Computer Organization and Architecture (EET 2211)

Chapter 3
A Top-Level View of Computer Function and Interconnection

Learning Objectives:

After studying this chapter, you should be able to:

- Understand the basic elements of an instruction cycle and the role of interrupts.
- Describe the concept of interconnection within a computer system.
- Assess the relative advantages of point-to-point interconnection compared to bus interconnection.
- Present an overview of QPI.
- Present an overview of PCIe.

Introduction:

- At a top level, a computer consists of CPU (central processing unit), memory, and I/O components.

- At a top level, we can characterize a computer system by describing :

(1) the external behavior of each component, that is, the data and control signals that it exchanges with other components, and

(2) the interconnection structure and the controls required to manage the use of the interconnection structure.

Contd.

- Top-level view of structure and function is important because it explains the nature of a computer and also provides understanding about the increasingly complex issues of performance evaluation.
- This chapter focuses on the basic structures used for computer component interconnection.
- The chapter begins with a brief examination of the basic components and their interface requirements.
- Then a functional overview is provided.
- Then the use of buses to interconnect system components has been explained.

3.1. Computer Components

All contemporary computer designs are based on the concepts of *von Neumann architecture.* It 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.

Programming in hardware

- The fig.1 shows a customized hardware.
- The system accepts data and produces results.
- If there is a particular computation to be performed, a configuration of logic components designed specifically for that computation could be constructed.

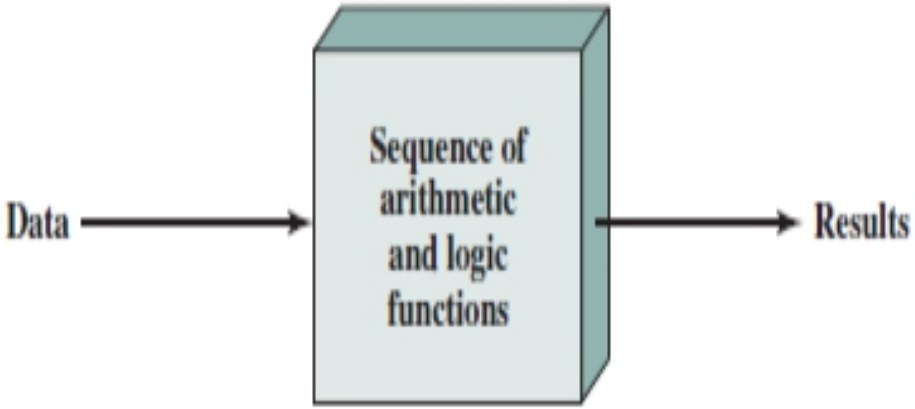


Fig.1. Programming in H/W.

- However, a rewiring of the hardware is required if a different computation is needed every time.

Programming in software

- Instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals.
- The fig.2 shows a general purpose hardware, that will perform various functions on data depending on control signals applied to the hardware.
- The system accepts data and control signals and produces results.

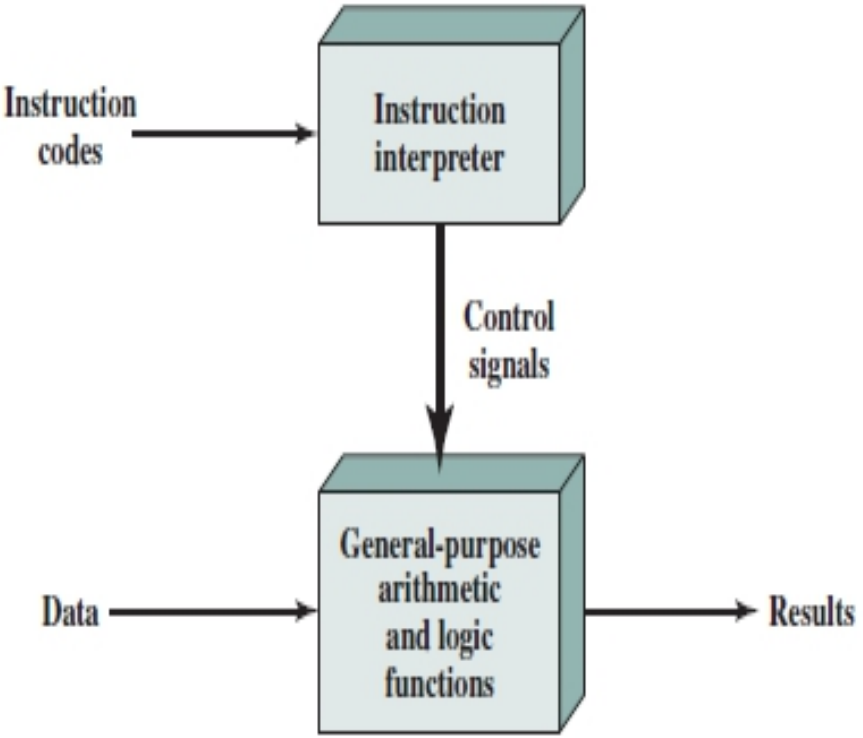


Fig.2. Programming in S/W.

How to supply the control signals?

- 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. Provide a unique code for each possible set of control signals and add to the general-purpose hardware a segment that can accept a code and generate control signals as shown in fig.2.
- Instead of rewiring the hardware for each new program, provide a new sequence of codes.
- Each code is an instruction, and part of the hardware interprets each instruction and generates control signals. To distinguish this new method of programming, a sequence of codes or instructions is called ***software***.

3.2 Computer Function

- The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory.
- The processor does the actual work by executing instructions specified in the program.

- Instruction processing consists of two steps:

1. The processor reads (*fetches*) instructions from memory one at a time and
2. Executes each instruction.

- Program execution consists of repeating the process of instruction fetch and instruction execution. The instruction execution may involve several operations and depends on the nature of the instruction

Computer Components: Top-Level View

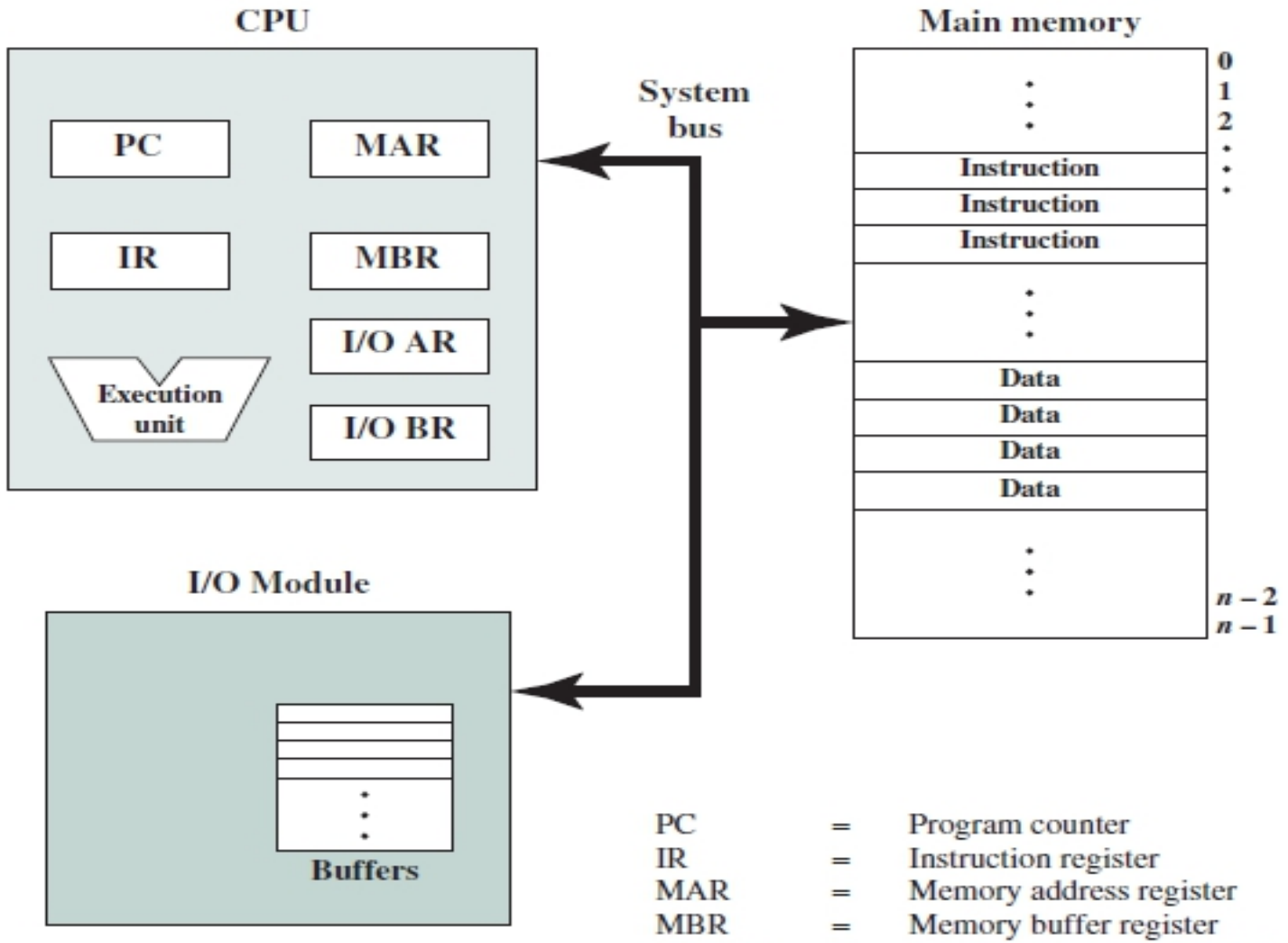


Fig.3. Computer Components : Top-Level View

Main Memory:

- Figure 3 illustrates these top-level components and suggests the interactions among them.

➢**Memory, or main memory:**

- An input device may fetch instructions and data sequentially. But the execution of a program may not be sequential always; it may jump around.
- Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence. Thus, there must be a place to temporarily store both instructions and data. That module is called *memory*, or *main memory.*
- The term ‘main memory’ has been used to distinguish it from external storage or peripheral devices.
- Von Neumann stated that the same memory could be used to store both instructions and data.

Central Processing Unit (CPU):

- The CPU exchanges data with memory by using two internal (to the CPU) registers:

1. **Memory Address Register (MAR)**: It specifies the address in memory for the next read or write.

2. Memory Buffer Register (MBR): It contains the data to be written into memory or receives the data read from memory.

- The CPU also contains:
- (I/O AR): It is an I/O address register which specifies a particular I/O device.
- (I/OBR): It is an I/O buffer register which is used for the exchange of data between an I/O module and the CPU.

Memory and I/O Module:

- **Memory Module:**
- It 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.
- **I/O module:**
- It 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.

Instruction Fetch and Execute:

- The processing required for a single instruction is called an **instruction cycle**.
- There are two steps referred to as the **fetch cycle** and the **execute cycle** as shown in the fig.4.

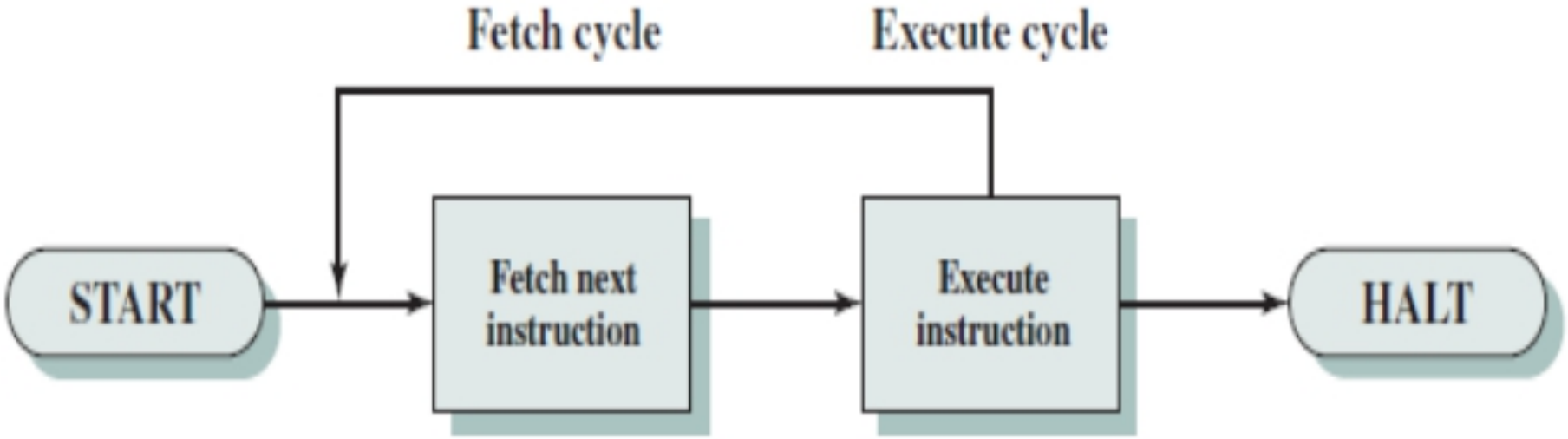


Fig.4. Basic Instruction Cycle

Contd.

- 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.

- Unless instructed, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e., the instruction located at the next higher memory address).

Contd.

- The fetched instruction is loaded into a register in the processor known as the **instruction register (IR)**.

- The instruction contains bits that specify the action the processor is to take.

- The processor interprets the instruction and performs the required action.

Contd.

- The processor performs the following four actions:
- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.

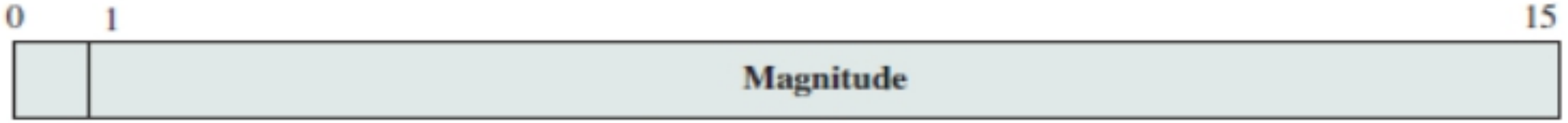
- **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.

- **Data processing:** The processor may perform some arithmetic or logic operation on data.

- **Control:** An instruction may specify that the sequence of execution be altered.

Characteristics of a Hypothetical Machine

(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

Fig.5. Characteristics of a Hypothetical Machine

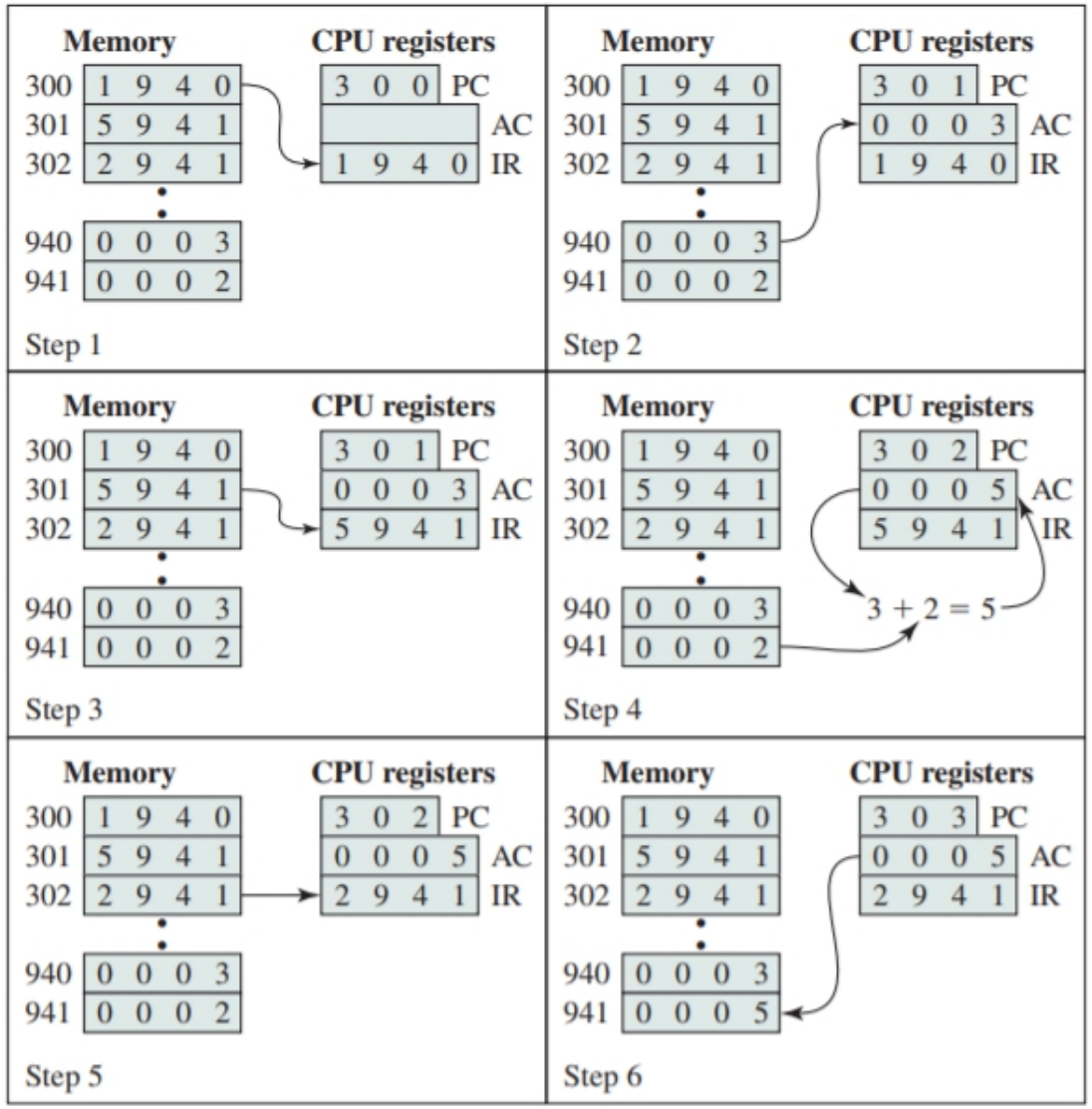


Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

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

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

Contd.

- An instruction's execution may involve a combination of these actions:
- Let us consider an example using a hypothetical machine that includes the characteristics listed in fig.5.
- 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.

Basic Instruction Cycle:

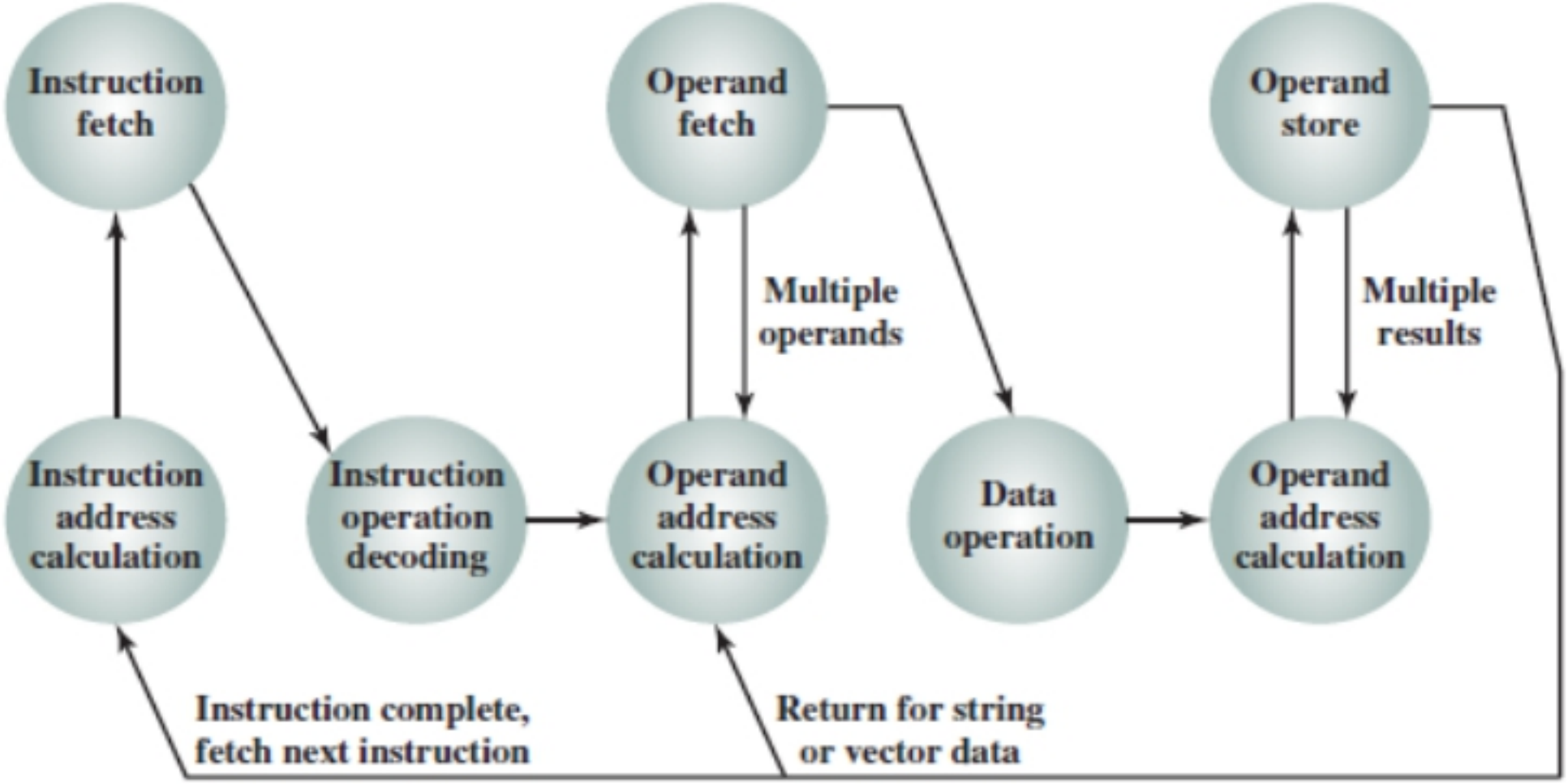


Fig.6. Instruction Cycle State Diagram

Contd.

- Fig.6. shows the state diagram of basic instruction cycle. The states can be described as follows:
- **Instruction address calculation (iac):** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction.
- For example, if each instruction is 16 bits long and memory is organized into 16-bit words, then add 1 to the previous address. If, instead, memory is organized as individually addressable 8-bit bytes, then add 2 to the previous address.
- **Instruction fetch (if):** Read instruction from its memory location into the processor.

Contd.

- **Instruction operation decoding (iod):** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand address calculation (oac):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- **Operand fetch (of):** Fetch the operand from memory or read it in from I/O.
- **Data operation (do):** Perform the operation indicated in the instruction.
- **Operand store (os):** Write the result into memory or out to I/O.

Contd.

- States in the upper part of fig.6. involve an exchange between the processor and either memory or an I/O module.
- States in the lower part of the diagram involve only internal processor operations.
- The oac state appears twice, because an instruction may involve a read, a write, or both.
- However, the action performed during that state is fundamentally the same in both cases, and so only a single state identifier is needed.

Thank You ！