Computing systems organization (Chapitres 1-2-3)

Francesco Bronzino
ArchiSys

* Slides based on Tanenbaum, "Structured Computer Organization" 5e

Topics

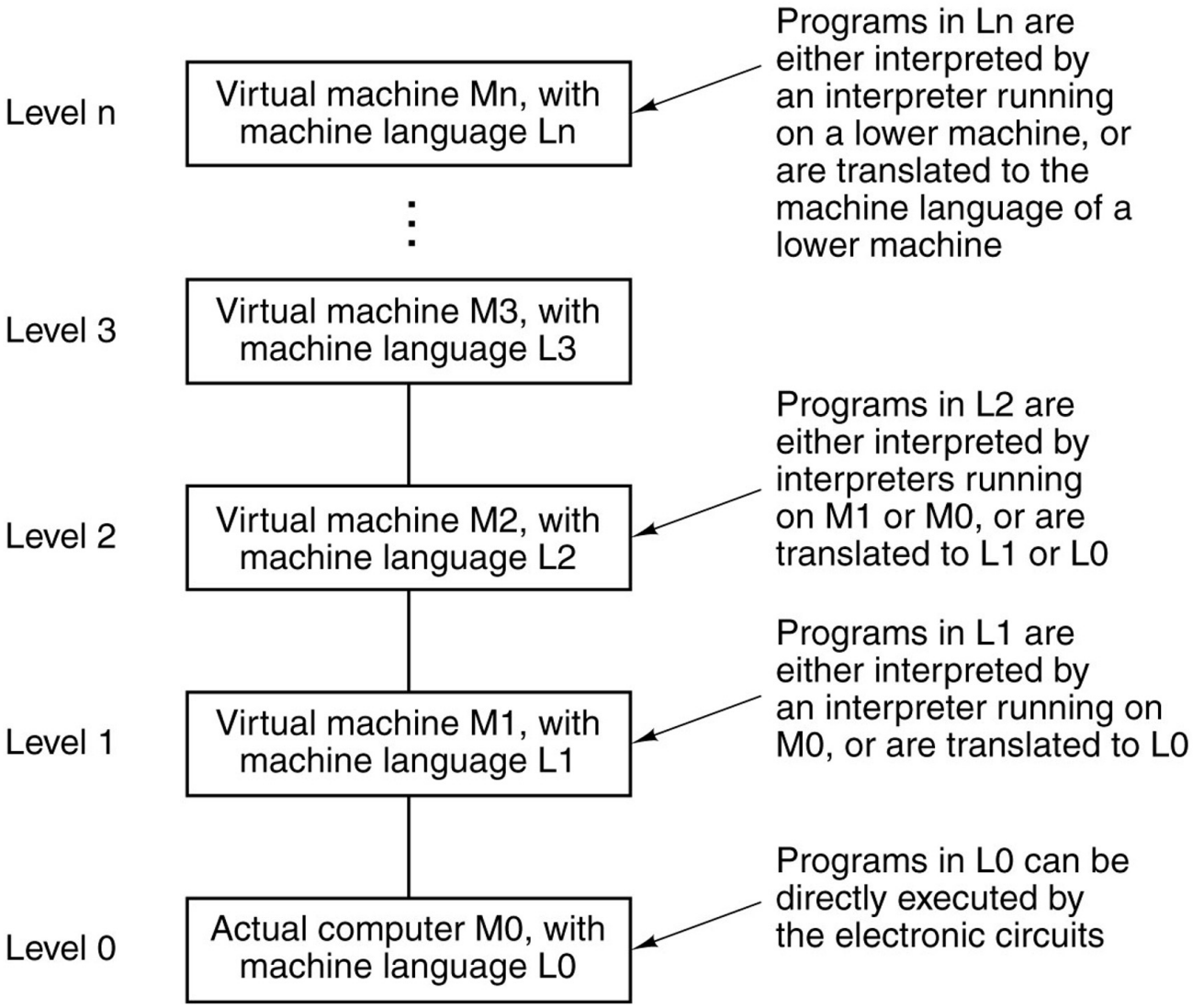
1. Introduction
2. Computer systems organization
3. The digital logic level

Introduction

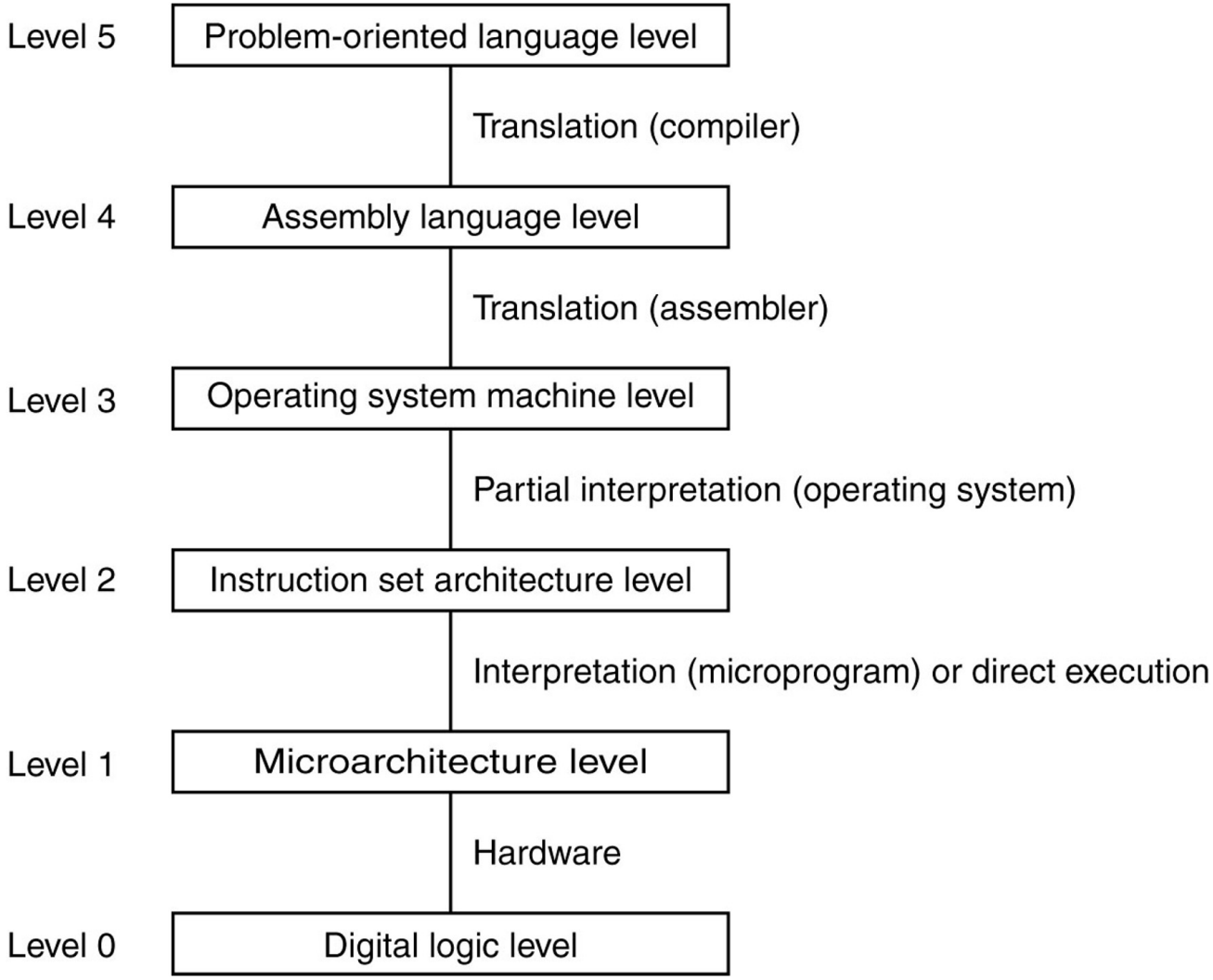
Bibliography update

- *"Structured Computer Organization."* Andrew Tanenbaum and Todd Austin. 6th Edition. Pearson
- Possible to find the PDF online (wink wink)

Languages, Levels, Virtual Machines



Contemporary Multilevel Machines



A six-level computer.

The support method for each level is indicated below it.

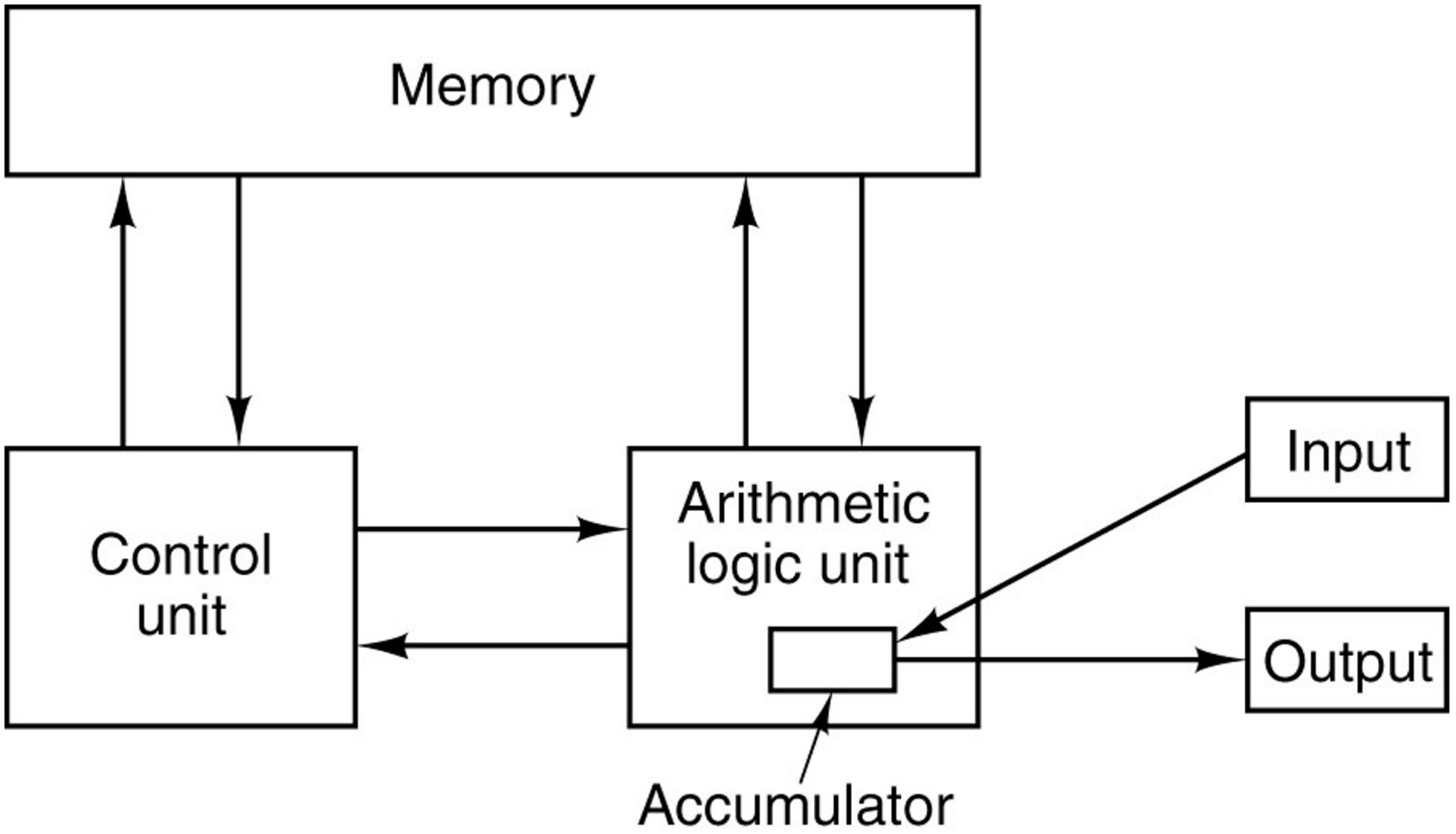
Evolution of Multilevel Machines

- Invention of microprogramming
- Invention of operating system
- Migration of functionality to microcode
- Elimination of microprogramming

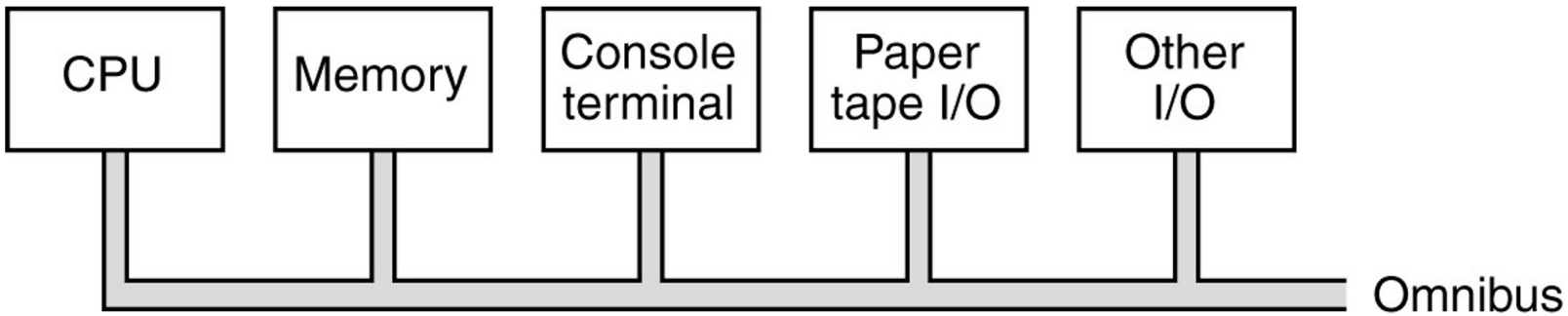
Computer Generations

- Zeroth Generation – Mechanical Computers (1642 – 1945)
- First Generation – Vacuum Tubes (1945 – 1955)
- Second Generation – Transistors (1955 – 1965)
- Third Generation – Integrated Circuits (1965 – 1980)
- Fourth Generation – Very Large Scale Integration (1980 – ?)

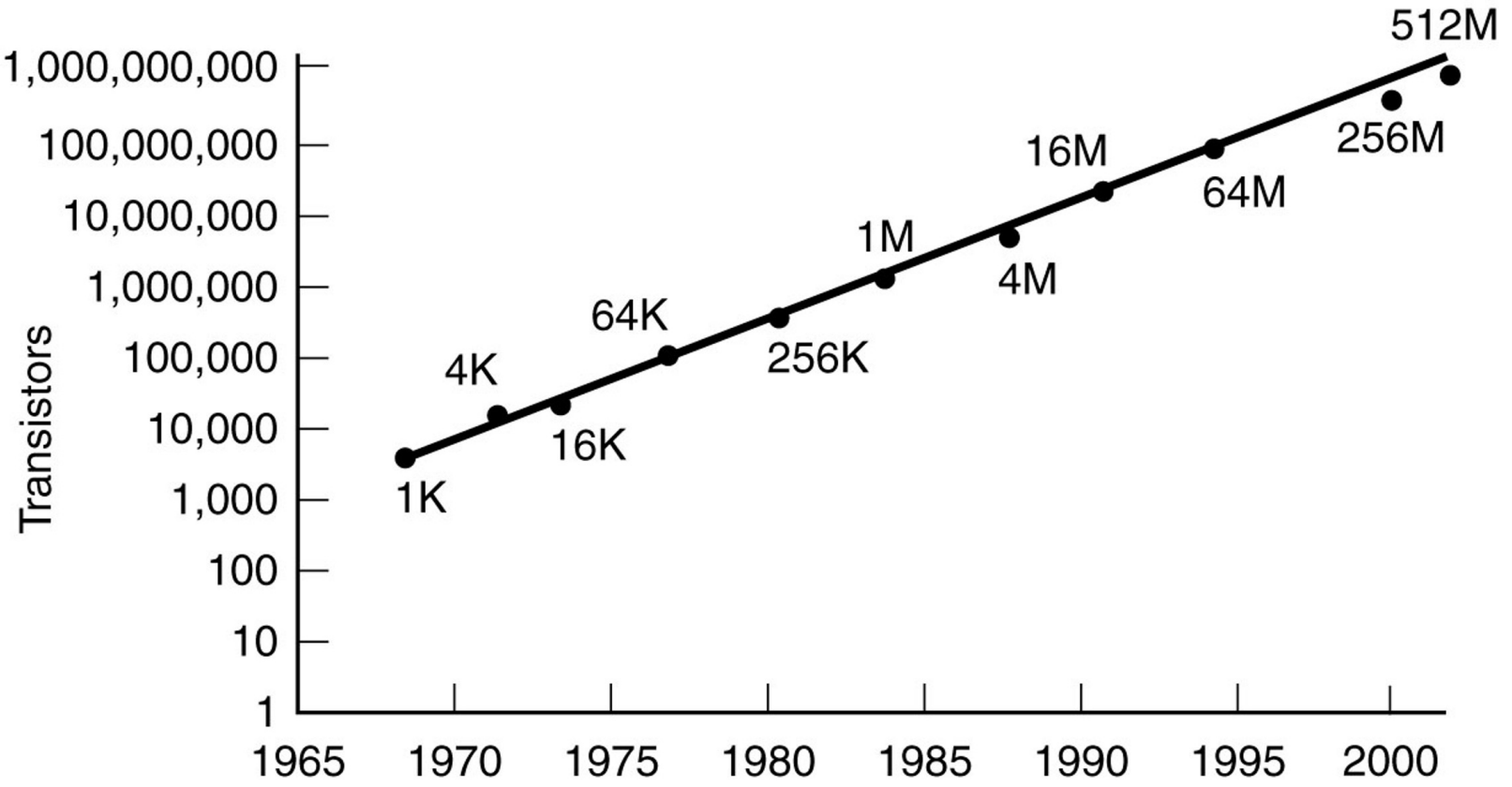
The Von Neumann Machine



PDP-8 Innovation – Single Bus



Technological and Economic Forces

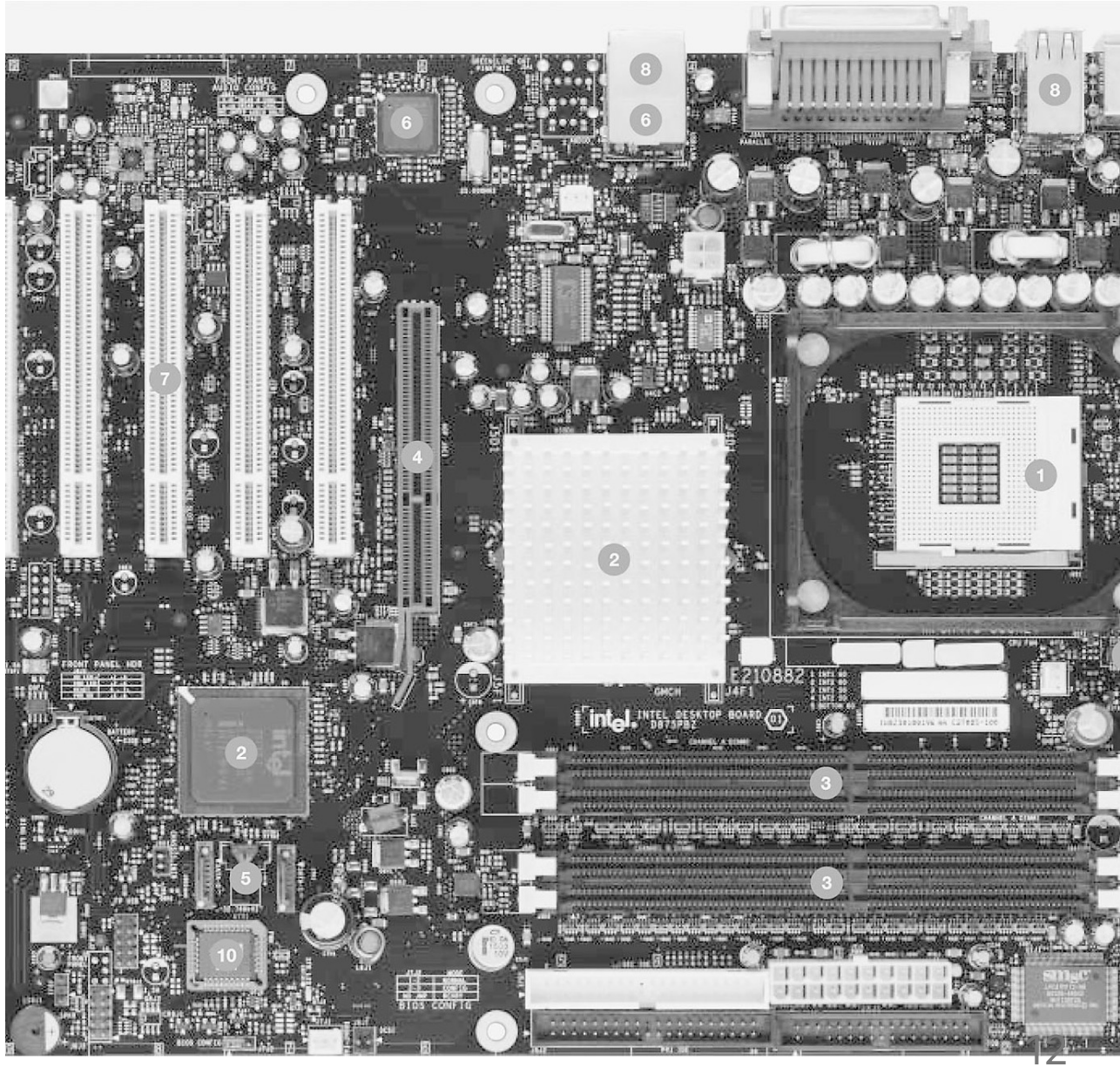


Moore's law predicts a 60-percent annual increase in the number of transistors that can be put on a chip.

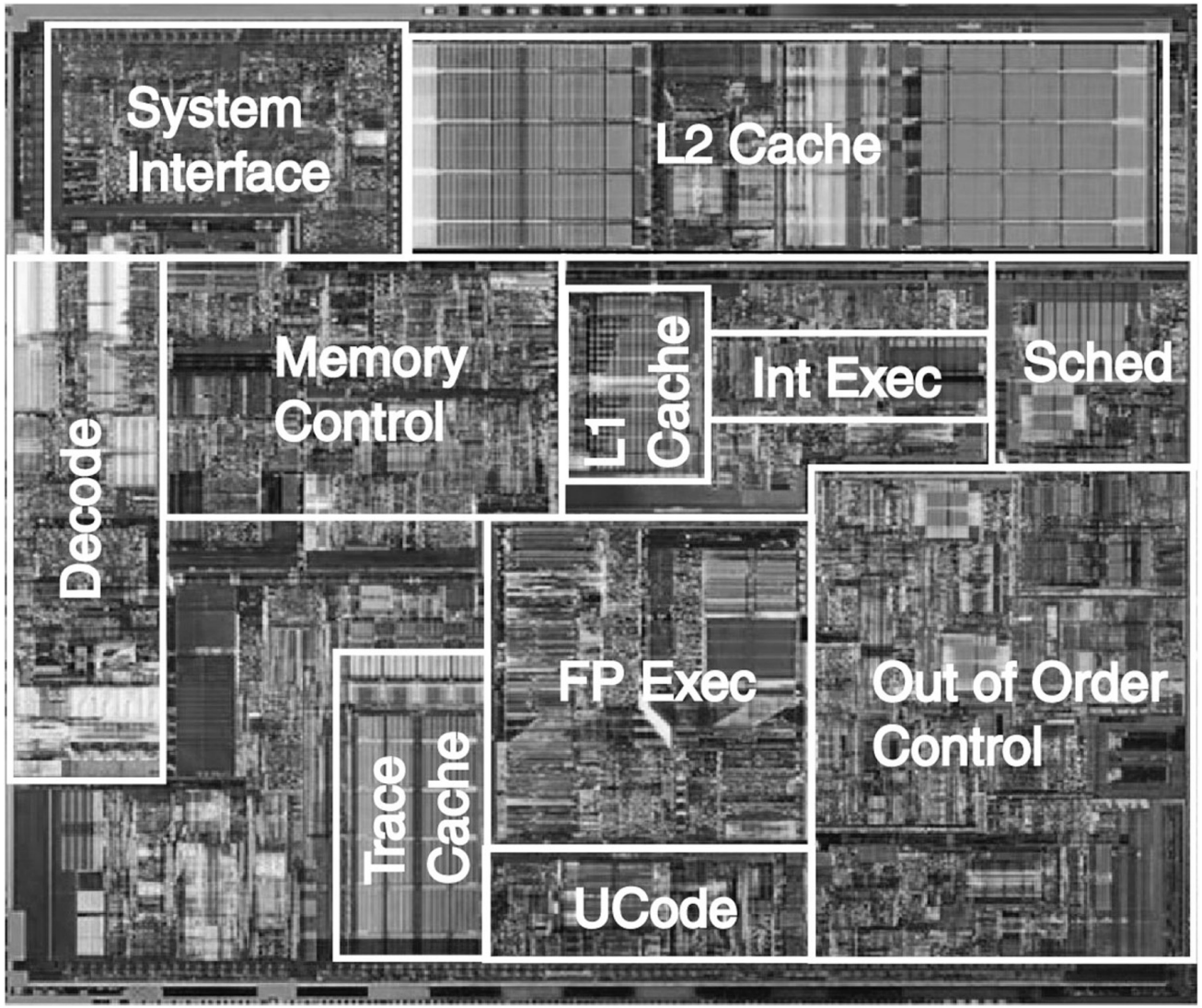
The data points given in this figure are memory sizes, in bits.

Personal Computer

1. Pentium 4 socket
2. 875P Support chip
3. Memory sockets
4. AGP connector
5. Disk interface
6. Gigabit Ethernet
7. Five PCI slots
8. USB 2.0 ports
9. Cooling technology
10. BIOS



CPU

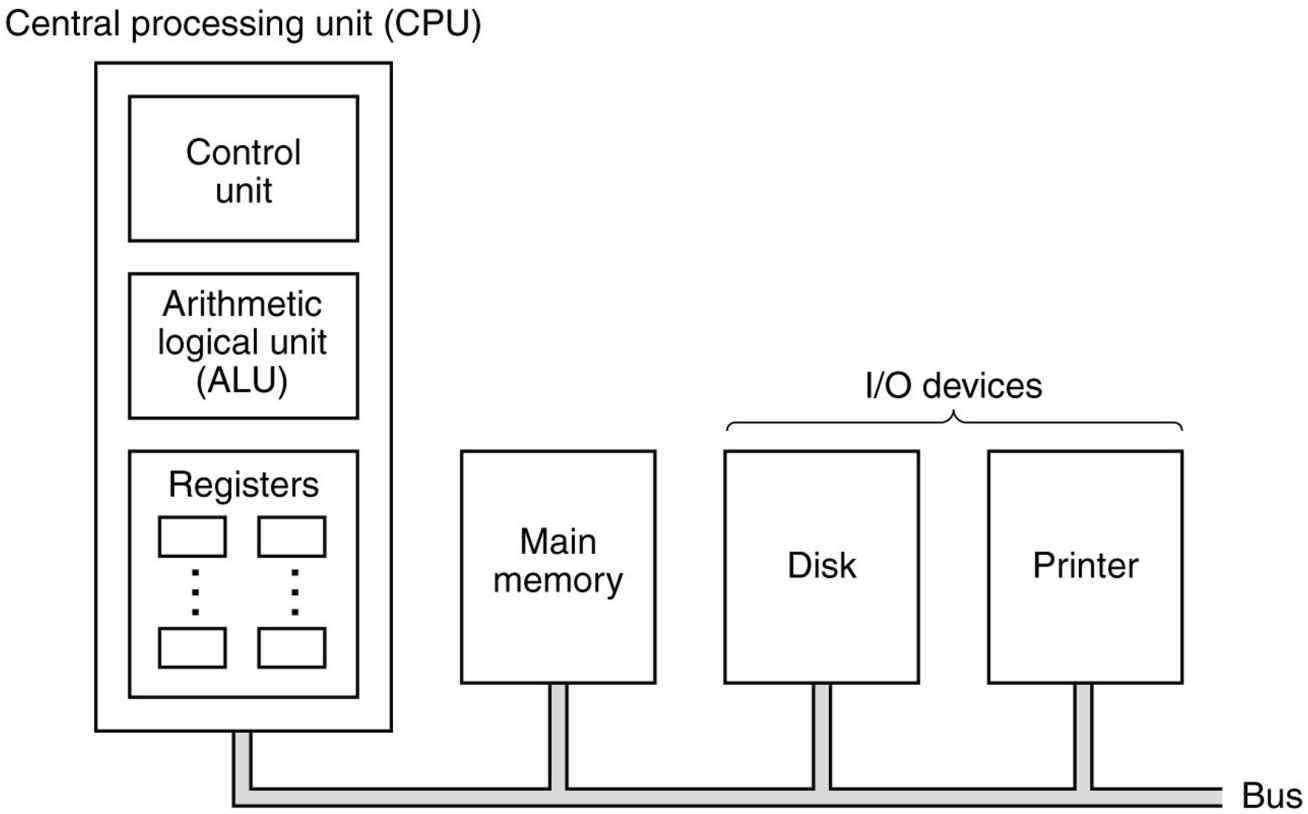


(Pentium 4 socket in the previous image)

Computer systems organization

```
A digital computer consists of an interconnected system of processors,
memories, and input/output devices
```

Central Processing Unit



The organization of a simple computer with one CPU and two I/O devices

Instruction Execution Steps

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

Interpreter (1)

```
public class Interp {
    static int PC;                  // program counter holds address of next instr
    static int AC;                  // the accumulator, a register for doing arithmetic
    static int instr;               // a holding register for the current instruction
    static int instr_type;          // the instruction type (opcode)
    static int data_loc;            // the address of the data, or –1 if none
    static int data;                // holds the current operand
    static boolean run_bit = true;  // a bit that can be turned off to halt the machine

    public static void interpret(int memory[ ], int starting_address) {
        // This procedure interprets programs for a simple machine with instructions having
        // one memory operand. The machine has a register AC (accumulator), used for
        // arithmetic. The ADD instruction adds an integer in memory to the AC, for example.
        // The interpreter keeps running until the run bit is turned off by the HALT instruction.
        // The state of a process running on this machine consists of the memory, the
        // program counter, the run bit, and the AC. The input parameters consist of
        // of the memory image and the starting address.
```

An interpreter for a simple computer (written in Java).

Interpreter (2)

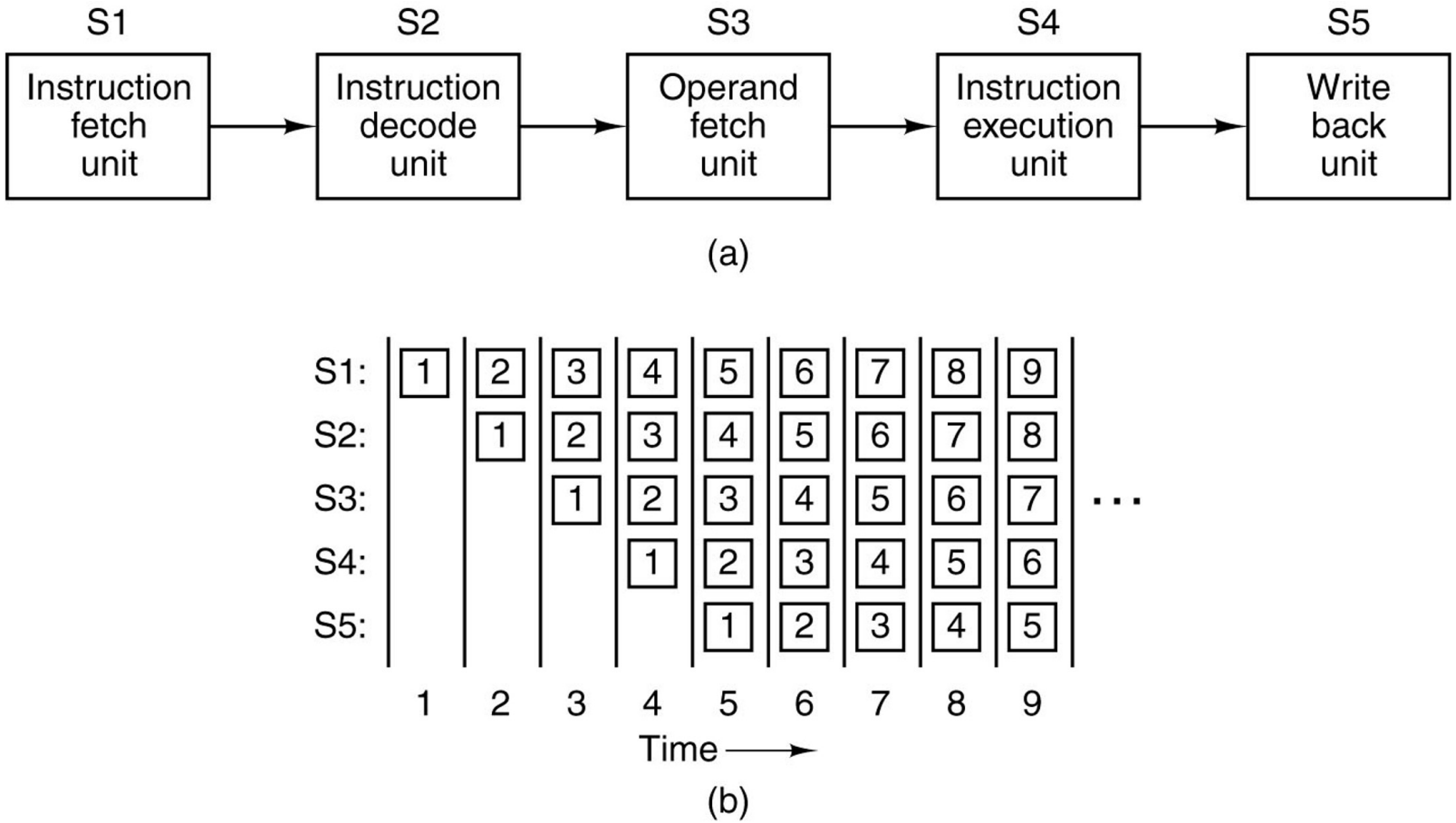
```
        PC = starting_address;
        while (run_bit) {
            instr = memory[PC];                    // fetch next instruction into instr
            PC = PC + 1;                           // increment program counter
            instr_type = get_instr_type(instr);    // determine instruction type
            data_loc = find_data(instr, instr_type); // locate data (–1 if none)
            if (data_loc >= 0)                     // if data_loc is –1, there is no operand
                data = memory[data_loc];           // fetch the data
            execute(instr_type, data);             // execute instruction
        }

    }

    private static int get_instr_type(int addr) { ... }
    private static int find_data(int instr, int type) { ... }
    private static void execute(int type, int data) { ... }
}
```

An interpreter for a simple computer (written in Java).

Design Principles for Modern Computers

- All instructions directly executed by hardware
- Maximize rate at which instructions are issued
- Instructions should be easy to decode
- Only loads, stores should reference memory
- Provide plenty of registers

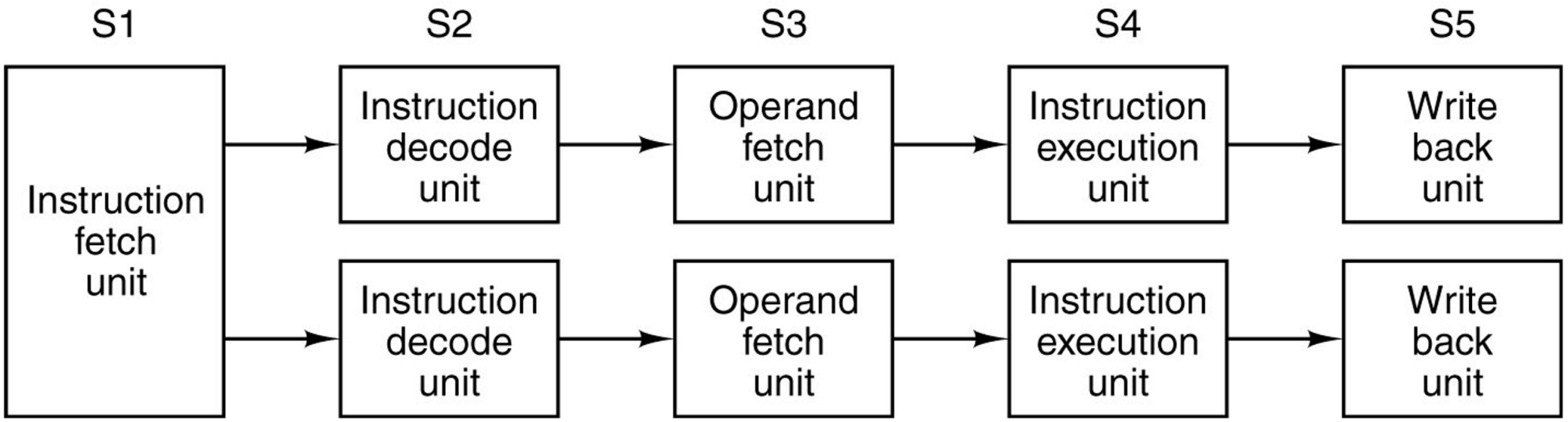
Instruction-Level Parallelism



(a) A five-stage pipeline

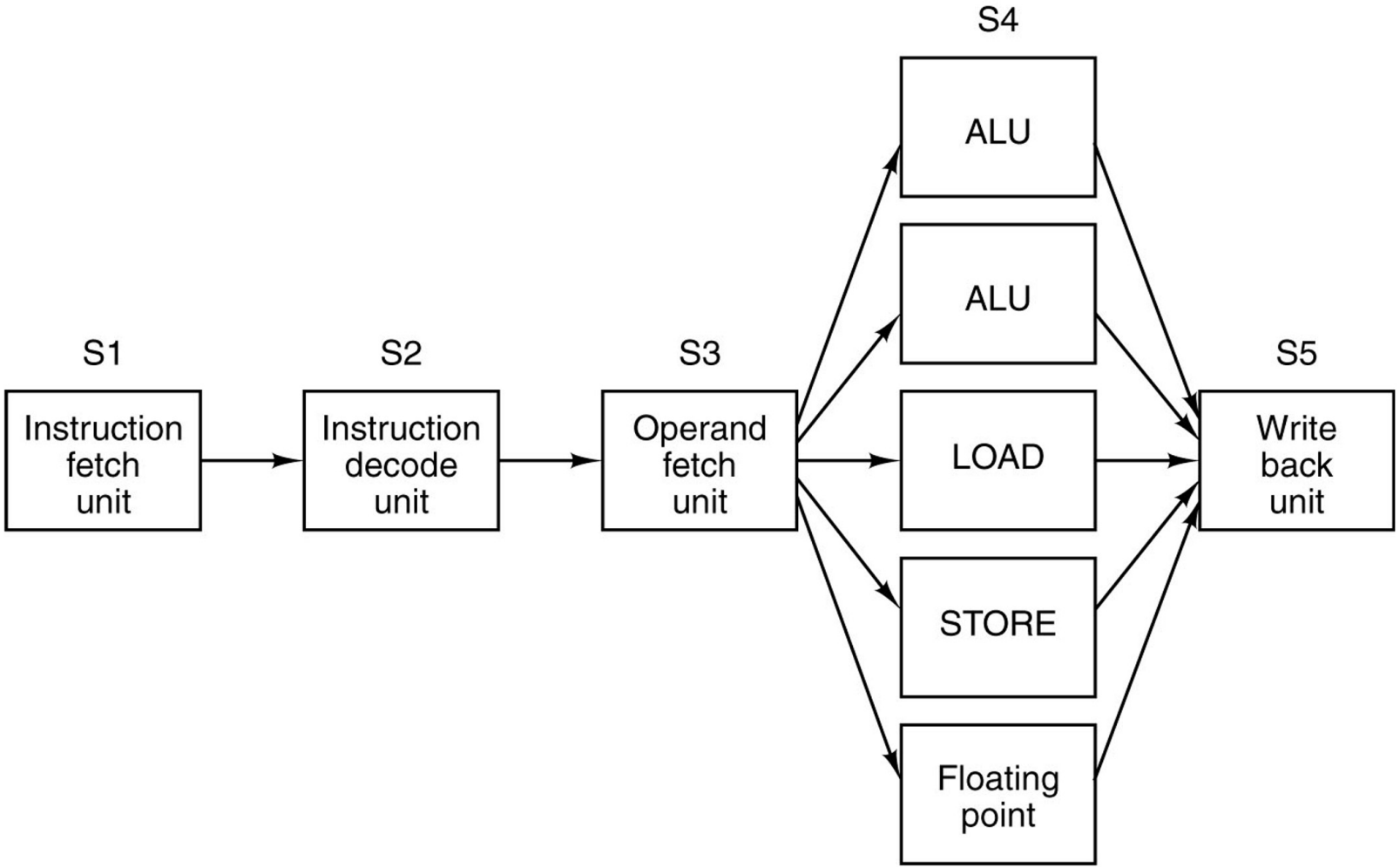
(b) The state of each stage as a function of time. Nine clock cycles are illustrated

Superscalar Architectures (1)



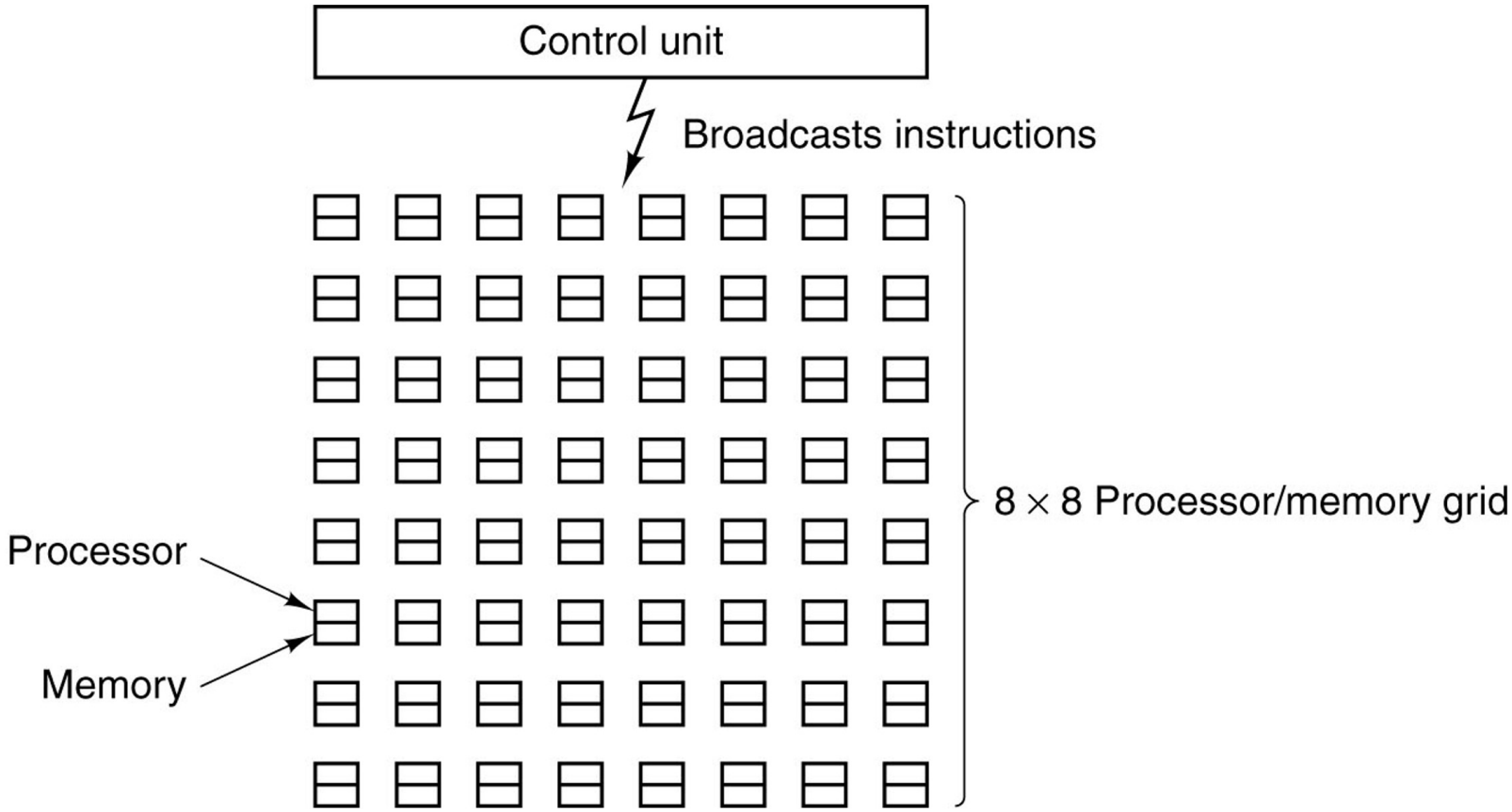
Dual five-stage pipelines with a common instruction fetch unit.

Superscalar Architectures (2)



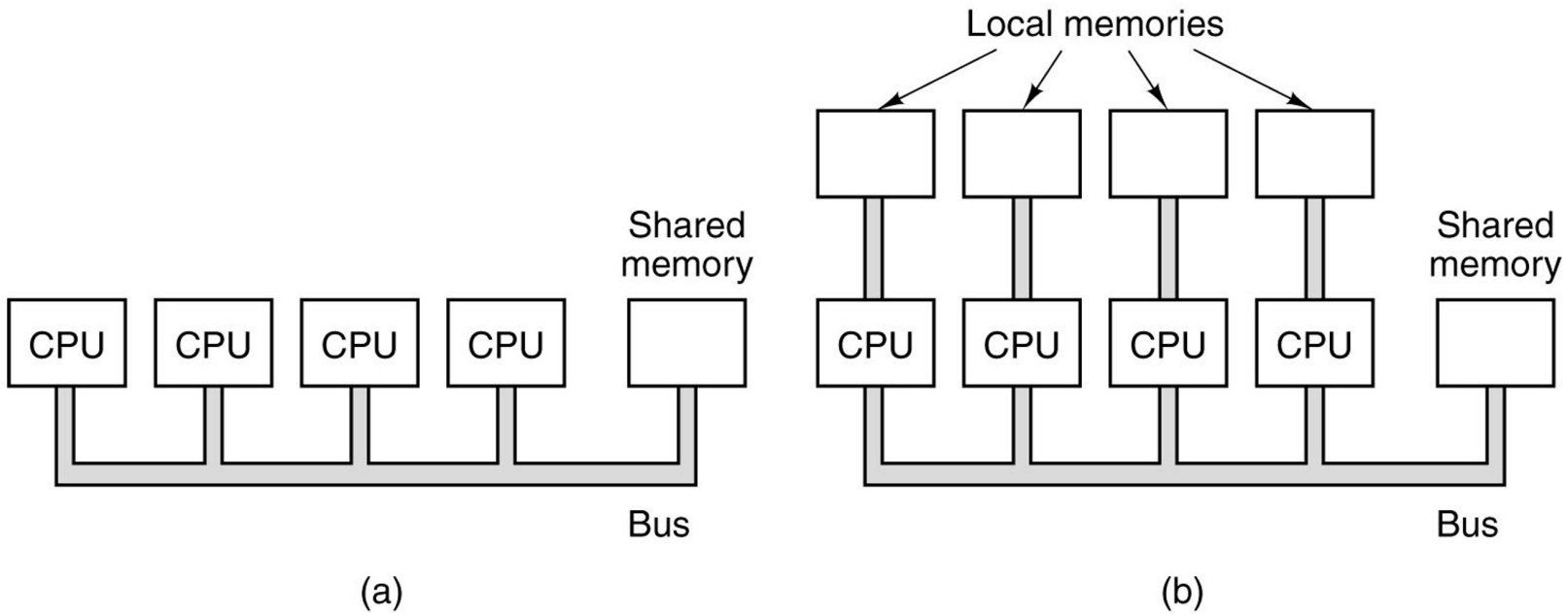
A superscalar processor with five functional units.

Processor-Level Parallelism (1)



An array of processors.

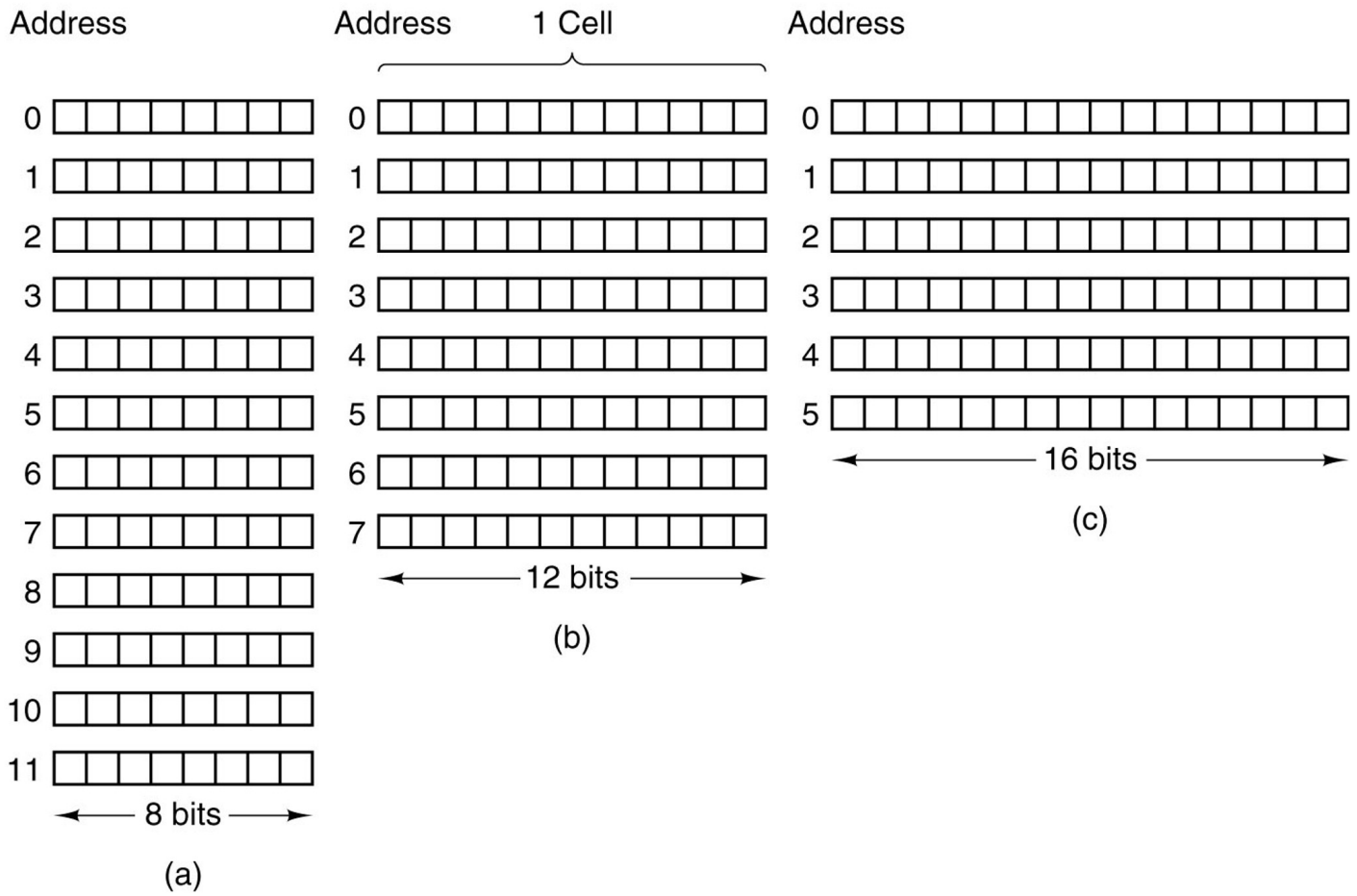
Processor-Level Parallelism (2)



(a) A single-bus multiprocessor.

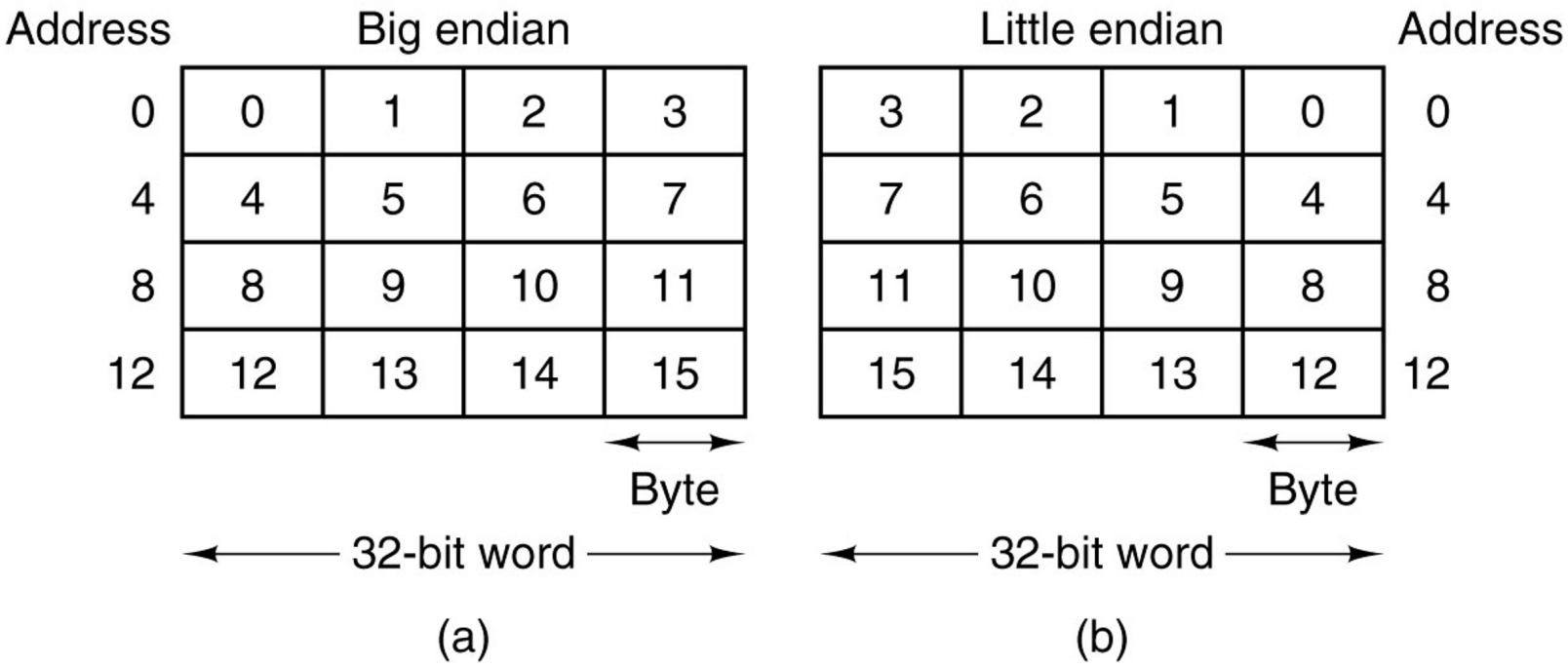
(b) A multicomputer with local memories.

Primary Memory



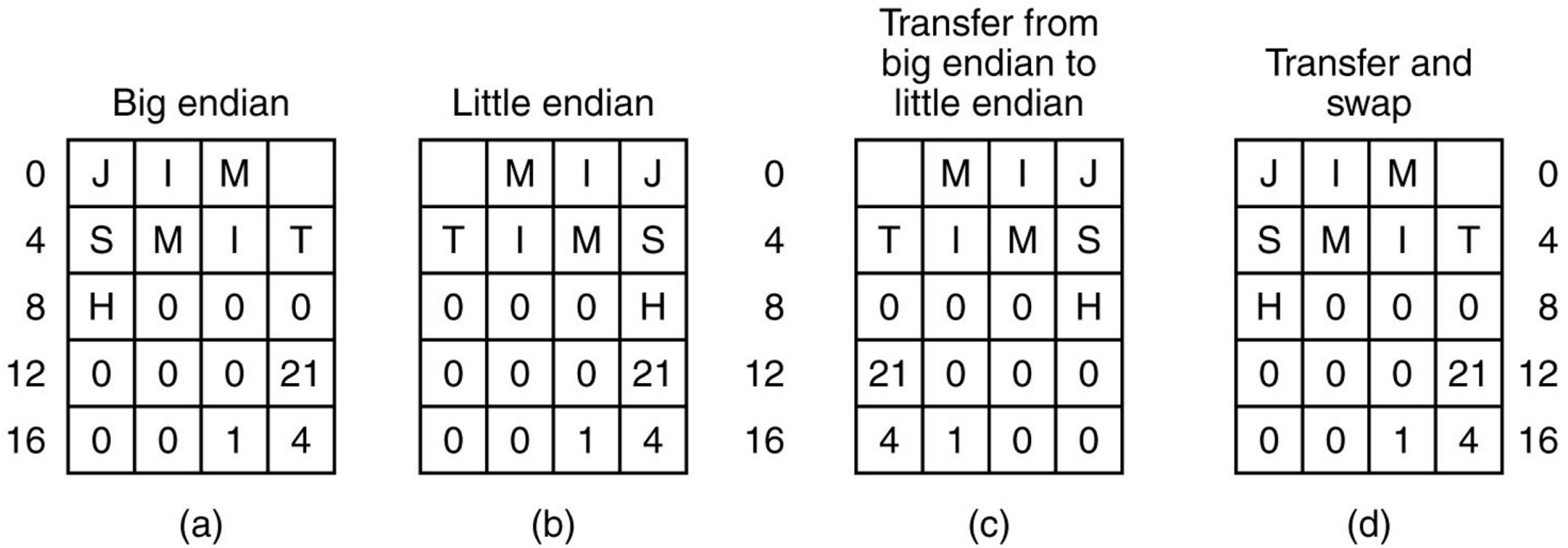
Three ways of organizing a 96-bit memory.

Byte Ordering (1)



(a) Big endian memory (b) Little endian memory

Byte Ordering (2)



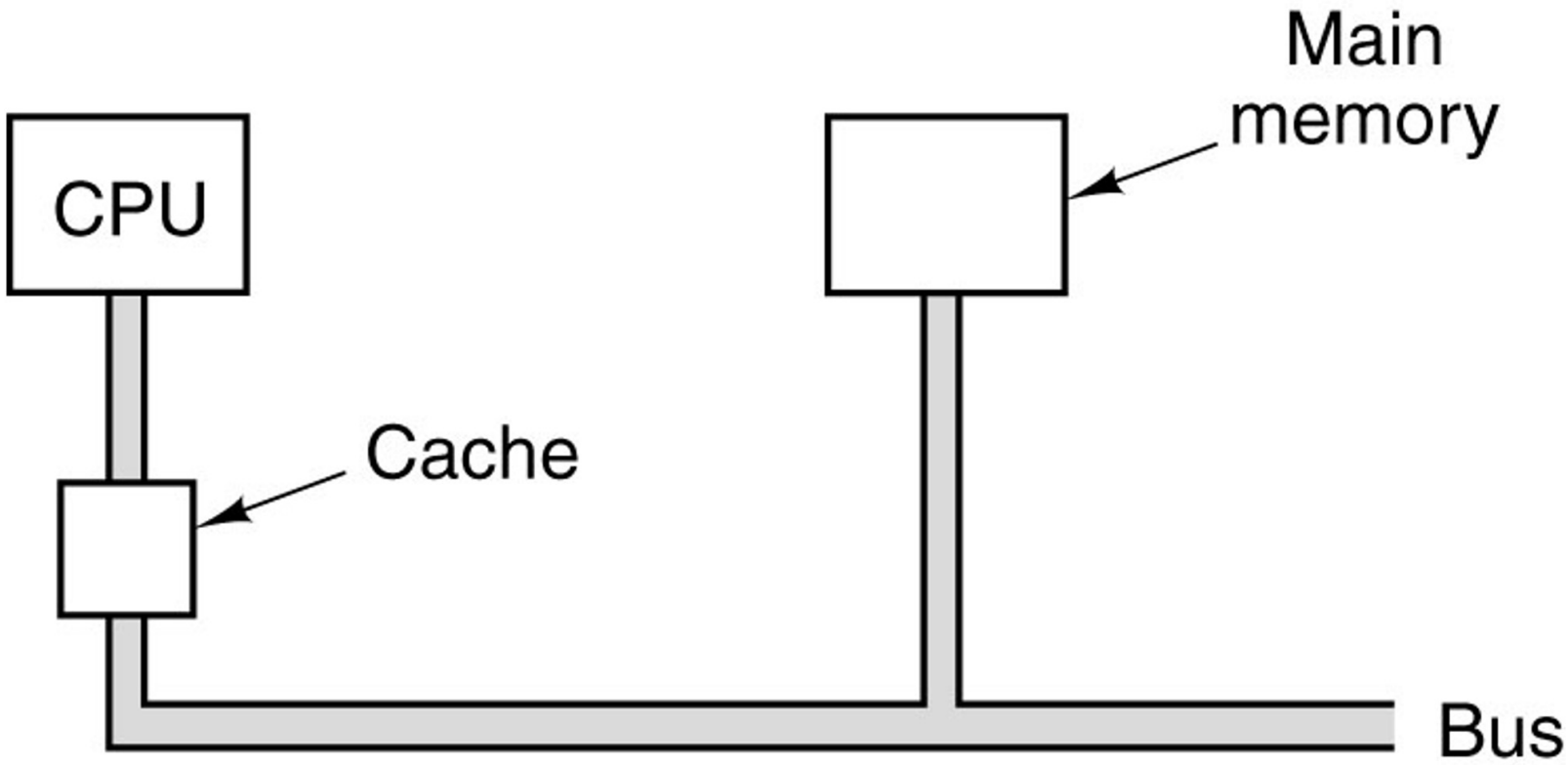
(a) A personal record for a big endian machine.

(b) The same record for a little endian machine.

(c) The result of transferring from big endian to little endian.

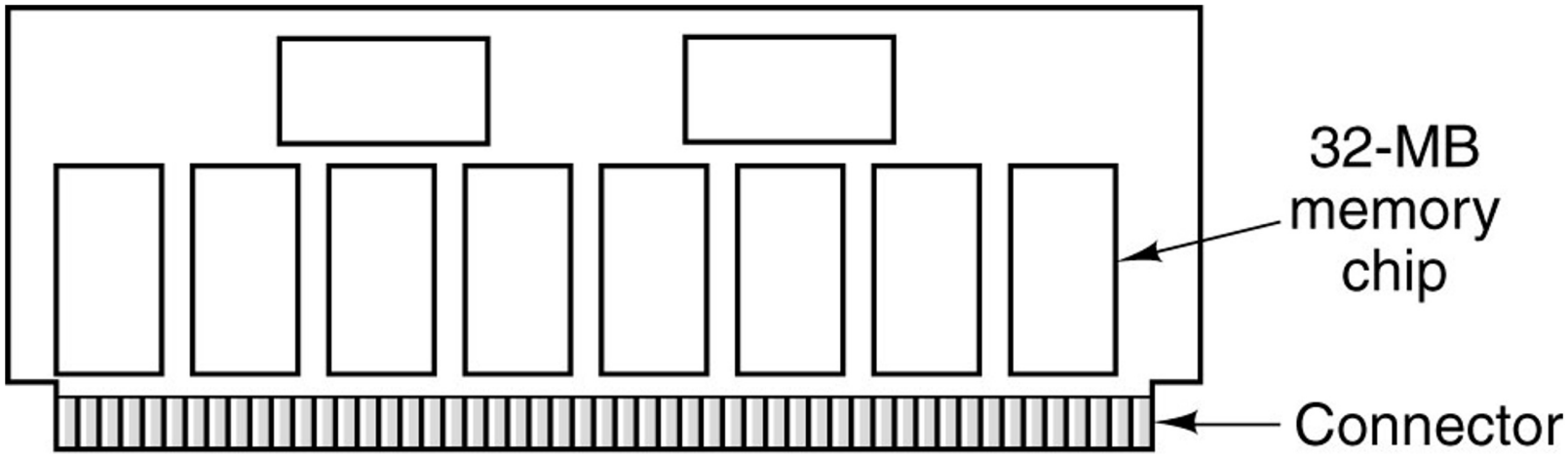
(d) The result of byte-swapping (c).

Cache Memory



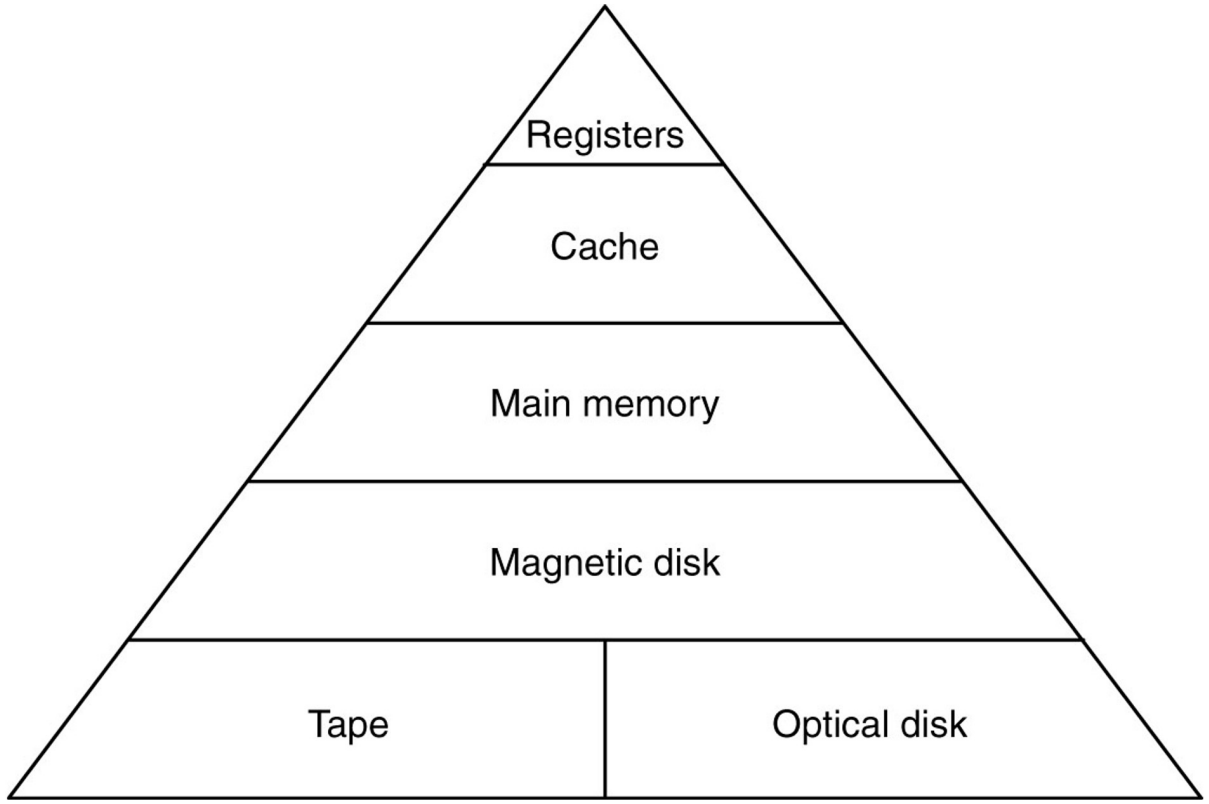
The cache is logically between the CPU and main memory.
Physically, there are several possible places it could be located.

Memory Packaging and Types



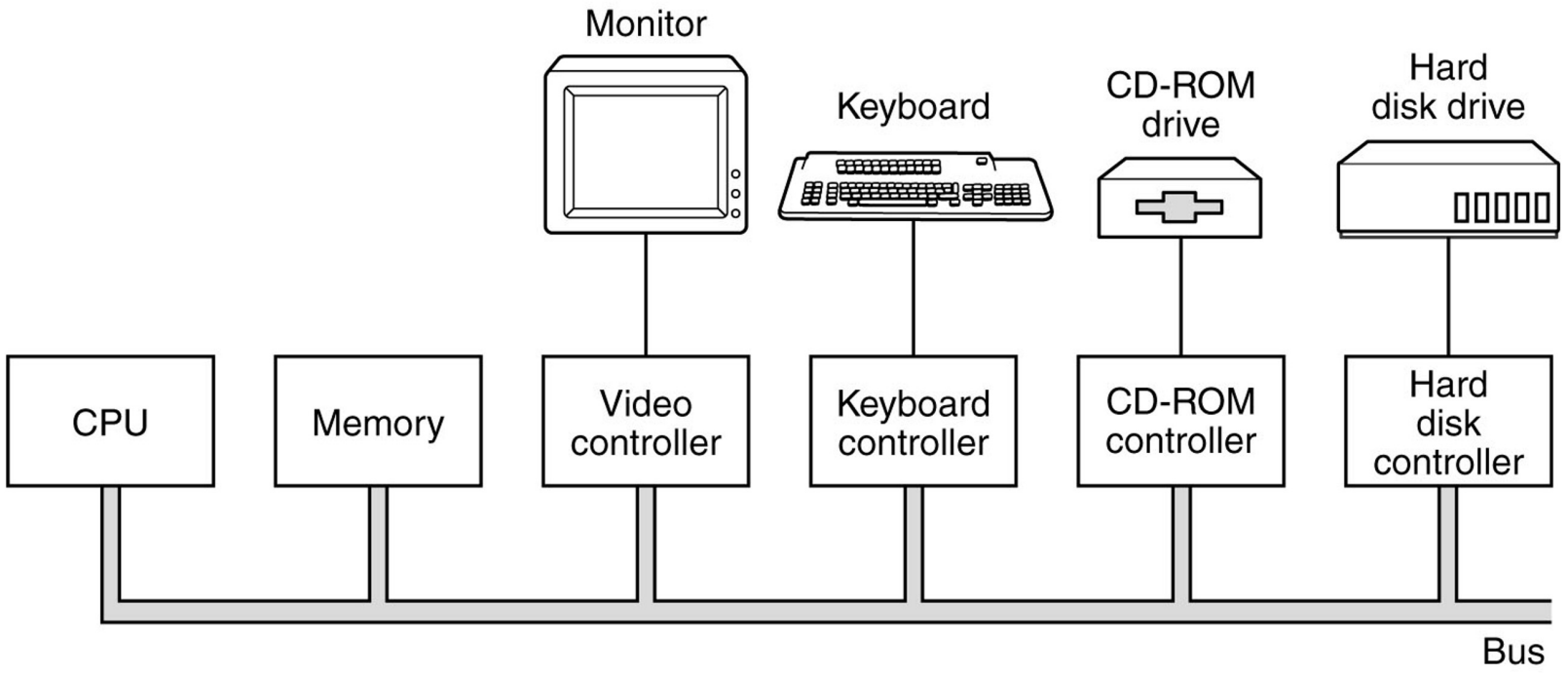
A single inline memory module (SIMM) holding 256 MB. Two of the chips control the SIMM.

Memory Hierarchies



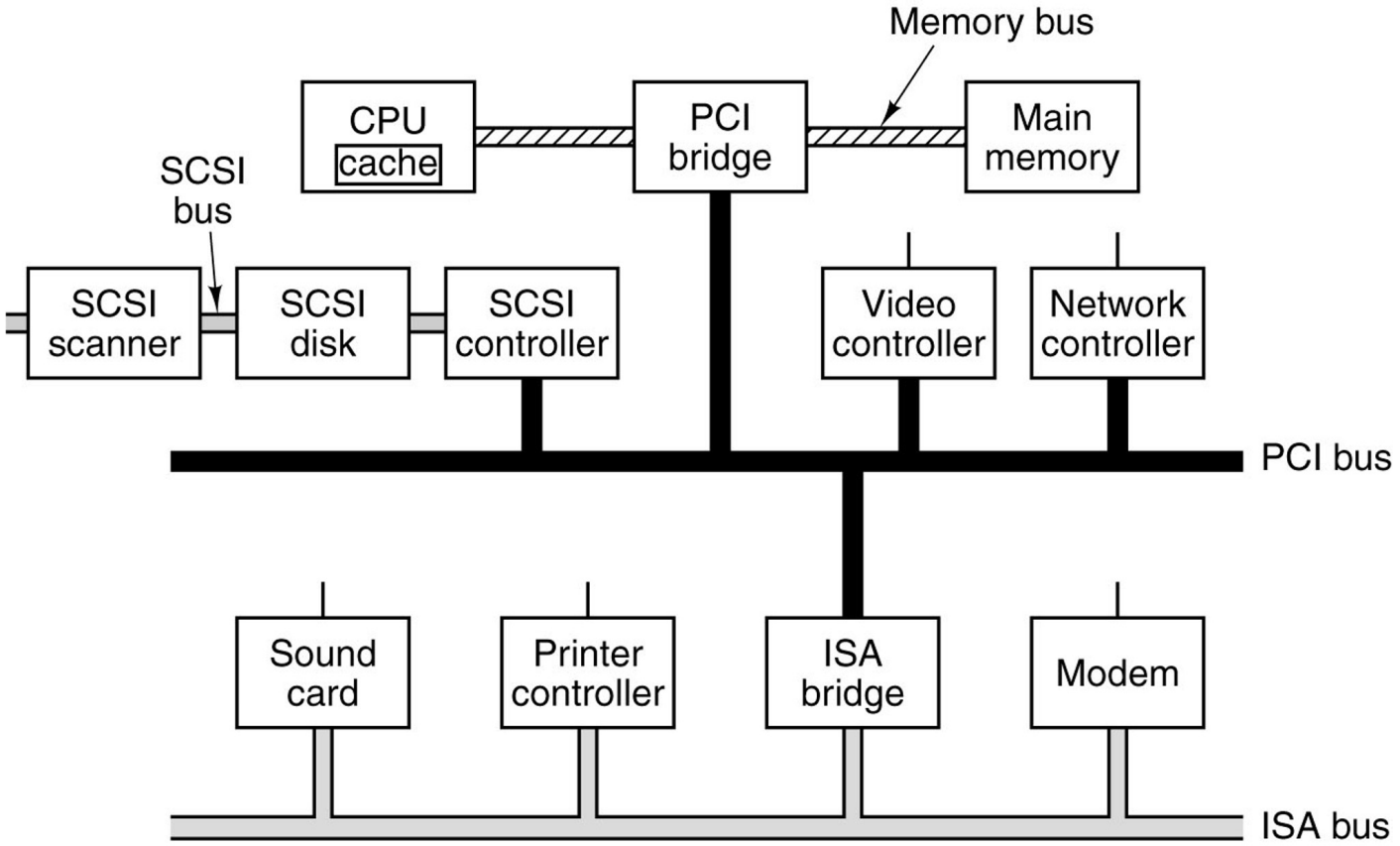
A five-level memory hierarchy.

Input / Output (1)



Logical structure of a simple personal computer.

Input / Output (2)



A typical modern PC with a PCI bus and an ISA bus.

The digital logic level

```
The computer’s real hardware
```

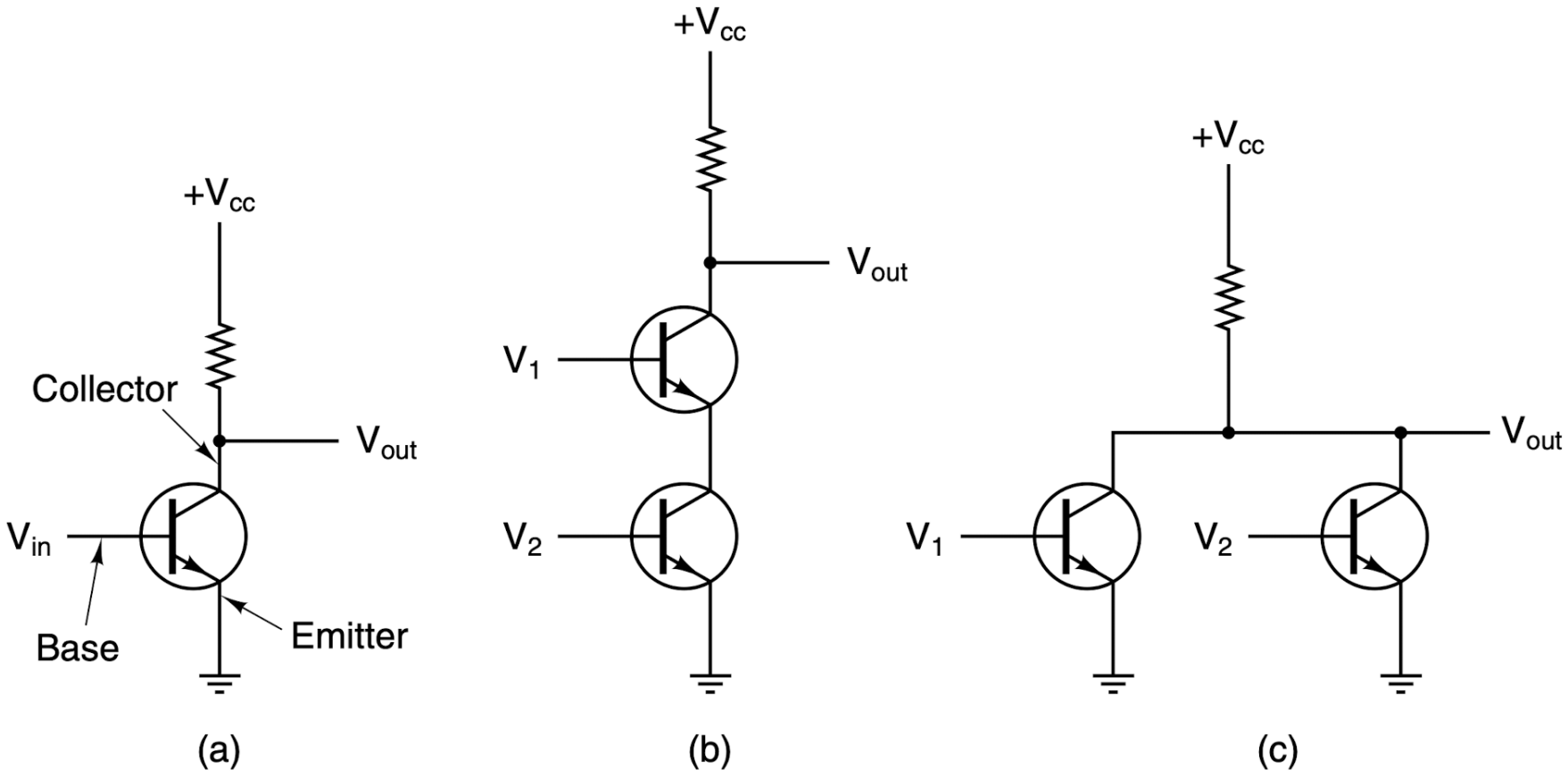
Overall notions

- The basic elements from which all digital computers are constructed are amazingly simple!
  - Small number of primitive elements combined in innumerable way
- Gates and boolean algebra
- Gate circuits
- Storing information

Gates (1)

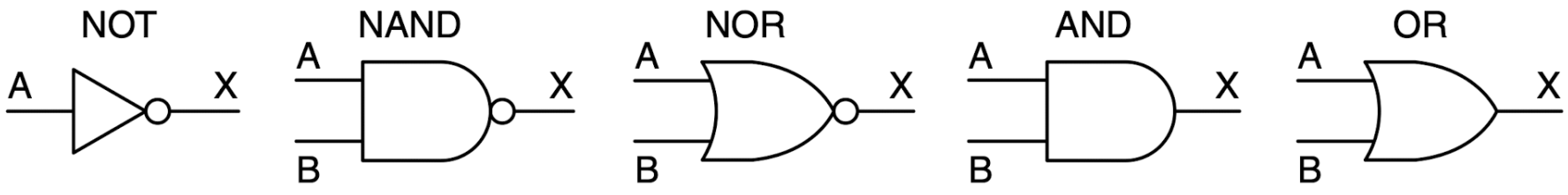
- A digital circuit is one in which only two logical values are present
- A signal between 0 and 0.5 volt represents one value
  - Voltages outside these two ranges are not permitted
- Gates compute various functions of these two-valued signal

Transistors



(a) A transistor inverter (b) A NAND gate. (c) A NOR gate.

Gates (2)



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

The symbols and functional behavior for the five basic gates.

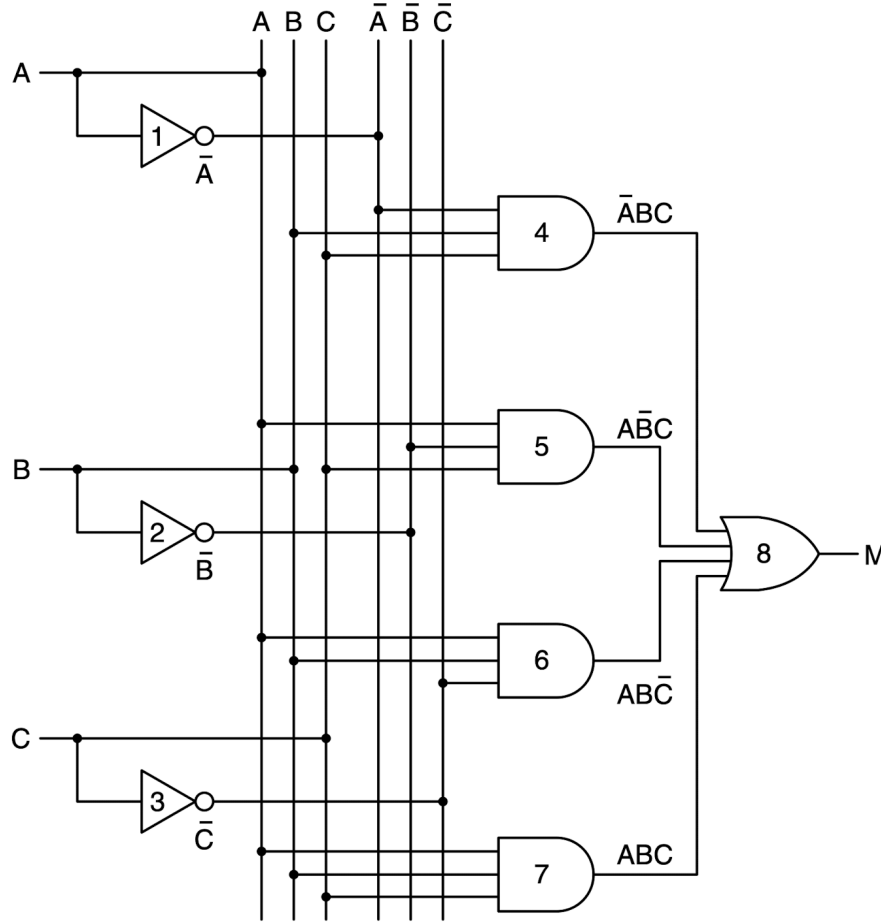
Boolean Logic (1)

(a) The truth table for the majority function of three variables.

(b) A circuit for (a).

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

(a)



(b)

Boolean Logic (2)

Method to implement a circuit for any Boolean function:

1. Write down the truth table for the function.
2. Provide inverters to generate the complement of each input.
3. Draw an AND gate for each term with a 1 in the result column.
4. Wire the AND gates to the appropriate inputs.
5. Feed the output of all the AND gates into an OR gate.

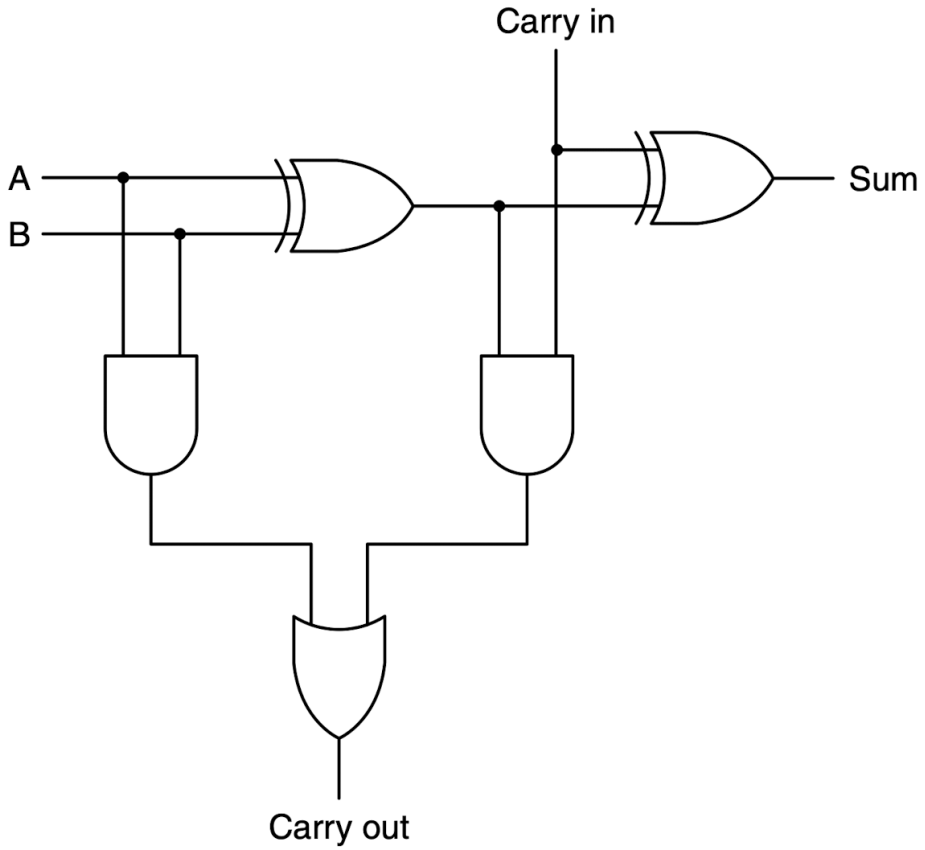
Integrated circuits

- Circuit designers try to reduce the number of gates in their products
  - Reduce the chip area needed to implement them
  - Minimize power consumption
  - Increase speed
- Gates are not manufactured or sold individually but rather in units called **Integrated Circuits**

Arithmetic operations

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)

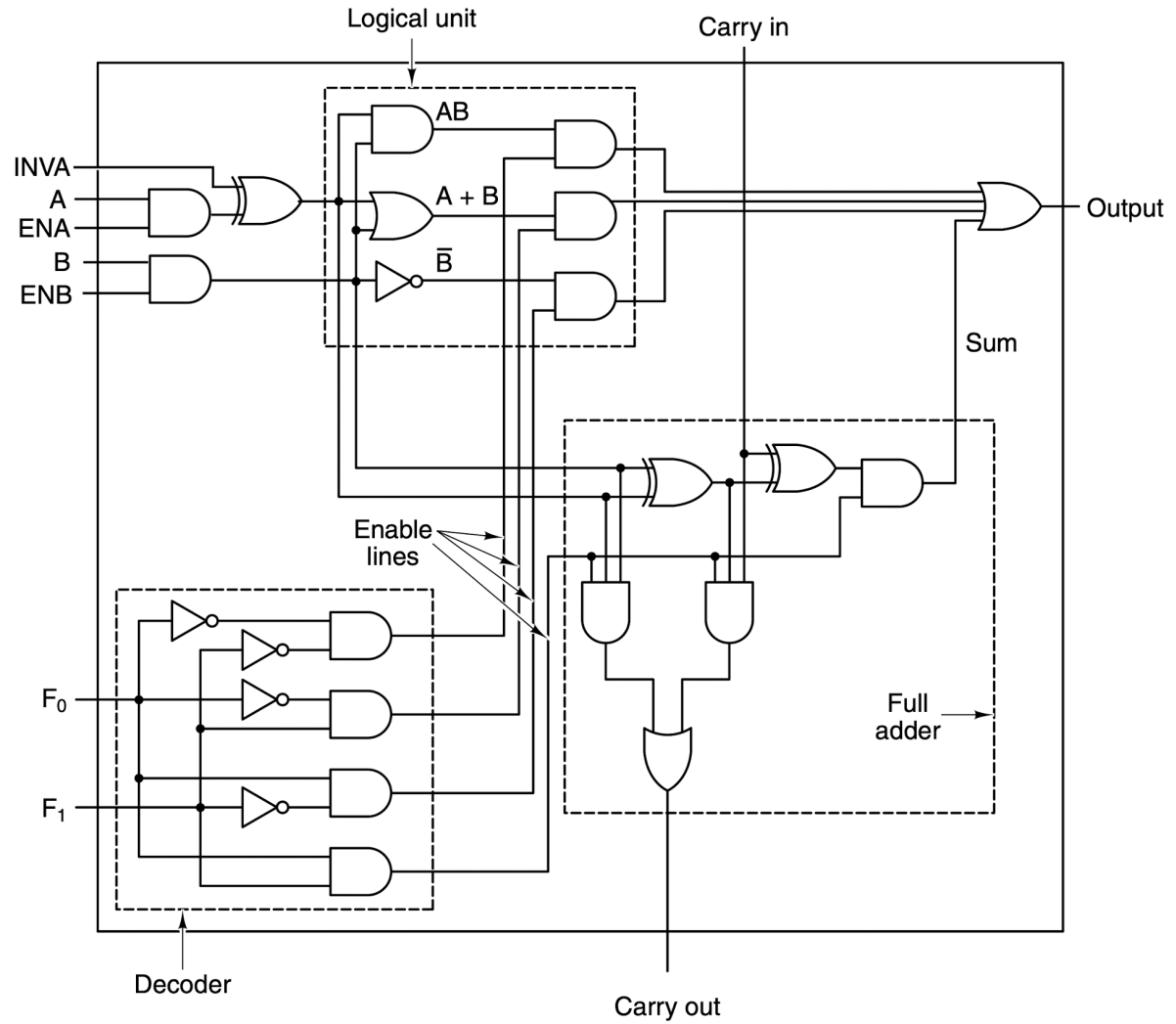


(b)

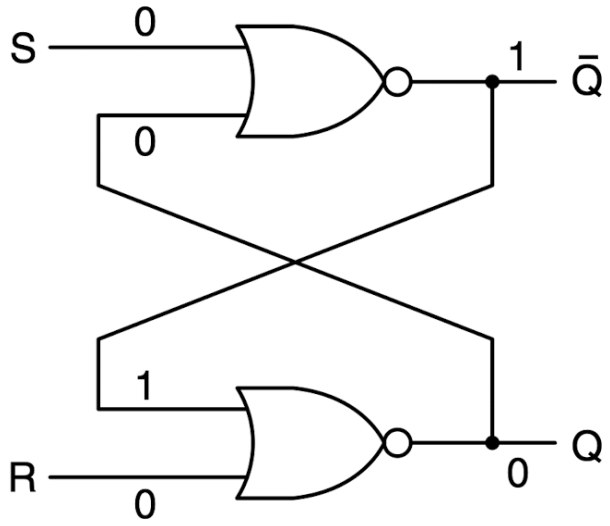
(a) Truth table for full adder. (b) Circuit for a full adder.

The Arithmetic Logic Unit

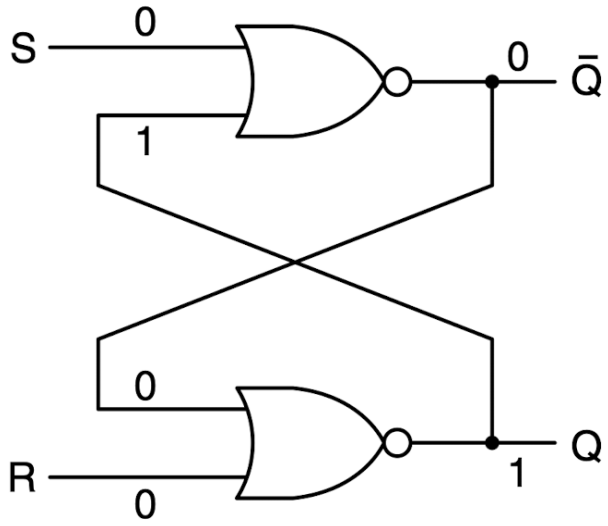
A 1-bit ALU



Memory (1)



(a)



(b)

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

(c)

((a) NOR latch in state 0. (b) NOR latch in state 1. (c) Truth table for NOR.

Memory (2)

- Not all memory works like this
- For example, DRAM uses capacitors
- Much more to say, we stop here…

Discussion: my TP1 solution