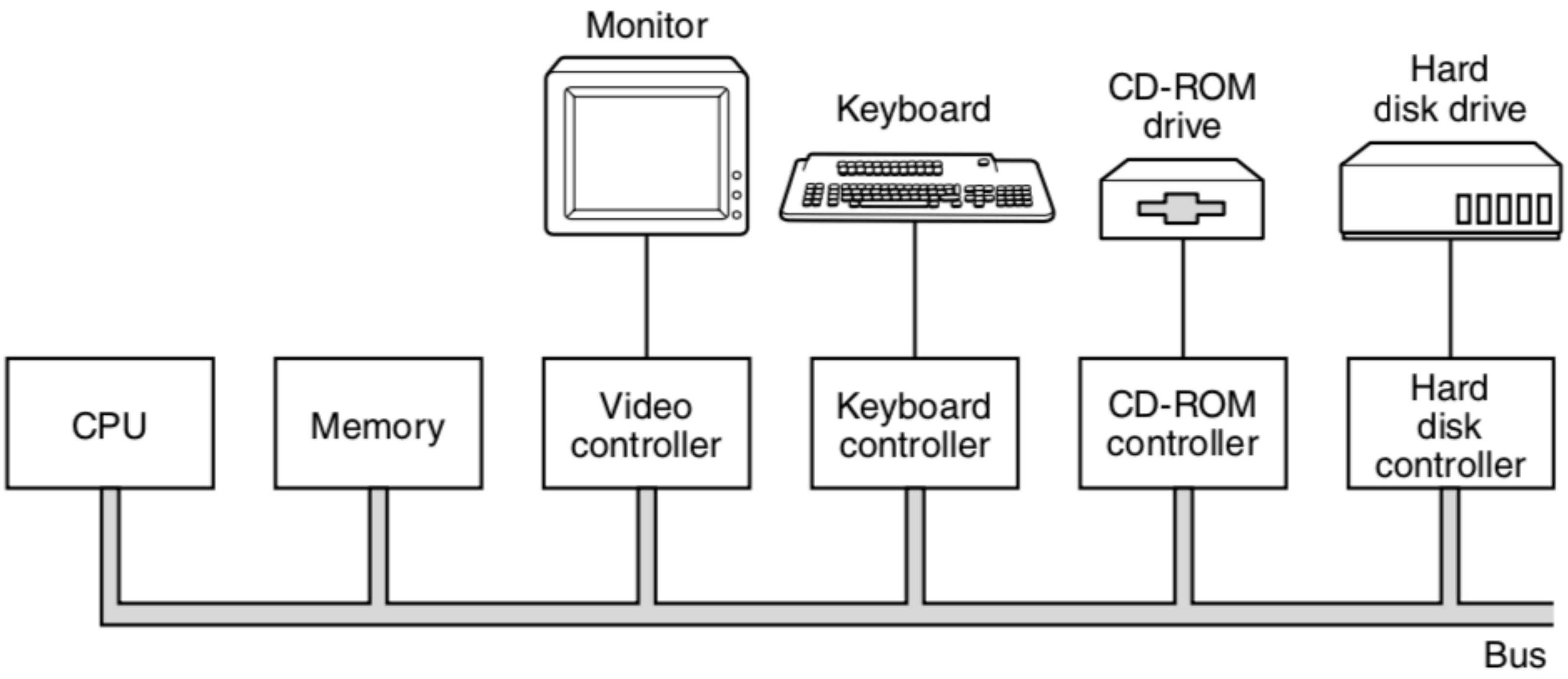
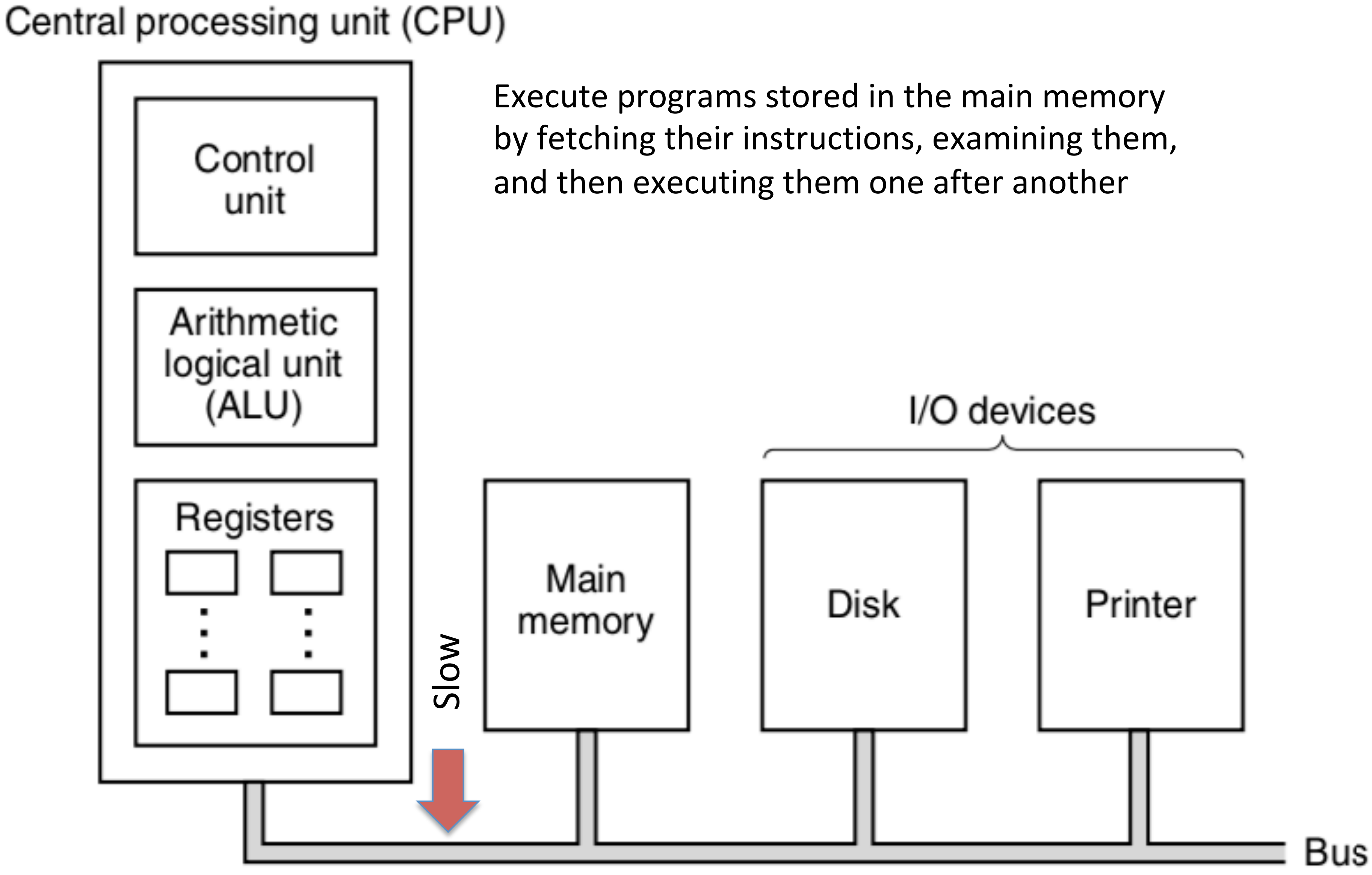
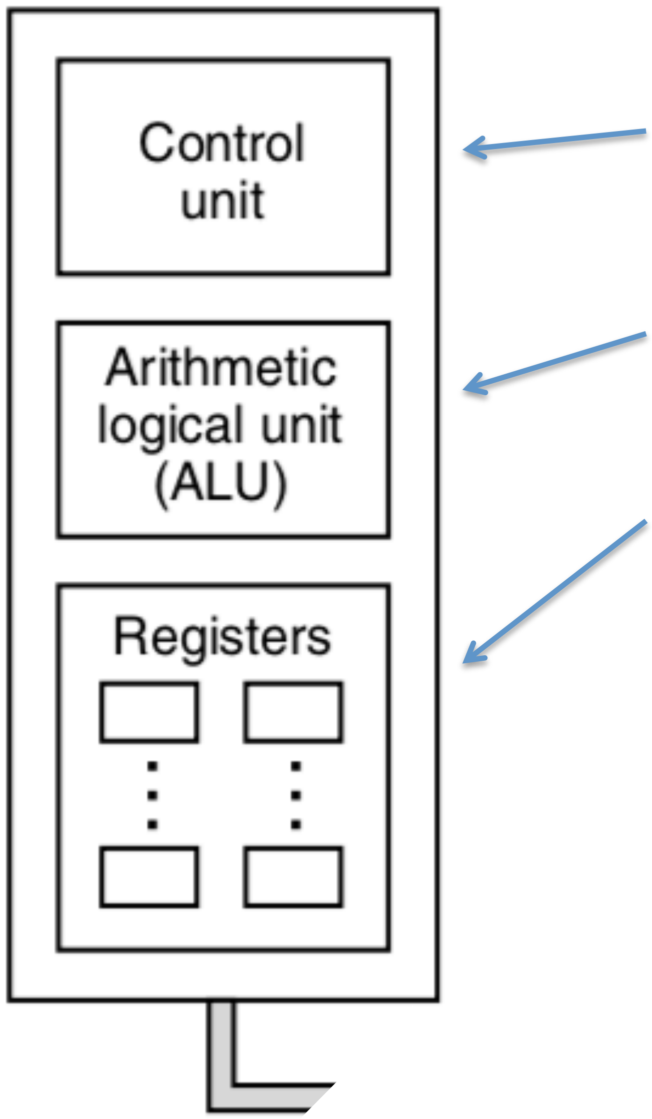
Organizacion interna de una computadora





Execute programs stored in the main memory by fetching their instructions, examining them, and then executing them one after another

Collection of parallel wires for transmitting address, data, and control signals



Fetch instructions from main memory and determine type

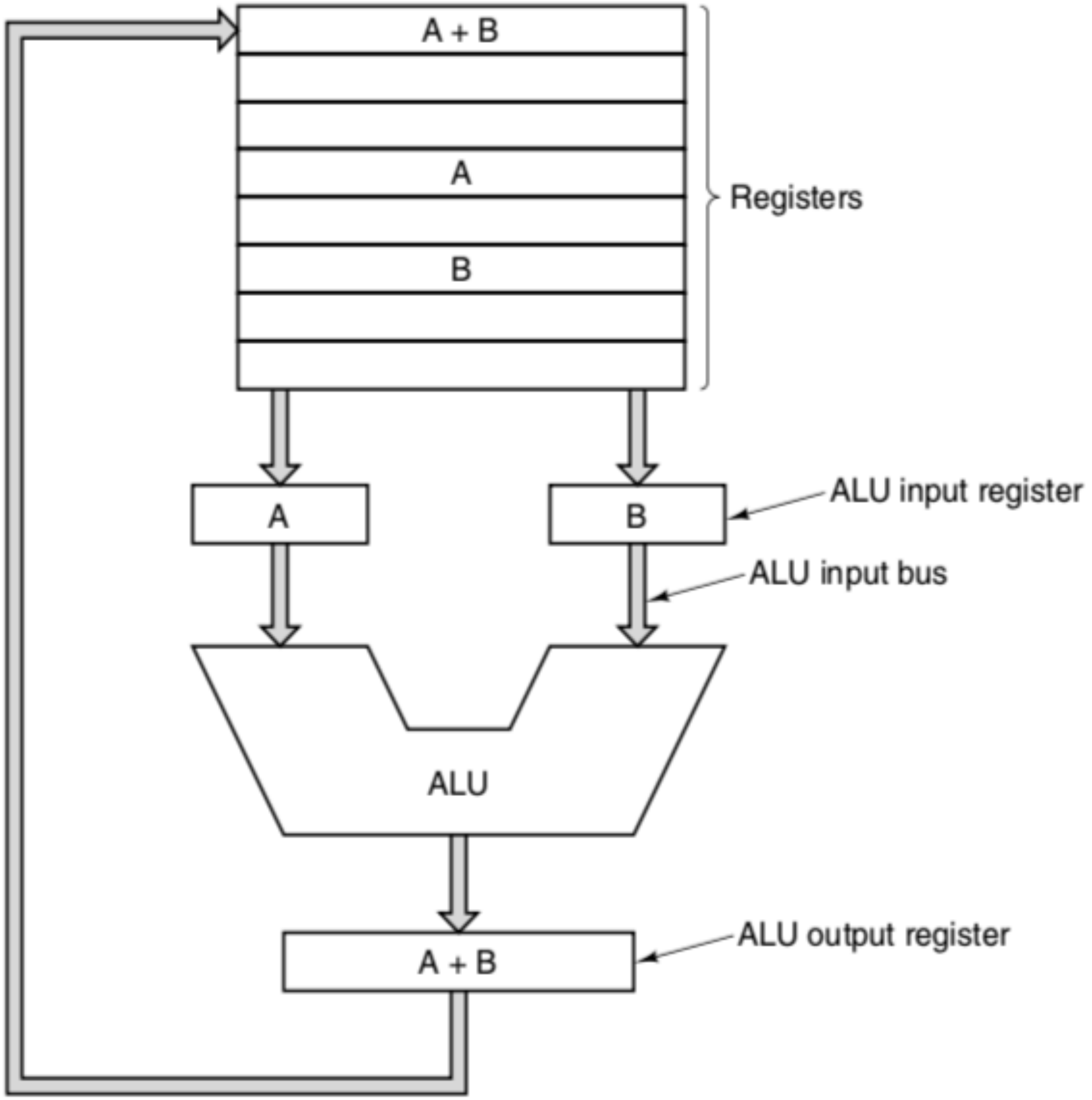
Perform operations such as addition needed to carry out the instructions

High-speed memory used to store temporary results and certain control information

- The most important register is the Program Counter (PC), which points to the next instruction to be fetched for execution
- Also important is the Instruction Register (IR), which holds the instruction currently being executed.

‘Words’’ are the units of data moved between memory and registers

Data Path



Most instructions can be divided into one of two categories: register-memory or register-register.

Instruction Execution

The CPU executes each instruction in a series of small steps. Roughly speaking, the steps are as follows:

1. Fetch the next instruction from memory into the instruction register.
2. Change the program counter to point to the following instruction.
3. Determine the type of instruction just fetched.
4. If the instruction uses a word in memory, determine where it is. Fetch the word, if needed, into a CPU register.
5. Execute the instruction.
6. Go to step 1 to begin executing the following instruction.

A program need not be executed by a ‘‘hardware’’ CPU.

Instead, a program can be carried out by having another program fetch, examine, and execute its instructions called an *interpreter.*

An interpreter breaks the instructions of its target machine into small steps. As a consequence, the machine on which the interpreter runs can be much simpler and less expensive than a hardware processor for the target machine would be.

The saving comes essentially from the fact that hardware is being replaced by software (the interpreter) and it costs more to replicate hardware than software.

After having specified the machine language, L, for a new computer, the design team can decide whether they want to build a hardware processor to execute programs in L directly or whether they want to write an interpreter to interpret programs in L instead.

Certain hybrid constructions are also possible, with some hardware execution as well as some software interpretation.

The more complex instructions were better because the execution of indivi- dual operations could sometimes be overlapped or otherwise executed in parallel using different hardware. For expensive, high-performance computers, the cost of this extra hardware could be readily justified. Thus expensive, high-performance computers came to have many more instructions than lower-cost ones.

Simple computers with interpreted instructions also had other benefits. Among the most important were

1. The ability to fix incorrectly implemented instructions in the field, or even make up for design deficiencies in the basic hardware.

2. The opportunity to add new instructions at minimal cost, even after delivery of the machine.

3. Structured design that permitted efficient development, testing, and documenting of complex instructions.

Reduced Instruction Set Computer (RISC)
vs.
Complex Instruction Set Computer (CISC)

In RISC: A small number of simple instructions that execute in one cycle.   Even if a RISC machine takes four or five instructions to do what a CISC machine does in one instruction, if the RISC instructions are 10 times as fast (because they are not interpreted), RISC wins.

Intel has been able to employ the same ideas even in a CISC architecture. Starting with the 486, the Intel CPUs contain a RISC core that executes the simplest (and typically most common) instructions in a single data path cycle, while interpreting the more complicated instructions in the usual CISC way.

Design Principles for Modern Computers

All Instructions Are Directly Executed by Hardware

All common instructions are directly executed by the hardware.

Maximize the Rate at Which Instructions Are Issued

Start as many instructions per second as possible.

Instructions Should be Easy to Decode

Making instructions regular, fixed length, with a small number of fields. The fewer different formats for instructions, the better.

Only Loads and Stores Should Reference Memory

Access to memory can take a long time, and the delay is unpredictable. All other instructions should operate only on registers.

Provide Plenty of Registers

Since accessing memory is relatively slow, many registers need to be provided.

Instruction-Level Parallelism

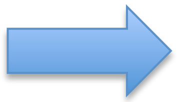
Making the chips run faster by increasing their clock speed has a limit to what is possible.
Consequently, most computer architects look to parallelism.

Parallelism comes in two general forms,

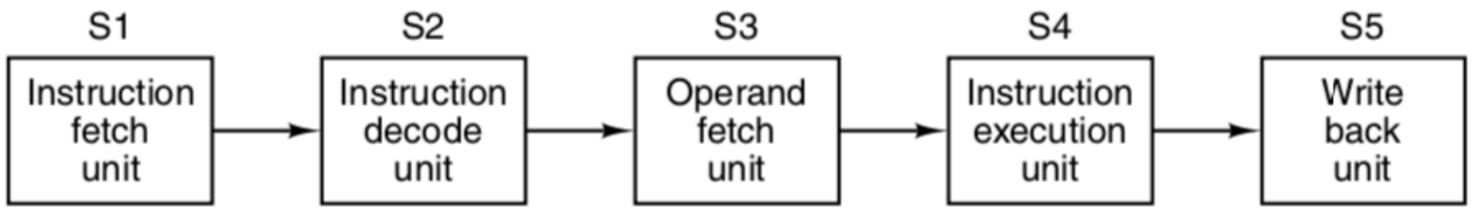
Instruction-level parallelism	Parallelism is exploited within individual instructions to get more instructions/sec out of the machine
Processor-level parallelism.	Multiple CPUs work together on the same problem.

Pipelining

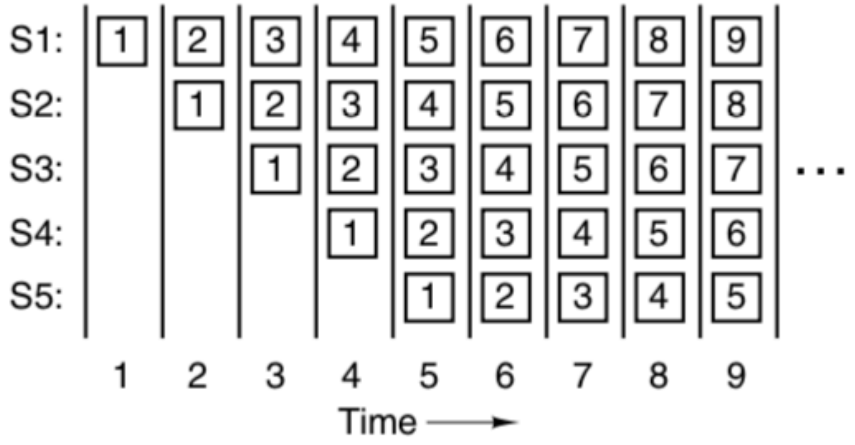
Fetching of instructions from memory is a major bottleneck in instruction execution speed



Fetch instructions from memory in advance, so they would be there when they were needed. These instructions are stored in a set of registers called the *prefetch* buffer.



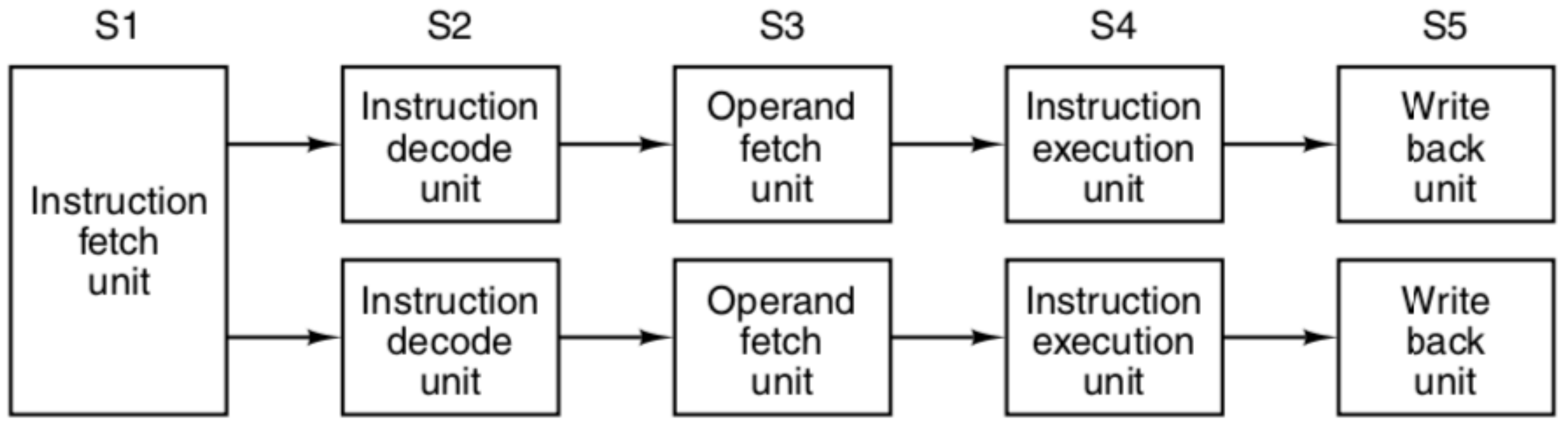
(a)



(b)

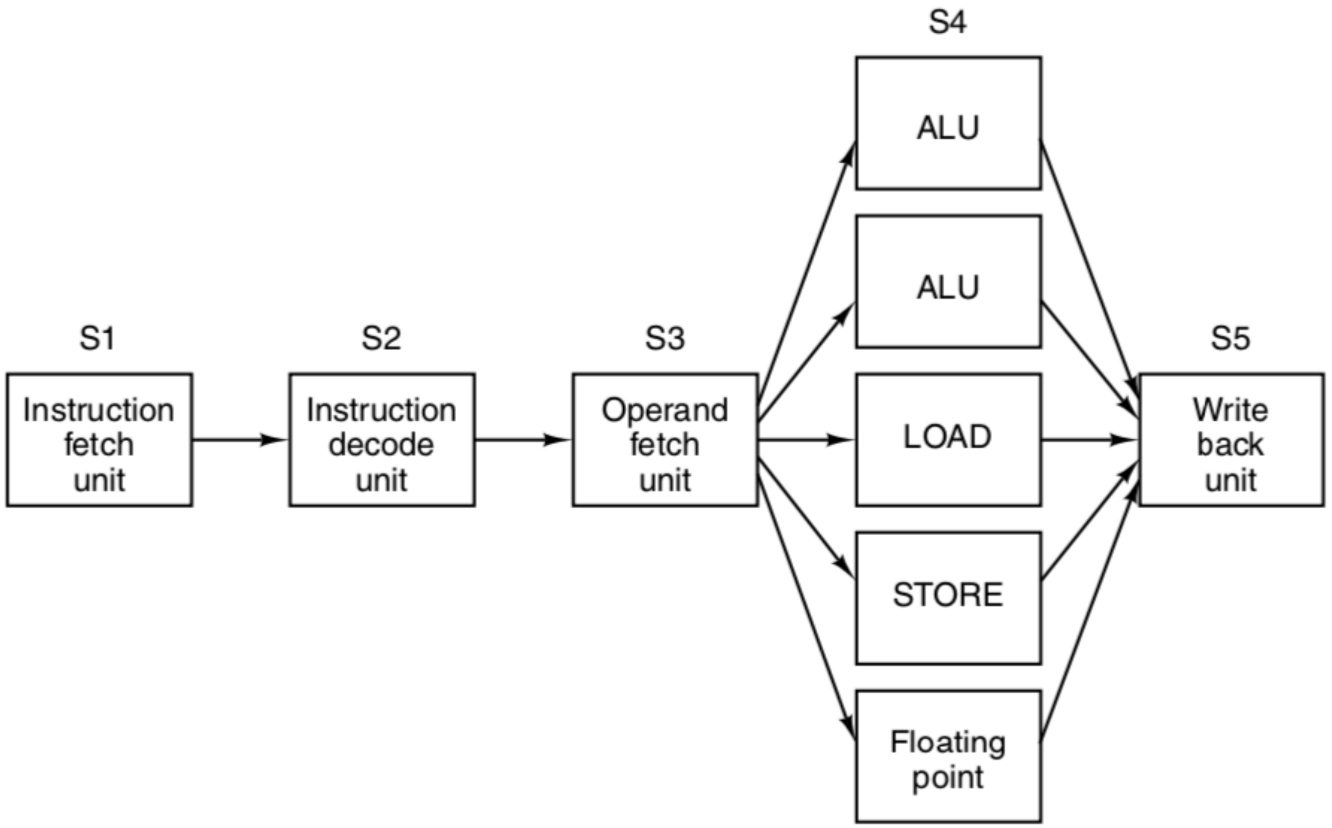
Superscalar Architectures

Here a single instruction fetch unit fetches pairs of instructions together and puts each one into its own pipeline, complete with its own ALU for parallel operation.



To be able to run in parallel, the two instructions must not conflict over resource usage (e.g., registers), and neither must depend on the result of the other. As with a single pipeline, either the compiler must guarantee this situation to hold (i.e., the hardware does not check and gives incorrect results if the instructions are not compatible), or conflicts are detected and eliminated during execution using extra hardware. (Pentium)

The basic idea is to have just a single pipeline but give it multiple functional units. For example, the Pentium II has a structure similar to this figure.



Implicit in the idea of a superscalar processor is that the S3 stage can issue instructions considerably faster than the S4 stage is able to execute them.

Multiprocessors

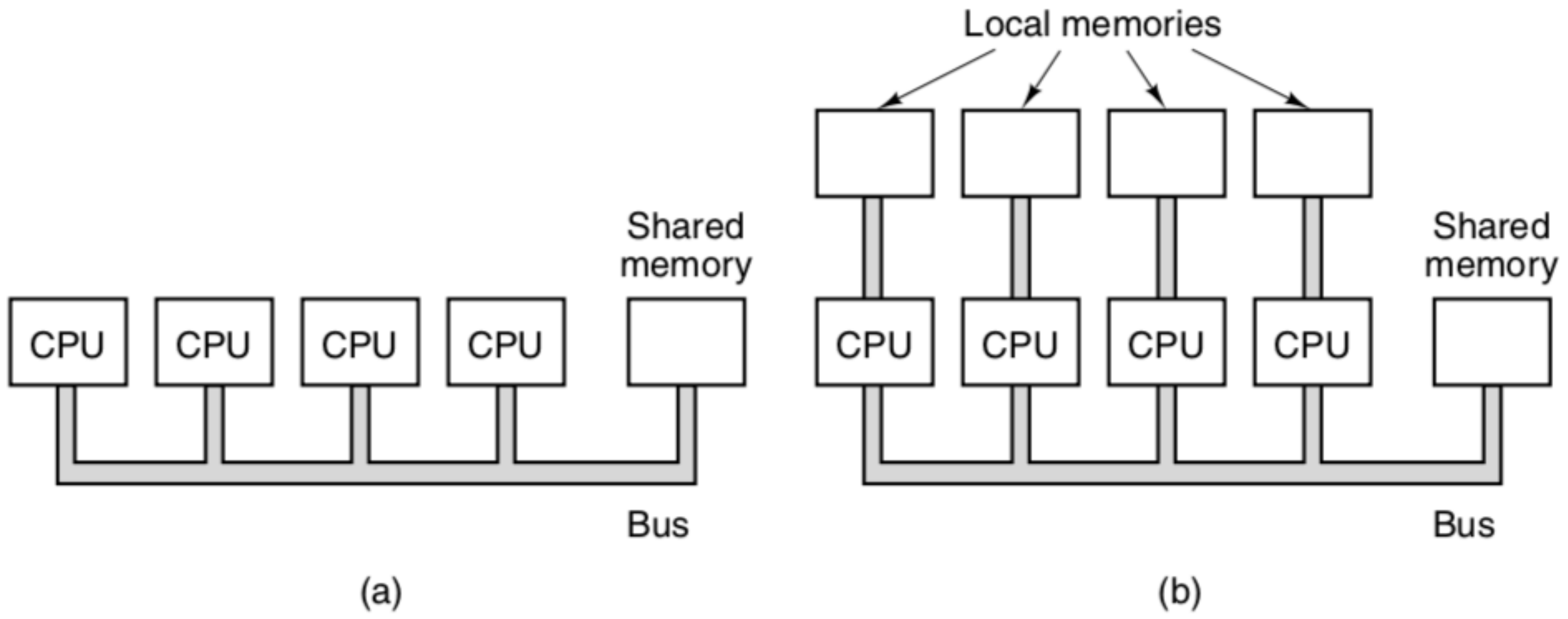
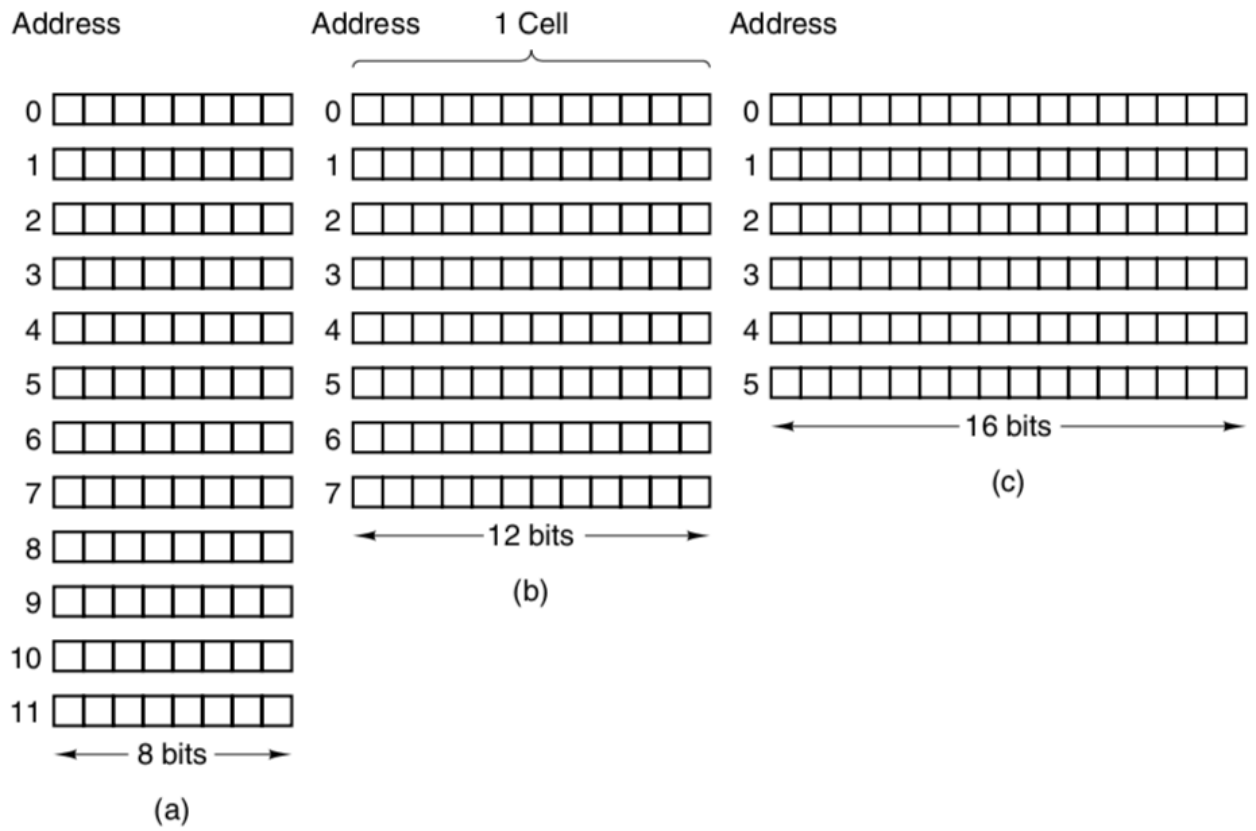


Figure 2-8. (a) A single-bus multiprocessor. (b) A multicomputer with local memories.

Primary memory

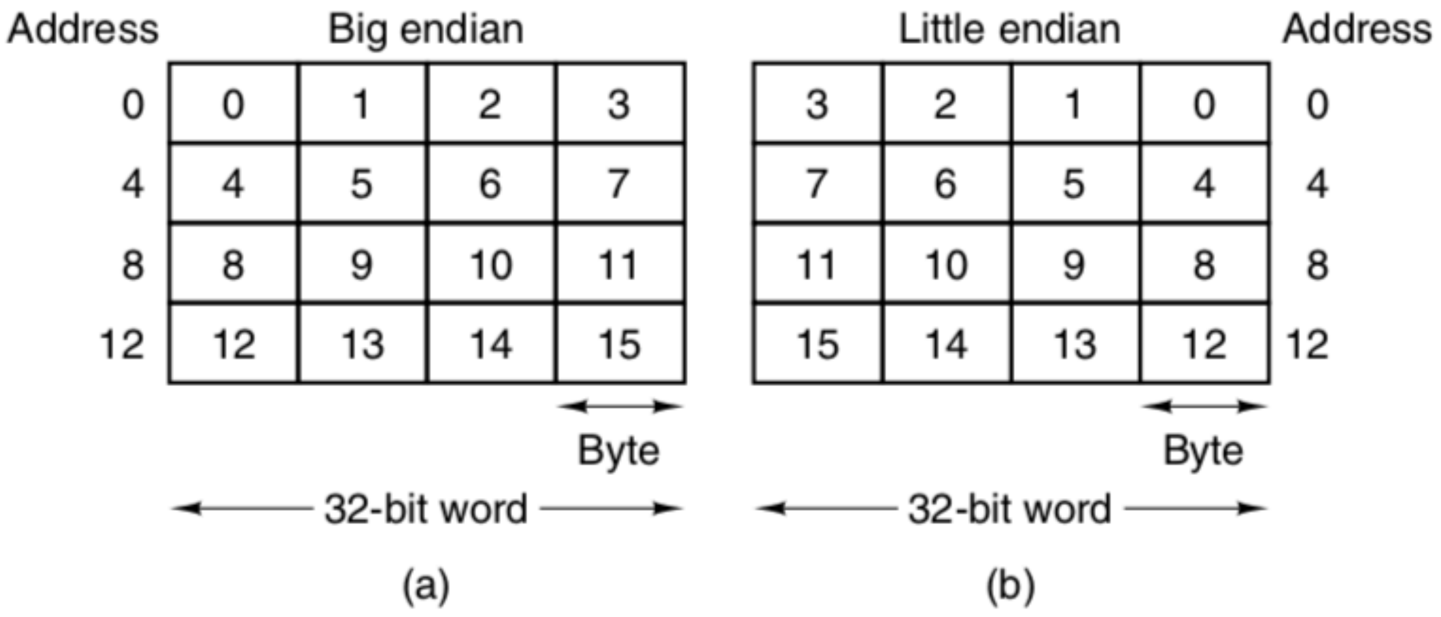
The basic unit of memory is the binary digit, called a bit



Nearly all computer manufacturers have standardized on an 8-bit cell, which is called a byte. Bytes are grouped into words. A computer with a 32-bit word has 4 bytes/word, whereas a computer with a 64-bit word has 8 bytes/word.

Byte Ordering

The bytes in a word can be numbered from left-to-right or right-to-left. At first it might seem that this choice is unimportant



Where the numbering begins at the ‘‘big’ end is called a big endian computer, in contrast to the little endian.

Error-Correcting Codes

As a simple example of an error-detecting code, consider a code in which a single parity bit is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). Such a code has a distance 2, since any single-bit error produces a codeword with the wrong parity.

It can be used to detect single errors. Whenever a word containing the wrong parity is read from memory, an error condition is signaled.

The program cannot continue, but at least no incorrect results are computed.

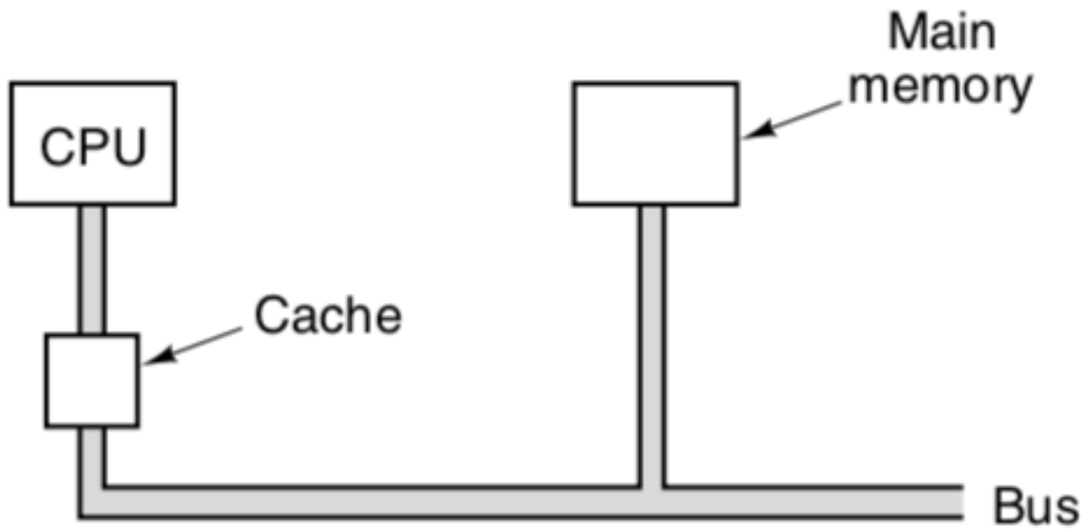
Imagine that we want to design a code with m data bits and r check bits that will allow all single-bit errors to be corrected.
Each of the 2^m legal memory words has n illegal codewords at a distance 1 from it. These are formed by systematically inverting each of the n bits in the n-bit codeword formed from it.
Thus each of the 2^m legal memory words requires n + 1 bit patterns dedicated to it (for the n possible errors and correct pattern). Since the total number of bit patterns is 2^n we must have $(n + 1)2^m \leq 2^n$.

Using $n = m + r$ this requirement becomes $(m + r + 1) \leq 2r$.

Given m, this puts a lower limit on the number of check bits needed to correct single errors.

Cache Memory

Historically, CPUs have always been faster than memories. As memories have improved, so have CPUs, preserving the imbalance.



The basic idea behind a cache is simple: the most heavily used memory words are kept in the cache. When the CPU needs a word, it first looks in the cache. Only if the word is not there does it go to main memory. If a substantial fraction of the words are in the cache, the average access time can be greatly reduced.

Examples of gates

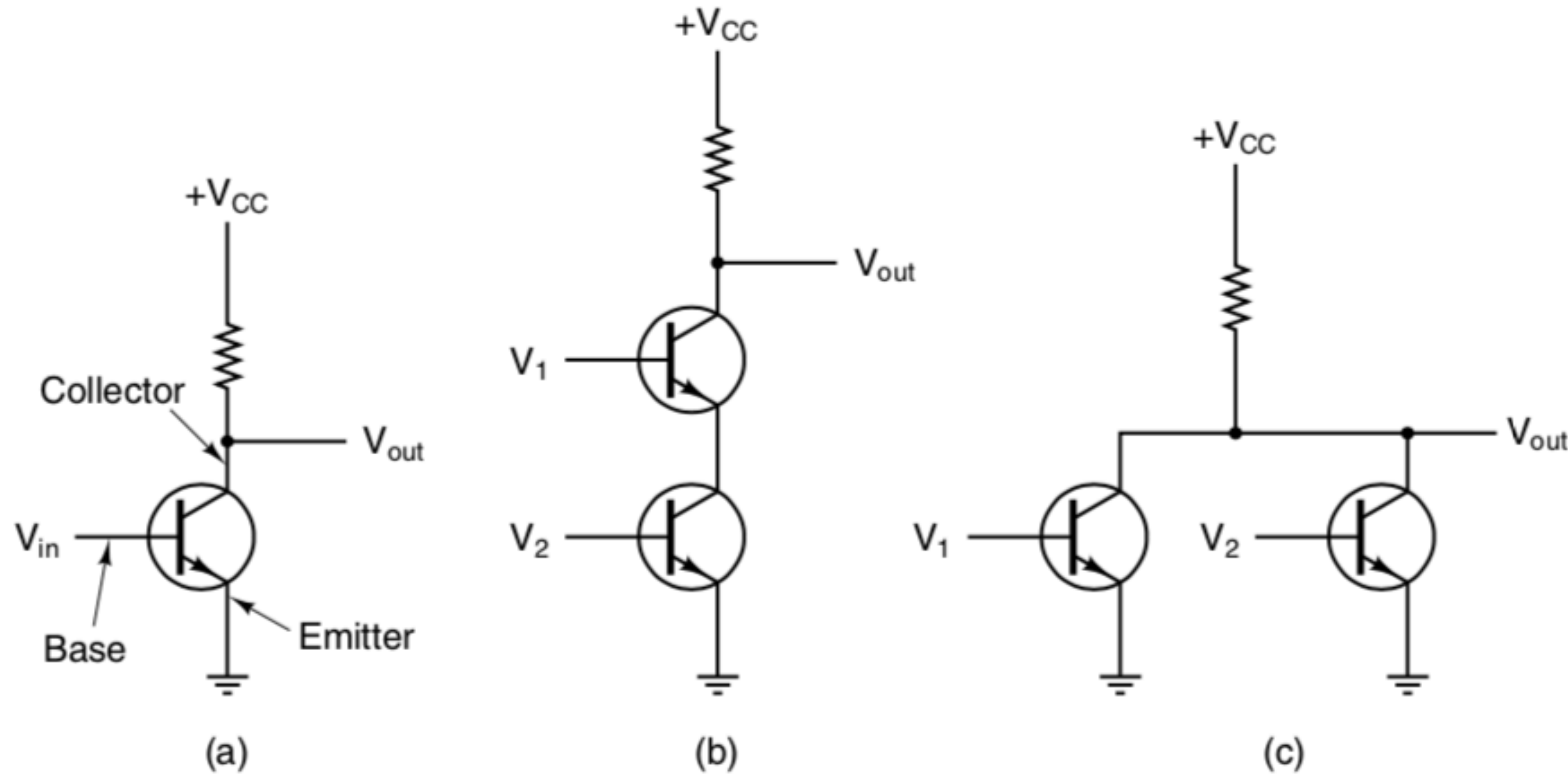


Figure 3-1. (a) A transistor inverter. (b) A NAND gate. (c) A NOR gate.