Computer Organization and Architecture Week 5

+

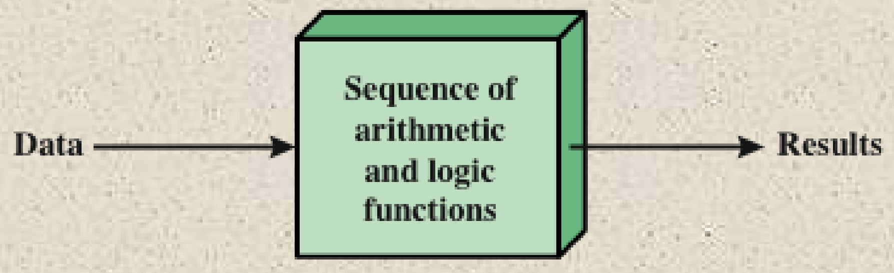
Computer Components

- Virtually all contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton

- Referred to as the *von Neumann architecture* and is based on three key concepts:
  - Data and instructions are stored in a single read-write memory
  - The contents of this memory are addressable by location, without regard to the type of data contained there
  - Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next

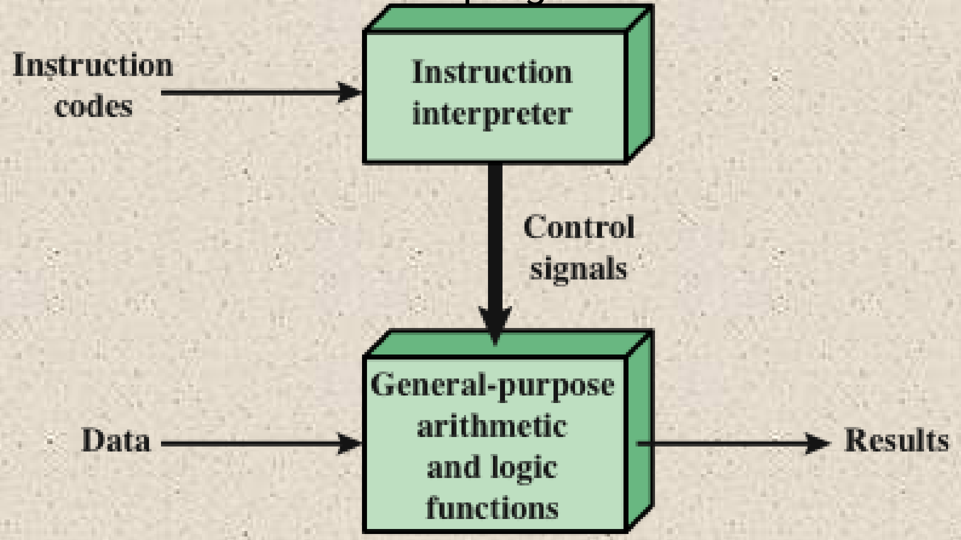
- *Hardwired program*
  - The result of the process of connecting the various components in the desired configuration as a form of programming.
  - The resulting "program" is in the form of hardware and is termed a hardwired program

Hardware and Software Approaches



(a) Programming in hardware

With general-purpose hardware, the system accepts data (configuration of arithmetic and logic functions) and control signals and produces results. Rewiring the hardware for each new program.



(b) Programming in software

The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data. For each step, a new set of control signals is needed. Let us provide a unique code for each possible set of control signals, and let us add to the general-purpose hardware a segment that can accept a code and generate control signals

Figure 3.1 Hardware and Software Approaches

Software

Programming is now much easier. Instead of rewiring the hardware for each new program

- A sequence of codes or instructions
- Part of the hardware interprets each instruction and generates control signals
- Provide a new sequence of codes for each new program instead of rewiring the hardware

Major components:

- CPU
  - Instruction interpreter
  - Module of general-purpose arithmetic and logic functions
- I/O Components
  - Input module
    - Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module
    - Means of reporting results

Software

I/O Components

One more component is needed. An input device will bring instructions and data in sequentially. But a program is not invariably executed sequentially; it may jump around

The CPU exchanges data with memory

Memory address register (MAR)

- Specifies the address in memory for the next read or write

Memory buffer register (MBR)

- Contains the data to be written into memory or receives the data read from memory

I/O address register (I/OAR)

- Specifies a particular I/O device

I/O buffer register (I/OBR)

- Used for the exchange of data between an I/O module and the CPU

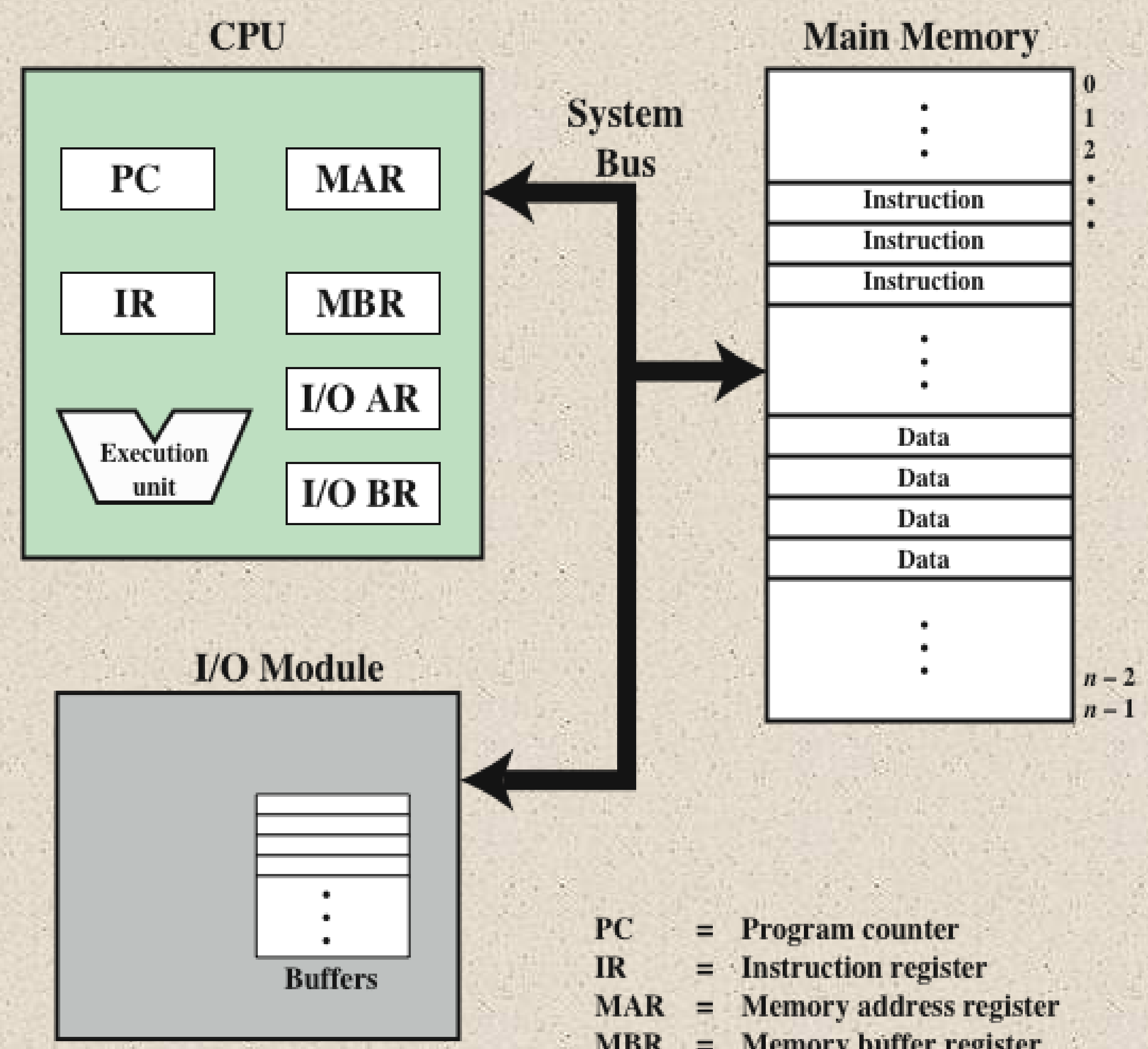
Operations on data may require access to more than just one element at a time in a predetermined sequence and must be a place to store temporarily both instructions and data.

MEMORY

MAR

MBR

Computer Components: Top Level View

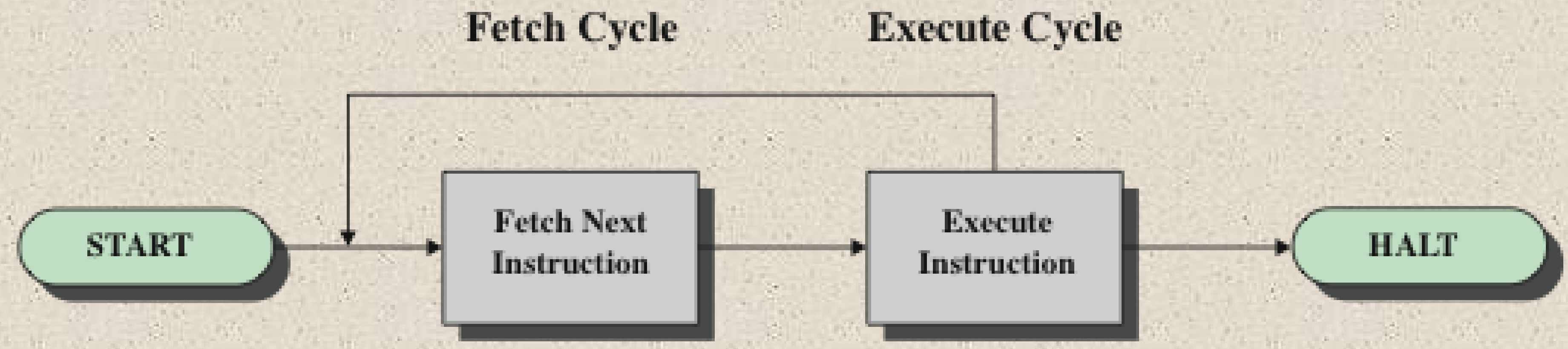


PC	=	Program counter
IR	=	Instruction register
MAR	=	Memory address register
MBR	=	Memory buffer register
I/O AR	=	Input/output address register
I/O BR	=	Input/output buffer register

Figure 3.2 Computer Components: Top-Level View

A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data. An I/O module transfers data from external devices to CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

+
Basic Instruction Cycle



Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.

Figure 3.3 Basic Instruction Cycle

+

Fetch Cycle

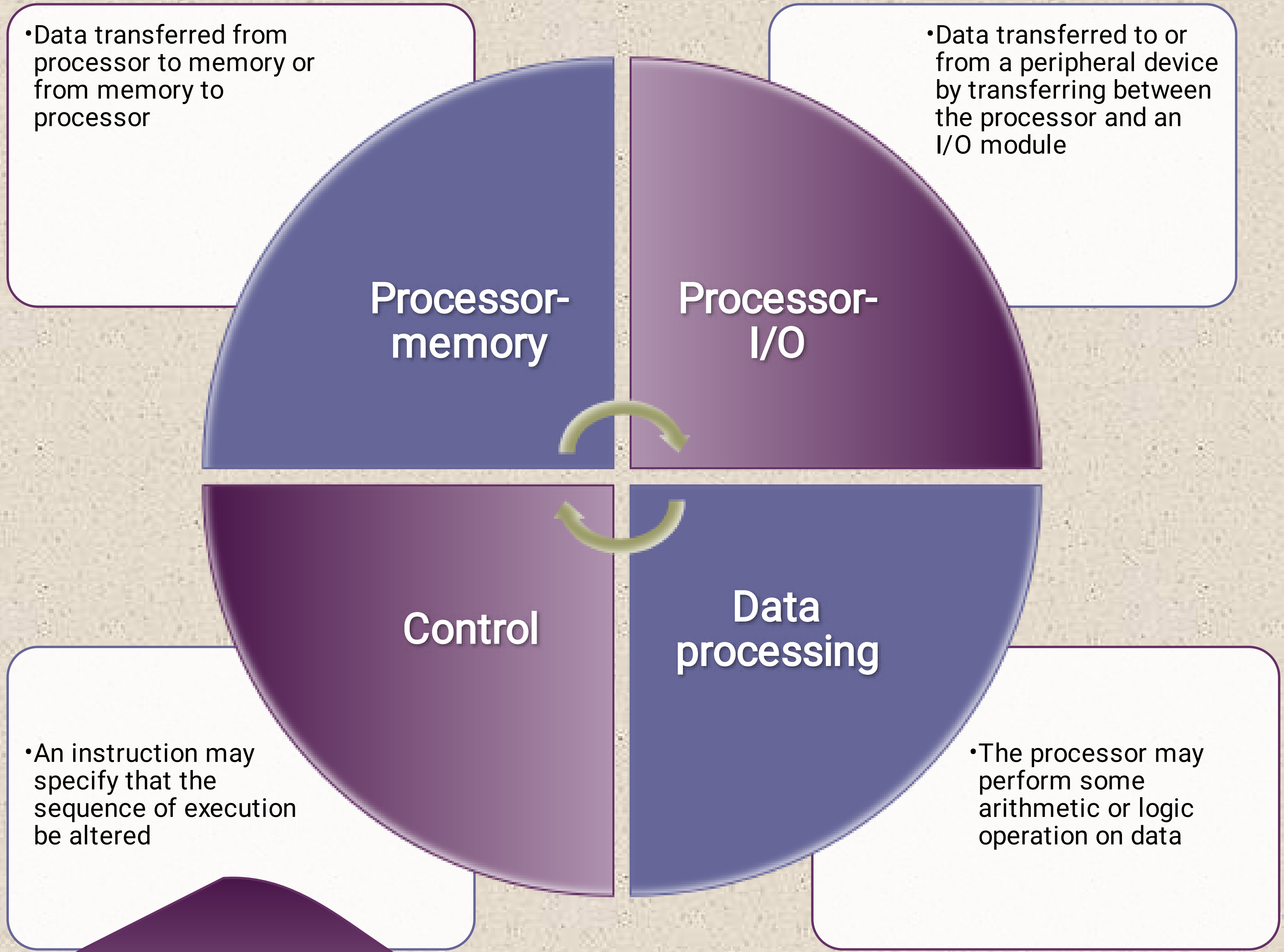
- At the beginning of each instruction cycle the processor fetches an instruction from memory
- In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next
- The processor increments the PC after each instruction fetch so that it will fetch the next instruction in sequence
- For example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to memory location 300, where the location address refers to a 16-bit word. The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on..

+

Fetch Cycle

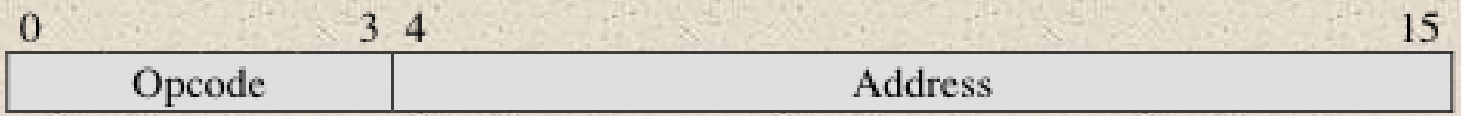
- The fetched instruction is loaded into the instruction register (IR)
- The processor interprets the instruction and performs the required action

Action Categories

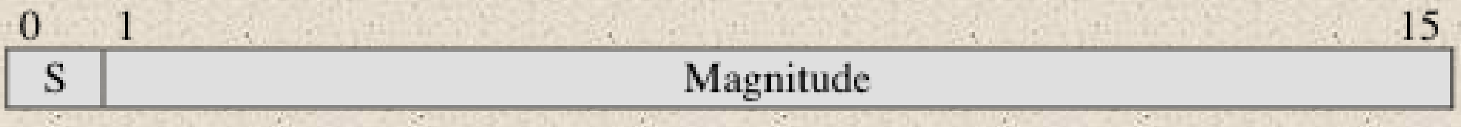


- Data transferred from processor to memory or from memory to processor
- Data transferred to or from a peripheral device by transferring between the processor and an I/O module
- An instruction may specify that the sequence of execution be altered
- The processor may perform some arithmetic or logic operation on data

For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor will remember this fact by setting the program counter to 182. Thus, on the next fetch cycle, the instruction will be fetched from location 182 rather than 150.



(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

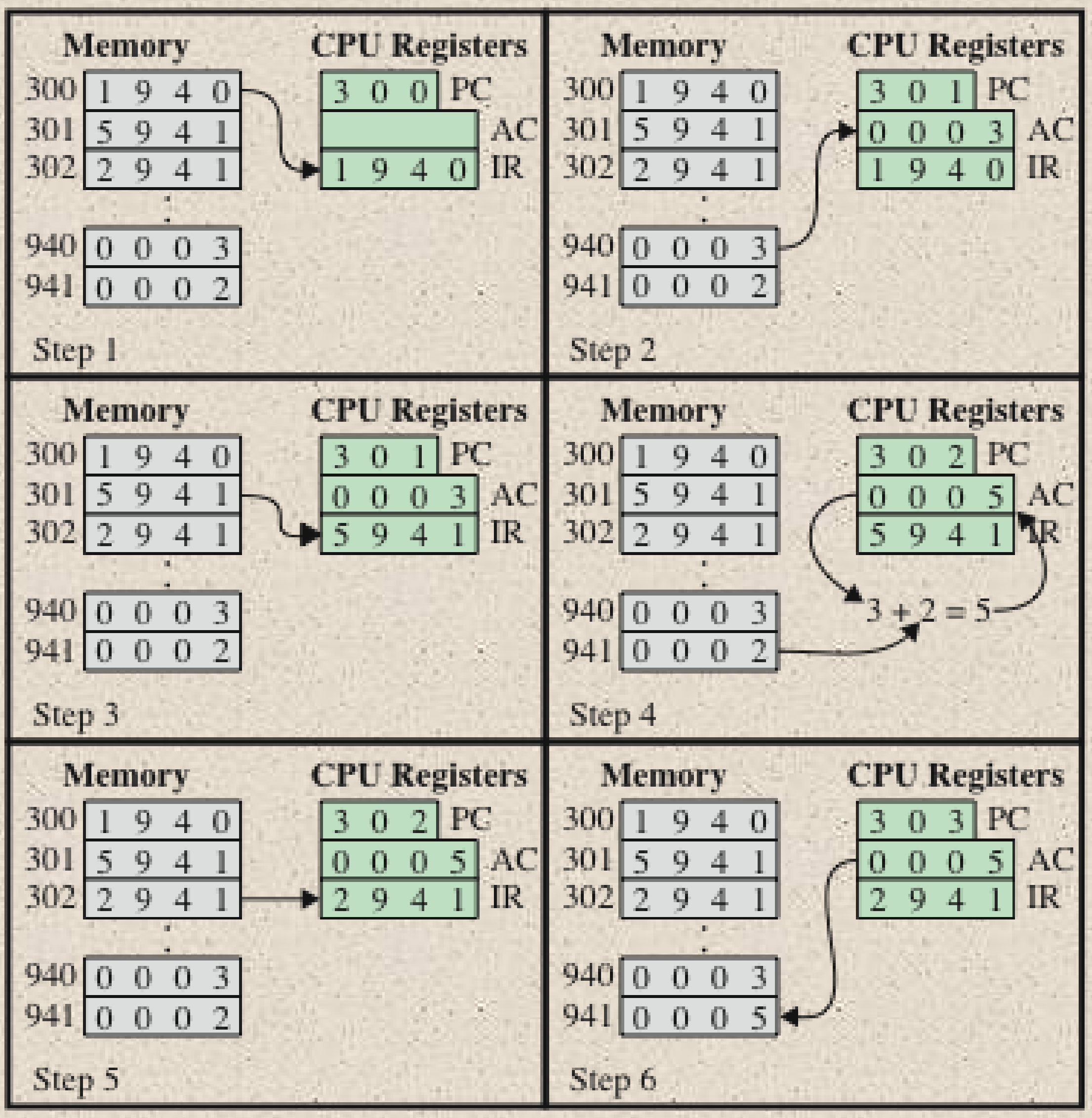
(d) Partial list of opcodes

The processor contains a single data register, called an accumulator (AC). Both instructions and data are 16 bits long. Thus, it is convenient to organize memory using 16-bit words. The instruction format provides 4 bits for the opcode, so that there can be as many as $2^4 = 16$ different opcodes, and up to $2^{12} = 4096$ (4K) words of memory can be directly addressed.

Figure 3.4 Characteristics of a Hypothetical Machine

Example of Program Execution

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the instruction register IR, and the PC is incremented.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded. The remaining 12 bits (three hexadecimal digits) specify the address (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301, and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added, and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302, and the PC is incremented.
6. The contents of the AC are stored in location 941



This example, three instruction cycles, each consisting of a fetch cycle and an execute cycle, are needed to add the contents of location 940 to the contents of 941

Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

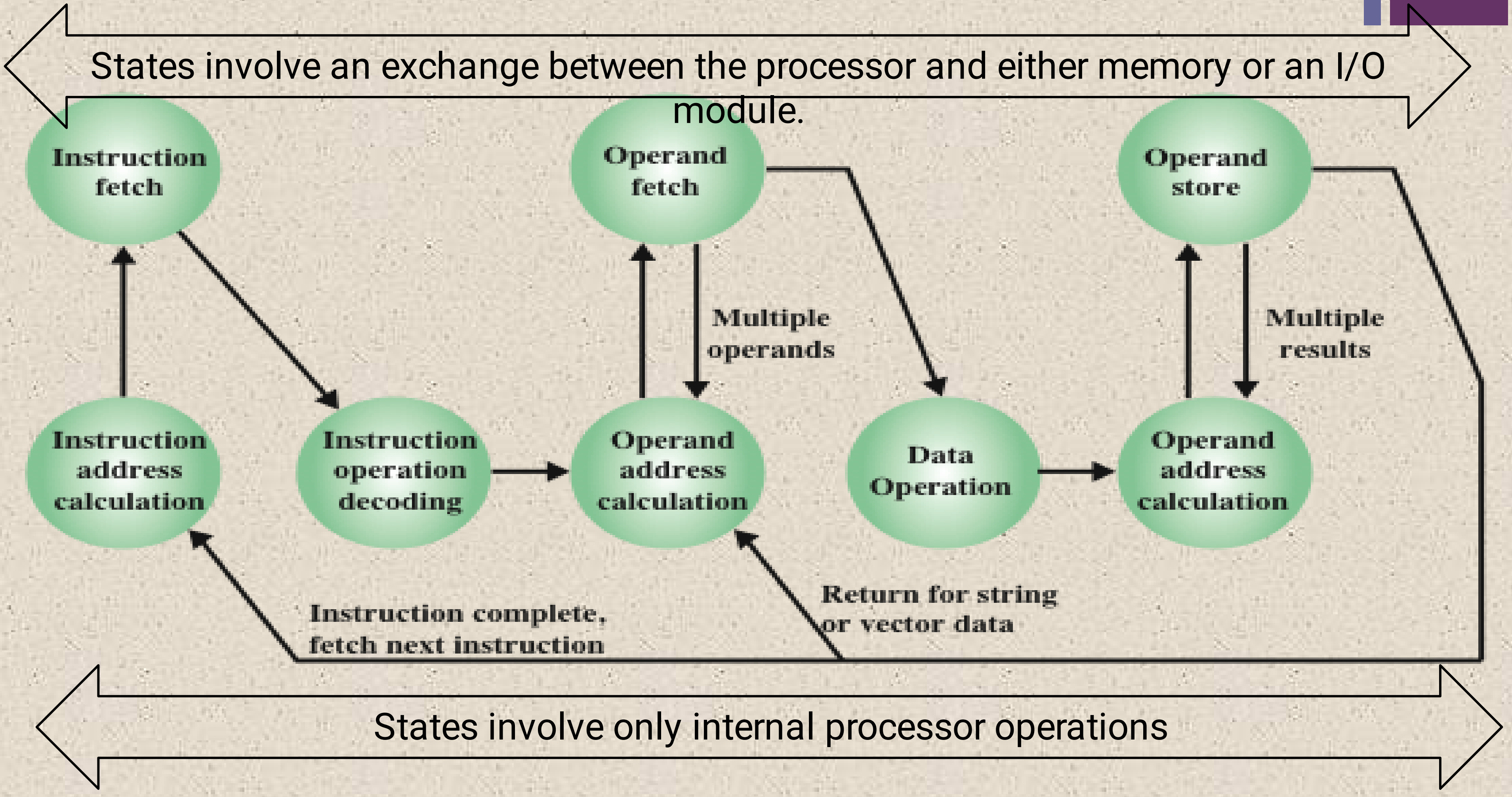
+ Instruction Cycle State

- For example, PDP-II processor includes and instructions, expresses symbolically as ADD B,A, that stores the sum of the contents of memory locations B and A into memory location A.
- A single instruction cycle with the following steps occurs:
  - Fetch the ADD instruction
  - Read the contents of memory location A into the processor.
  - Read the contents of memory location B into the processor. In order that the contents A are not lost, the processor must have at least two registers for storing memory values, rather than a single accumulator.
  - Add the two values.
  - Write the result from the processor to memory location A.

+

Instruction Cycle State Diagram

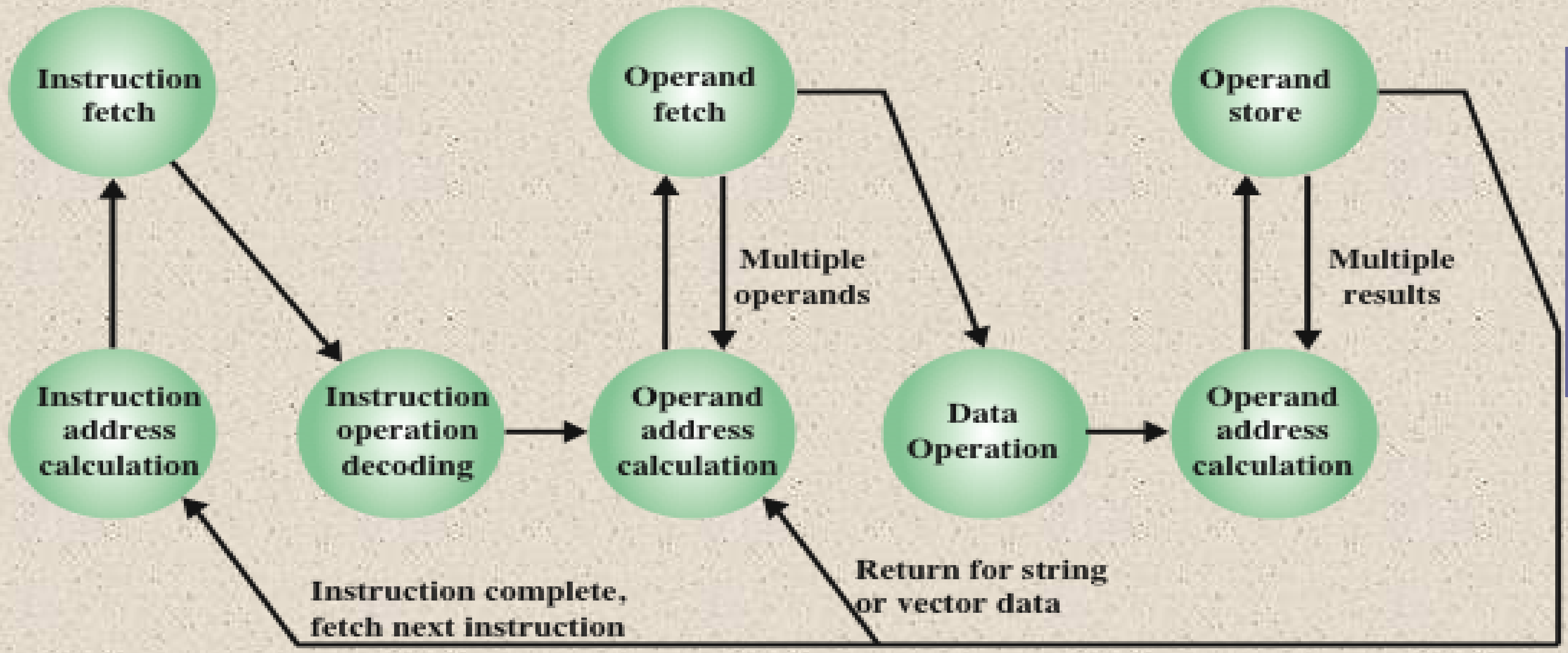
The execution cycle for a particular instruction may involve more than one reference to memory. Also, instead of memory references, an instruction may specify an I/O operation. Indicates, this would involve repetitive operand fetch and/or store operations.



States involve an exchange between the processor and either memory or an I/O module.
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Return for string or vector data
Instruction complete, fetch next instruction
States involve only internal processor operations

Figure 3.6 Instruction Cycle State Diagram

+


Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Instruction complete, fetch next instruction
Return for string or vector data

vector (one-dimensional array) of numbers or a string (one-dimensional array) of characters

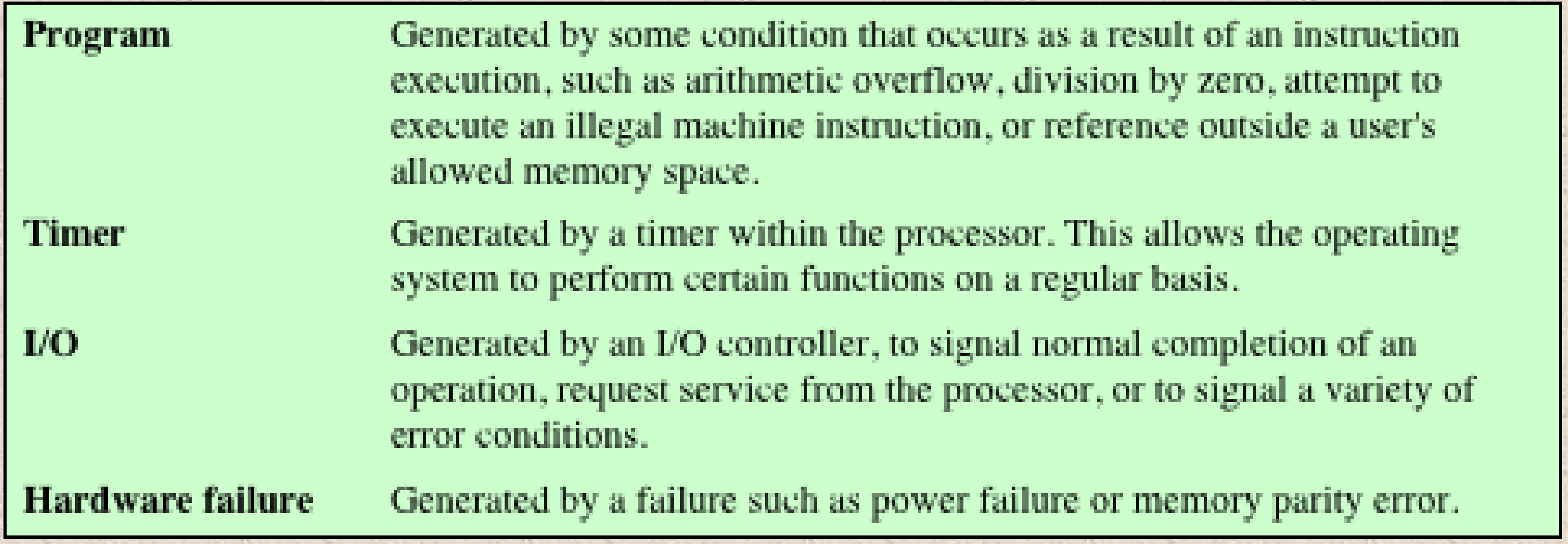
if: Read instruction from its memory location into the processor.
iac: Determine the address of the next instruction to be executed.
iod: Analyze instruction to determine type of operation to be performed and operand(s) to be used.
oac: If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
of: Fetch the operand from memory or read it in from I/O.
do: Perform the operation indicated in the instruction.
os: Write the result into memory or out to I/O.

The **oac** state appears twice, because an instruction may involve a read, a write, or both

Classes of Interrupts

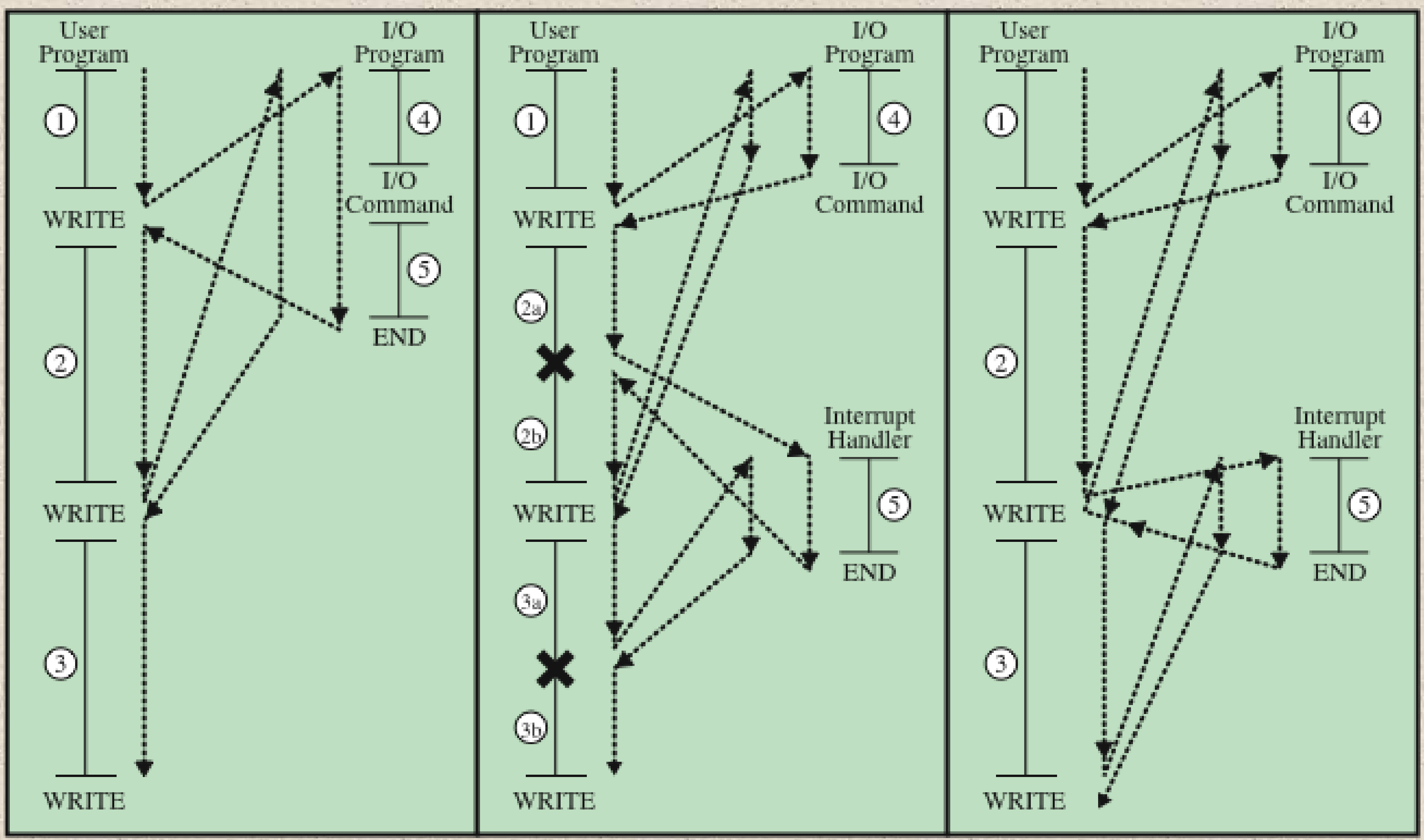
Virtually all computers provide a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor.

Following are the most common 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, or 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, request service from the processor, or to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

Program Flow Control



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

✖ = interrupt occurs during course of execution of user program

Figure 3.7 Program Flow of Control Without and With Interrupts

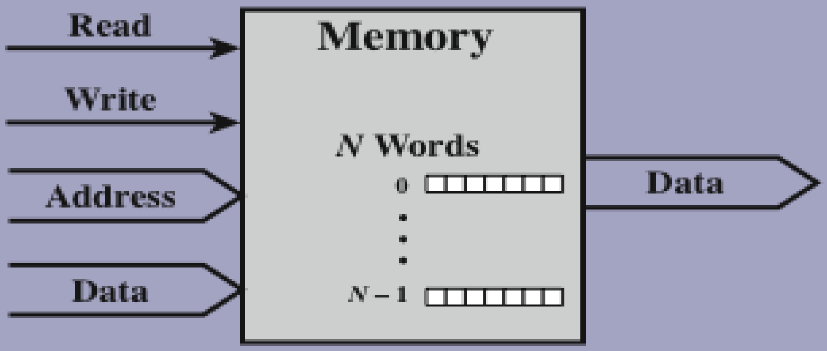
+

I/O Function

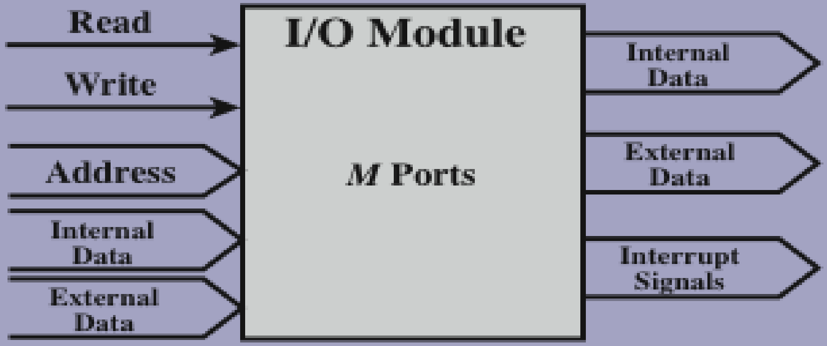
- I/O (e.g., a disk controller) module can exchange data directly with the processor
- Processor can read data from or write data to an I/O module
  - Processor identifies a specific device that is controlled by a particular I/O module
  - I/O instructions rather than memory referencing instructions
- In some cases it is desirable to allow I/O exchanges to occur directly with memory
  - The processor grants to an I/O module the authority to read from or write to memory so that the I/O memory transfer can occur without tying up the processor
  - The I/O module issues read or write commands to memory relieving the processor of responsibility for the exchange
  - This operation is known as direct memory access (DMA)

+

Computer Modules



a memory module will consist of *N words of equal length.* Each word is assigned a unique numerical address (0, 1, …, *N - 1)*



to an external device as a *port* and give each a unique address (e.g., 0, 1, …, *M - 1)*

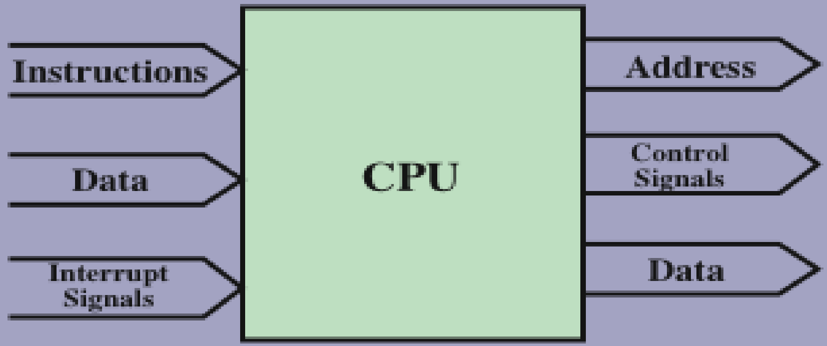


Figure 3.15 Computer Modules

interconnection structure

The interconnection structure must support the following types of transfers:

Memory to processor

Processor reads an instruction or a unit of data from memory

Processor to memory

Processor writes a unit of data to memory

I/O to processor

Processor reads data from an I/O device via an I/O module

Processor to I/O

Processor sends data to the I/O device

I/O to or from memory

An I/O module is allowed to exchange data directly with memory without going through the processor using direct memory access

Bus Interconnection

A communication pathway connecting two or more devices

- Key characteristic is that it is a shared transmission medium

Signals transmitted by any one device are available for reception by all other devices attached to the bus

- If two devices transmit during the same time period their signals will overlap and become garbled

Typically consists of multiple communication lines

- Each line is capable of transmitting signals representing binary 1 and binary 0

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy

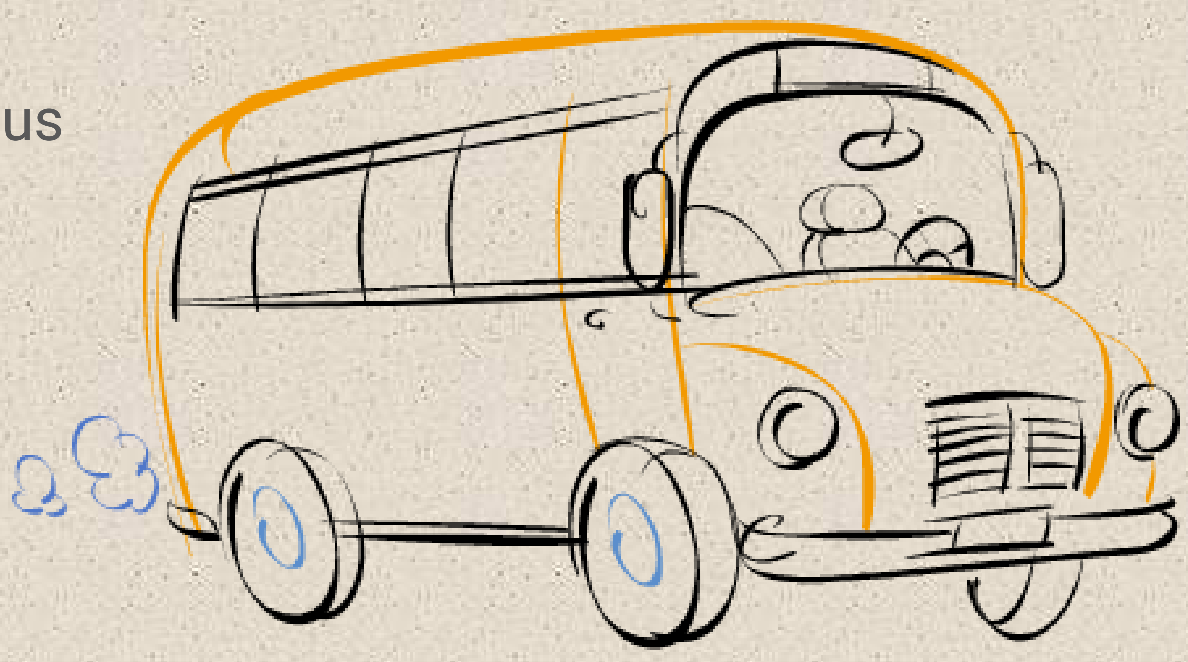
System bus

- A bus that connects major computer components (processor, memory, I/O)

The most common computer interconnection structures are based on the use of one or more system buses

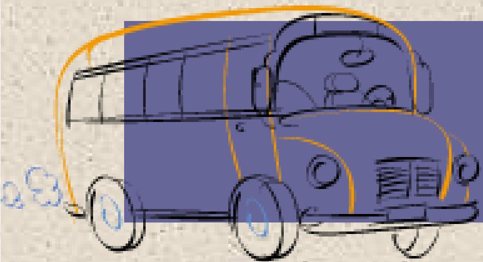
Data Bus

- Data lines that provide a path for moving data among system modules
- May consist of 32, 64, 128, or more separate lines
- The number of lines is referred to as the *width* of the data bus
- The number of lines determines how many bits can be transferred at a time
- The width of the data bus is a key factor in determining overall system performance



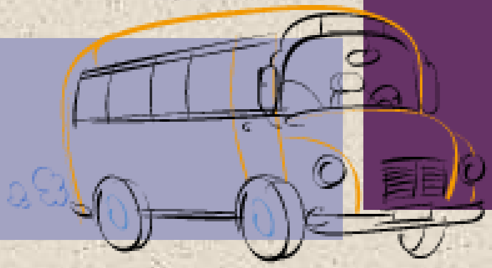
+

Address Bus



- Used to designate the source or destination of the data on the data bus
  - If the processor wishes to read a word of data from memory it puts the address of the desired word on the address lines
- Width determines the maximum possible memory capacity of the system
- Also used to address I/O ports
  - The higher order bits are used to select a particular module on the bus and the lower order bits select a memory location or I/O port within the module

Control Bus



- Used to control the access and the use of the data and address lines
- Because the data and address lines are shared by all components there must be a means of controlling their use
- Control signals transmit both command and timing information among system modules
- Timing signals indicate the validity of data and address information
- Command signals specify operations to be performed

Bus Interconnection Scheme

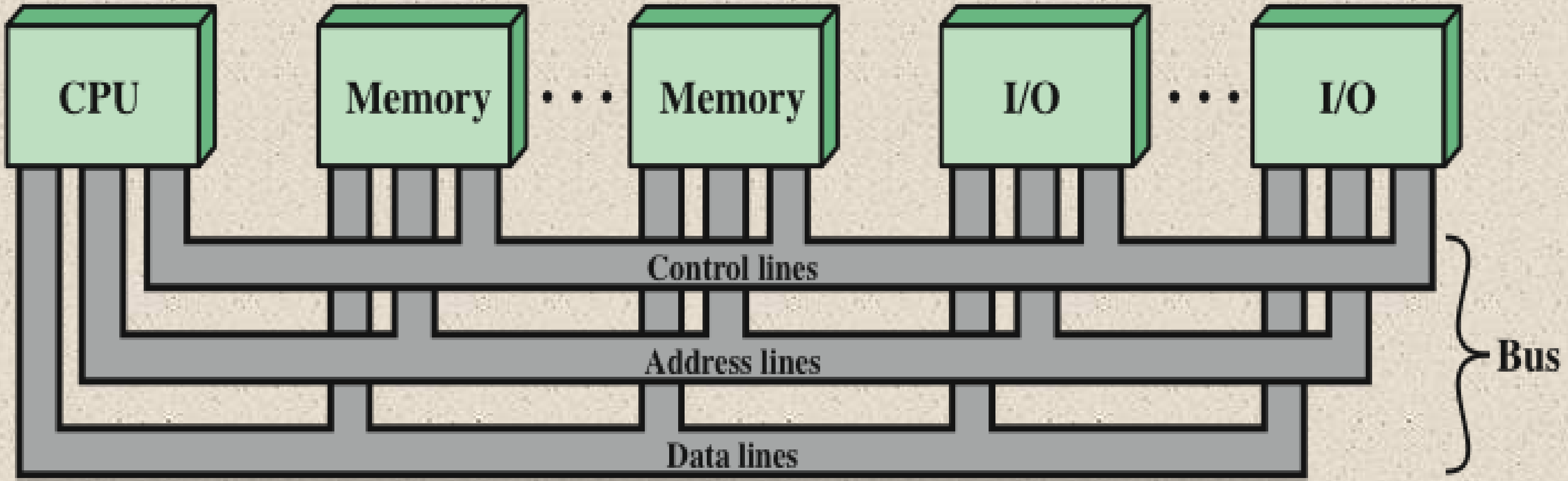
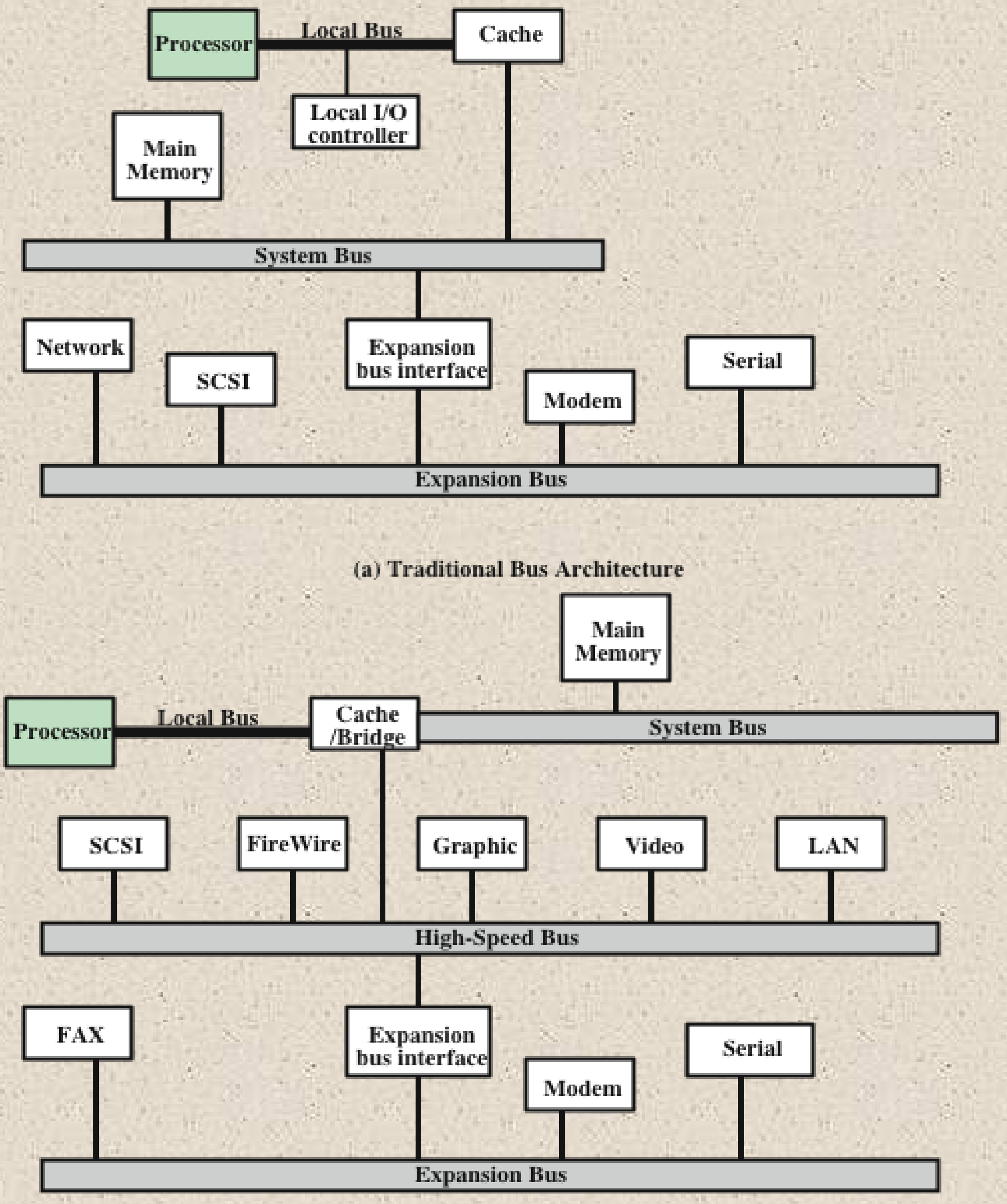


Figure 3.16 Bus Interconnection Scheme

Bus Configurations



(a) Traditional Bus Architecture

(b) High-Performance Architecture

Figure 3.17 Example Bus Configurations