Chapter 01
Architecture & Organization

- **Architecture:** is those **attributes visible to the programmer**
  - **Instruction set**, **number of bits** used for data representation, **I/O mechanisms**, addressing techniques.
  - e.g. Is there a **multiply instruction**?
- **Organization:** is **how features are implemented**
  - **Control signals, interfaces, memory technology**.
  - e.g. **Is there a hardware multiply unit** or **is it done by repeated addition?**

Structure & Function

- **Structure:** is the **way in which components relate to each other**

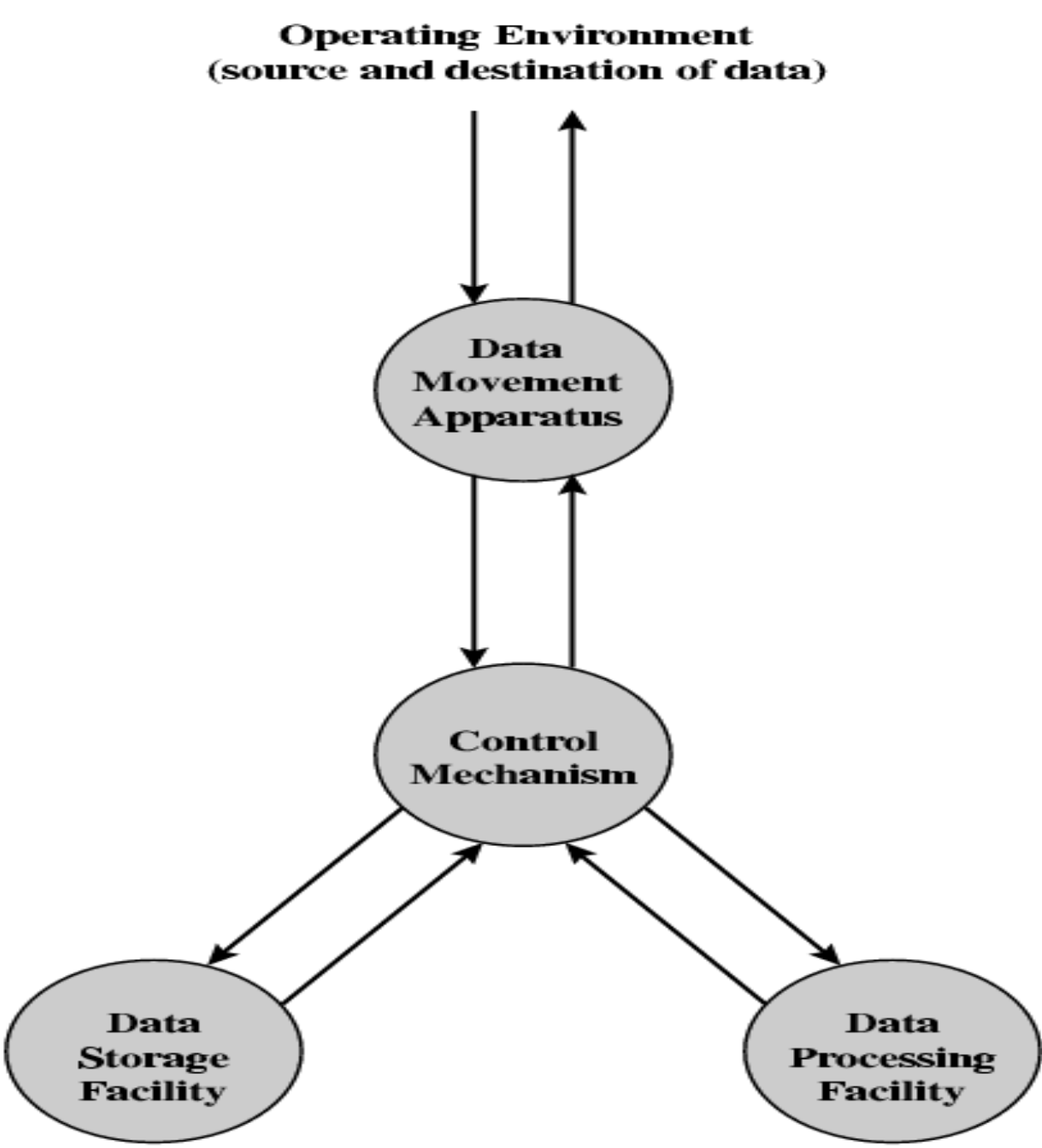
- **Function:** is the **operation of individual components** as part of the structure

Function

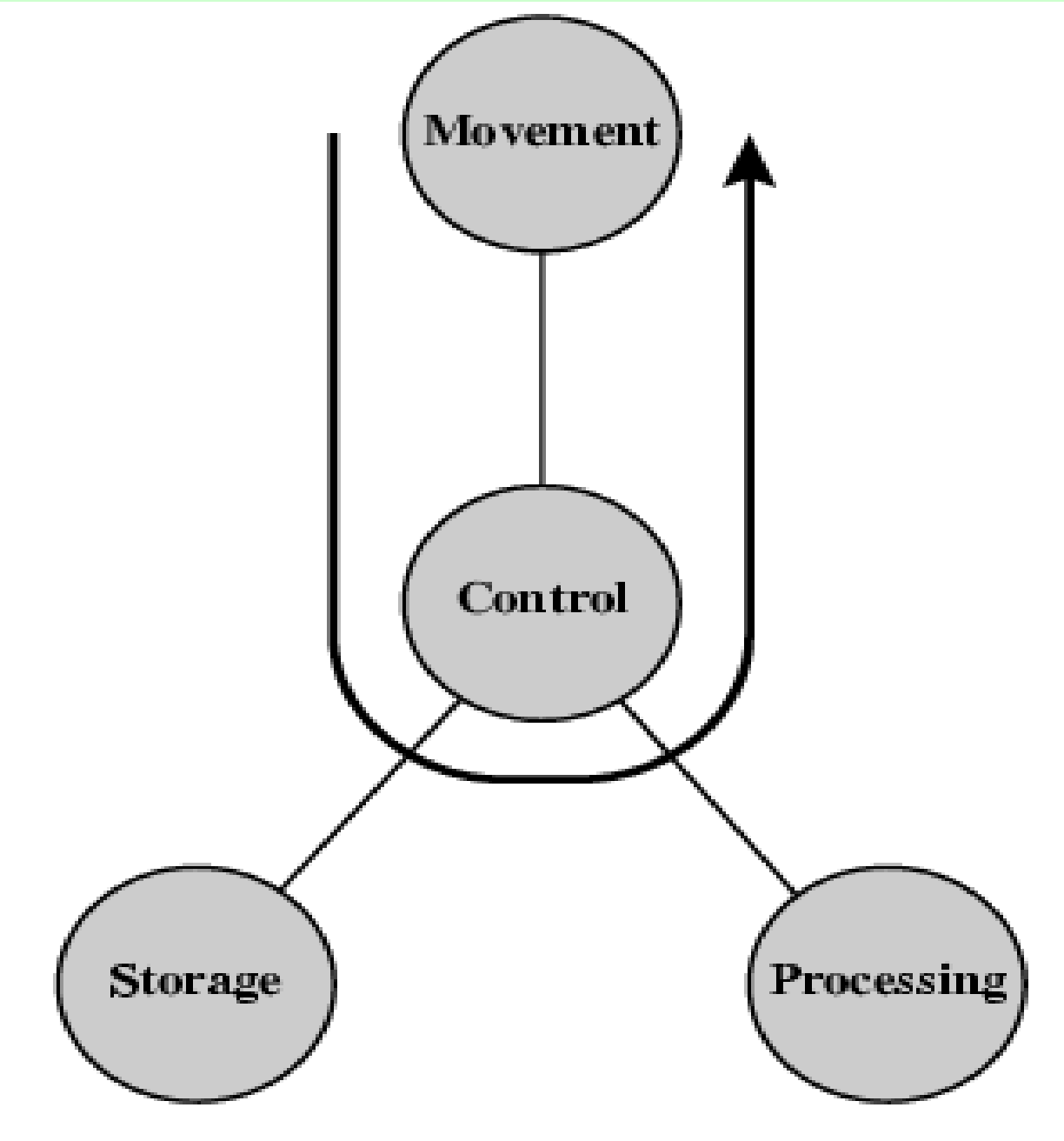
All computer functions are:

- Data processing
- Data storage
- Data movement
- Control

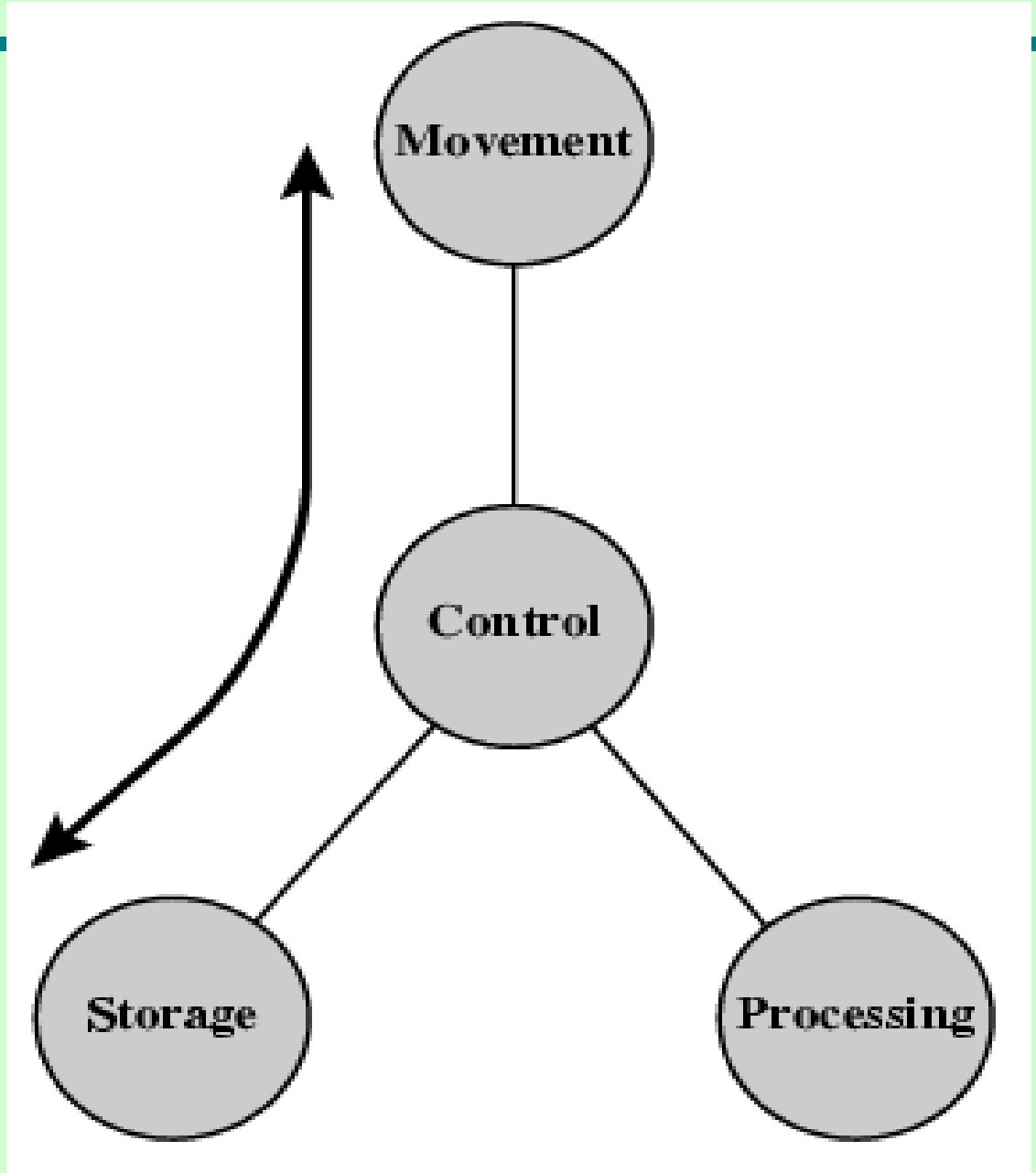
Functional View



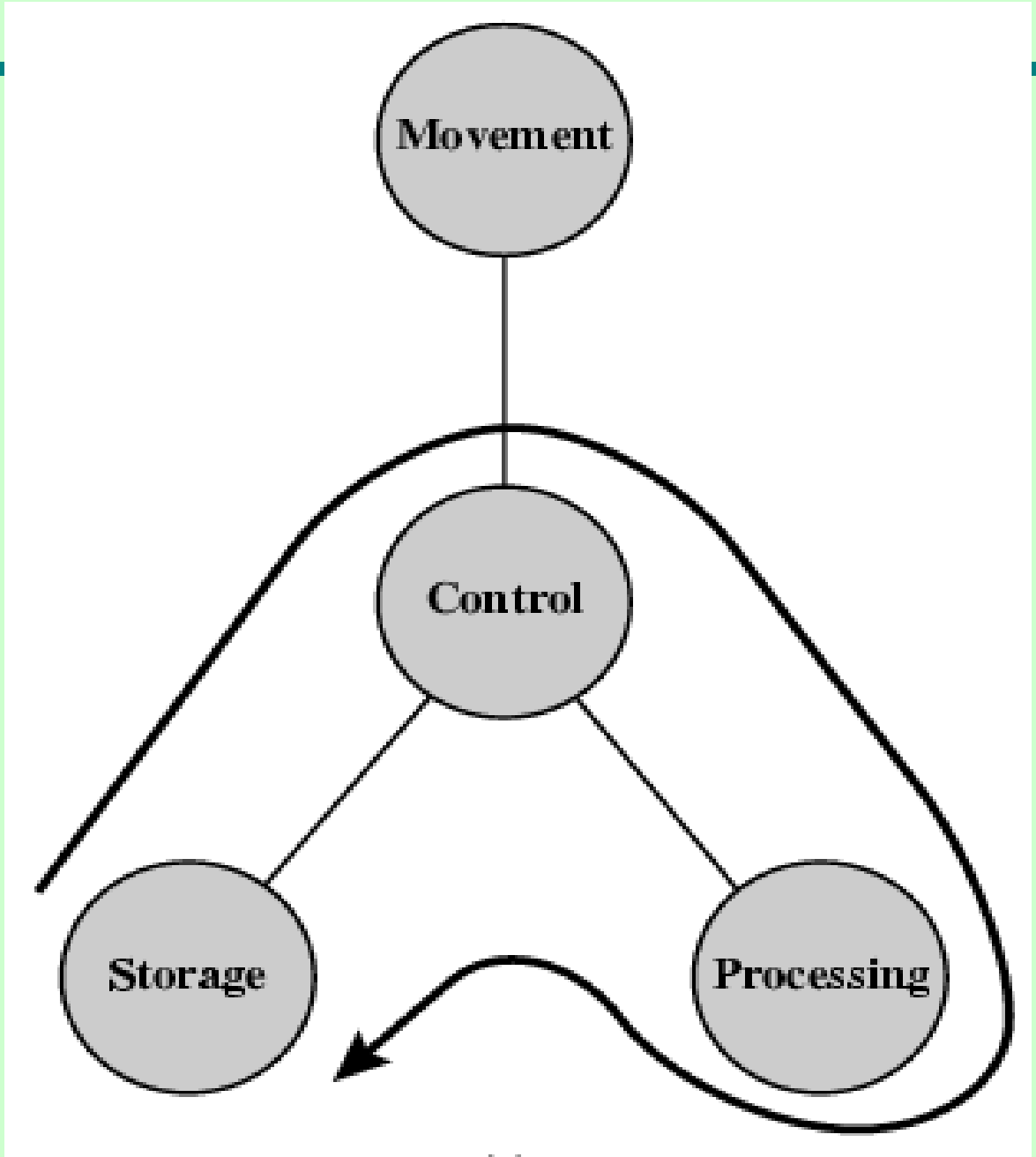
Operations (a) Data movement



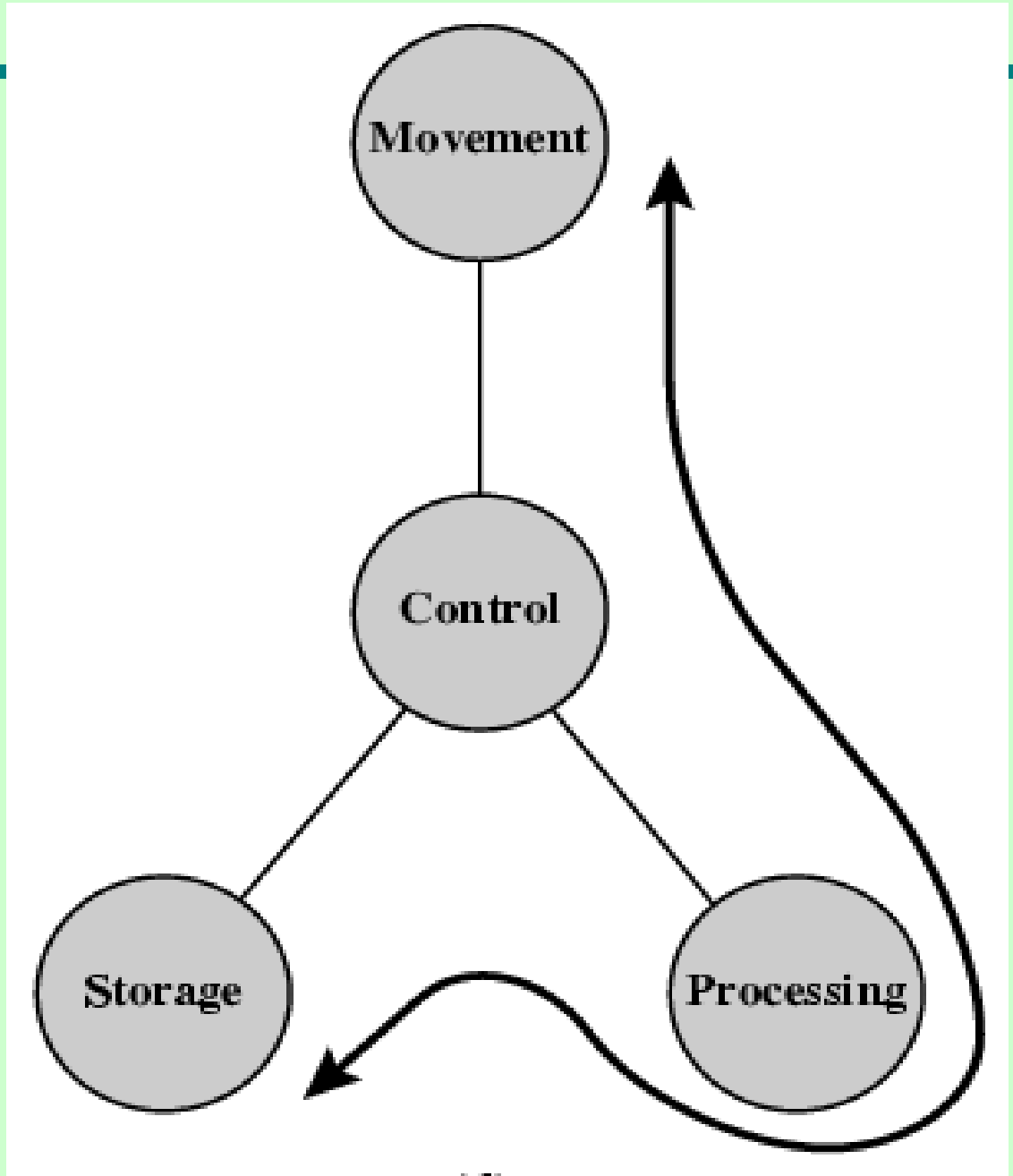
Operations (b) Storage



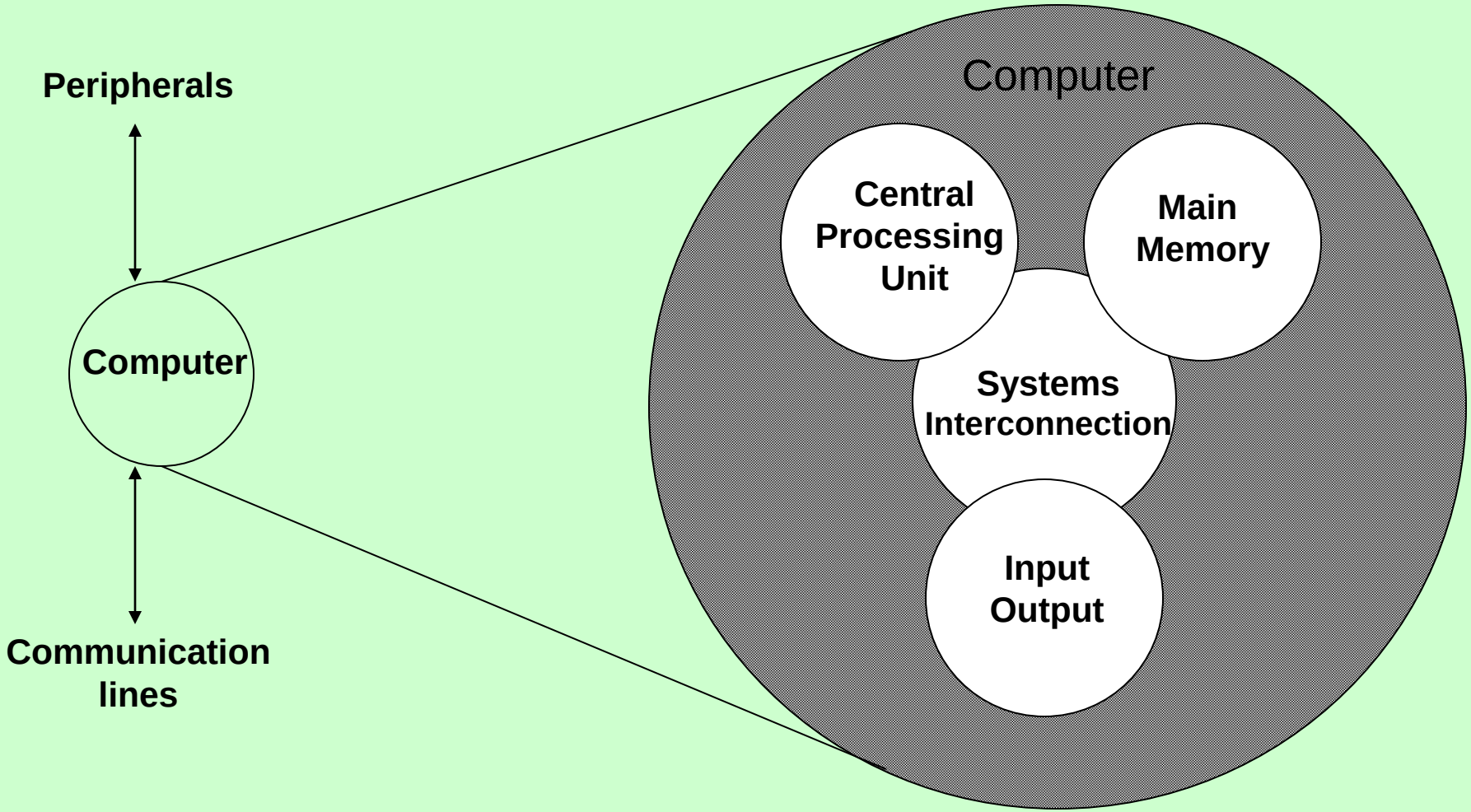
Operation (c) Processing from/to storage



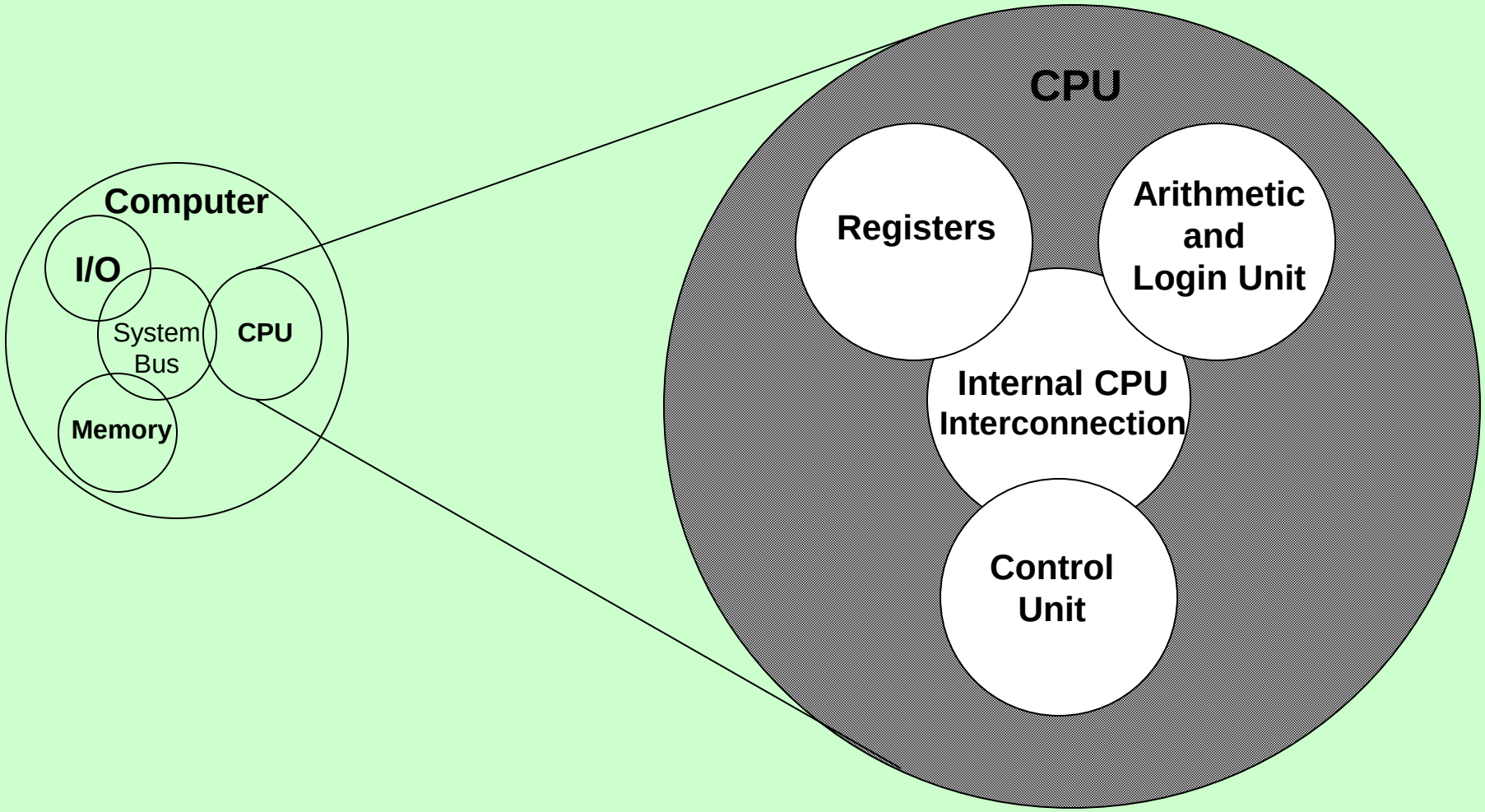
Operation (d) Processing from storage to I/O



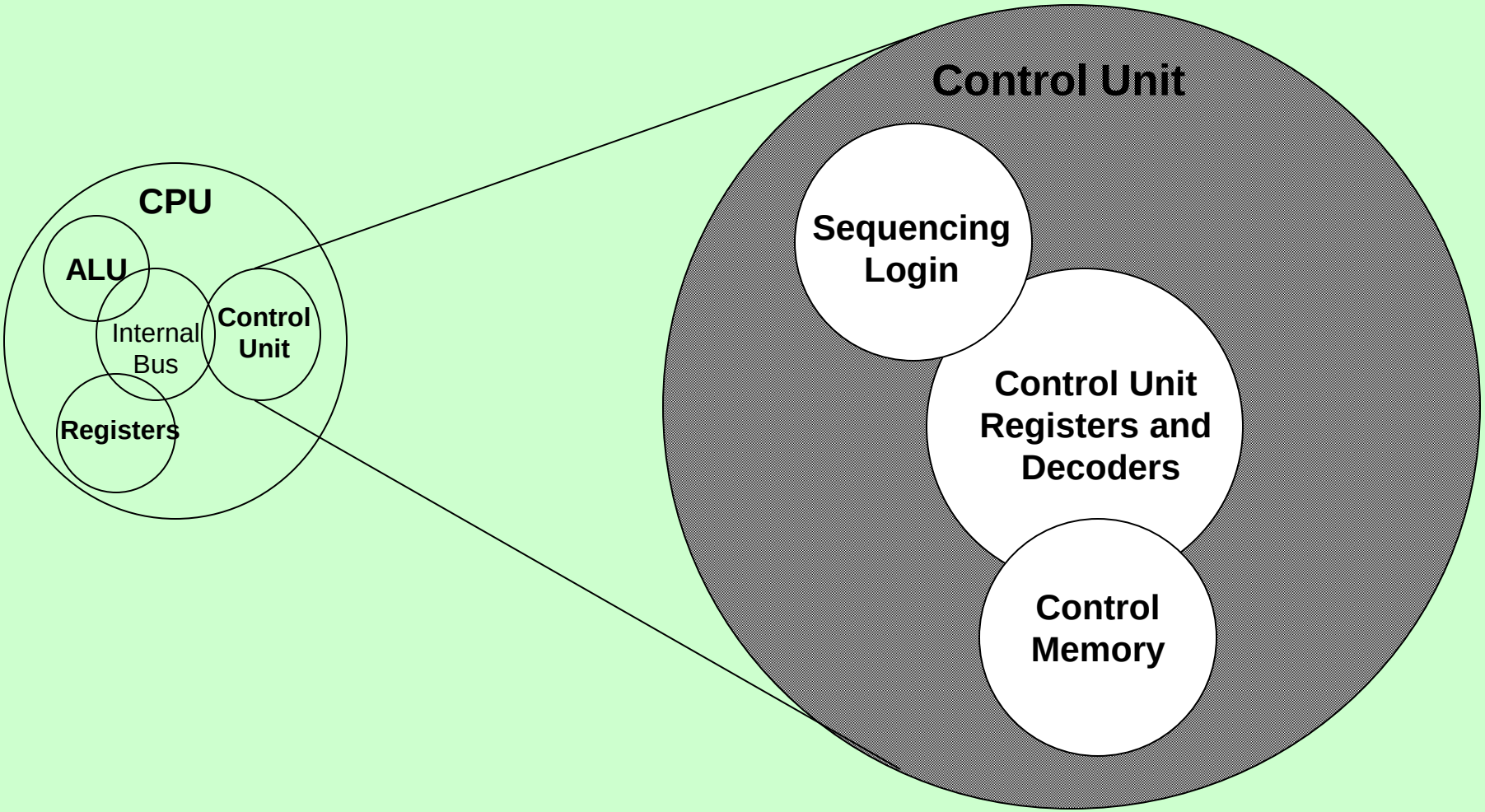
Structure - Top Level



Structure - The CPU



Structure - The Control Unit



Chapter 02
Generations of Computer

- **Vacuum tube - 1946-1957**
- **Transistor - 1958-1964**
- **Small scale integration - 1965** on
  - **Up to 100 devices** on a chip
- **Medium scale integration** - to **1971**
  - **100-3,000 devices** on a chip
- **Large scale integration - 1971-1977**
  - **3,000 - 100,000 devices** on a chip
- **Very large scale integration - 1978 -1991**
  - **100,000 - 100,000,000 devices** on a chip
- **Ultra large scale integration – 1991 -**
  - **Over 100,000,000 devices** on a chip

Moore's Law

- **Gordon Moore – co-founder of Intel**
- **Number of transistors on a chip will double every year**
- **Since 1970's** development has slowed a little
  - **Number of transistors doubles every 18 months**
- **Cost of a chip has remained almost unchanged**

Speeding it up

- **Pipelining**
- **On board cache**
- **On board L1 & L2 cache**
- **Branch prediction**
- **Data flow analysis**
- **Speculative execution**

Performance Balance

- **Processor speed increased**
- **Memory capacity increased**
- **Memory speed lags behind processor speed**

Pipelining	• Processor moves data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously
Branch prediction	• Processor looks ahead in the instruction code fetched from memory and predicts which branches, or groups of instructions, are likely to be processed next
Data flow analysis	• Processor analyzes which instructions are dependent on each other's results, or data, to create an optimized schedule of instructions
Speculative execution	• Using branch prediction and data flow analysis, some processors speculatively execute instructions ahead of their actual appearance in the program execution, holding the results in temporary locations, keeping execution engines as busy as possible

Solutions

- Increase number of bits retrieved at one time
  - Make DRAM "wider" rather than "deeper"
- Change DRAM interface
  - Cache
- Reduce frequency of memory access
  - More complex cache and cache on chip
- Increase interconnection bandwidth
  - High speed buses
  - Hierarchy of buses

Key is Balance

- Processor components
- Main memory
- I/O devices
- Interconnection structures

Improvements in Chip Organization and Architecture

- **Increase hardware speed of processor**
  - Fundamentally **due to shrinking logic gate size**
    - **More gates**, packed more tightly, **increasing clock rate**
    - **Propagation time** for signals **reduced**
- **Increase size and speed of caches**
  - **Dedicating part of processor chip**
    - **Cache access times drop significantly**
- **Change processor organization and architecture**
  - **Increase** effective **speed of execution**
  - **Parallelism**

Problems with Clock Speed and Login Density

- **Power**
  - **Power density increases with density of logic and clock speed**
  - **Dissipating heat**
- **RC delay**
  - **Delay increases as RC product increases**
  - **Wire interconnects thinner, increasing resistance**
  - **Wires closer together, increasing capacitance**
- **Memory latency**
  - **Memory speeds lag processor speeds**
- **Solution:**
  - More emphasis on **organizational and architectural approaches**

More Complex Execution Logic

- **Enable parallel execution of instructions**
- **Pipeline works like assembly line**
  - —Different stages of execution of different instructions at same time along pipeline
- **Superscalar allows multiple pipelines within single processor**
  - —**Instructions that do not depend on one another can be executed in parallel**

Diminishing Returns

- **Internal organization of processors complex**
  - **Can get a great deal of parallelism**
  - **Further significant increases likely to be relatively modest**
- **Benefits from cache are reaching limit**
- **Increasing clock rate runs into power dissipation problem**
  - Some fundamental physical limits are being reached

New Approach – Multiple Cores

- **Multiple processors on single chip**
  - —Large shared cache
- **Within a processor, increase in performance proportional to square root of increase in complexity**
- **If software can use multiple processors, doubling number of processors almost doubles performance**
- **So, use two simpler processors on the chip rather than one more complex processor**
- With two processors, larger caches are justified (Power consumption of memory logic less than processing logic)

Performance Assessment
Clock Speed

- **Key parameters**
  - **Performance, cost, size, security, reliability, power consumption**
- **System clock speed (**In **Hz** or multiples of)
  - **Clock rate, clock cycle, clock tick, cycle time**
- **Instruction execution in discrete steps**
  - **Fetch, decode, load and store, arithmetic or logical**
  - **Usually require multiple clock cycles per instruction**
- **Pipelining gives simultaneous execution of instructions**
- So, clock speed is not the whole story

Instruction Execution Rate

- **Millions of instructions per second (MIPS)**
- **Millions of floating point instructions per second (MFLOPS)**

Performance Balance

While processor power has raced ahead at breakneck speed, other critical components of the computer have not kept up. The result is a need to look for performance balance: **an adjusting of the organization and architecture to compensate for the mismatch among the capabilities of the various components.**

The interface between processor and main memory is the most crucial pathway in the entire computer because it is responsible for carrying a constant flow of program instructions and data between memory chips and the processor. **If memory or the pathway fails to keep pace with the processor's insistent demands, the processor stalls in a wait state, and valuable processing time is lost.**

There are a number of ways that a system architect can **attack this problem**, all of which are reflected in contemporary computer designs.

Consider the following examples:

- **Increase the number of bits that are retrieved at one time by making DRAMs "wider" rather than "deeper"** and by using wide bus data paths.
- **Change the DRAM interface to make it more efficient by including a cache** or other buffering scheme **on the DRAM chip**.
- **Reduce the frequency of memory access by** incorporating increasingly complex and efficient **cache structures between the processor and main memory**. This includes the incorporation of **one or more caches on the processor chip as well as on an off-chip cache close to the processor chip.**
- **Increase the interconnect bandwidth between processors and memory by using higher-speed buses** and by using **a hierarchy of buses** to buffer and structure data flow.

There are three approaches to achieving increased processor speed:

- **Increase the hardware speed of the processor**. This increase is fundamentally due to **shrinking the size of the logic gates on the processor chip**, so that **more gates can be packed together** more tightly and to **increasing the clock rate**. **With gates closer together, the propagation time for signals is significantly reduced, enabling a speeding up of the processor**.
- **Increase the size and speed of caches that are interposed between the processor and main memory**.
- **Make changes to the processor organization and architecture that increase the effective speed of instruction execution**. Typically, this involves **using parallelism** in one form or another.

However, **simply relying on increasing clock rate for increased performance runs into the power dissipation problem** already referred to.

The faster the clock rate, the greater the amount of power to be dissipated, and some fundamental physical limits are being reached.

With all of these difficulties in mind, **designers have** turned to a fundamentally **new approach to improving performance**: **placing multiple processors on the same chip, with a large shared cache**. The use of multiple processors on the same chip, also referred to as multiple cores, or **multicore, provides the potential to increase performance without increasing the clock rate.**

Embedded system: A combination of computer hardware and software, designed to perform a dedicated function. - an **antilock braking system in a car.**

ARM Evolution: ARM is **a family of RISC-based microprocessors and microcontrollers designed by ARM Inc.**, Cambridge, England. **ARM chips are high-speed processors that are known for their small die size and low power requirements**. ARM is probably the **most widely used embedded processor architecture**

ARM processors are designed to meet the needs of three system categories:

- **Embedded real-time systems:** Systems for storage, automotive body and power-train, industrial, and networking applications
- **Application platforms:** Devices running open operating systems including Linux, Palm OS, Symbian OS, and Windows CE in wireless, consumer entertainment and digital imaging applications
- **Secure applications:** Smart cards, SIM cards, and payment terminals

Clock Speed and Instructions per Second

THE SYSTEM CLOCK **Operations performed by a processor**, **are governed by a system clock**. **The speed of a processor is dictated by the pulse frequency produced by the clock**, measured in **Hertz (Hz).**

The execution of an instruction involves a number of discrete steps, such as fetching the instruction from memory, decoding the various portions of the instruction, loading and storing data, and performing arithmetic and logical operations.

Thus, **most instructions on most processors require multiple clock cycles to complete**. **Some instructions may take only a few cycles, while others require dozens**. In addition, **when pipelining is used, multiple instructions are being executed simultaneously**. Thus, a straight comparison of **clock speeds on different processors does not tell** the whole story **about performance**.

INSTRUCTION EXECUTION RATE

A processor is driven by **a clock** with a **constant frequency f** or, equivalently, a **constant cycle time *T* , where *T*=1/f** . Define the **instruction count, Ic, for a program** as the number of machine instructions executed for that program

An important parameter is the **average cycles per instruction CPI for a program**.

If all instructions required the same number of clock cycles, then CPI would be a constant value for a processor.

However, on any give processor, the number of clock cycles required varies for different types of instructions, such as load, store, branch, and so on.

Let **CPIi be the number of cycles required for instruction type i**. and **Ii be the number of executed instructions of type i** for a given program. Then we can calculate an overall CPI as follows:

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{I_c}$$

The processor time T needed to execute a given program can be expressed as

$$T = I_c \times CPI \times \tau$$

A **common measure of performance for a processor** is the **rate at which instructions are executed**, expressed as **millions of instructions per second (MIPS),** referred to as the **MIPS rate**.

We can express the MIPS rate in terms of the clock rate and CPI as follows:

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

For example, consider the execution of a program which results in the execution of **2 million instructions** on a **400-MHz processor**. The program consists of **four major types of instructions**.

The **instruction mix and the CPI for each instruction type are given below** based on the result of a program trace experiment:

Instruction Type	CPI	Instruction Mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

The average CPI when the program is executed on a uniprocessor with the above trace results is $CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24$. The corresponding MIPS rate is $(400 \times 10^6)/(2.24 \times 10^6) \approx 178$.

The average CPI when the program is executed on a uniprocessor with the above trace results is

$$CPI = 0.6 + (2 \times 0.18) + (4 \times 0.12) + (8 \times 0.1) = 2.24.$$

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

$$\text{MIPS rate is } (400 \times 10^6)/(2.24 \times 10^6) \approx 178.$$

Another common performance measure deals only with floating-point instructions. These are common in many scientific and game applications. Floating point performance is expressed as millions of floating-point operations per second (MFLOPS), defined as follows:

$$\text{MFLOPS rate} = \frac{\textit{Number of executed floating-point operations in a program}}{\textit{Execution time} \times 10^6}$$

2.10. A benchmark program is run on a 40 MHz processor. The executed program consists of 100,000 instruction executions, with the following instruction mix and clock cycle count:

Instruction Type	Instruction Count	Cycles per Instruction
Integer arithmetic	45000	1
Data transfer	32000	2
Floating point	15000	2
Control transfer	8000	2

Determine the effective CPI, MIPS rate, and execution time for this program.

Answer to 2.10

$$CPI = \frac{\sum_{i=1}^{n} (CPI_i \times I_i)}{I_c}$$

$$\text{MIPS rate} = \frac{I_c}{T \times 10^6} = \frac{f}{CPI \times 10^6}$$

CPI =1.55

MIPS=25.8

Execution time=3.87 nsec

2.11. Consider two different machines, with two different instruction sets, both of which have a clock rate of 200 MHz. The following measurements are recorded on the two machines running a given set of benchmark programs:

Instruction Type	Instruction Count (millions)	Cycles per Instruction
Machine A		
Arithmetic and logic	8	1
Load and store	4	3
Branch	2	4
Others	4	3
Machine A		
Arithmetic and logic	10	1
Load and store	8	2
Branch	2	4
Others	4	3

a. Determine the effective CPI, MIPS rate, and execution time for each machine.

b. Comment on the results.

Answer to 2.11

CPI_A = ((8*1 + 4*3 + 2*4 + 4*3) * 10^6) / ((8+4+2+4) * 10^6) = **2.22**

MIPS_A = (200*10^6) / (2.22*10^6) = **90**

CPU_A = (18*10^6 *2.2) / (200*10^6) = **0.2 Second**

CPI_B = ((10*1 + 8*2+ 2*4 + 4*3) * 10^6) / ((10+8+2+4) * 10^6) = **1.92**

MIPS_B = (200*10^6) / (1.92*10^6) = **104**

CPU_B = (24*10^6 *1.92) / (200*10^6) = **0.23 Second**

Although machine B has a higher MIPS than machine A , It requires a longer CPU time to execute the same set of benchmark programs

2.11 a

$$CPI_A = \frac{\sum CPI_i \times I_i}{I_c} = \frac{(8\times 1 + 4\times 3 + 2\times 4 + 4\times 3)\times 10^6}{(8+4+2+4)\times 10^6} \approx 2.22$$

$$MIPS_A = \frac{f}{CPI_A \times 10^6} = \frac{200\times 10^6}{2.22\times 10^6} = 90$$

$$CPU_A = \frac{I_c \times CPI_A}{f} = \frac{18\times 10^6 \times 2.2}{200\times 10^6} = 0.2 \text{ s}$$

$$CPI_B = \frac{\sum CPI_i \times I_i}{I_c} = \frac{(10\times 1 + 8\times 2 + 2\times 4 + 4\times 3)\times 10^6}{(10+8+2+4)\times 10^6} \approx 1.92$$

$$MIPS_B = \frac{f}{CPI_B \times 10^6} = \frac{200\times 10^6}{1.92\times 10^6} = 104$$

$$CPU_B = \frac{I_c \times CPI_B}{f} = \frac{24\times 10^6 \times 1.92}{200\times 10^6} = 0.23 \text{ s}$$

2.11 b. Although machine B has a higher MIPS than machine A, it requires a longer CPU time to execute the same set of benchmark programs

2.12. Early examples of CISC and RISC design are the VAX 11/780 and the IBM RS/6000, respectively. Using a typical benchmark program, the following machine characteristics result:

Processor	Clock Frequency	Performance	CPU Time
VAX 11/780	5 MHz	1 MIPS	12 *x* seconds
IBM RS/6000	25 MHz	18 MIPS	*x* seconds

The final column shows that the VAX required 12 times longer than the IBM measured in CPU time.

a. What is the relative size of the instruction count of the machine code for this benchmark program running on the two machines?

b. What are the *CPI* values for the two machines?

Answer to 2.12

a:

[(MIPS rate) / 10^6] = Ic / T

Ic = T * [(MIPS rate) / 10^6]

The ratio of **Instruction Count** of **RS / 6000** to the **VAX** is:

(18*1)/(1*12) = **1.5**

b:

For the VAX , **CPI** = (5 MHz) / (1 MIPS) = **5**

For the RS/6000, **CPI** = (25 MHz) / (18 MIPS) = **1.39**

2.13. Four benchmark programs are executed on three computers with the following results:

	Computer A	Computer B	Computer C
Program 1	1	10	20
Program 2	1000	100	20
Program 3	500	1000	50
Program 4	100	800	100

The table shows the execution time in seconds, with 100,000,000 instructions executed in each of the four programs. Calculate the MIPS values for each computer for each program. Then calculate the arithmetic and harmonic means assuming equal weights for the four programs, and rank the computers based on arithmetic mean and harmonic mean.

Answer to 2.13

MIPS = Ic / (T * 10^6) = 100000000/ (T * 10^6) = **100/T**

	Computer A	Computer B	Computer C
Program 1	**100**	**10**	**5**
Program 2	**0.1**	**1**	**5**
Program 3	**0.2**	**0.1**	**2**
Program 4	**1**	**0.125**	**1**

	Arithmetic mean	Rank	Harmonic mean	Rank
Computer A	25.325	1	0.25	2
Computer B	2.8	3	0.21	3
Computer C	3.26	2	2.1	1

2.14. The following table, based on data reported in the literature [HEAT84], shows the execution times, in seconds, for five different benchmark programs on three machines.

Benchmark	Processor		
	R	M	Z
E	417	244	134
F	83	70	70
H	66	153	135
I	39,449	35,527	66,000
K	772	368	369

a. Compute the speed metric for each processor for each benchmark, normalized to machine R. That is, the ratio values for R are all 1.0. Other ratios are calculated using Equation (2.5) with R treated as the reference system. Then compute the arithmetic mean value for each system using Equation (2.3). This is the approach taken in [HEAT84].

b. Repeat part (a) using M as the reference machine. This calculation was not tried in [HEAT84].

c. Which machine is the slowest based on each of the preceding two calculations?

d. Repeat the calculations of parts (a) and (b) using the geometric mean, defined in Equation (2.6). Which machine is the slowest based on the two calculations?

2.14 a:
Normalized to R

Benchmark	Processor		
	R	M	Z
E	1.00	1.71	3.11
F	1.00	1.19	1.19
H	1.00	0.43	0.49
I	1.00	1.11	0.60
K	1.00	2.10	2.09
Arithmetic mean	1.00	1.31	1.50

2.14 b:
Normalized to M

Benchmark	Processor		
	R	M	Z
E	0.59	1.00	1.82
F	0.84	1.00	1.00
H	2.32	1.00	1.13
I	0.90	1.00	0.54
K	0.48	1.00	1.00
Arithmetic mean	1.01	1.00	1.10

2.14 c:Recall that the larger the ratio, the higher the speed. Based on (a) R is the slowest machine, by a significant amount. Based on (b), M is the slowest machine, by a modest amount.

2.14 d:
Normalized to R

Benchmark	Processor		
	R	M	Z
E	1.00	1.71	3.11
F	1.00	1.19	1.19
H	1.00	0.43	0.49
I	1.00	1.11	0.60
K	1.00	2.10	2.09
Geometric mean	1.00	1.15	1.18

Normalized to M:

Benchmark	Processor		
	R	M	Z
E	0.59	1.00	1.82
F	0.84	1.00	1.00
H	2.32	1.00	1.13
I	0.90	1.00	0.54
K	0.48	1.00	1.00
Geometric mean	0.87	1.00	1.02

$$r_G = \left(\prod_{i=1}^{n} r_i \right)^{1/n}$$

Using the geometric mean, R is the slowest no matter which machine is used for normalization

Benchmarks

- **Programs designed to test performance**
- **Written in high level language**
  - Portable
- Represents style of task
  - Systems, numerical, commercial
- Easily measured
- Widely distributed
- **E.g. System Performance Evaluation Corporation (SPEC)**
  - **CPU2006 for computation bound**
    - **17 floating point programs in C, C++, Fortran**
    - **12 integer programs in C, C++**
    - **3 million lines of code**
  - **Speed and rate metrics**
    - **Single task and throughput**

SPEC Speed Metric

- **Single task**
- Base **runtime** defined for each benchmark **using reference machine**
- Results are reported as **ratio of reference time to system run time**
  - **$Tref_i$ execution time for benchmark i on reference machine**
  - **$Tsut_i$ execution time of benchmark i on test system**

$$r_i = \frac{Tref_i}{Tsut_i}$$

- **Overall performance calculated by averaging ratios for all 12 integer benchmarks**
  - **Use geometric mean**
    - Appropriate for normalized numbers such as ratios

$$r_G = \left(\prod_{i=1}^{n} r_i \right)^{1/n}$$

SPEC Rate Metric

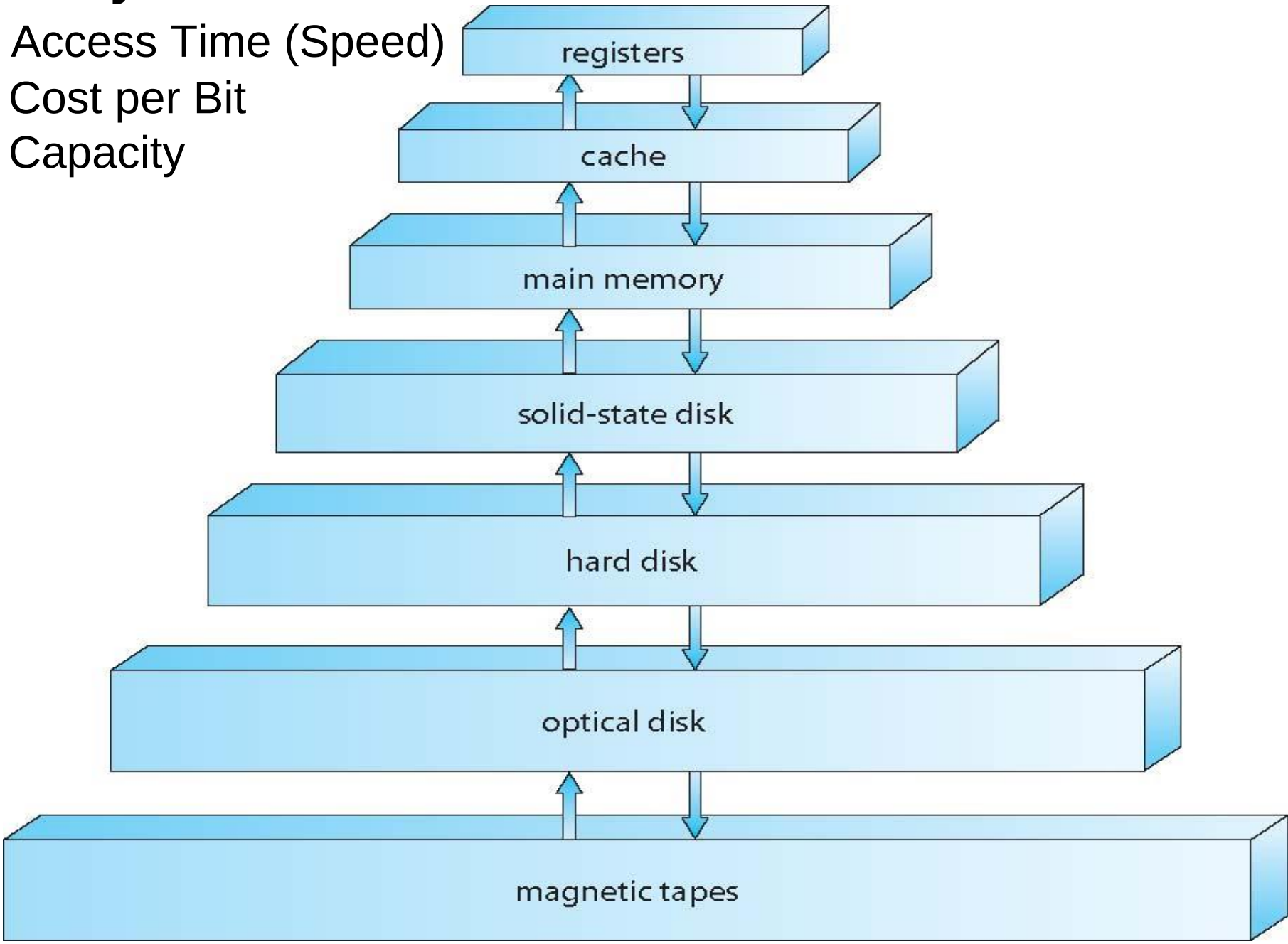
- **Measures throughput or rate of a machine carrying out a number of tasks**
- **Multiple copies of benchmarks run simultaneously**
  - Typically, **same as number of processors**
- Ratio is calculated as follows:
  - $Tref_i$ **reference execution time for benchmark i**
  - N **number of copies run simultaneously**
  - Tsuti **elapsed time from start of execution of program on all N processors until completion** of all copies of program
  - Again, a **geometric mean** is calculated

$$r_i = \frac{N \times Tref_i}{Tsut_i}$$

Memory Characteristics:

- Access Time (Speed)
- Cost per Bit
- Capacity

Storage-Device Hierarchy



Chapter 03
What is a program?

- **A sequence of steps**
- **For each step, an arithmetic or logical operation is done**
- **For each operation, a different set of control signals is needed**

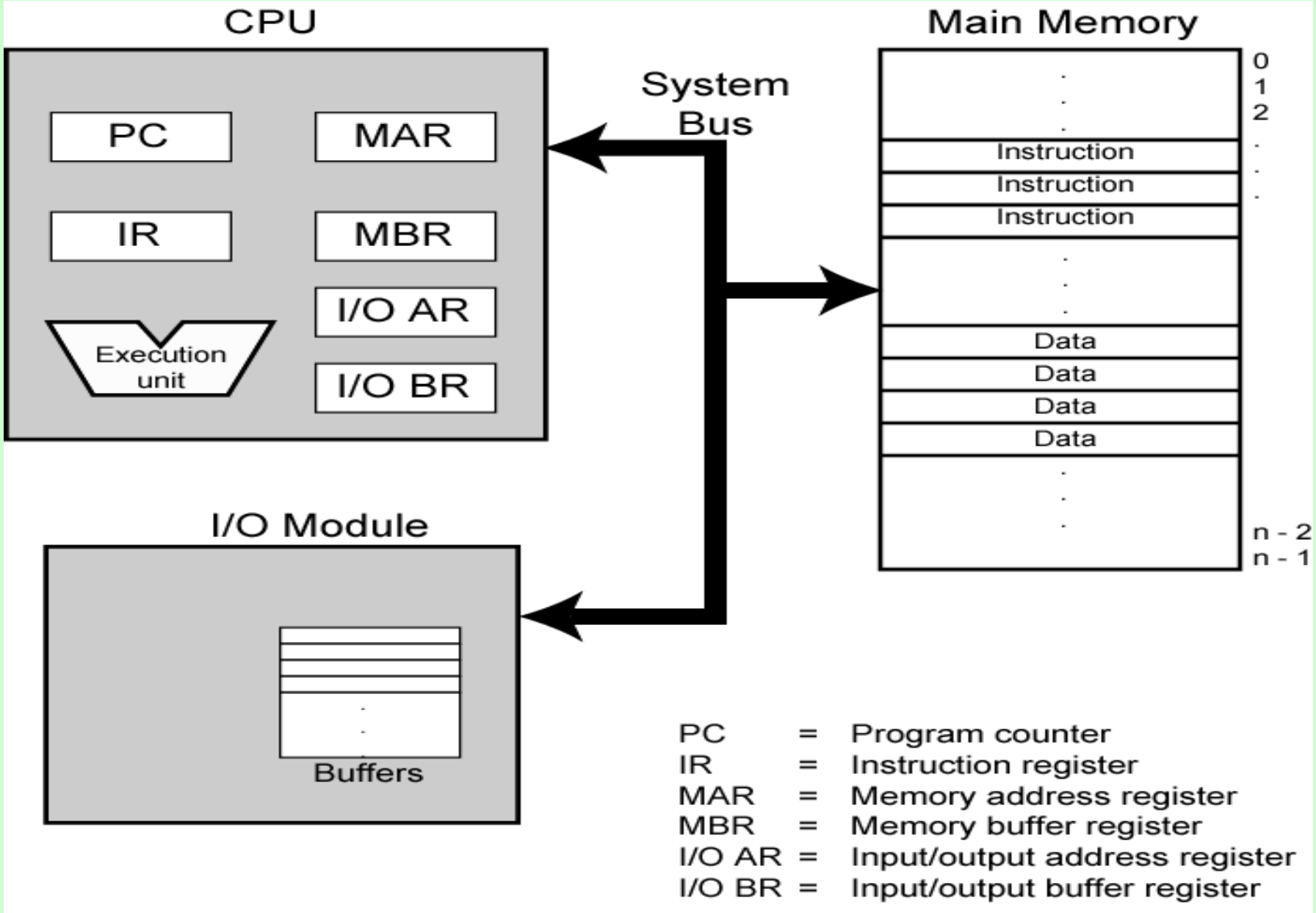
Function of Control Unit

- **For each operation a unique code is provided**
  - —e.g. ADD, MOVE
- **A hardware segment accepts the code and issues the control signals**

Components

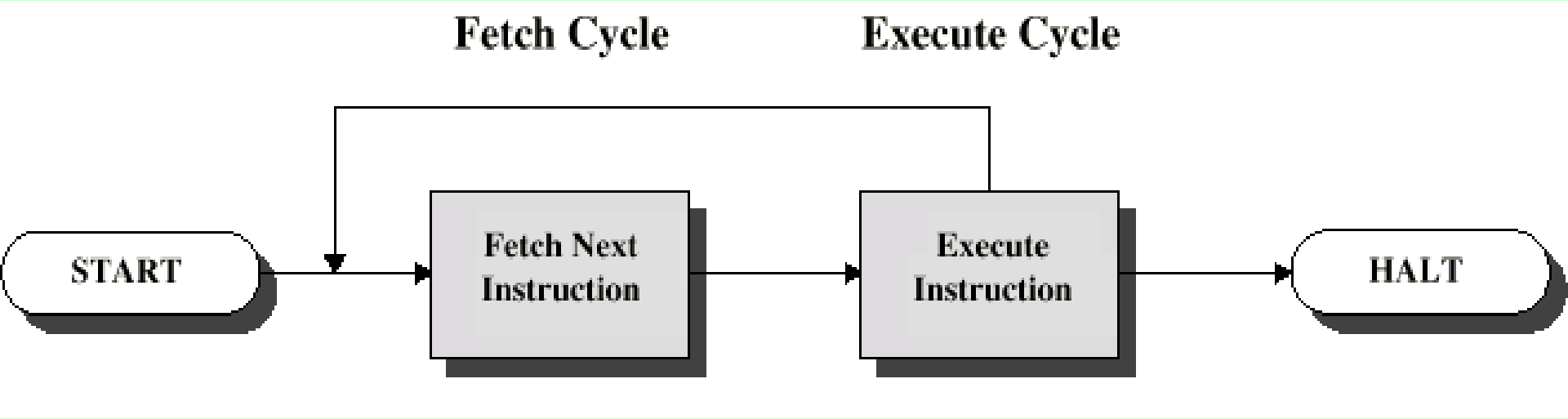
- The **Control Unit and the Arithmetic and Logic Unit constitute the Central Processing Unit**
- Data and instructions need to get into the system and results out
  - **Input/output**
- Temporary storage of code and results is needed
  - **Main memory**

Computer Components: Top Level View



Instruction Cycle

- **Two steps:**
  - **Fetch**
  - **Execute**



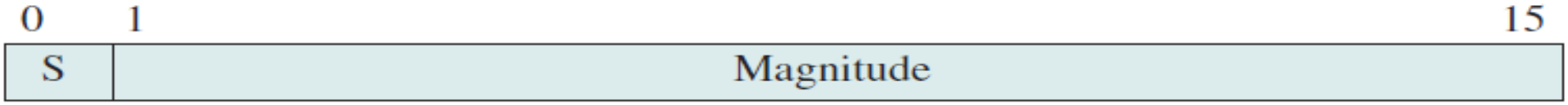
Fetch Cycle

- **Program Counter (PC) holds address of next instruction to fetch**
- **Processor fetches instruction from memory location pointed to by PC**
- **Increment PC**
  - —Unless told otherwise
- **Instruction loaded into Instruction Register (IR)**
- **Processor interprets instruction and performs required actions**

Execute Cycle

- Processor-memory
  - **data transfer between CPU and main memory**
- Processor - I/O
  - **Data transfer between CPU and I/O module**
- Data processing
  - Some **arithmetic or logical operation on data**
- Control
  - Alteration of **sequence of operations**
  - e.g. **jump**
- **Combination of above**

(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

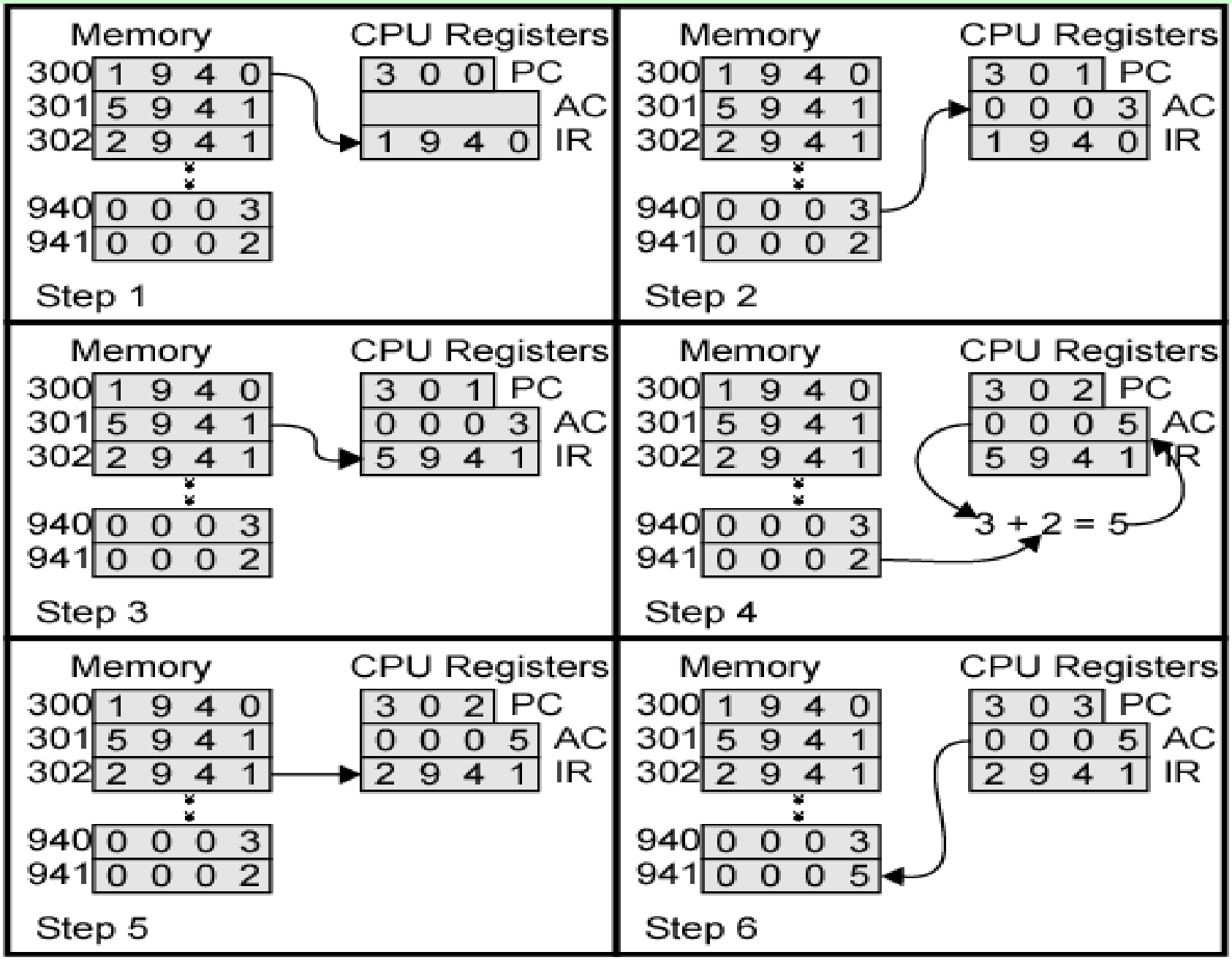
(c) Internal CPU registers

0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

Example of Program Execution



Interrupts

- **Mechanism by which other modules (e.g. I/O) may interrupt normal sequence of processing**
- **Program**
  - e.g. **overflow, division by zero**
- **Timer**
  - Generated by internal processor timer
  - Used in pre-emptive **multi-tasking**
- **I/O**
  - from **I/O controller**
- **Hardware failure**
  - e.g. **memory parity error**

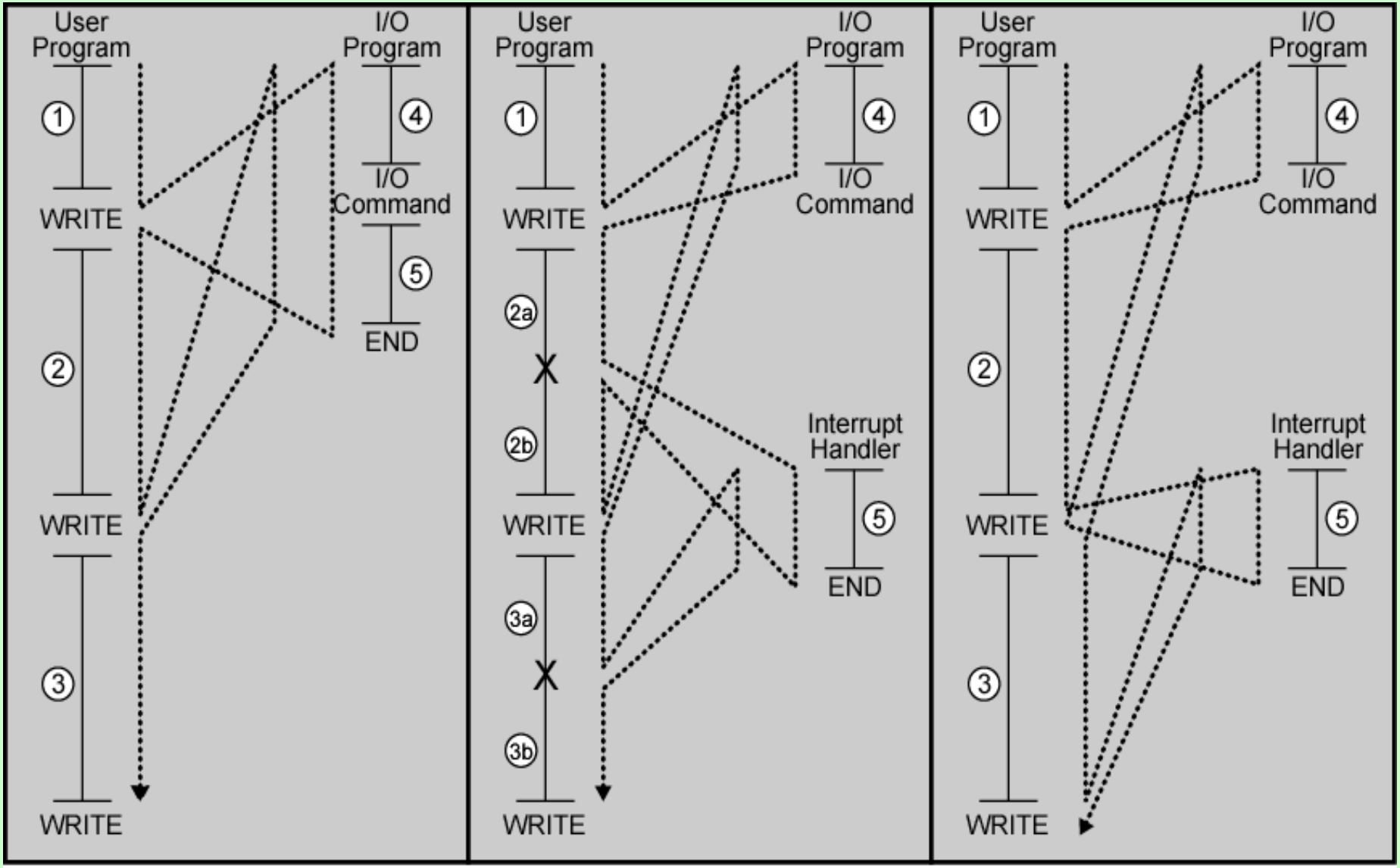
Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal sequencing of the processor.

Table 1.1 Classes of Interrupts

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
Hardware failure	Generated by a failure, such as power failure or memory parity error.

Interrupts are provided primarily as a way to improve processor utilization. For example, most I/O devices are much slower than the processor. Suppose that the processor is transferring data to a printer using the instruction cycle scheme. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many thousands or even millions of instruction cycles (Figure 1.5a)

Program Flow Control



(a) No interrupts

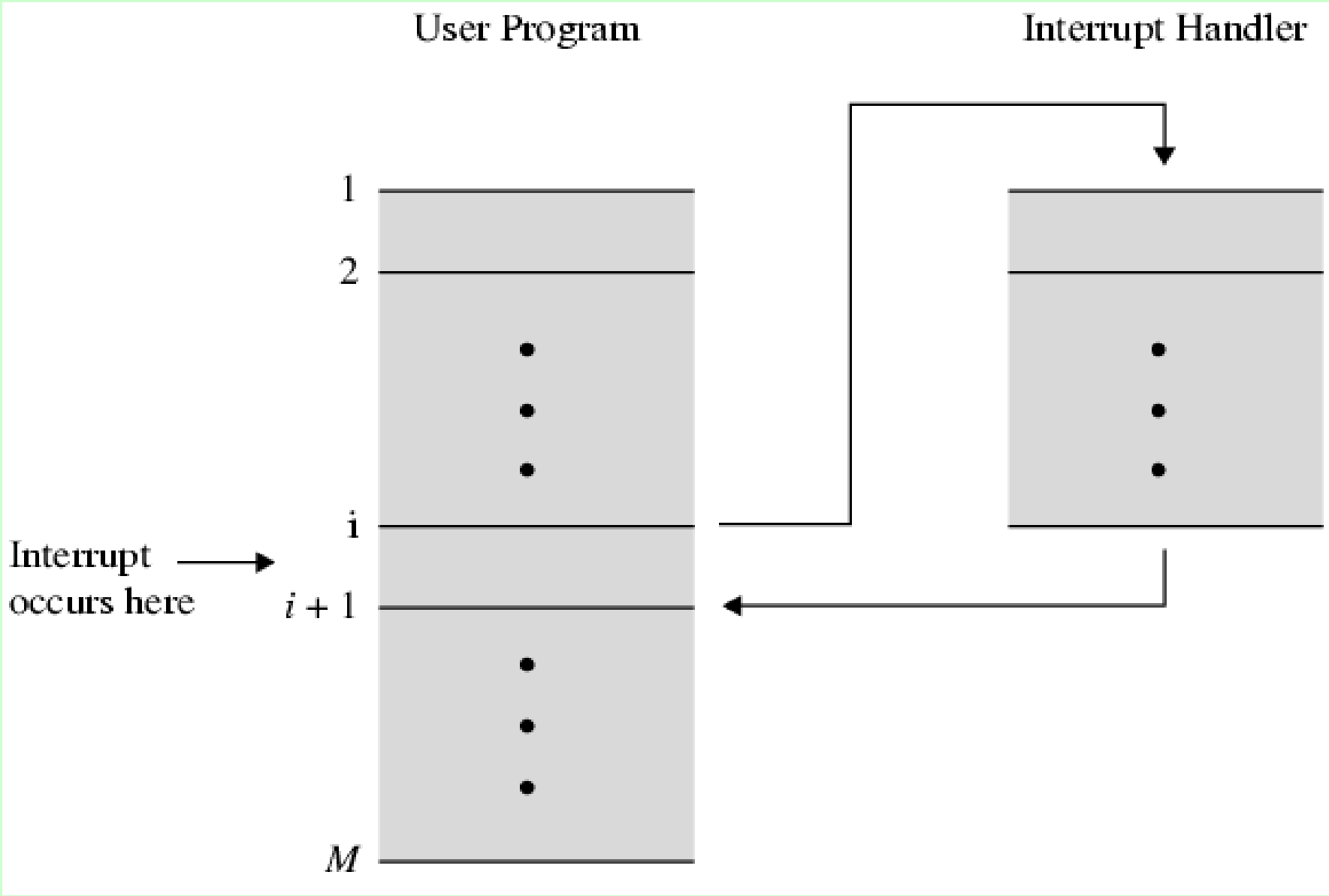
(b) Interrupts; short I/O wait

(c) Interrupts; long I/O wait

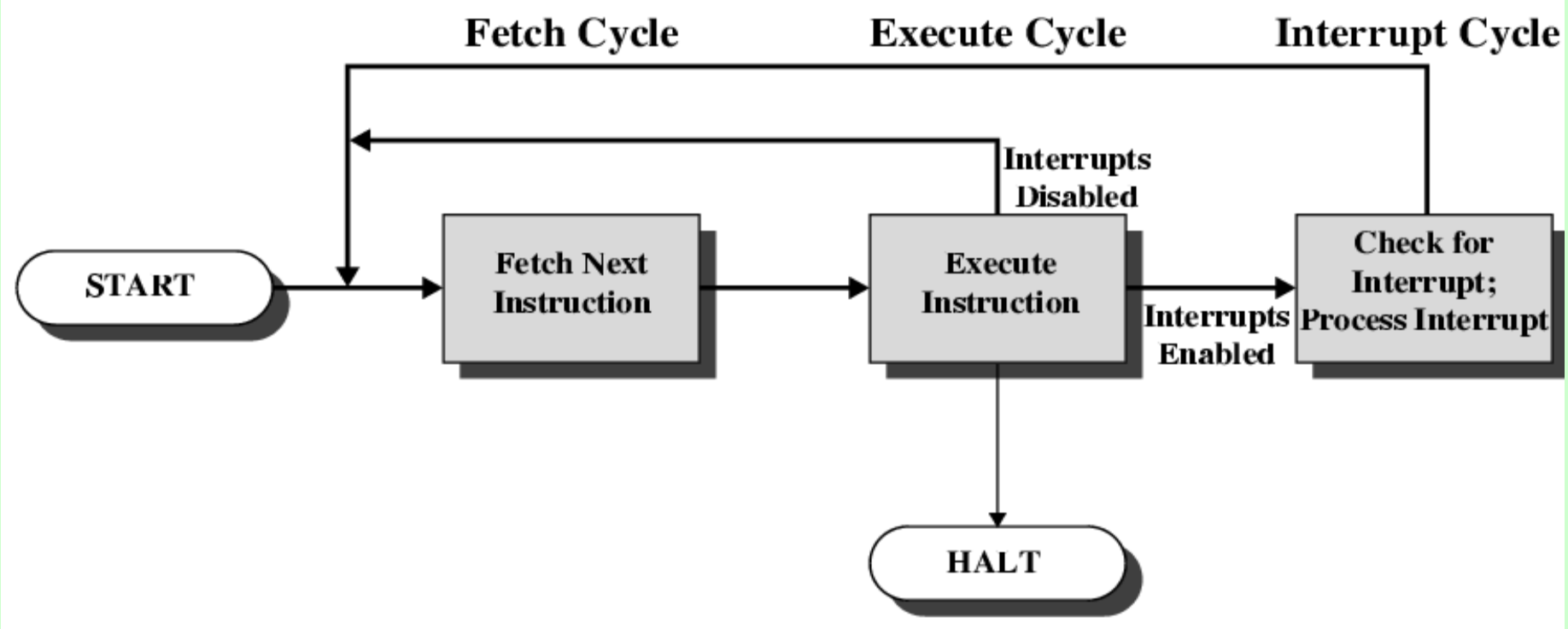
Interrupt Cycle

- **Added to instruction cycle**
- **Processor checks for interrupt**
  - Indicated by an **interrupt signal**
- **If no interrupt, fetch next instruction**
- If **interrupt pending**:
  - **Suspend execution** of current program
  - **Save context**
  - **Set PC to start address of interrupt handler** routine
  - **Process interrupt**
  - **Restore context and continue interrupted program**

Transfer of Control via Interrupts



Instruction Cycle with Interrupts



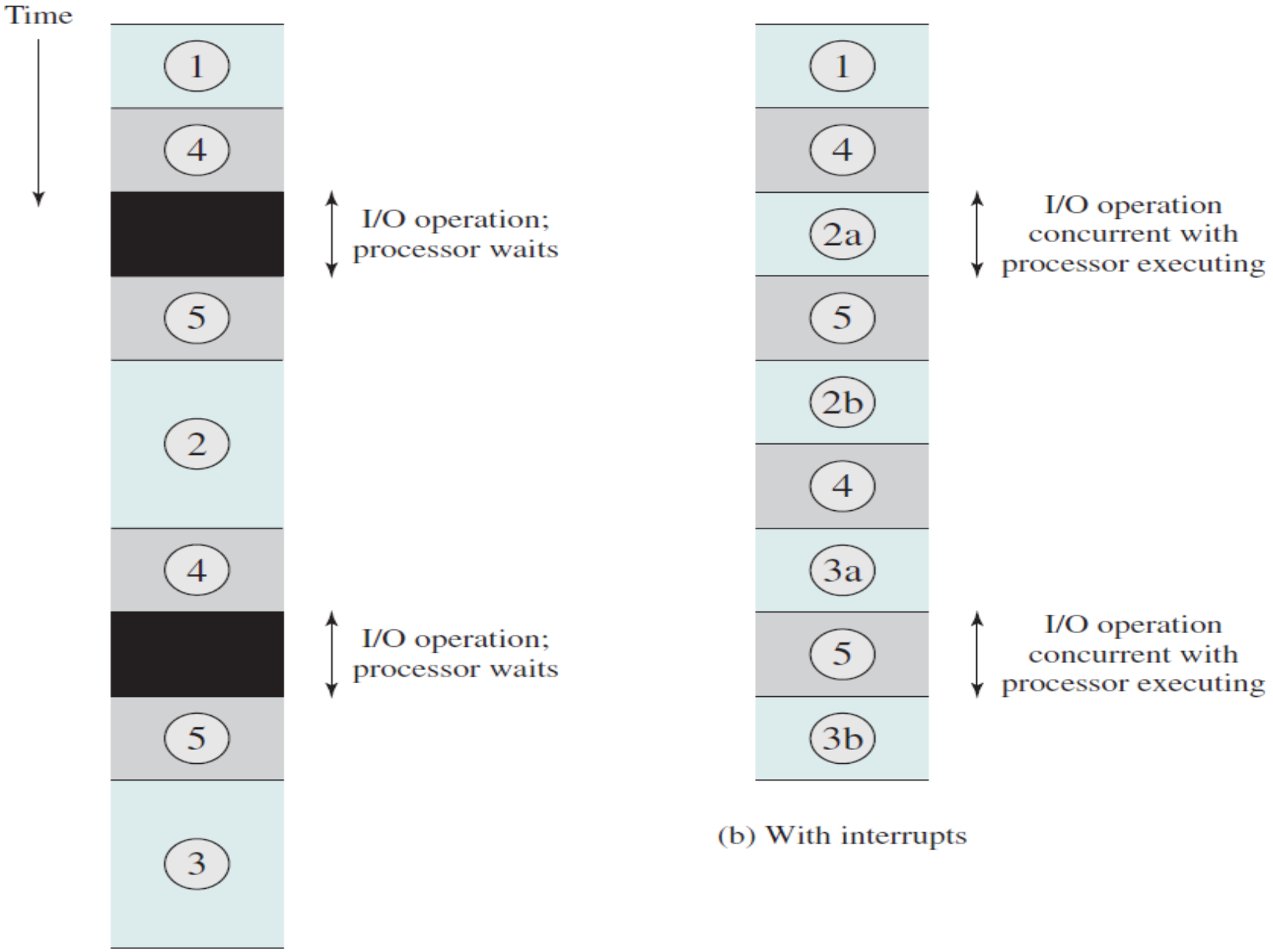
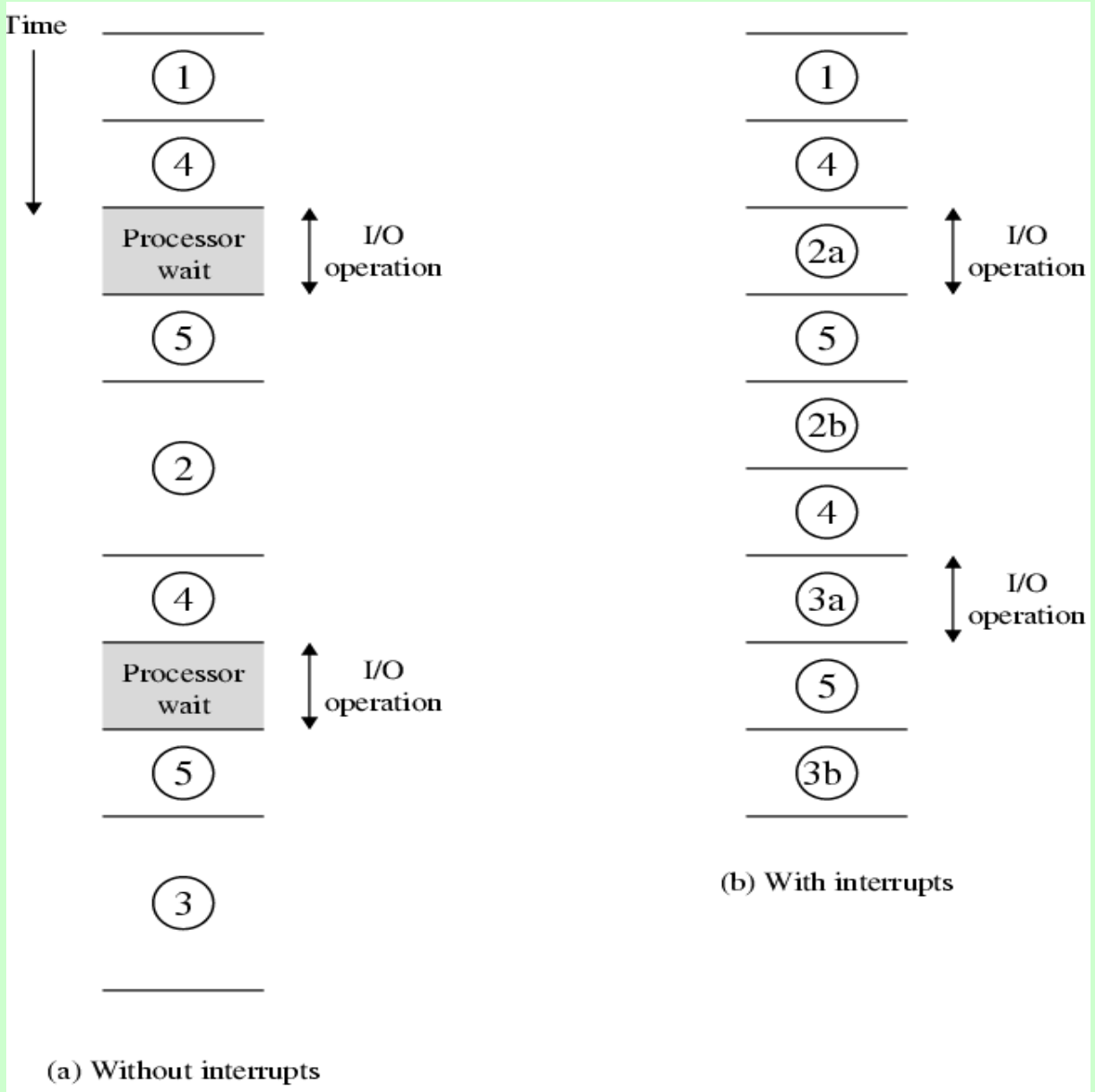


Figure 1.8 Program Timing: Short I/O Wait

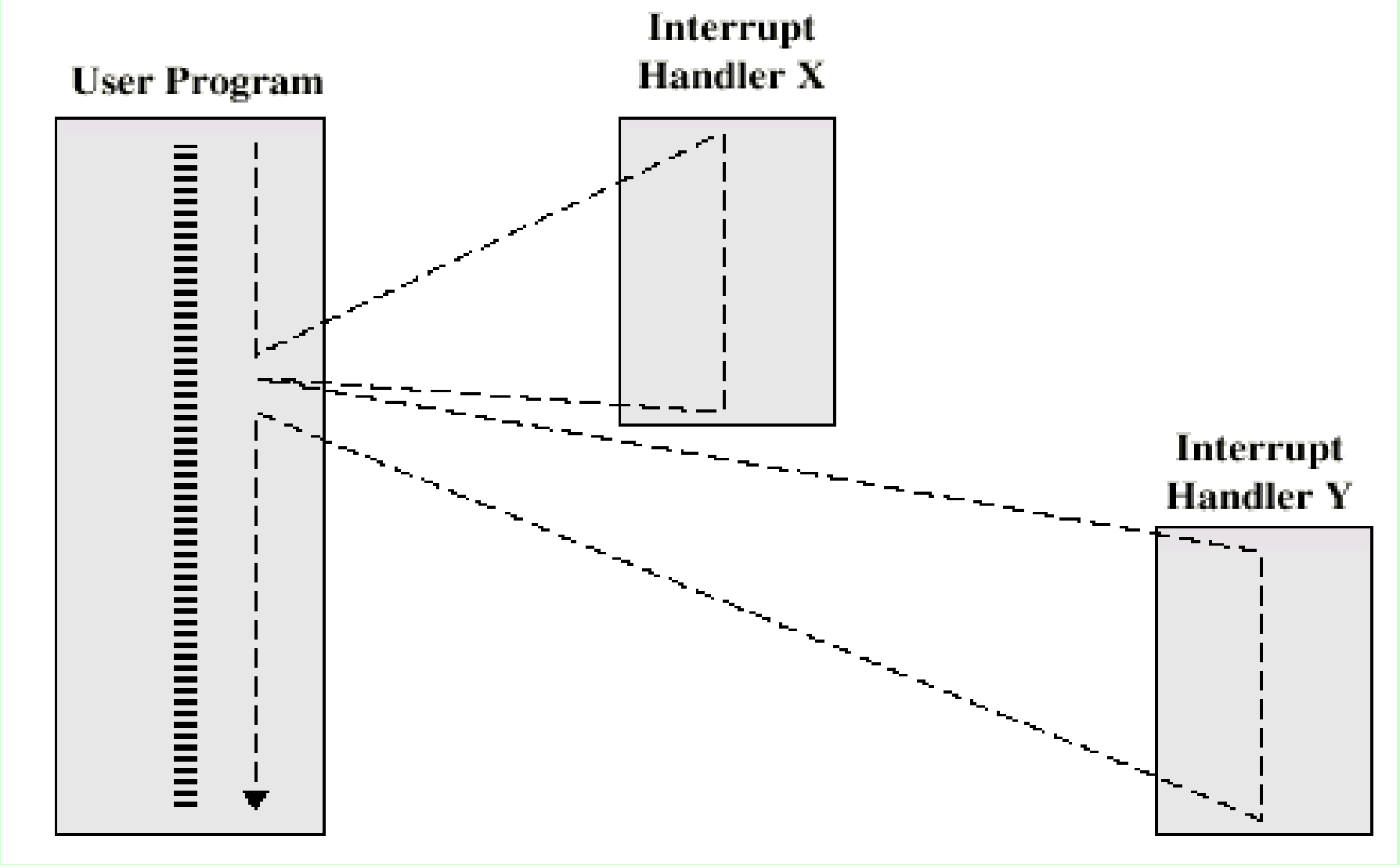
Program Timing
Short I/O Wait



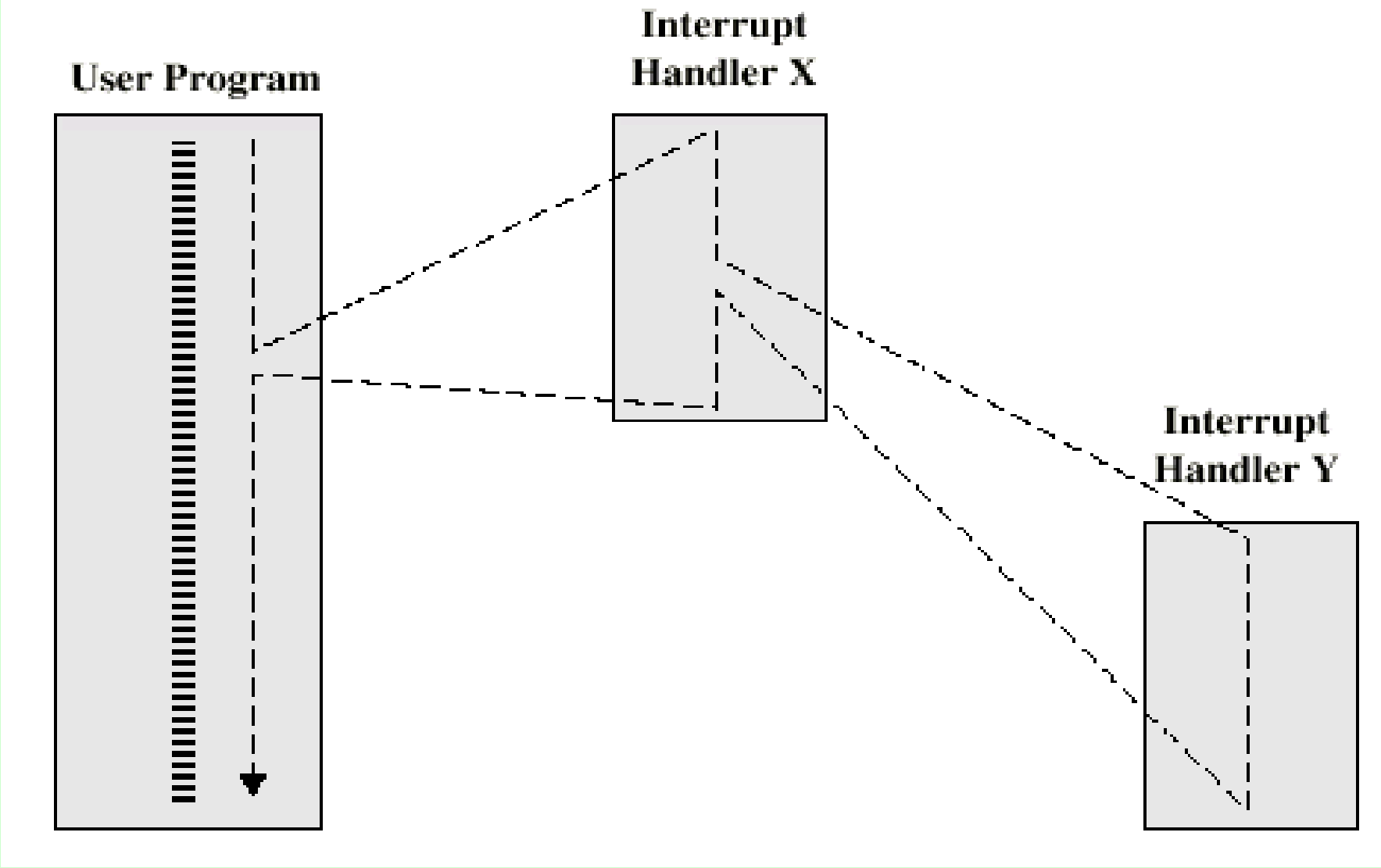
Multiple Interrupts

- **Disable interrupts**
  - **Processor will ignore further interrupts whilst processing one interrupt**
  - **Interrupts remain pending and are checked after first interrupt has been processed**
  - **Interrupts handled in sequence** as they occur
- **Define priorities**
  - **Low priority interrupts can be interrupted by higher priority interrupts**
  - When higher priority interrupt has been processed, processor returns to previous interrupt

Multiple Interrupts - Sequential



Multiple Interrupts – Nested



Time Sequence of Multiple Interrupts