Lecture 00.01 Introduction to embedded computing

embedded computer

Embedded computers comprise the vast majority of computers, yet are perhaps the least familiar to the uninitiated. A computer in a kitchen appliance, smart thermostat, or pacemaker is *embedded*—that is, it has most of the following characteristics: it

- performs a specific, limited function;
- performs its function in real-time;
- is small;
- is inexpensive;
- is low-power;
- is ruggedly packaged.

central processing unit (CPU)
integrated circuit (IC)
semiconductor
doping
microprocessor

The *central processing unit (CPU)* of any computer is the electronic circuitry that performs a computer's most basic instructions, such as arithmetic, logic, and input/output. A CPU is typically an *integrated circuit* (IC, i.e. "microchip" or just "chip"), which is made of a single silicon chip, the size of a human fingernail, transformed into a circuit containing *billions* of elements, such as transistors, diodes, and capacitors, by a process in which the *semiconductor* material silicon is selectively *doped* with impurities, locally changing its conductivity. An IC CPU is called a *microprocessor*.

microcontroller (μC)

Many embedded computers are *microcontrollers* (μC), which are integrated circuits that include CPUs, memory, and input/output peripherals—classes of computing components that will be described in this chapter. Images of a microprocessor and a microcontroller are found in Figure 00.1.

instructions
machine language
language levels
assembly language
opcodes
program

The *instructions* carried out by a CPU are rather basic, yet combinations of them can be vastly more complex. This is analogous to words, perhaps relatively simple, alone, being combined to form complex phrases, sentences, and books. In fact, this is why a CPU's instructions are said to form a *machine language*—literally just numbers with predefined meanings, often represented in binary. These languages are very cumbersome for humans to write useful software with, as we will see, so *higher level* languages are developed. The first level above machine language is called *assembly language*, which typically assigns a more descriptive mnemonic to represent each instruction ("*opcodes*"). A list of languages by level is given in Table 00.1. A *program* is a sequence of instructions that performs some task.

programming languages

Higher-level languages such as C, Python, and Matlab are called *programming languages*. They have the important quality of *microprocessor independence*—that is, they can be written for multiple processors, then

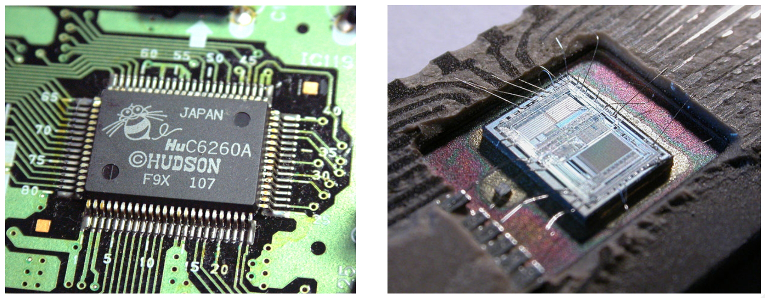

Figure 00.1: (left) a Hudson HuC5260A microprocessor (Baz1521, 2018) and (right) an Intel 8742 microcontroller—including a 12 MHz CPU, 128 B RAM, 2048 B EPROM, and input/output peripherals—broken open (Sameli, 2018).

translated into lower-level, processor-specific languages. There are two archetypical ways this translation occurs for a program:

compilation When a program is *compiled*, it is converted *en masse* into machine code before it is processed. This is often considered to be optimal for performance because each high-level function of the program is converted directly to processor-specific instructions. **compiled**

interpretation When a program is *interpreted*, its high-level functions have been pre-translated to machine code such that it can be directly processed "on the fly" without compilation. This is often considered more convenient and easier to implement than compilation. **interpreted**

Most modern programming languages provide both options.[1] Due to typically stringent performance and memory requirements, embedded programs often use compilation.

The *C programming language* is the language-of-choice for embedded computer programming. It can be thought of as being as close as possible to machine language without being processor-specific. It deals with numbers and arithmetic and memory addresses, which is what a processor does, also. That is, it is a low-level, *general-purpose language*. A limitation of the core language is that it lacks functions for basic operations like reading or writing to a file. The *C standard library* augments this functionality. **C programming language** **general-purpose language** **C standard library**

An advantage of its compactness is its relatively small size, a key advantage for embedded computing. Other key advantages of C for embedded computing include its ubiquity (C compilers are available for

[1]Previously, it was common to refer to a specific language as "compiled" or "translated," but this terminology is increasingly obsolete.

Table 00.1: categories of programming languages descending from high-level to low-level.

type	**examples**
graphical	LabVIEW, Simulink
scripting	Bash, Perl, BASIC
typically interpreted	MATLAB, Python, Ruby
typically compiled	C, C++, Java
assembly	symbolic codes
machine	numerical codes

most processors), speed (a result of a small language and good compilers), and energy efficiency (in that executing fewer cycles requires less power). Although the vast majority of embedded computer programming is in C, recently languages such as Python (which is, itself, written in C) have captured a small sliver of the embedded market (Altera, 2018).