   - Both

2) Static in-order pipeline scheduling
   - ARM Cortex-A53
   - Intel Core i7
   - Both

3) 14-stage pipeline
   - ARM Cortex-A53
   - Intel Core i7
   - Both

4) Dynamic multiple issue
   - ARM Cortex-A53
   - Intel Core i7
   - Both

5) Predecode stage
   - ARM Cortex-A53
   - Intel Core i7
   - Both

6) Dynamic speculative pipeline

scheduling

- ARM Cortex-A53
- Intel Core i7
- Both

Provide feedback on this section