

Lecture No.3

Lecture Outlines

1.4 The Evolution of the Intel x86 Architecture

1.5 Embedded Systems

The Internet of Things
Embedded Operating Systems
Application Processors versus Dedicated Processors
Microprocessors versus Microcontrollers Embedded
versus Deeply Embedded Systems

1.6 ARM Architecture

ARM Evolution Instruction Set
Architecture ARM Products

Learning Objectives

After studying this lecture, you should be able to:

- ◆ Present an overview of the evolution of the x86 architecture.
- ◆ Define embedded systems and list some of the requirements and constraints that various embedded systems must meet.

1.4 THE EVOLUTION OF THE INTEL x86 ARCHITECTURE

Throughout this book, we rely on many concrete examples of computer design and implementation to illustrate concepts and to illuminate trade-offs. Numerous systems, both contemporary and historical, provide examples of important computer architecture design features. But the book relies principally on examples from two processor families: the Intel x86 and the ARM architectures. The current x86 offerings represent the results of decades of design effort on **complex instruction set computers (CISCs)**. The x86 incorporates the sophisticated design principles once found only on mainframes and supercomputers and serves as an excellent example of CISC design. An alternative approach to processor design is the **reduced instruction set computer (RISC)**. The ARM architecture is used in a wide variety of embedded systems and is one of the most powerful and best-designed RISC-based systems on the market. In this section and the next, we provide a brief overview of these two systems.

In terms of market share, Intel has ranked as the number one maker of microprocessors for non-embedded systems for decades, a position it seems unlikely to yield. The evolution of its flagship microprocessor product serves as a good indicator of the evolution of computer technology in general.

Table 1.3 shows that evolution. Interestingly, as microprocessors have grown faster and much more complex, Intel has actually picked up the pace. Intel used to develop microprocessors one after another, every four years. But Intel hopes to keep rivals at bay by trimming a year or two off this development time, and has done so with the most recent x86 generations.

Table 1.3 Evolution of Intel Microprocessors**(a) 1970s Processors**

	4004	8008	8080	8086	8088
Introduced	1971	1972	1974	1978	1979
Clock speeds	108 kHz	108 kHz	2 MHz	5 MHz, 8 MHz, 10 MHz	5 MHz, 8 MHz
Bus width	4 bits	8 bits	8 bits	16 bits	8 bits
Number of transistors	2,300	3,500	6,000	29,000	29,000
Feature size (μm)	10	8	6	3	6
Addressable memory	640 bytes	16 KB	64 KB	1 MB	1 MB

(b) 1980s Processors

	80286	386TM DX	386TM SX	486TM DX CPU
Introduced	1982	1985	1988	1989
Clock speeds	6–12.5 MHz	16–33 MHz	16–33 MHz	25–50 MHz
Bus width	16 bits	32 bits	16 bits	32 bits
Number of transistors	134,000	275,000	275,000	1.2 million
Feature size (μm)	1.5	1	1	0.8–1
Addressable memory	16 MB	4 GB	16 MB	4 GB
Virtual memory	1 GB	64 TB	64 TB	64 TB
Cache	—	—	—	8 kB

(c) 1990s Processors

	486TM SX	Pentium	Pentium Pro	Pentium II
Introduced	1991	1993	1995	1997
Clock speeds	16–33 MHz	60–166 MHz,	150–200 MHz	200–300 MHz
Bus width	32 bits	32 bits	64 bits	64 bits
Number of transistors	1.185 million	3.1 million	5.5 million	7.5 million
Feature size (μm)	1	0.8	0.6	0.35
Addressable memory	4 GB	4 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	8 kB	8 kB	512 kB L1 and 1 MB L2	512 kB L2

(d) Recent Processors

	Pentium III	Pentium 4	Core 2 Duo	Core i7 EE 4960X
Introduced	1999	2000	2006	2013
Clock speeds	450–660 MHz	1.3–1.8 GHz	1.06–1.2 GHz	4 GHz
Bus width	64 bits	64 bits	64 bits	64 bits
Number of transistors	9.5 million	42 million	167 million	1.86 billion
Feature size (nm)	250	180	65	22
Addressable memory	64 GB	64 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	512 kB L2	256 kB L2	2 MB L2	1.5 MB L2/15 MB L3
Number of cores	1	1	2	6

It is worthwhile to list some of the highlights of the evolution of the Intel product line:

- **8080:** The world's first general-purpose microprocessor. This was an 8-bit machine, with an 8-bit data path to memory. The 8080 was used in the first personal computer, the Altair.
- **8086:** A far more powerful, 16-bit machine. In addition to a wider data path and larger registers, the 8086 sported an instruction cache, or queue, that prefetches a few instructions before they are executed. A variant of this processor, the 8088, was used in IBM's first personal computer, securing the success of Intel. The 8086 is the first appearance of the x86 architecture.
- **80286:** This extension of the 8086 enabled addressing a 16-MB memory instead of just 1 MB.
- **80386:** Intel's first 32-bit machine, and a major overhaul of the product. With a 32-bit architecture, the 80386 rivaled the complexity and power of minicomputers and mainframes introduced just a few years earlier. This was the first Intel processor to support multitasking, meaning it could run multiple programs at the same time.
- **80486:** The 80486 introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining. The 80486 also offered a built-in math coprocessor, offloading complex math operations from the main CPU.
- **Pentium:** With the Pentium, Intel introduced the use of superscalar techniques, which allow multiple instructions to execute in parallel.
- **Pentium Pro:** The Pentium Pro continued the move into superscalar organization begun with the Pentium, with aggressive use of register renaming, branch prediction, data flow analysis, and speculative execution.
- **Pentium II:** The Pentium II incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently.
- **Pentium III:** The Pentium III incorporates additional floating-point instructions: The Streaming SIMD Extensions (SSE) instruction set extension added 70 new instructions designed to increase performance when exactly the same operations are to be performed on multiple data objects. Typical applications are digital signal processing and graphics processing.
- **Pentium 4:** The Pentium 4 includes additional floating-point and other enhancements for multimedia.
- **Core:** This is the first Intel x86 microprocessor with a dual core, referring to the implementation of two cores on a single chip.
- **Core 2:** The Core 2 extends the Core architecture to 64 bits. The Core 2 Quad provides four cores on a single chip. More recent Core offerings have up to 10 cores per chip. An important addition to the architecture was the Advanced Vector Extensions instruction set that provided a set of 256-bit, and then 512-bit, instructions for efficient processing of vector data.

Almost 40 years after its introduction in 1978, the x86 architecture continues to dominate the processor market outside of embedded systems. Although the organization and technology of the x86 machines have changed dramatically over the decades, the instruction set architecture has evolved to remain backward compatible with earlier versions. Thus, any program written on an older version of the x86 architecture can execute on newer versions. All changes to the instruction set architecture have involved additions to the instruction set, with no subtractions. The rate of change has been the addition of roughly one instruction per month added to the architecture [ANTH08], so that there are now thousands of instructions in the instruction set.

The x86 provides an excellent illustration of the advances in computer hardware over the past 35 years. The 1978 8086 was introduced with a clock speed of 5 MHz and had 29,000 transistors. A six-core Core i7 EE 4960X introduced in 2013 operates at 4 GHz, a speedup of a factor of 800, and has 1.86 billion transistors, about 64,000 times as many as the 8086. Yet the Core i7 EE 4960X is in only a slightly larger package than the 8086 and has a comparable cost.

1.5 EMBEDDED SYSTEMS

The term *embedded system* refers to the use of electronics and software within a product, as opposed to a general-purpose computer, such as a laptop or desktop system. Millions of computers are sold every year, including laptops, personal computers, workstations, servers, mainframes, and supercomputers. In contrast, billions of computer systems are produced each year that are embedded within larger devices. Today, many, perhaps most, devices that use electric power have an embedded computing system. It is likely that in the near future virtually all such devices will have embedded computing systems.

Types of devices with embedded systems are almost too numerous to list. Examples include cell phones, digital cameras, video cameras, calculators, microwave ovens, home security systems, washing machines, lighting systems, thermostats, printers, various automotive systems (e.g., transmission control, cruise control, fuel injection, anti-lock brakes, and suspension systems), tennis rackets, toothbrushes, and numerous types of sensors and actuators in automated systems.

Often, embedded systems are tightly coupled to their environment. This can give rise to real-time constraints imposed by the need to interact with the environment. Constraints, such as required speeds of motion, required precision of measurement, and required time durations, dictate the timing of software operations. If multiple activities must be managed simultaneously, this imposes more complex real-time constraints.

Figure 1.14 shows in general terms an embedded system organization. In addition to the processor and memory, there are a number of elements that differ from the typical desktop or laptop computer:

- There may be a variety of interfaces that enable the system to measure, manipulate, and otherwise interact with the external environment. Embedded systems often interact (sense, manipulate, and communicate) with external world through sensors and actuators and hence are typically reactive systems; a

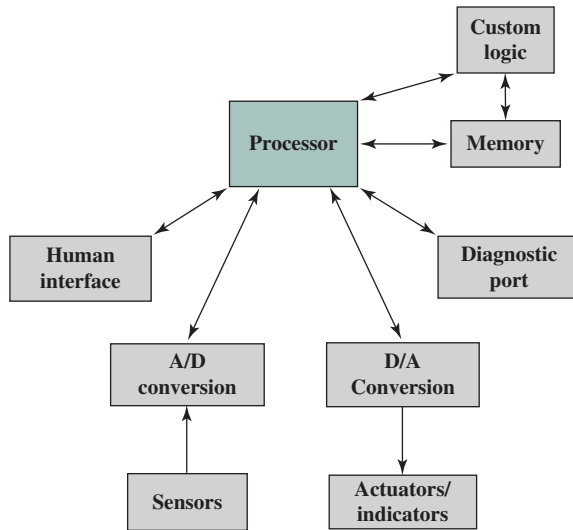


Figure 1.14 Possible Organization of an Embedded System

reactive system is in continual interaction with the environment and executes at a pace determined by that environment.

- The human interface may be as simple as a flashing light or as complicated as real-time robotic vision. In many cases, there is no human interface.
- The diagnostic port may be used for diagnosing the system that is being controlled—not just for diagnosing the computer.
- Special-purpose field programmable (FPGA), application-specific (ASIC), or even nondigital hardware may be used to increase performance or reliability.
- Software often has a fixed function and is specific to the application.
- Efficiency is of paramount importance for embedded systems. They are optimized for energy, code size, execution time, weight and dimensions, and cost.

There are several noteworthy areas of similarity to general-purpose computer systems as well:

- Even with nominally fixed function software, the ability to field upgrade to fix bugs, to improve security, and to add functionality, has become very important for embedded systems, and not just in consumer devices.
- One comparatively recent development has been of embedded system platforms that support a wide variety of apps. Good examples of this are smartphones and audio/visual devices, such as smart TVs.

Embedded Operating Systems

There are two general approaches to developing an embedded operating system (OS). The first approach is to take an existing OS and adapt it for the embedded application. For example, there are embedded versions of Linux, Windows, and Mac, as well as other commercial and proprietary operating systems specialized for embedded systems. The other approach is to design and implement an OS intended solely for embedded use. An example of the latter is TinyOS, widely used in wireless sensor networks. This topic is explored in depth in [STAL15].

Application Processors versus Dedicated Processors

In this subsection, and the next two, we briefly introduce some terms commonly found in the literature on embedded systems. **Application processors** are defined by the processor's ability to execute complex operating systems, such as Linux, Android, and Chrome. Thus, the application processor is general-purpose in nature. A good example of the use of an embedded application processor is the smartphone. The embedded system is designed to support numerous apps and perform a wide variety of functions.

Most embedded systems employ a **dedicated processor**, which, as the name implies, is dedicated to one or a small number of specific tasks required by the host device. Because such an embedded system is dedicated to a specific task or tasks, the processor and associated components can be engineered to reduce size and cost.

Microprocessors versus Microcontrollers

As we have seen, early **microprocessor** chips included registers, an ALU, and some sort of control unit or instruction processing logic. As transistor density increased, it became possible to increase the complexity of the instruction set architecture, and ultimately to add memory and more than one processor. Contemporary microprocessor chips, as shown in Figure 1.2, include multiple cores and a substantial amount of cache memory.

A **microcontroller** chip makes a substantially different use of the logic space available. Figure 1.15 shows in general terms the elements typically found on a microcontroller chip. As shown, a microcontroller is a single chip that contains the processor, non-volatile memory for the program (ROM), volatile memory for input and output (RAM), a clock, and an I/O control unit. The processor portion of the microcontroller has a much lower silicon area than other microprocessors and much higher energy efficiency. We examine microcontroller organization in more detail in Section 1.6.

Also called a “computer on a chip,” billions of microcontroller units are embedded each year in myriad products from toys to appliances to automobiles. For example, a single vehicle can use 70 or more microcontrollers. Typically, especially for the smaller, less expensive microcontrollers, they are used as dedicated processors for specific tasks. For example, microcontrollers are heavily utilized in automation processes. By providing simple reactions to input, they can control machinery, turn fans on and off, open and close valves, and so forth. They are integral parts of modern industrial technology and are among the most inexpensive ways to produce machinery that can handle extremely complex functionalities.

Microcontrollers come in a range of physical sizes and processing power. Processors range from 4-bit to 32-bit architectures. Microcontrollers tend to be much slower than microprocessors, typically operating in the MHz range rather than the GHz speeds of microprocessors. Another typical feature of a microcontroller is that it does not provide for human interaction. The microcontroller is programmed for a specific task, embedded in its device, and executes as and when required.

Embedded versus Deeply Embedded Systems

We have, in this section, defined the concept of an embedded system. A subset of embedded systems, and a quite numerous subset, is referred to as **deeply embedded systems**. Although this term is widely used in the technical and commercial

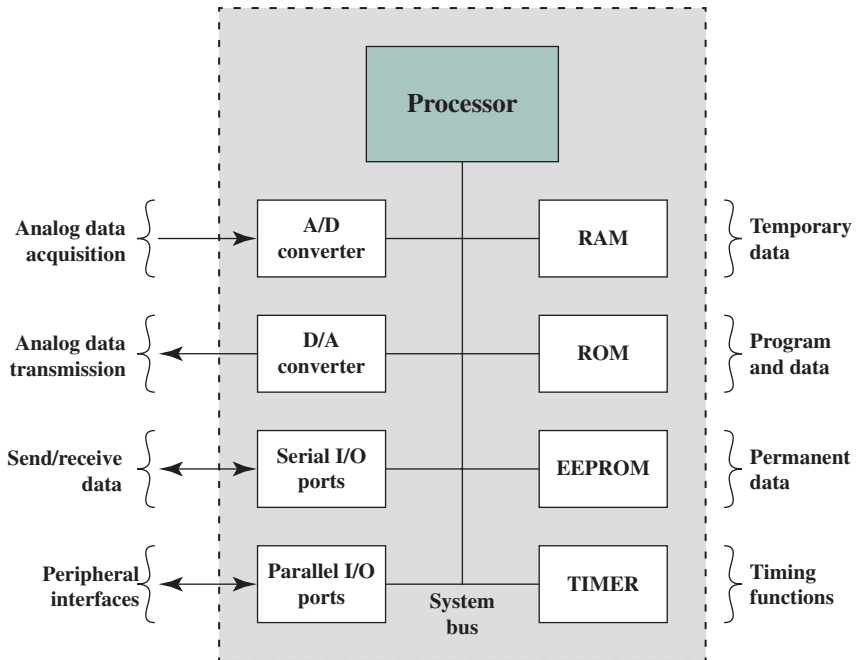


Figure 1.15 Typical Microcontroller Chip Elements

literature, you will search the Internet in vain (or at least I did) for a straightforward definition. Generally, we can say that a deeply embedded system has a processor whose behavior is difficult to observe both by the programmer and the user. A deeply embedded system uses a microcontroller rather than a microprocessor, is not programmable once the program logic for the device has been burned into ROM (read-only memory), and has no interaction with a user.

Deeply embedded systems are dedicated, single-purpose devices that detect something in the environment, perform a basic level of processing, and then do something with the results. Deeply embedded systems often have wireless capability and appear in networked configurations, such as networks of sensors deployed over a large area (e.g., factory, agricultural field). The Internet of things depends heavily on deeply embedded systems. Typically, deeply embedded systems have extreme resource constraints in terms of memory, processor size, time, and power consumption.

1.6 ARMARCHITECTURE

The ARM architecture refers to a processor architecture that has evolved from RISC design principles and is used in embedded systems. Chapter 15 examines RISC design principles in detail. In this section, we give a brief overview of the ARM architecture.

ARM Evolution

ARM is a family of RISC-based microprocessors and microcontrollers designed by ARM Holdings, Cambridge, England. The company doesn't make processors but instead designs microprocessor and multicore architectures and licenses them to manufacturers. Specifically, ARM Holdings has two types of licensable products: processors and processor architectures. For processors, the customer buys the rights to use ARM-supplied design in their own chips. For a processor architecture, the customer buys the rights to design their own processor compliant with ARM's architecture.

ARM chips are high-speed processors that are known for their small die size and low power requirements. They are widely used in smartphones and other handheld devices, including game systems, as well as a large variety of consumer products. ARM chips are the processors in Apple's popular iPod and iPhone devices, and are used in virtually all Android smartphones as well. ARM is probably the most widely used embedded processor architecture and indeed the most widely used processor architecture of any kind in the world [VANC14].

The origins of ARM technology can be traced back to the British-based Acorn Computers company. In the early 1980s, Acorn was awarded a contract by the British Broadcasting Corporation (BBC) to develop a new microcomputer architecture for the BBC Computer Literacy Project. The success of this contract enabled Acorn to go on to develop the first commercial RISC processor, the Acorn RISC Machine (ARM). The first version, ARM1, became operational in 1985 and was used for internal research and development as well as being used as a coprocessor in the BBC machine.

In this early stage, Acorn used the company VLSI Technology to do the actual fabrication of the processor chips. VLSI was licensed to market the chip on its own and had some success in getting other companies to use the ARM in their products, particularly as an embedded processor.

The ARM design matched a growing commercial need for a high-performance, low-power-consumption, small-size, and low-cost processor for embedded applications. But further development was beyond the scope of Acorn's capabilities. Accordingly, a new company was organized, with Acorn, VLSI, and Apple Computer as founding partners, known as ARM Ltd. The Acorn RISC Machine became Advanced RISC Machines.¹²

Instruction Set Architecture

The ARM instruction set is highly regular, designed for efficient implementation of the processor and efficient execution. All instructions are 32 bits long and follow a regular format. This makes the ARM ISA suitable for implementation over a wide range of products.

Augmenting the basic ARM ISA is the Thumb instruction set, which is a re-encoded subset of the ARM instruction set. Thumb is designed to increase the performance of ARM implementations that use a 16-bit or narrower memory data bus, and to allow better code density than provided by the ARM instruction set. The Thumb instruction set contains a subset of the ARM 32-bit instruction set recoded into 16-bit instructions. The current defined version is Thumb-2.

The ARM and Thumb-2 ISAs are discussed in Chapters 12 and 13.

ARM Products

ARM Holdings licenses a number of specialized microprocessors and related technologies, but the bulk of their product line is the Cortex family of microprocessor architectures. There are three Cortex architectures, conveniently labeled with the initials A, R, and M.

CORTEX-A/CORTEX-A50 The Cortex-A and Cortex-A50 are application processors, intended for mobile devices such as smartphones and eBook readers, as well as consumer devices such as digital TV and home gateways (e.g., DSL and cable Internet modems). These processors run at higher clock frequency (over 1 GHz), and support a memory management unit (MMU), which is required for full feature OSs such as Linux, Android, MS Windows, and mobile OSs. An MMU is a hardware module that supports virtual memory and paging by translating virtual addresses into physical addresses; this topic is explored in Chapter 8.

The two architectures use both the ARM and Thumb-2 instruction sets; the principal difference is that the Cortex-A is a 32-bit machine, and the Cortex-A50 is a 64-bit machine.

CORTEX-R The Cortex-R is designed to support real-time applications, in which the timing of events needs to be controlled with rapid response to events. They can run at a fairly high clock frequency (e.g., 200MHz to 800MHz) and have very low response latency. The Cortex-R includes enhancements both to the instruction set and to the processor organization to support deeply embedded real-time devices. Most of these processors do not have MMU; the limited data requirements and the limited number of simultaneous processes eliminates the need for elaborate hardware and software support for virtual memory. The Cortex-R does have a Memory Protection Unit (MPU), cache, and other memory features designed for industrial applications. An MPU is a hardware module that prohibits one program in memory from accidentally accessing memory assigned to another active program. Using various methods, a protective boundary is created around the program, and instructions within the program are prohibited from referencing data outside of that boundary.

Examples of embedded systems that would use the Cortex-R are automotive braking systems, mass storage controllers, and networking and printing devices.

CORTEX-M Cortex-M series processors have been developed primarily for the microcontroller domain where the need for fast, highly deterministic interrupt management is coupled with the desire for extremely low gate count and lowest possible power consumption. As with the Cortex-R series, the Cortex-M architecture has an MPU but no MMU. The Cortex-M uses only the Thumb-2 instruction set. The market for the Cortex-M includes IoT devices, wireless sensor/actuator networks used in factories and other enterprises, automotive body electronics, and so on.

There are currently four versions of the Cortex-M series:

- **Cortex-M0:** Designed for 8- and 16-bit applications, this model emphasizes low cost, ultra low power, and simplicity. It is optimized for small silicon die size (starting from 12k gates) and use in the lowest cost chips.
- **Cortex-M0+:** An enhanced version of the M0 that is more energy efficient.
- **Cortex-M3:** Designed for 16- and 32-bit applications, this model emphasizes performance and energy efficiency. It also has comprehensive debug and trace features to enable software developers to develop their applications quickly.
- **Cortex-M4:** This model provides all the features of the Cortex-M3, with additional instructions to support digital signal processing tasks.