LECTURA 2

1 Introduction

← INICIO

Modern computer systems built from the most sophisticated microprocessors and extensive memory hierarchies achieve their high performance through a combination of dramatic improvements in technology and advances in computer architecture. Advances in technology have resulted in exponential growth rates in raw speed (i.e., clock frequency) and in the amount of logic (number of transistors) that can be put on a chip. Computer architects have exploited these factors in order to further enhance performance using architectural techniques, which are the main subject of this book.

Microprocessors are over 30 years old: the Intel 4004 was introduced in 1971. The functionality of the 4004 compared to that of the mainframes of that period (for example, the IBM System/370) was minuscule. Today, just over thirty years later, workstations powered by engines such as (in alphabetical order and without specific processor numbers) the AMD Athlon, IBM PowerPC, Intel Pentium, and Sun UltraSPARC can rival or surpass in both performance and functionality the few remaining mainframes and at a much lower cost. Servers and supercomputers are more often than not made up of collections of microprocessor systems.

It would be wrong to assume, though, that the three tenets that computer architects have followed, namely *pipelining*, *parallelism*, and the *principle of locality*, were discovered with the birth of microprocessors. They were all at the basis of the design of previous supercomputers. The advances in technology made their implementations more practical and spurred further refinements. The microarchitectural techniques that are presented in this book rely on these three tenets to translate the architectural specification – the *instruction set architecture* – into a partition of *static* blocks whose *dynamic* interaction leads to their intended behavior.

In the next few pages, we give an extremely brief overview of the advances in technology that have led to the development of current microprocessors. Then, the remainder of this chapter is devoted to defining performance and means to measure it.

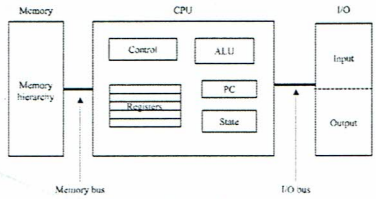

Figure 1.1. The von Neumann machine model.

1.1 A Quick View of Technological Advances

1.1.1 The von Neumann Machine Model

Modern microprocessors have sophisticated engineering features. Nonetheless, they still follow essentially the conventional von Neumann machine model shown in Figure 1.1. The von Neumann model of a stored program computer consists of four blocks:

- A *central processing unit* (CPU) containing an arithmetic-logical unit (ALU) that performs arithmetic and logical operations, registers whose main function is to be used for high-speed storage of operands, a control unit that interprets instructions and causes them to be executed, and a program counter (PC) that indicates the address of the next instruction to be executed (for some authors the ALU and the control unit constitute separate blocks).
- A *memory* that stores instructions, data, and intermediate and final results. Memory is now implemented as a hierarchy.
- An *input* that transmits data and instructions from the outside world to the memory.
- An *output* that transmits final results and messages to the outside world.

Under the von Neumann model, the *instruction execution cycle* proceeds as follows:

1. The next instruction (as pointed to by the PC) is fetched from memory.
2. The control unit decodes the instruction.
3. The instruction is executed. It can be an ALU-based instruction, a load from a memory location to a register, a store from a register to the memory, or a testing condition for a potential branch.
4. The PC is updated (incremented in all cases but a successful branch).
5. Go back to step 1.

Of course, in the more than sixty years of existence of stored program computers, both the contents of the blocks and the basic instruction execution cycle sequence have been optimized thoroughly. In this book, we shall look primarily at what can be found on a microprocessor chip. At the outset, a microprocessor chip contained the CPU. Over the years, the CPU has been enhanced so that it could be pipelined. Several functional units have replaced the single ALU. Lower levels of the memory hierarchy (caches) have been integrated on the chip. Recently, several microprocessors and their low-level caches coexist on a single chip, forming *chip multiprocessors* (CMPs). Along with these (micro)architectural advances, the strict sequential execution cycle has been extended so that we can have several instructions proceed concurrently in each of the basic steps. In Chapters 2 through 6, we examine the evolution of single processor chips; Chapters 7 and 8 are devoted to multiprocessors.

1.1.2 Technological Advances

The two domains in which technological advances have had the most impact on the performance of microprocessor systems have been the increases in clock frequency and the shrinking of the transistor features leading to higher logic density. In a simplistic way, we can state that increasing clock frequency leads to faster execution; and more logic yields implementation of more functionality on the chip.

Figure 1.2 shows the evolution of the clock frequencies of the Intel microprocessors from the inception of the 4004 in 1971 to the chip multiprocessors of 2003 and beyond. From 1971 to 2002, we see that processor raw speeds increased at an exponential rate. The frequency of the 4004, a speck on the figure that cannot be seen because of the scaling effect, was 1.08 MHz, a factor of 3,000 less than the 3.4 GHz of the Pentium 4. This corresponds roughly to a doubling of the frequency every 2.5 years. To be fair, though, although the Pentium 4 was the fastest processor in 2003, there were many mainframes in 1971 that were faster than the 4004. Some supercomputers in the late 1960s, such as the IBM System 360/91 and Control Data 6600/7600 – two machines that will be mentioned again in this book – had frequencies of over 100 MHz. However, if they were two orders of magnitude faster than the 4004, they were about six orders of magnitude more expensive. After 2003, the frequencies stabilize in the 3 GHz range. We shall see very shortly the reason for such a plateau.

The number of transistors that can be put on a chip has risen at the same pace as the frequency, but without any leveling off. In 1965, Gordon Moore, one of the founders of Intel, predicted that "the number of transistors on a given piece of silicon would double every couple of years." Although one can quibble over the "couple of years" by a few months or so, this prediction has remained essentially true and is now known as *Moore's law*. Figure 1.3 shows the exponential progression of transistors in the Intel microprocessors (notice the log scale). There is a growth factor of over 2,000 between the 2,300 transistors of the Intel 4004 and the 1.7 billion transistors of the 2006 Intel dual-core Itanium (the largest number of transistors in the

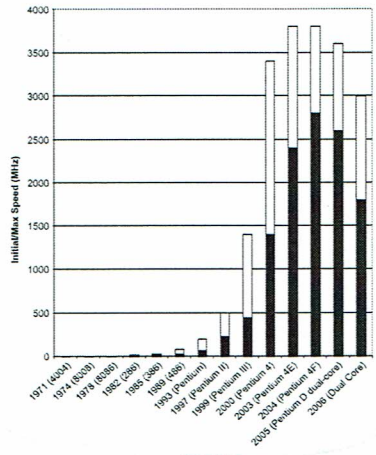

Figure 1.2. Evolution of Intel microprocessors speeds (black bars: frequency at introduction; white bars: peak frequency).

microprocessors of Figure 1.2 is a little over half a billion for a dual-core Extreme). It is widely believed that Moore's law can be sustained until 2025.

Besides allowing implementers to use more logic (i.e., provide more functionality) or more storage (i.e., mitigate the imbalance between processor speed and memory latency), consequences of Moore's law include reduced costs and better reliability.

However, the exponential growths in speed and logic density do not come without challenges. First, of course, the feature size of the manufacturing process, which is closely related to transistor sizes, cannot be reduced indefinitely, and the number of transistors that can be put on a chip is physically bounded. Second, and of primary importance now, is the amount of power that is dissipated. Figure 1.4 illustrates this point vividly (again, notice the log scale). Third, at multigigahertz frequencies, the on-chip distance (i.e., sum of wire lengths) between producer and consumer of

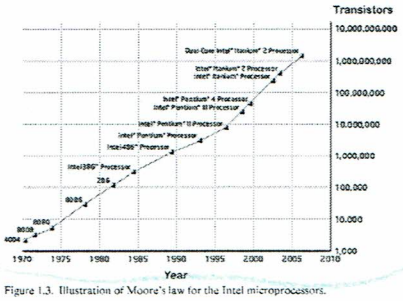

Figure 1.3. Illustration of Moore's law for the Intel microprocessors.

information becomes a factor and influences microarchitectural design decisions. We will elaborate more on these design problems in Chapter 9.

Now we can come back to the source of the leveling of speeds in Figure 1.2 after 2002. The power requirements as shown in Figure 1.4 and the growth in the number of transistors as shown by Moore's law limit the speeds that can be attained, because the switching (or dynamic) power dissipation is related to the frequency, and the

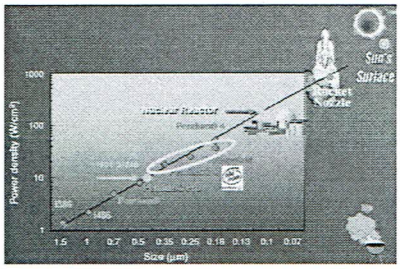

Figure 1.4. Power dissipation in selected Intel microprocessors (from Fred Pollack, Micro32 keynote).

static power (leakage) dissipation is related to the number of transistors on the chip. Therefore, Intel and other manufacturers have capped the frequencies at their 2003 level. An exception is the IBM Power6, introduced in 2007, which is clocked at slightly over 4.7 GHz. However, even with this latest development, the exponential speed increase has stopped. The persistence of Moore's law and the resulting increase in the amount of logic that can be put on a chip have led to the emergence of CMPs.

FIN

1.2 Performance Metrics

Raw speed and the number of transistors on a chip give a good feel for the progress that has been made. However, in order to assess the performance of computer systems, we need more precise metrics related to the execution of programs.

From a user's viewpoint, what is important is how fast her programs execute. Often, though, this time of execution depends on many factors that have little to do with the processor on which the programs execute. For example, it may depend on the operating system (which version of Windows or Linux), the compiler (how well optimized it is), and network conditions (if the user's workstation is sharing I/O devices with other workstations on a local area network). Moreover, the programs that user A is interested in can be completely different from those of interest to programmer B. Thus, to perform meaningful architectural comparisons between processors and their memory hierarchies or to assess the benefits of some components of the design, we need:

- Metrics that reflect the underlying architecture in a program-independent fashion.
- A suite of programs, or *benchmarks*, that are representative of the intended load of the processor.

Neither of these goals is easy to attain in a scientific manner.

1.2.1 Instructions Per Cycle (*IPC*)

Metrics to assess the microarchitecture of a processor and of its memory hierarchy, (i.e., caches and main memory), should be defined independently of the I/O subsystem. The execution time of a program whose code and data reside in the memory hierarchy will be denoted EX_{CPU} to indicate the context in which it is valid.

The execution time of a program depends on the number of instructions executed and the time to execute each instruction. In a first approximation, if we assume that all instructions take the same time to execute, we have

$$EX_{CPU} = \textit{Number of instructions} \times \textit{Time to execute one instruction} \quad (1)$$

Now, *Time to execute one instruction* can be expressed as a function of the cycle time (the reciprocal of the clock frequency) and the number of cycles to execute an

3 Superscalar Processors

3.1 From Scalar to Superscalar Processors

In the previous chapter we introduced a five-stage pipeline. The basic concept was that the instruction execution cycle could be decomposed into nonoverlapping stages with one instruction passing through each stage at every cycle. This so-called *scalar* processor had an ideal throughput of 1, or in other words, ideally the number of instructions per cycle (*IPC*) was 1.

If we return to the formula giving the execution time, namely,

$$EX_{CPU} = Number\ of\ instructions \times CPI \times cycle\ time$$

we see that in order to reduce EX_{CPU} in a processor with the same ISA – that is, without changing the number of instructions, N – we must either reduce CPI (increase IPC) or reduce the cycle time, or both. Let us look at the two options.

The only possibility to increase the ideal IPC of 1 is to radically modify the structure of the pipeline to allow more than one instruction to be in each stage at a given time. In doing so, we make a transition from a scalar processor to a *superscalar* one. From the microarchitecture viewpoint, we make the pipeline *wider* in the sense that its representation is not linear any longer. The most evident effect is that we shall need several functional units, but, as we shall see, each stage of the pipeline will be affected.

The second option is to reduce the cycle time through an increase in clock frequency. In order to do so, each stage must perform less work. Therefore, a stage must be decomposed into smaller stages, and the overall pipeline becomes *deeper*.

Modern microprocessors are therefore both wider and deeper than the five-stage pipeline of the previous chapter. In order to study the design decisions that are necessary to implement the concurrency caused by the superscalar effect and the consequences of deeper pipelines, it is convenient to distinguish between the *front-end* and the *back-end* of the pipeline. The front-end, which corresponds to the IF and ID stages, now must fetch and decode several instructions at once. The number m of instructions brought (ideally) into the pipeline at each cycle defines the processor as an m-*way* superscalar. The back-end, which corresponds to the

EX, Mem, and WB stages, must execute and write back several instructions concurrently.

Superscalar microprocessors can be divided into two categories. In both cases, instructions proceed in the front-end in program order. In *in-order*, or *static*, superscalar processors, the instructions leave the front-end in strict program order and all data dependencies are resolved before the instructions are passed to the back-end. In *out-of-order*, or *dynamic*, superscalar processors, the instructions can leave the front-end and execute in the back-end before some of their program-order predecessors. In a dynamic superscalar, the WB stage, now called the *commit* stage, must be designed in such a way that the semantics of the program are respected, that is, the results must be stored in the order intended by the source code. Out-of-order processors are arguably more complex to design and to implement than in-order ones. It is estimated that for the same number of functional units, they require 30% more logic, and, naturally, the design complexity translates into longer time to market.

Theoretically, if we have an m-way superscalar with a clock frequency k times that of a scalar one with the same ISA, we should realize an mk speedup. While this theoretical speedup can be approached for small m and k, there are behavioral, design, and technological factors that limit the increase in m and k. We give here a nonexhaustive list of the most salient ones:

- In order to sustain the concurrent execution of many instructions, that is, have a large m, we need to uncover a large amount of *instruction-level parallelism* (ILP). This is especially difficult in an in-order processor, wherein instructions that have (RAW) dependencies cannot leave the front-end until the dependencies are resolved. In an out-of-order processor there is more latitude to find independent instructions, but it is at the cost of increased hardware and design complexity. While some researchers envisioned 8-way and even 16-way dynamic superscalars, there is no current implementation that is more than 6-way, and the trend is not towards increasing m.
- A second factor that limits m is that the back-end requires at least m functional units. Although implementing a large number, say n, of functional units is not limiting with respect to the amount of logic needed, the number of forwarding paths grows as n^2. Some of these paths will necessarily be long, and the wire lengths may prevent forwarding in a single cycle.
- Several factors limit improvements in cycle time. A first constraint, already mentioned, is that power dissipation increases with frequency. A second constraint is that pipeline registers (i.e., the stable storage between stages, must be written and read at each cycle), thus providing a physical lower bound on the cycle time.
- Deep pipelines, (i.e., those having a large k), have a significant drawback. The resolution of some speculative action, like a branch prediction, will be known only late in the pipeline. In the case of a misspeculation, recovery is delayed and may contribute to a loss in performance (lower IPC) compared to a shorter

pipeline. The trend to deep pipes, exemplified by the Pentium 4 with a branch misprediction penalty of 20 cycles (the pipeline was up to 31 stages in the Pentium D) vs. only 10 stages in the Pentium III, was reversed in the Intel Core architecture, which has a 14-stage pipeline.

In-order superscalar microprocessors, the logical successors of the pipeline processors of the previous chapter, were the first in production. When both in-order and out-of-order microprocessors became available, the relative ease of implementation of in-order processors allowed their manufacture with faster clock speed. Until the mid 1990s, the performance of in-order "speed demons" tended to dominate that of the out-of-order "brainiacs," which had slower clock speed but higher IPC. After that time frame, the out-of-order processors started to take over, thanks in part to Moore's law, which allowed more functionality and more hardware assists on chip.

Today high-performance single-processor microprocessors are out-of-order ones. The speed advantage of the in-order processors has disappeared since power dissipation has capped the increases in clock frequency. Because of the limit imposed on speed by power dissipation, because of the continuous validity of Moore's law, which allows more logic on a chip, and because the performance that can be attained with a single processor cannot be increased without adding extreme design complexity, the single processor is being replaced by multiprocessors. Whether multiprocessors on a chip (CMP) will consist of simple in-order or complex out-of-order processors, or a mix, is still an open question (cf. Chapters 7 and 8). Thus, it is important to study both types of microarchitectures.

This chapter gives overviews of the instruction pipelines of an in-order processor, the DEC Alpha 21164 (vintage 1994), and of an out-of-order one, the Intel Pentium P6 architecture started in the early 1990s and still in use, as exemplified by the Intel Pentium III (announced in 1999) and the recent Pentium Core. Whereas we can describe the Alpha instruction pipeline with the knowledge that we have so far, the transition from an in-order to an out-of-order processor requires the introduction of new concepts such as *register renaming*, *reorder buffer*, and *reservation stations*. In this chapter, the basic concepts are explained. Detailed and/or alternate implementations, for components of both the front-end (such as branch prediction and register renaming) and the back-end (such as instruction scheduling and load speculation), will be presented in Chapters 4 and 5. Because of their historical importance and because they have strongly influenced the design of current microprocessors, we will also present the scoreboard of the CDC 6600 and Tomasulo's algorithm for the IBM System 360/91.

Because the focus of this book is on microarchitecture, we do not emphasize compiler techniques that can enhance performance. Compiler optimizations are quite important for in-order processors, for they can remove statically some data dependencies. We conclude the chapter with an introduction to the design paradigm of *very long instruction word* (VLIW)/*explicitly parallel instruction computing* (EPIC). In VLIW/EPIC processors, the compiler is responsible for detecting

and scheduling the instruction-level parallelism. The compiler decides which operations can execute in the same cycle under the constraints of the concurrency offered by the functional units and the data flow of the program. We will use the Intel Itanium as the example general-purpose processor. The VLIW technique is also widely used in embedded processors where programs are short and can be hand-tuned.

Finally, the reader may have noticed that so far we have not mentioned the memory hierarchy design and whether memory latencies might be better tolerated in static or dynamic superscalars. Part of the answer is that for long latencies, like those occurring on misses that percolate up to main memory, both architectures will suffer performance degradation. Latencies are of the order of 100 cycles, and it is impossible to schedule statically enough instructions to cover that span of time. In the case of dynamic superscalars, many queuing structures will be full before the memory access is resolved, and the pipeline(s) will be stalled because of structural hazards. Since context switching, which requires saving and restoring states, takes longer than a memory access, this is not a viable solution. An alternative is to implement in hardware several contexts – that is, fast, stable storage units – that can hold the state of a process. Context switching now can be very fast, especially for lightweight processes, or *threads*, that share a common address space. This technique, called *multithreading*, will be discussed in detail in Chapter 8.

3.2 Overview of the Instruction Pipeline of the DEC Alpha 21164

3.2.1 General Organization

The DEC Alpha 21164 (Figure 3.1), introduced in 1994, is a four-way in-order RISC superscalar processor. Its clock frequency was 300 MHz, and there were 9.3 million transistors on chip. The RISC label means that the Alpha ISA is of the load–store type with all instructions being register–register except for those accessing memory. There are 32 64-bit integer registers and 32 floating-point registers, and an additional 8 integer registers used only by the Pal code (recall Section 2.3.1). Since it is four-way, up to four instructions can pass through each stage of the front-end on any given cycle. Its predecessor, the Alpha 21064, was only two-way.

The back-end consists of four functional units. There are two integer pipelines and two floating-point pipelines. Both integer pipelines can perform the simple arithmetic and logical operations, including the address computation for load–store instructions. One integer pipe has a shifter and multiplier associated with it. The multiply operation is partially pipelined, and its latency depends on the size of the data (32 or 64 bits).The other integer pipe handles branches. Integer operations other than multiply, conditional moves, and divide have a latency of 1 cycle. Integer divide is handled in software. One floating-point unit is dedicated to multiplication: the other one handles all other operations, including divide – the only operation that is not pipelined. The latency of all floating-point operations except divide is 4 cycles.

The instruction pipeline is shown in Figure 3.2. As can be seen, the front-end takes 4 cycles. The overall number of stages is seven for the integer pipelines and