Chapter 7
The Variety Of Processors And Computational Engines

Douglas Comer. (2017). *Essentials of Computer Architecture* (2nd ed.). CRC Press.

M. Morris Mano, Charles R. Kime. (2015). *Logic and computer design fundamentals* (5th ed.). Pearson.

Contents

1. The Harvard and Von Neumann Architectures

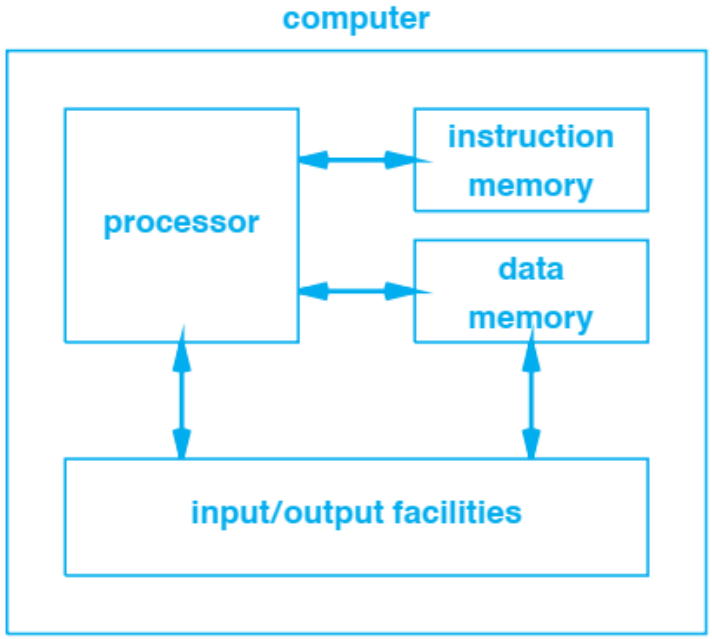

Figure 4.1 Illustration of the Harvard Architecture that uses two memories, one to hold programs and another to store data.

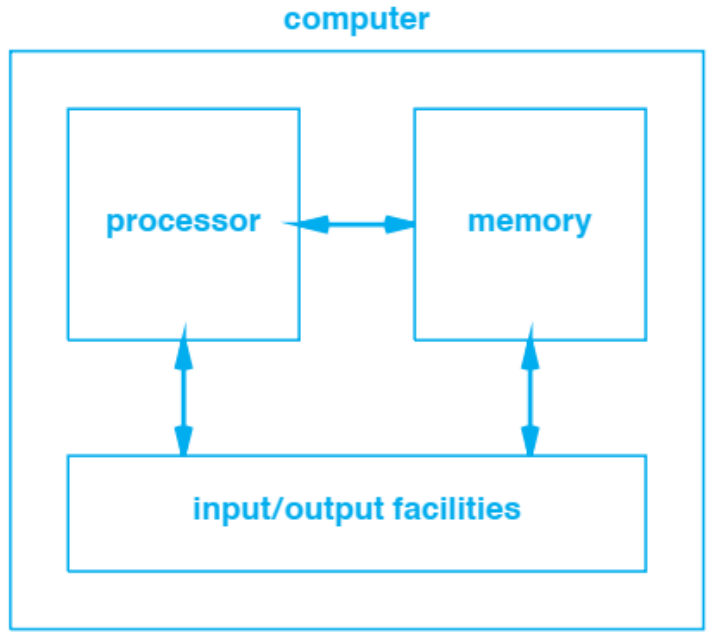

Figure 4.2 Illustration of the Von Neumann Architecture. Both programs and data can be stored in the same memory.

The Von Neumann Architecture offers complete flexibility: at any time, an owner can change how much of the memory is devoted to programs and how much to data.

Because it offers flexibility, the Von Neumann Architecture, which uses a single memory to hold both programs and data, has become pervasive: almost all computers follow the Von Neumann approach.

2. Hierarchical Structure And Computational Engines

- A large processor, such as a modern, general-purpose CPU, is so complex that no human can understand the entire processor as a single unit.
- To control the complexity, computer architects use a hierarchical approach in which subparts of the processor are designed and tested independently before being combined into the final design.

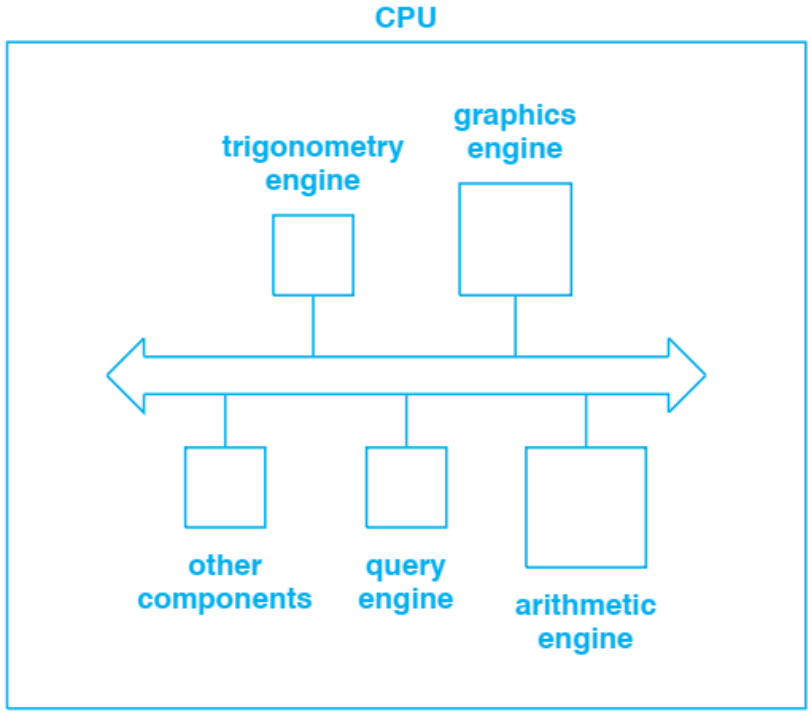

Figure 4.3 An example of a CPU that includes multiple components. The large arrow in the center of the figure indicates a central interconnect mechanism that the components use to coordinate.

3. Structure of a Conventional Processor

- Although a practical processor contains many subcomponents with complex interconnections among them, we can view a processor as having five conceptual units:
  - Controller
  - Arithmetic Logic Unit (ALU)
  - Local data storage (typically, registers)
  - Internal interconnection(s)
  - External interface(s) (I/O buses)

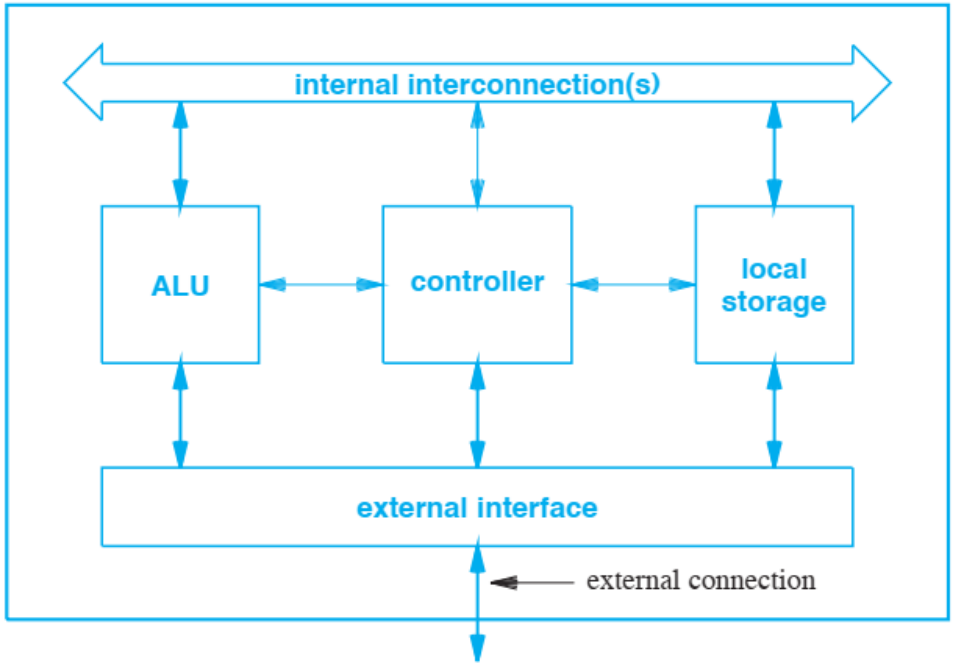

Figure 4.4 The five major units found in a conventional processor. The external interface connects to the rest of the computer system.

3. Structure of a Conventional Processor

- *Controller*
  - The controller forms the heart of a processor. Controller hardware has overall responsibility for program execution.
- *Arithmetic Logic Unit (ALU)*
  - We think of the ALU as the main computational engine in a processor.
  - The ALU performs all computational tasks, including integer arithmetic, operations on bits (e.g., left or right shift), and Boolean (logical) operations (e.g., Boolean and, or, exclusive or, and not).
  - However, an ALU does not perform multiple steps or initiate activities. Instead, the ALU only performs one operation at a time, and relies on the controller to specify exactly what operation to perform on the operand values.

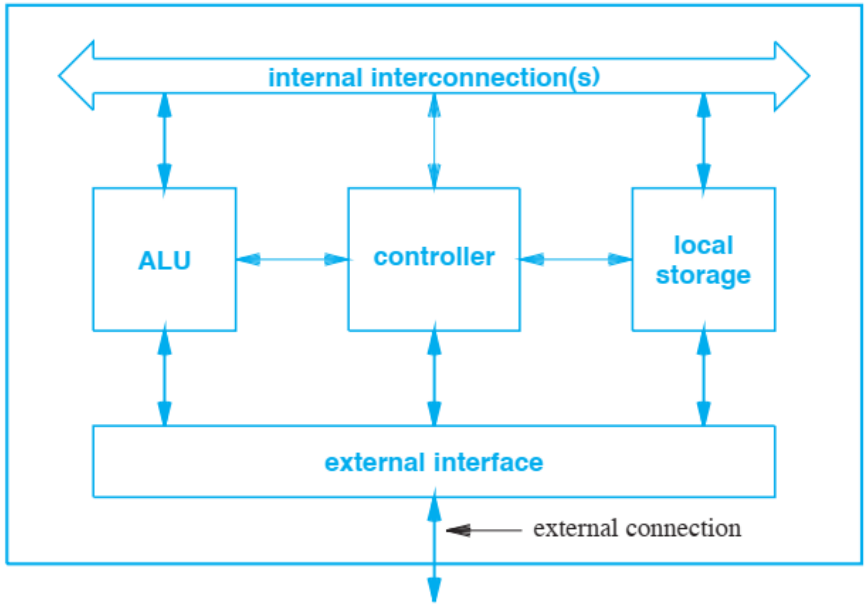

Figure 4.4 The five major units found in a conventional processor. The external interface connects to the rest of the computer system.

3. Structure of a Conventional Processor

- Local Data Storage
  - A processor must have at least some local storage to hold data values such as operands for arithmetic operations and the result.
  - Local storage usually takes the form of hardware registers – values must be loaded into the hardware registers before they can be used in computation.
- *Internal Interconnection(s)*
  - A processor contains hardware mechanisms that are used to transfer values between the other hardware units.
  - Architects sometimes use the term data path to describe an internal interconnection.
- *External Interface(s)*
  - The external interface unit handles all communication between the processor and the rest of the computer system. In particular, the external interface manages communication between the processor and external memory and I/O devices.

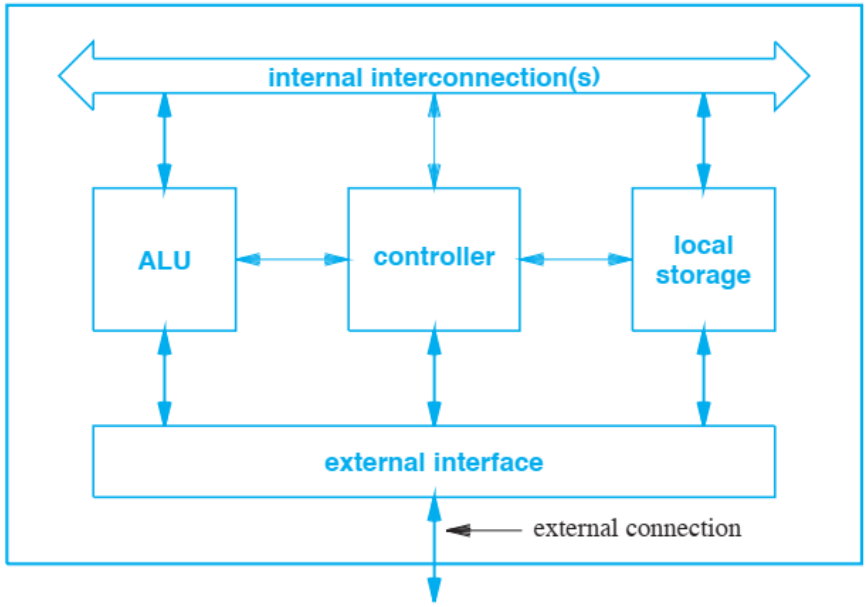

Figure 4.4 The five major units found in a conventional processor. The external interface connects to the rest of the computer system.

4. Processor Categories and Roles

- Coprocessors
- Microcontrollers
- Embedded system processors
- General-purpose processors

4. Processor Categories and Roles

- *Coprocessors*
  - A coprocessor operates in conjunction with and under the control of another processor. Usually, a coprocessor consists of a special-purpose processor that performs a single task at high speed.
  - For example, some CPUs use a coprocessor known as a floating point accelerator to speed the execution of arithmetic operations – when a floating point operation occurs, the CPU automatically passes the necessary values to the coprocessor, obtains the result, and then continues execution.

4. Processor Categories and Roles

- *Microcontrollers*
  - A microcontroller consists of a programmable device dedicated to the control of a physical system.
  - For example, microcontrollers run physical systems such as the engine in a modern automobile, the landing gear on an airplane, and the automatic door in a grocery store. In many cases, a microcontroller performs a trivial function that does not require much traditional computation. Instead, a microcontroller tests sensors and sends signals to control devices.

4. Processor Categories and Roles

- *Embedded System Processors*
  - An embedded system processor runs sophisticated electronic devices such as a wireless router or smart phone.
  - The processors used for embedded systems are usually more powerful than the processors that are used as microcontrollers, and often run a protocol stack used for communication. However, the processor may not contain all the functionality found on more general-purpose CPUs.
- *General-purpose Processors*
  - General-purpose processors are the most familiar and need little explanation. For example, the CPU in a PC is a general-purpose processor.

5. The Fetch-Execute Cycle

- How does a programmable processor access and perform steps of a program?
- The underlying mechanism is known as the *fetch-execute cycle*.
- To implement fetch-execute, a processor has an instruction pointer that automatically moves through the program in memory, performing each step. That is, each programmable processor executes two basic functions repeatedly. Algorithm 4.1 presents the two fundamental steps.

Algorithm 4.1

```
Repeat forever {
        Fetch: access the next step of the program from the
               location in which the program has been stored.
        Execute: perform the step of the program.
}
```

6. Clock Rate and Instruction Rate

- How fast does the fetch-execute cycle operate?
  - The answer depends on the processor, the technology used to store a program, and the time required to execute each instruction.
- Most processors use a *clock* to control the rate at which the underlying digital logic operates.
  - Although a higher clock rate usually means higher processing speed, it is important to realize that the clock rate does not give the rate at which the fetch-execute cycle proceeds.
- In particular, in most systems, the time required for the *execute* portion of the cycle depends on the instruction being executed.
  - The time also varies among basic arithmetic operations: integer multiplication or division requires more time than integer addition or subtraction.
  - Floating point computation is especially costly because floating point operations usually require more clock cycles than equivalent integer operations. Floating point multiplication or division stands out as especially costly – a single floating point division can require orders of magnitude more clock cycles than an integer addition.

6. Clock Rate and Instruction Rate

- The fetch-execute cycle may not proceed at a fixed rate because the time taken to execute an instruction depends on the operation being performed.
  - An operation such as multiplication requires more time than an operation such as addition.

7. Control: Getting Started and Stopping

- How does the processor start running the fetch-execute cycle? What happens after the processor executes the last step in a program?
- Because a processor runs the fetch-execute cycle indefinitely, a system must be designed to ensure that there is always a next step to execute.
- In a dedicated system, the same program executes repeatedly; in a general-purpose system, an operating system runs when no application is running.