COURSE GUIDE

IFT 212
COMPUTER ARCHITECTURE AND ORGANIZATION

Course Team Greg Onwodi **(**Developer/Writer**) - NOUN**
J. B. Awotunde (Reviewer) - UniIlorin
Prof Joshua Abah (Course Editor)

NATIONAL OPEN UNIVERSITY OF NIGERIA

National Open University of Nigeria
Headquarters
University Village
Plot 91, Cadastral Zone
Nnamdi Azikiwe Expressway
Jabi, Abuja

Lagos Office
14/16 Ahmadu Bello Way
Victoria Island, Lagos

e-mail: centralinfo@nou.edu.ng
URL: www.nou.edu.ng

Printed 2009

Reviewed and Reprinted 2025

ISBN:978-978-786-494-4

MAIN COURSE

CONTENTS **PAGE**

MODULE 1 ORGANIZATION AND ARCHITECTURE

UNIT 1 INTRODUCTION TO COMPUTER ARCHITECTURE AND ORGANIZATION

CONTENTS

1.0 INTRODUCTION

Despite the variety and pace of change in the computer field, certain fundamental concepts consistently apply throughout. The application of these concepts depends on the current state of technology and the price/performance objectives of the designer.

Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in their organization. In a class of computers called microcomputers, the relationship between architecture and organization is very close. Changes in technology not only influence organizations but also result in the introduction of more powerful and complex architectures. However, because a computer organization must be designed to implement a particular architectural specification, a thorough treatment of organization requires a detailed examination of the architecture as well.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- Explain the operational units of a computer system.
- Outline types of operands and operations specific by machine instruction.
- Explain opcodes, operands, and addressing modes

3.0 MAIN CONTENT

3.1 COMPUTER ORGANIZATION AND ARCHITECTURE

Although it is difficult to give a precise definition, a consensus exists about the general area covered by it. Computer organization refers to the operational units and their interconnection that realize the architectural specification.

Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e. g numbers, characters), I/O mechanism, and techniques for addressing memory. Organizational attributes include hardware details transparent to the programmer, such as control signals; interfaces between the computer peripherals, and memory technology used.

In computer engineering, computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. The architecture of a system refers to its structure in terms of separately specified components of that system and their interrelationships.

Computer architecture consists of rules and methods or procedures that describe the implementation, and functionality of the computer systems. We can define computer architecture based on its performance, efficiency, reliability, and cost of the computer system. It deals with software and hardware technology standards.

3.2 STRUCTURE AND FUNCTION

A computer is a computer system, contemporary computers contain millions of elementary electronic components.

- **Structure:** How the components are interrelated.
- **Function:** The operation of each component as part of the structure.

In terms of description, there are two choices: starting at the bottom and building up to a complete description, or beginning with a top view and decomposing the system into its subparts. Evidence from several fields suggests that the top-down approach is the clearest and most effective.

The approach taken is that the computer be described from the top down.

Both the structure and functioning of a computer are simple. Figure 1 depicts the basic functions that a computer can perform. In general terms, there are only four:

- Data processing
- Data storage
- Data movement
- Control

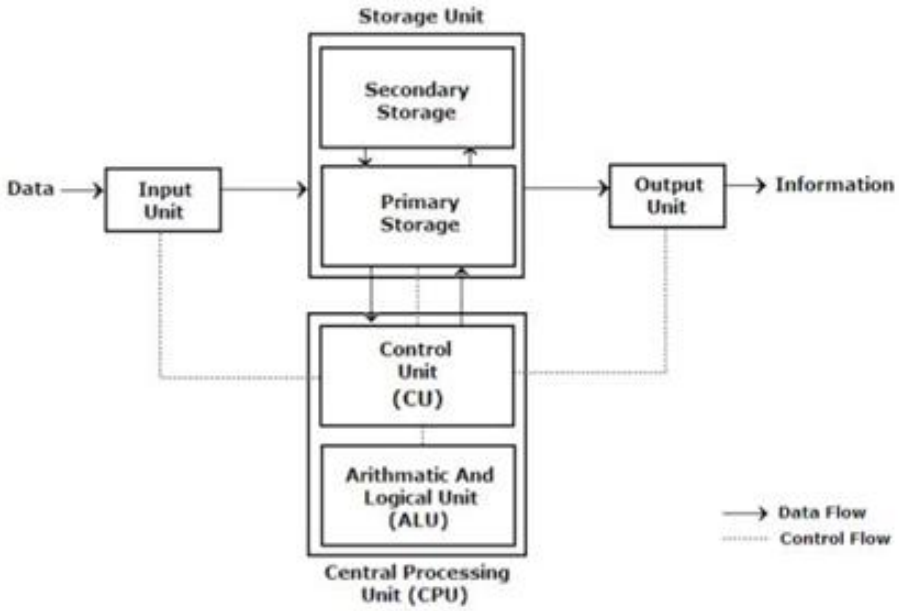


Figure 1: The Basic Functions of Computer

The computer, of course, must be able to process data. The data may take a wide variety of forms, and the range of processing requirements ID broad. It is also essential that a computer stores data. Even if the computer is processing data on the fly (i.e. data come in and get processed and the results go out immediately) the computer must temporarily store at least. Those pieces of data that are being worked on at any given moment. Files of data are stored on the computer for subsequent retrieval and update.

The computer must be able to move data between itself and the outside world. The computer's operating environment consists of devices that serve as either sources or destinations of data. When data are received

from or delivered to a device that is directly connected to the computer, the process is known as input-output (I/O), and the device is referred to as a peripheral. When data are moved over longer distances, to or from a remote device, the process is known as data communications. Finally, there must be control of these three functions. Ultimately, this control is exercised by the individuals who provide the computer with instructions. Within the computer, a control unit manages the resources of the computer and orchestrates the performance of its functional parts in response to those instructions.

There are four main structural components

- **The central processing unit (CPU**): Controls the operations of the computer and performs its data processing functions; often simply referred to as a processor.
- Main memory: Stores data
- **I/O:** Moves data between the computer and its external environment.
- **System interconnections**: Some mechanism that provides for communication among CPU, main memory, and I/O. A common example of system interconnection is through a system bus, consisting of several conducting wires to which all the other components attach.

However, the most interesting and complex component is the CPU. Its major structural components are as follows:

- **Control unit**: Controls the operations of the CPU and hence the computer.
- **Arithmetic and logic unit (ALU**): Performs the computer data processing functions.
- Registers: Provides storage internal to the CPU.
- **CPU interconnection**: Some mechanism that provides for communication among the control unit, ALU, and registers.

3.3 COMPUTER COMPONENTS

Virtually all contemporary computer designs are based on concepts developed by John Von Neumann at the Institute for Advanced Studies Princeton. Such a design is 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 sequentially (unless explicitly modified) from one instruction to the next.

There is a small set of basic logic components that can be combined in various ways to store binary data and to perform arithmetic and logical operations on that data. If there is a particular computation to be performed, a configuration of logic components designed specifically for that computation could be constructed. We can think 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.*

Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions. This set of hardware will perform various functions on data depending on control signals applied to the hardware. In the original case of customized hardware, the system accepts data and produces results Figure 2a. With general-purpose hardware, the system accepts data and control signals and produces results. Thus, instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals.

How shall control signals be supplied? The answer is simple but subtle. The entire program is 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 2b).

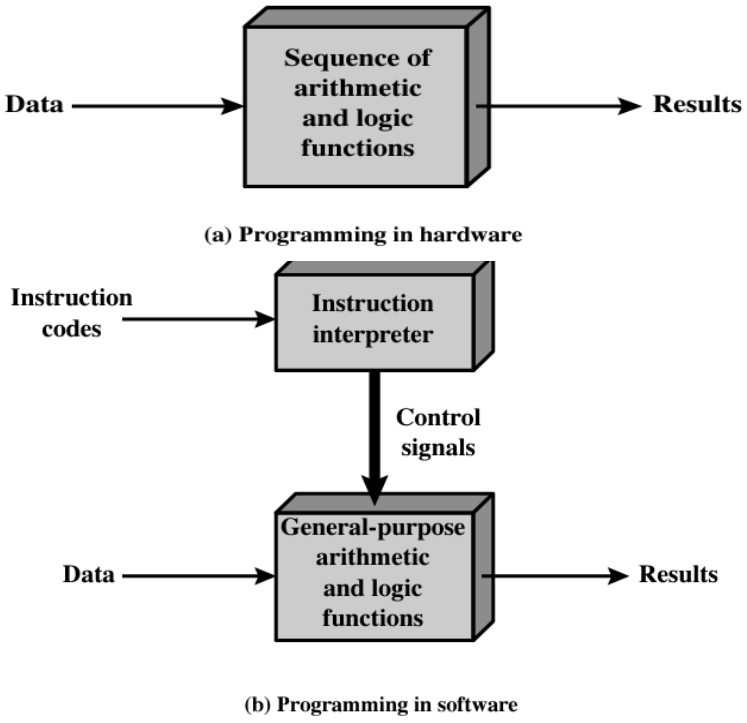


Figure 2. Hardware and Software Approaches

Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes. Each code is, in effect, 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.*

Figure 2b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions. These two constitute the CPU. Several other components are needed to yield a functioning computer. Data and instructions must be put into the system. For this, we need some sort of input module. This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system. A means of reporting results is needed, and this is in the form of an output module. Taken together, these are referred to as I10 *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 ma, jump around (e.g., the IAS jump instruction). 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 store temporarily both instructions and data. That module is called *memory,* or *main memory* to distinguish it from external storage of peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data.

Figure 3 illustrates these top-level components and suggests the interaction among them. The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) registers: a memory address register (MAR), which specifies the address in memory for the next read or write, and a memory buffer register (MBR), which contains the data to be written into memory receives the data read from memory. Similarly, an I/0 address register (I/OAR specifies a particular 1/0 device. An I/0 buffer (I/OBR) register is used for the $_{ex}$change of data between an I/0 module and the CPU.

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. A 1/0 module transfers data from external devices' CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

Having looked briefly at these major components, we now turn to an overview of how these components function together to execute programs.

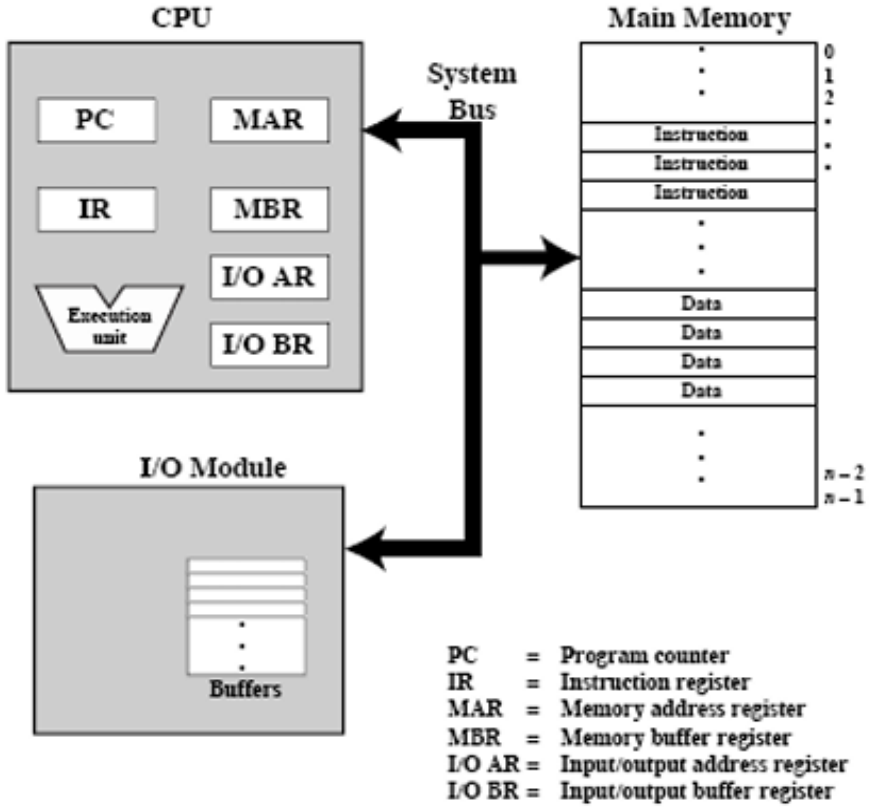


Figure 3. Computer Components Top-level View

The key elements of program execution. In its simplest form, instruction processing consists of two steps: The processor reads *(fetches)* instructions from memory one at a time and 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 (see, for example, the lower portion of Figure 2.4).

The processing required for a single instruction is called an *instruction cycle.*

The two steps are referred to as the *fetch cycle* and the *execute 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.

3.4 Instruction Fetch and Execute

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 told otherwise, the processor. Using the simplified two-step description given previously, the instruction cycle is depicted in Figure 4.

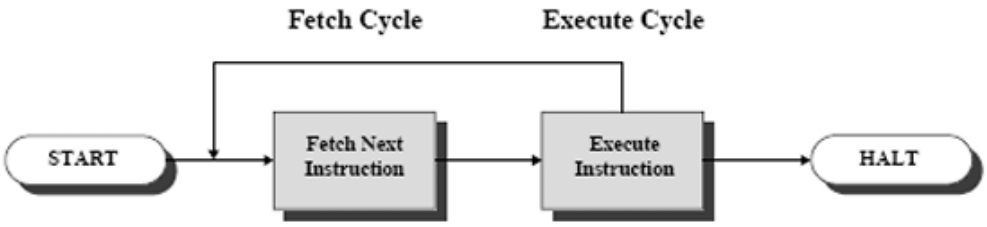


Figure 4. Basic Instruction Cycle

Explain an instruction fetch using the components of Figure 3

1) The PC holds the address of the next instruction to execute. The contents of the PC are placed on the System Bus and the PC is incremented to the next instruction to be executed.
2) The instruction from Main Memory is retrieved and placed into the IR using the System Bus.

Note: The MAR and MBR registers are also used in the process but for now we will ignore their use for simplicities sake.

The processor will then interpret the instruction and perform an action. What are these possible actions? 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). So, for example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300. 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. This sequence may be altered, as explained presently.

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. In general, these actions fall into four categories:

i. Processor-memory: Data may be transferred from processor to memory or from memory to processor.
ii. Processor-I/O: Data may be transferred to or from a peripheral

device be transferring between the processor and an I/O module.

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

iv. **Control:** An instruction may specify that the sequence of execution is altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction is 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.

An instruction's execution may involve a combination of these actions. 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, aup to 212 = 4096 (4K) words of memory can be directly addressed. Address 941 and stores the result in the latter location. Three instructions, which be described as three fetch and three execute cycles, are required:

1. The PC contains 300, the address of the first instruction. This instruction value is 1940 in hexadecimal) is loaded into the instruction register IR anPC is incremented. Note that this process involves the use of a memory dress register (MAR) and a memory buffer register (MBR). For simply these intermediate registers are ignored.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is loaded. The remaining 12 bits (three hexadecimal digits) specify the ac (940) from which data are to be loaded.
3. The next instruction (5941) is fetched from location 301 and incremented.
4. The old contents of the AC and the contents of location 941 are added an result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the F is incremented.
6. The contents of the AC are stored in location 941.

In this example, three instruction cycles, each consisting of a fetch cycle execute cycle, are needed to add the contents of location 940 to the contents C With a more complex set of instructions, fewer cycles would be needed. Some processors, for example, included instructions that contain more than one address. Thus the execution cycle for a particular instruction on such prop could involve more than one reference to memory. Also, instead of memory references, an instruction may specify an I/O operation. Figure 5 shows the characteristics of a hypothetical machine.

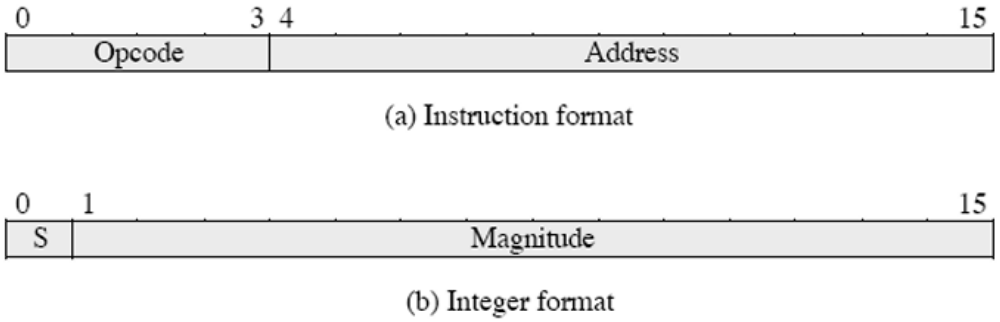


Figure 5. Characteristics of Hypothetical Machine

Program Counter (PC) = Address of Instruction
Instruction Register (IR) = Instruction begin 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

For example, the PDP-11 processor includes an instruction, expressed physically as ADD B, A, that stores the sum of the contents of memory location B into memory location A. A single instruction cycle with the following steps

- Fetch the ADD instruction.
- Read the contents of memory location A into the processor.
- Read the contents of memory location B into the processor. To contents of A are not lost, the processor must have at least two registers storing memory values, rather than a single accumulator.
- Add the two values
- Write the result from the processor to memory location A.

Thus, the execution cycle for a particular instruction may involve more than one reference to memory. Also, instead of memory references, an instructor specifies an I/O operation.

For any given instruction cycle, some states -null and others may be visited more than once. The states can be described as follows:

Instruction address calculation (ac): 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 ad- dress. If, instead, memory is organized as individually addressable 8-bit bytes, then add 2 to the previous address.

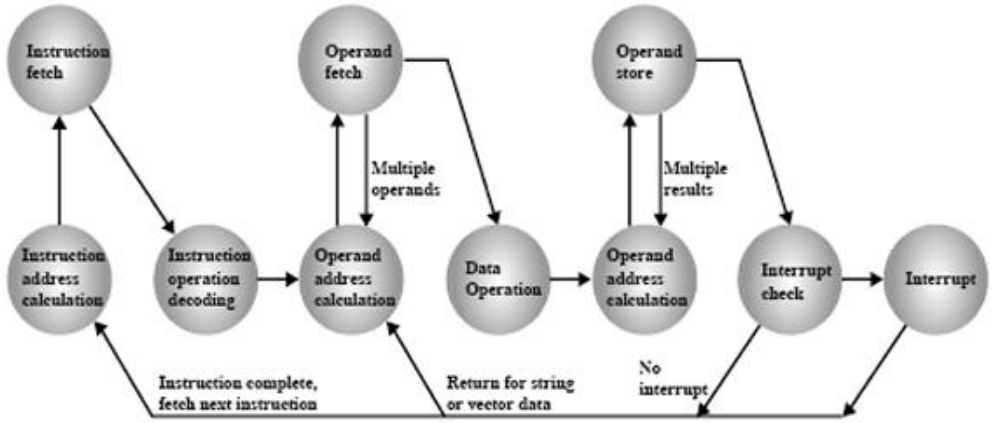


Figure 6. The Instruction Cycle State with Interrupts

Instruction fetch (if): Read instruction from its memory location into the processor.

Instruction operation decoding (iod): Analyze instruction to determine the 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 1/O. Data operation (do): Perform the operation indicated in the instruction. Operand store (os): Write the result into memory or out to I/O. States in the upper part of Figure 6 involve an exchange between the processor and either memory or a 1/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. Also note that the diagram allows for multiple operands and multiple results because some instructions on some machines require this. For example, the PDP-11 instruction ADD A, B results in the following sequence of states: iac, if, iod, oac, of, oac, of, do, oac, os.

Finally, on some machines, a single instruction can specify an operation to be performed on a vector (one-dimensional array) of numbers or a string (one- dimensional array) of characters. As Figure 6 indicates, this would involve repetitive operand fetch and/or store operations.

Table 1. Classes of Interrupts

Program	Generated by some conditions that occur 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 regularly.
I/O	Generated by an I/O controller, to signal normal completion of an operation to signal a variety of error conditions.
Hardware failure	Generated by a failure such as power failure or memory parity error.

Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What is the primary difference between computer organization and computer architecture?
   A. Organization deals with software while architecture deals with hardware
   B. Architecture refers to attributes visible to programmers while organization refers to operational units and their interconnections
   C. Organization is more important than architecture in system design
   D. There is no difference between the two terms

2. Which of the following is NOT one of the four basic functions of a computer?
   A. Data processing
   B. Data storage
   C. Data encryption
   D. Data movement

3. What are the main structural components of a computer system?
   A. CPU, Main memory, I/O, System interconnections
   B. Hardware, Software, Data, Procedures
   C. Input, Processing, Output, Storage
   D. Registers, ALU, Control Unit, Cache

Self-Assessment Exercises 2

Fill in the gaps in the sentences below with the most suitable words:

1. The _____ processing unit (CPU) controls the operations of the computer and performs its data processing functions.

2. The Von Neumann architecture is based on three key concepts, one of which is that data and _____ are stored in a single read-write memory.
3. The instruction cycle consists of two main steps: the _____ cycle and the _____ cycle.

4.0 CONCLUSION

Computer architecture and organization form the foundation of modern computing systems. Architecture defines what the system can do - the instruction set, data types, addressing modes, and interface specifications visible to programmers. Organization, on the other hand, determines how these architectural specifications are implemented through hardware components and their interconnections. The Von Neumann architecture remains the dominant model, with its key principles of stored program concept, sequential execution, and unified memory for instructions and data. Understanding the relationship between structure and function, along with the basic computer components (CPU, memory, I/O, and system interconnections), provides the essential knowledge needed to comprehend how modern computers operate and execute instructions.

5.0 SUMMARY

This unit introduced the fundamental concepts of computer architecture and organization. Computer architecture refers to the attributes of a system visible to programmers, including instruction sets, data types, and addressing mechanisms. Computer organization deals with the operational units and their interconnections that realize the architectural specifications. The four basic functions of a computer are data processing, data storage, data movement, and control. A computer system consists of four main structural components: the CPU (which includes the control unit, ALU, and registers), main memory, I/O systems, and system interconnections. The Von Neumann architecture, based on the stored program concept, sequential execution, and unified memory, forms the foundation of modern computer design. The instruction cycle, consisting of fetch and execute phases, describes how computers process individual instructions.

6.0 TUTOR-MARKED ASSIGNMENT

1. Explain the distinction between computer architecture and computer organization. Provide two examples of architectural attributes and two examples of organizational attributes. (10 marks)
2. Describe the four basic functions of a computer system and explain how these functions interact during program execution. (8 marks)

3. The Von Neumann architecture is fundamental to modern computer design. List and explain the three key concepts on which this architecture is based. Discuss one advantage and one limitation of this architectural approach. (12 marks)

Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. B
2. C
3. A

Self-Assessment Exercise 2
1. Central
2. Instructions
3. Fetch, execute

UNIT 2 INSTRUCTION SETS CHARACTERISTICS

CONTENTS

1.0 INTRODUCTION

One boundary where the computer designer and the computer programmer can view the same machine is the machine instruction set. From the designers' point of view, the machine instruction set provides the functional requirements for the processor. Implementing the processor is a task that largely involves implementing the machine instruction set.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- Explain the instruction format
- Understand the instruction length and characteristics

3.0 MAIN CONTENT

3.1 INSTRUCTION FORMATS

An instruction format defines the layout of the bits of an instruction in terms of its constituent fields. An instruction format must include an opcode and implicitly or explicitly, zero or more operands, and The format must implicitly and explicitly, indicate the addressing mode for each operand. For most instruction sets, more than one instruction format is used.

3.1.1 INSTRUCTION LENGTH

The most basic design issue to be faced is the instruction format length. These decisions effects and are affected by, memory size, memory organization bus structure process complexity, and processor speed. This decision determines the richness and flexibility of the machine.

3.2 INSTRUCTION SETS CHARACTERISTICS

The operation of the processor is determined by the instructions it executes referred to as machine instructions or computer instructions. The collection of different instructions that the processor can execute is referred to as the processor's instruction set.

3.2.1 ELEMENTS OF MACHINE INSTRUCTION

These elements are as follows:

- **Operation code:** Specifies the operation to be performed (e.g., ADD, I/O). The operation is specified by a binary code, known as the operation code or opcode.
- **Source operand reference:** This operation may involve one or more source operands, that is operands that are inputs for the operation
- **Results from operands reference:** The operation may produce a result
- **Next instruction reference:** This tells the processor where to fetch the next instruction after the execution of this instruction is complete.

The address of the next instruction to be fetched could be either a real address or a virtual address, depending on the architecture. Generally, the distinction is transparent to the instruction set architecture. In most cases, the next instruction to be fetched immediately follows the current instruction. In most cases, there is no explicit reference to the next instruction when an explicit reference is needed then the main memory or virtual memory address must be supplied. Source and result operands can be in one of four areas.

- **Main or virtual memory:** As with the next instruction references, the main or virtual memory address must be supplied.
- **Processor register:** With rare exception, a processor contains one or more registers that may be referenced by machine instructions. If only one register exits reference to it may be implicit. If more than one register exists, then each register is assigned a unique name or number, and the instruction must contain the number of the designed register
- **Immediate:** The value of the operand is contained in a field in the

instruction being executed.

- **I/O device:** The instruction must specify the I/O module and device for operation. If memory-mapped I/O is used, this is just another main or virtual memory address

Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. Which of the following is NOT an element of a machine instruction?
   A. Operation code
   B. Source operand reference
   C. Memory address register
   D. Next instruction reference

2. What does the opcode specify in an instruction?
   A. The memory location of data
   B. The operation to be performed
   C. The size of the operand
   D. The addressing mode

3. How many different opcodes can be represented with 4 bits?
   A. 4
   B. 8
   C. 12
   D. 16

3.2.2 INSTRUCTION REPRESENTATION

In a computer, each instruction is represented by a sequence of bits. The instruction is divided into fields corresponding to the constituent elements of the instruction. Opcodes are represented by abbreviations called mnemonics that indicate their operation. Common examples include:

ADD add
SUB SUBTRACT
MUL multiply
DIV divide
LOAD Load data form memory STOR Store data to memory

Operands are also represented in a symbolic manner. For example the instruction ADD, R, Y.

This may mean adding the value contained in data location Y to the contents of register R. In this example, Y refers to the address of a location in memory, and R refers to a particular register. Note that the operation is performed on the contents of a location, not on its address:

Thus, it is possible to write a machine-language program in symbolic form.

X= 413
Y= 414

A simple program accepts this symbolic input, converts opcodes and operand references to binary form, and constructs binary machine instructions. However, symbolic machine language remains a useful tool for describing machine instructions, and we will use it for that purpose.

Assume that variables X and Y correspond to locations 413 and 414, respectively. Assuming a simple set of machine instructions, this operation can be accomplished with three instructions.

1. Load a register with the content of memory location 413.
2. Add the contents of memory location 414 to the register.
3. Store the contents of the register in memory location 413.

3.3 INSTRUCTION SET DESIGN

One of the most interesting and most analyzed, aspects of computer design is instruction set is very complex because it affects so many aspects of the computer system. The instruction defines any of the functions performed by the processor and thus has a significant effect on the implementation of the process. The instruction set is the programmer's means of controlling the processor. Thus, programmer requirements must be considered in designing the instruction set. The most important of these fundamental design issues include the following:

- **Operation repertoire:** How many and which operations to provide and how complex operations should be.
- **Data types:** The various types of data upon which operations are performed.
- Instruction format: Instruction length (in nits) number of assesses size of various fields and so on.
- **Registers:** Number of processor registers that can be referenced by instructions and their use.
- **Addressing:** The mode or modes by which the address of an operand is specified.

These issues are highly interrelated and must be considered together in designing an instruction set.

4.0 CONCLUSION

Despite the variety and pace of change in the computer field, certain fundamental concept applies consistently throughout. The application of

these concepts depends on the current state of technology and the price/performance objectives of the designer.

5.0 SUMMARY

Computer organization refers to the operational units and their interconnections that realize the architectural specification.

Computer architecture refers to those attributes of a system visible to a programmer or those attributes that have a direct impact on the logical execution of a program. The collection of different instructions that the processor can execute is referred to as the processor's instruction set and an to instruction format defines the layout of the bits of instruction, in terms of its constituents' fields.

Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. C
2. B
3. D

6.0 TUTOR- MARKED ASSIGNMENT

1. What in general terms is the distinction between computer organization and computer architecture?
2. What are the four main functions of a computer?
3. List and briefly explain five important instruction set design issues

7.0 REFERENCES/ FURTHER READING

Hennessy, J. L., & Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Elsevier.

Patterson, D. A., Brooks Jr, F. P., Sutherland, I. E., & Thacker, C. P. (2011). *Computer architecture*. Elsevier Science.

Null, L. (2023). *Essentials of Computer Organization and Architecture*. Jones & Bartlett Learning.

Sloss, A; Symes, D; and Wright, C.ARM system developers guide an Fransisco Morgan Kaufmann, 2004

MODULE 2 COMPUTER ARITHMETIC

UNIT 1: The Arithmetic Implementation
UNIT 2: Control Flow Design/Implementation

UNIT 1 THE ARITHMETIC IMPLEMENTATION

1.0 INTRODUCTION

This unit focuses on the most complex aspect of the ALU, computer arithmetic. Computer arithmetic is commonly performed on two very different types of numbers: integer and floating point. In both cases, the representation chosen is a crucial design issue and is treated first.

Computer arithmetic is the branch of computer science that deals with the representation and manipulation of numerical quantities in a computer system. Here are some basic concepts and operations involved in computer arithmetic:

1. Number systems: Computers use different number systems to represent numerical quantities, including binary (base 2), decimal (base 10), and hexadecimal (base 16) systems. In binary system, each digit can only be either 0 or 1, while in decimal system, each digit can be any of the 10 digits from 0 to 9.
2. Arithmetic operations: The basic arithmetic operations used in computer arithmetic are addition, subtraction, multiplication, and division. These operations are usually performed using arithmetic circuits within the CPU.
3. Overflow: In computer arithmetic, overflow occurs when the result of an arithmetic operation is too large to be represented in the available number of bits. This can result in incorrect or unexpected results.
4. Floating-point arithmetic: Floating-point arithmetic is used to represent and perform operations on non-integer numbers. It involves representing a number as a combination of a mantissa (or significand) and an exponent.

5. Round-off errors: Round-off errors occur in floating-point arithmetic due to the limited precision of the number representation. This can result in small inaccuracies in the computed results.
6. Bitwise operations: Bitwise operations are used to manipulate individual bits in a number. The basic bitwise operations include AND, OR, XOR, and NOT.
7. Two's complement representation: Two's complement representation is a method of representing negative numbers in binary. In this representation, the most significant bit is used as a sign bit, with 0 indicating a positive number and 1 indicating a negative number.

Overall, computer arithmetic is a fundamental aspect of computer science and is used in a wide range of applications, including scientific computing, financial analysis, and digital signal processing.

2.0 OBJECTIVES

At the end of this unit, you should be able to
Recognize and explain the importance of various bases in computing.
Perform arithmetic operations with floating-point numbers.
Describe the fixed-point number representation and its applications.

3.1 THE ARITHMETIC AND LOGIC UNIT

The arithmetic and logic unit (ALU) is that part of the computer that performs arithmetic and logical operations on data. All of the other elements of the computer system- Control unit, registers memory, I/0- are there mainly to bring into the ALU for it to process and then take the result back out.

An ALU and all electronic components in the computers are based on the use of simple digital logic devices that can store binary digits and perform simple Boolean logic operations. Data are presented to the ALU in registers and the results of an operation are stored in registers. These registers are temporary storage locations within the processor that are connected by signal paths to the ALU. The ALU may also set flags as the result of an operation. For example, an overflow flag is set to 1 if the result of a computation exceeds the length of the register into which it is to be stored. The flag values are also stored in registers within the processor. The control unit provides signals that control the operation of the ALU and the movement of the data into and out of the ALU.

Representing and storing numbers were the basic operations of the computers of earlier times. The real go came when computation, manipulating numbers like adding and multiplying came into the picture. These operations are handled by the computer's **arithmetic logic unit**

(ALU). The ALU is the mathematical brain of a computer. The first ALU (Arithmetic Logic Unit) was indeed the INTEL 74181, which was implemented as part of the 7400 series TTL (Transistor-Transistor Logic) integrated circuits. It was released by Intel in 1970.
ALU is a digital circuit that provides arithmetic and logic operations. It is the fundamental building block of the central processing unit of a computer. A modern central processing unit(CPU) has a very powerful ALU and it is complex in design. In addition to ALU modern CPU contains a control unit and a set of registers. Most of the operations are performed by one or more ALUs, which load data from the input register. Registers are a small amount of storage available to the CPU. These registers can be accessed very fast. The control unit tells ALU what operation to perform on the available data. After calculation/manipulation, the ALU stores the output in an output register.

3.2 INTEGER REPRESENTATION

In the binary number, arbitrary numbers can be represented with just the digits zero and the minis sign, and the period or radix point.
$-1101.0101_{2} = -13.3125_{10}$
For purposes of computer storage and processing, however, we do not have the benefits of minus signs and periods. Only binary digits (0 and 1) may be used
to represent numbers. If we are limited to non-negative integers, the representation is straight forward.
An 8-bit word can represent the numbers from 0 to 255, including

00000000 = 0
00000001 = 1
00101001 = 41
10000000 = 128
11111111 = 255

In general, if an n-bit sequence of binary digits is interpreted as an unsigned integer, A it value is

$$A = \sum_{i=0}^{n-1} 2i\ ai$$

In going from the first to the second equation, we require that the least significant *n* - 1 bits do not change between the two representations. Then we get to next to the last equation, which is only true if all of the bits in positions theorem *2* are 1. Therefore, the sign-extension rule works.
Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What is the range of numbers that can be represented using 8-bit

unsigned binary?
A. 0 to 127
B. -128 to 127
C. 0 to 255
D. -255 to 255

2. In two's complement representation, what does the most significant bit represent?
A. The magnitude of the number
B. The sign of the number
C. The decimal point location
D. The base of the number system

3. What is the primary advantage of two's complement representation?
A. It uses less memory
B. It simplifies arithmetic operations
C. It allows larger numbers
D. It is easier to understand

Fixed-point representation

Finally, we mention that the representations discussed in this section are sometimes referred to as fixed points. This is because the radix point (binary point) is fixed and assumed to be to the right of the rightmost digit. The programmer can use the representation for binary fractions by scaling the numbers so that the binary poor implicitly positioned at some other location.

Negative Number Representation

Sign Magnitude

Sign magnitude is a very simple representation of negative numbers. In sign-magnitude, the first bit is dedicated to representing the sign and hence it is called the sign bit.

The sign bit ‘1’ represents a negative sign.

The sign bit ‘0’ represents a positive sign.

In the sign-magnitude representation of n-bit number, the first bit will represent the sign, and the rest n-1 bits represent the magnitude of the number.

For example,

+25 = 011001

Where 11001 = 25

And 0 for ‘+’

-25 = 111001

Where 11001 = 25

And 1 for ‘-‘.

Range of number represented by sign magnitude method = -(2n-1-1) to +(2n-1-1) (for n bit number)

But there is one problem in sign-magnitude and that is we have two

representations of 0
+0 = 000000
– 0 = 100000

2's complement method

To represent a negative number in this form, first we need to take the 1's complement of the number represented in simple positive binary form and then add 1 to it.

For example:
(-8)10 = (1000)2
1's complement of 1000 = 0111
Adding 1 to it, 0111 + 1 = 1000
So, $(-8)^{10} = (1000)^2$
Please don't get confused with $(8)^{10}$ =1000 and $(-8)^{10}$=1000 as with 4 bits, we can't represent a positive number more than 7. So, 1000 is representing -8 only.
Range of number represented by 2's complement = $(-2^{n-1}$ to $2^{n-1} – 1)$

Floating point representation of numbers

32-bit representation floating point numbers IEEE standard

Normalization

- Floating point numbers are usually normalized
- The exponent is adjusted so that the leading bit (MSB) of the mantissa is 1
- Since it is always 1 there is no need to store it
- Scientific notation where numbers are normalized to give a single digit before the decimal point like in a decimal system e.g. 3.123×10^3

Some insight into two complement addition and subtraction can be gained by looking at a geometric depiction. The circle in the upper half of each part of the figure is formed by selecting the appropriate segment of the number line and joining the endpoints. Note that when the numbers are laid out on a circle, the twos complement of any number are horizontally opposite that number (indicated by dashed horizontal lines). Starting at any number on the circle, we can add positive *k* (or subtract negative k), to that number by moving *k* positions clockwise, and we can subtract positive *k (of* add negative *k)* from that number by moving *k* positions counterclockwise. If an arithmetic operation results in traversal of the point where the endpoints are joined, an incorrect answer is given (overflow).

The central element is a binary adder, which presents two numbers for addition and produces a sum and an overflow indication. The binary adder treats the two numbers as unsigned integers. In addition, the two numbers are presented to the adder from two registers, designated in this case as A and B registers. The result may be stored in one of these registers or a third. The overflow indication is stored in a 1-bit overflow flag (0 = no overflow; I = overflow). For subtraction, the

4.0 CONCLUSION

Numbers are represented in binary form and the algorithms used for basic

arithmetic operators are add, subtract, multiply, and divide

5.0 SUMMARY

- An ALU and all electronic components in the digital logic devices that store binary digits and perform simple Boolean logic operations
- Overflow rule occurs when two numbers positive or negative numbers are added and the result of the addition has the opposite sign.
- Subtraction flow is to subtract one number (subtracted) from another (minuend) take the two compliments (negation) of the subtrahend and hold it to the minuend.

Floating point numbers are expressed as a number (significant) multiplied by a constant (base) raised to some integer power (exponent). It can be used to represent very large and very small numbers.

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1

1. C
2. B
3. B

6.0 TUTOR- MARKED ASSIGNMENT

1. What is a sign-extension rule for two compliment numbers?
2. Find the following differences using two complement arithmetic:

a. 1111011 b. 10101110 c. 111110010111
-100100 -111-1-1 -111010010101

7.0 Reference and further reading

Null, L. (2023). *Essentials of Computer Organization and Architecture*. Jones & Bartlett Learning.

Englander, I., & Wong, W. (2021). *The architecture of computer hardware, systems software, and networking: An information technology approach*. John Wiley & Sons.

Swartzlander, E. editor computer Arithimetic, volumes I and II. Los Alamitiss, CA IEEE Computer society press, 1990.

UNIT 2 CONTROL FLOW DESIGN/OPERATION

1.0 Introduction

The execution of an instruction involves the execution of a sequence of sub-steps, generally called cycles. For example, an execution may consist of fetch, indirect, execute, and interrupt cycles. Each cycle is in turn made up is a sequence of more fundamental operations called micro-operations. A single micro-operation generally involves transfer between registers a register and an external bus, or a simple ALU operation.

2.0 At the end of this unit, you should be able to

- Understand that each cycle is in turn made up of a sequence of more fundamental operations called micro-operations.
- Identify hardwired implementation
- Explain micro-programmed control

3.1 MICRO OPERATIONS

The prefix micro refers to the fact that each step is very simple and accomplishes very little. To design a control unit each of the smaller cycles involves a series of steps each of which involves the processor registers. We refer to these steps as micro-operations. Micro operations are the functional, or atomic operations of a processor.

Three. Now, we turn to the question of how these functions are performed or, more specifically, how the various elements of the processor are controlled to provide these functions. Thus, we turn to a discussion of the control unit, which controls the operation of the processor.

We have seen that the operation of a computer, in executing a program, consists of a sequence of instruction cycles, with one machine instruction per cycle. Of course, we must remember that this sequence of instruction cycles is not necessarily the same as the *written sequence* of instructions that make up the program, because of the existence of branching instructions. What we are referring to here is the execution *time sequence* of instructions.

We have further seen that each instruction cycle is made up of several smaller units. One subdivision that we found convenient is fetch, indirect,

execute, and interrupt, with only fetch and execute cycles always occurring.

To design a control unit, however, we need to break down the description further. In our discussion of pipelining in Chapter 12, we began to see that further decomposition is possible. We will see that each of the smaller cycles involves

a series of steps, each of which involves the processor registers. We will refer to these steps as micro-operations. The prefix *micro* refers to the fact that each step is very simple and accomplishes very little. Figure 15.1 depicts the relationship among the various concepts we have been discussing. To summarize, the execution of a program consists of the sequential execution of instructions. Each instruction is executed during an instruction cycle made up of shorter subcycles (e.g., fetch, indirect, execute, interrupt). The execution of each subcycle involves one or more shorter operations, that is, micro- operations.

Micro-operations are the functional, or atomic, operations of a processor. In this section, we will. examine micro-operations to gain an understanding of how the events of any instruction cycle can be described as a sequence of such m' operations. A simple example will be used. In the remainder of this chapter.

-then show how the concept of micro-operations serves as a guide to the design control unit. Figure 7 displayed the contituent element of a program execution.

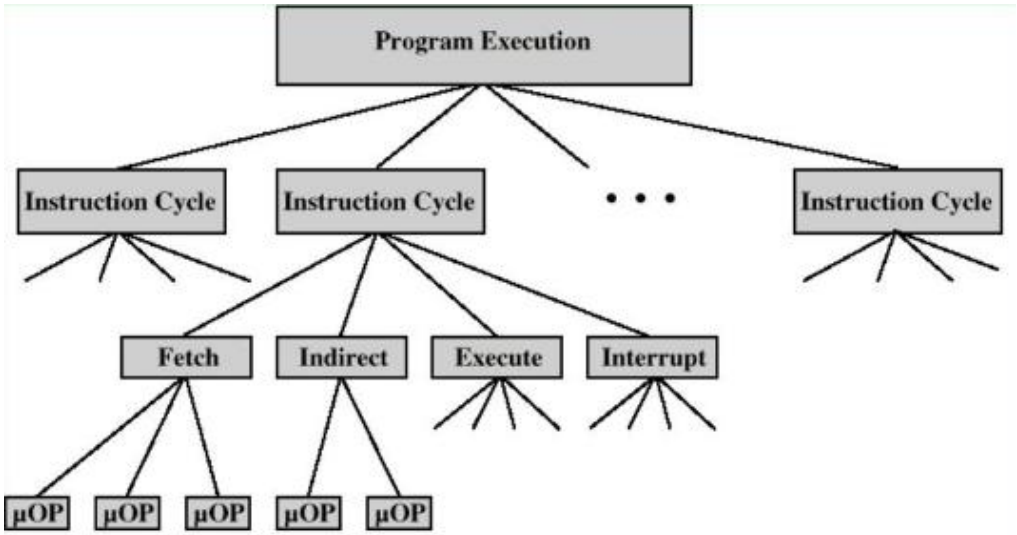


Figure 7. Contituent element of a program execution

'We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory.

Memory address register (MAR): Is connected to the address lines of the bus. It specifies the address in memory for a read or write operation.

Memory buffer register (MBR): Is connected to the data lines of the system --

_ It contains the value to be stored in memory or the last value read from melr

_ Program counter (PC): Holds the address of the next instruction to be fetched

Instruction register (IR): Holds the last instruction fetched.

Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers. An example appears in Figure 5 at the beginning of the fetch cycle, the address of the next instruction to be executed is in the program counter (PC); in this case, the address is 1100100. The first steto move that addresses to the memory address register

(MAR) because this is only registered and connected to the address lines of the system bus. The second step bring in the instruction. The desired address (in the MAR) is placed on the adder. We have seen that the operation of a computer, in executing a program, consists of a sequence of instruction cycles, with one machine instruction per cycle. Of course, we must remember that this sequence of instruction cycles is not necessarily the same as the *written sequence* of instructions that make up the program, because of the existence of branching instructions. What we are referring to here is the execution *time sequence* of instructions.

We have further seen that each instruction cycle is made up of several smaller units. One subdivision that we found convenient is fetch, indirect, execute, and interrupt, with only fetch and execute cycles always occurring.

To design a control unit, however, we need to break down the description further. We will see that each of the smaller cycles involves a series of steps, each of which involves the processor registers. We will refer to these steps as micro-operations. The prefix *micro* refers to the fact that each step is very simple and accomplishes very little. Figure 15.1 depicts the relationship among the various concepts we have been discussing. To summarize, the execution of a program consists of the sequential execution of instructions. Each instruction is executed during an instruction cycle made up of shorter subcycles (e.g., fetch, indirect, execute, interrupt). The execution of each subcycle involves one or more shorter operations, that is, micro-operations.

Micro-operations are the functional, or atomic, operations of a processor. bus, the control unit issues a READ command on the control bus, and the result appears on the data bus and is copied into the memory buffer register (MBR). We also need to increment the PC by the instruction length to get ready for the next instruction. Because these two actions (read word from memory, increment PC) do not interfere with each other, we can do them simultaneously to save time. The third step is to move the contents of the MBR to the instruction register (IR). This frees up the MBR for use during a possible indirect cycle.

Thus, the simple fetch cycle consists of three steps and four micro-operations. Each micro-operation involves the movement of data into or out of a register. So long as these movements do not interfere with one another, several of them can take place during one step, saving time. Symbolically, we can write this sequence of events as follows:
t_1: MAR E- (PC) t2: MBR <-- Memory PC <- (PC) + I t3: IR <-- (MBR)
where I is the instruction length. We need to make several comments about this sequence. We assume that a clock is available for timing purposes and that it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit. The notation (t_i, t_2, t_3) represents successive time units. In words, we have
I First-time unit: Move contents of PC to MAR.
Second-time unit: Move contents of the memory location specified by MAR to MBR. Increment by I the contents of the PC.
Third-time unit: Move contents of MBR to IR.
Note that the second and third micro-operations both take place during the second time unit. The third micro-operation could have been grouped with the fourth without affecting the fetch operation:
t_1: MAR <- (PC) t_2: MBR <- Memory t3: PC E- (PC) + I IR <- (MBR)
The groupings of micro-operations must follow two simple rules:
The proper sequ**en**ce of events must be followed. Thus (MAR - (PC)) must precede (MBR - Memory) because the memory read operation makes use of the address in the MAR.
Conflicts must be avoided. One should not attempt to read to and write from the same register in a one-time unit, because the results would be unpredictable. For example, the micro-operations (MBR ¢-- Memory) and (IR <- MBR) should not occur during the same time unit.
A final point worth noting is that one of the micro-operations involves an addition. To avoid duplication of circuitry, this addition could be performed by the ALU. The use of the ALU may involve additional micro-operations, depending on the functionality of the ALU and the organization of the processor. Whereas micro-operations are ignored in that figure, this discussion shows the micro-operations needed to perform the sub-cycles of the instruction cycle.
Once an instruction is fetched, the next step is to fetch source operands. Continuing our simple example, let us assume a one-address instruction format, with direct and indirect addressing allowed. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle.

$$t_1: MAR\ <-\ (IR(Address))$$
$$t_2: MBR\ F-\ Memory$$
$$t_3: IR(Address)\ F-\ (MBR(Address))$$

The address field of the instruction is transferred to the MAR. This is then used to fetch the address of the operand. Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than

an indirect address.
The IR is now in the same state as if indirect addressing had not been used and it is ready for the execution cycle. We skip that cycle for a moment, to consider t interrupt cycle.
After the execute cycle, a test is made to determine whether any-:-_abled interrupts have occurred. If so, the interrupt cycle occurs. The nature of the cycle varies greatly from one machine to another. We present a very simple sequeof events, as illustrated in Figure 12.8. We have
t_1: MBR E- (PC)
t_2: MAR F- Save Address PC F- Routine Address
t_3: Memory E- (MBR)
In the first step, the contents of the PC are transferred to the MBR, so that u- can be saved for return from the interrupt. Then the MAR is loaded with the add- .at which the contents of the PC are to be saved, and the PC is loaded with the add to the MAR and PC, respectively. In any case, once this is done, the final step is to store the MBR, which contains the old value of the PC, in memory. The processor is now ready to begin the next instruction cycle.
The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-operations and, in each case, the same micro-operations are repeated each time around.
This is not true of the execution cycle. Because of the variety of opcodes, there are several different sequences of micro-operations that can occur. Let us consider several hypothetical examples.
First, consider an add instruction:
ADD R1, X
which adds the contents of location X to register R1. The following sequence of micro-operations might occur:
We begin with the IR containing the ADD instruction. In the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory
location is read. Finally, the contents of RI and MBR are added by the ALLT.
Again. This is a simplified example. Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALt inputs or outputs in some intermediate registers.
Let us look at two more complex examples. A common instruction is increment and skip if zero:
The content of location X is incremented by l. If the result is 0, the next instruction is skipped. A possible sequence of micro-operations is
ti: MAR <-- (IR(address)) t2: MBR- F- Memory
tz : MBR <-- (MBR) + 1
tu: Memory <- (MBR)
If ((MBR) = 0) then (PC F - (PC) + I)
The new feature introduced here is the conditional action. The PC is incremented if (MBR) = 0. This test and action can be implemented as

one micro-operation. Note also that this micro-operation can be performed during the same time unit during which the updated value in MBR is stored back in memory.
It is worth pondering the minimal nature of the control unit. The control unit is the engine that runs the entire computer. It does this based only on knowing the instructions to be executed and the nature of the results of arithmetic and logical operations (e.g., positive, overflow, etc.). It never gets to see the data being processed or the actual results produced. It controls everything with a few control signals to points within the processor and a few control signals to the system bus.
Self-Assessment Exercises 1

Fill in the gaps in the sentences below with the most suitable words:

1. _____ are the functional, or atomic, operations of a processor.

2. The fetch cycle consists of _____ steps and _____ micro-operations.

3. A _____ control unit uses fixed logic circuits while a _____ control unit stores control signals in memory.

INTERNAL PROCESSOR ORGANIZATION

Figure 8 indicates the use of a variety of data paths. The complexity of this type of organization should be clear. Using an internal processor bus, Figure 8 can be rearranged. A single internal bus connects the ALU and all processor registers.
CPU with Internal Bus.

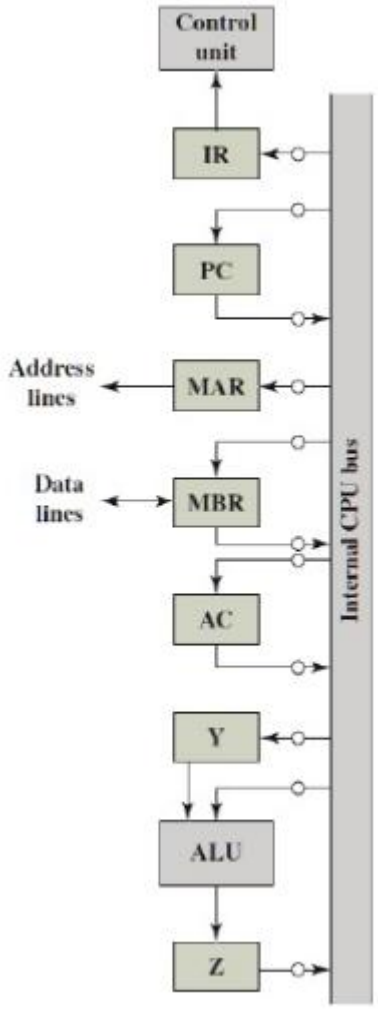


Figure 8. CPU with internal bus

Gates and control signals are provided for the movement of data onto and off the bus from each register. Additional control signals control data transfer to and from the system (external) bus and the operation of the ALU.

Two new registers, labeled Y and Z, have been added to the organization. These are needed for the proper operation of the ALU. When an operation involving two operands is performed, one can be obtained from the internal bus, but the other must be obtained from another source. The AC could be used for this purpose, but this limits the flexibility of the system and would not work with a processor with multiple general-purpose registers. Register Y provides temporary storage for the other input. The ALU is a combinatorial circuit with no internal storage. Thus, when control signals activate an ALU function, the input to the ALU is transformed into the output. Thus, the output of the ALU cannot be directly connected to the bus, because this output would feed back to the input. Register Z provides temporary output storage. With this arrangement, an operation to add a value from memory to the AC would

have the following steps:
t_1: MAR <- (IR(address))
t_2: MBR E- Memory
t_3: Y <-(MBR)
t4: Z f- (AC) + (Y)
t_s: AC F- (Z)
Other organizations are possible, but, in general, some sort of internal bus or set of internal buses is used. The use of common data paths simplifies the interconnection layout and the control of the processor. Another practical reason for the use of an internal bus is to save space.
To illustrate some of the concepts introduced thus far in this chapter, let us consider the Intel 8085. Its organization is shown in Figure 9. Several key components that may not be self-explanatory are:
Incrementer/decrementer address latch: Logic that can add 1 to or subtract 1 from the contents of the stack pointer or program counter. This saves time by avoiding the use of the ALU for this purpose.
Interrupt control: This module handles multiple levels of interrupt signals.
Serial I/O control: This module interfaces to devices that communicate 1 bit at a time. These signals are the interface between the 8085 processor and the rest of the system (Figure 10).

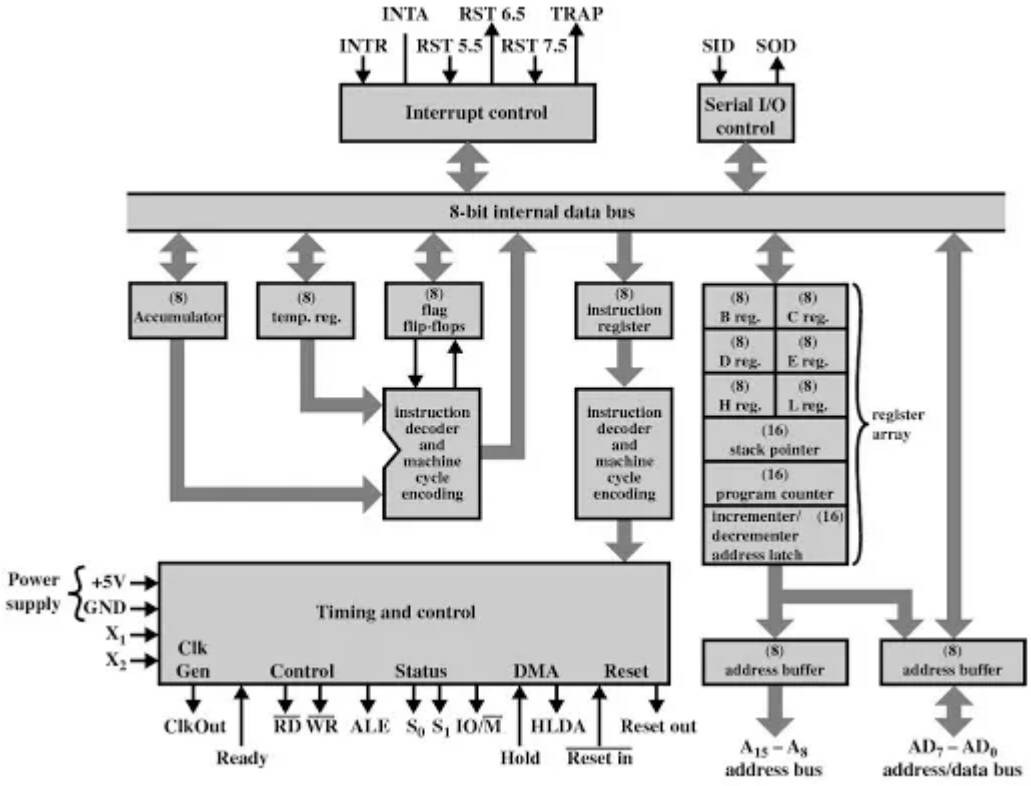


Figure 9. Intel 8085 CPU Block Diagram

The control unit is identified as having two components labeled (1) in decoder and machine cycle encoding and (2) in timing and control. The timing of processor operations is synchronized by the clock trolled by the control unit with control signals. Each instruction cycle i, into from one to five *machine cycles;* each machine cycle is in turn diN from three to five states. Each state lasts one clock cycle. During a state. The son performs one or a set of simultaneous micro-operations as determined control signals.

The number of machine cycles is fixed for a given instruction but one instruction to another. Machine cycles are defined to be equivalent cesses. Thus, the number of machine cycles for an instruction depends on a bar of times the processor must communicate with external devices. For e an instruction consists of two 8-bit portions, and then two machine, cycles fetch the instruction. If that instruction involves a 1-byte memory or 1/0 then a third machine cycle is required for execution.

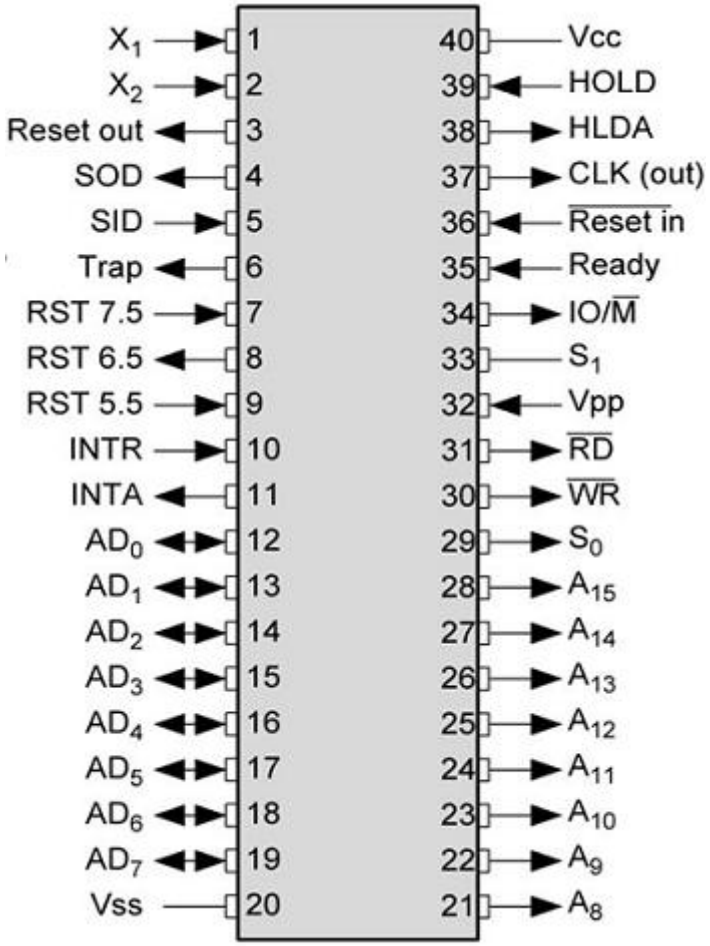


Figure 10. Inter 8085 pin configuration

Figure 11 gives an example of 8085 timing, showing the value of external control signals. Of course, at the same time, the control unit generates

internal control signals that control internal data transfers. The diagram shows the instruction cycle for an OUT instruction. Three machine cycles (Ml, M2, M3) are needed. During the first, the OUT instruction is fetched. The second machine cycle fetches the second half of the instruction, which contains the number of the 1/O device selected for output. During the third cycle, the contents of the AC are written out to the selected device over the data bus.

The Address Latch Enabled (ALE) pulse signals the start of each machine cycle from the control unit. The ALE pulse alerts external circuits. During timing state T_1 of machine cycle M_r, the control unit sets the IO/M signal to indicate that this is a memory operation. Also, the control unit causes the contents of the PC to be placed on the

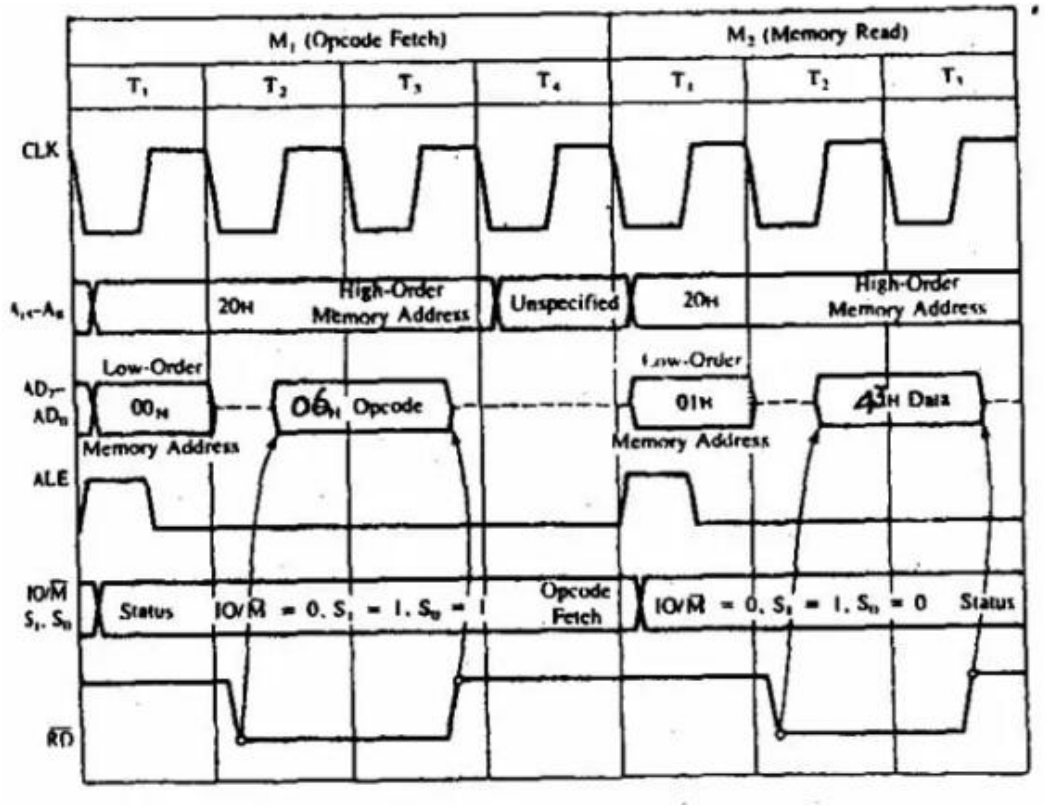


Figure 11. The timing diagram for inter 8085 out instruction

The timing diagram for inter 8085 out instruction addressed memory module places the contents of the addressed memory vocation on the address/data bus. The control unit sets the Read Control (RD) signal to indicate a read, but it waits until T_3 to copy the data from the bus. This gives the memory module time to put the data on the bus and for the signal levels to stabilize. The final state, T_4, is a bus *idle* state during which the processor decodes the instruction. The remaining machine cycles proceed similarly.

Finally, consider a subroutine call instruction. As an example, consider a branch and-save-address instruction:

BSA X
The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be used for return. This is a straightforward technique for providing subroutine calls. The following micro-operations suffice:
t_1: MAR E- (IR(address)) MBR ~ (PC)
t_2: PC <-- (IR(address)) Memory <-- MBR)
t_3: PC <- (PC) + I
The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in the IR. The later address is also incremented to provide the address of the instruction for the next - instruction cycle.
We have seen that each phase of the instruction cycle can be decomposed into a sequence of elementary micro-operations. In our example, there is one sequence eac= for the fetch, indirect, and interrupt cycles, and, for the execute cycle, there is one sequence of micro-operations for each opcode. To complete the picture, we need to tie sequences of micro-operations together, and this is done in Figure 15.3. We assume a new 2-bit register called the *instruction cycle code (ICC).* The ICC designates the state of the processor in terms of which portion of the cycle it is in:
00: Fetch 01: Indirect
10: Execute 11:
Interrupt
At the end of each of the four cycles, the ICC is set appropriately. The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle. For both the fetch and execute cycles, the next cycle depends on the state of the system.
Of course, this is a simplified example. The flowchart for an actual processor would be more complex. In any case, we have reached the point in our discussion in which the operation of the processor is defined as the performance of a sequence of micro-operations. We can now consider how the control unit causes this sequence to occur of tbp r of the interrupt-processing routine. These two actions may each be single micro-operation. However, because most processors provide multiple tyr and/or levels of interrupts, it may take one or more additional micro-operations to obtain the Save Address and the Routine Address before they can be transfer the events of any instruction cycle can be described as a sequence of such micro operations. A simple example will be used. In the remainder of this chapter, we then show how the concept of micro- operations serves as a guide to the design of the control unit.
THE FETCH CYCLE
We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. Four registers are involved:

- **Memory address register (MAR):** Is connected to the address lines of the system bus. It specifies the address in memory for a read or

write operation.

- **Memory buffer register (MBR):** Is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from memory.
- **Program counter (PC):** Holds the address of the next instruction to be fetched.
- **Instruction register (IR):** Holds the last instruction fetched.

Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers. At the beginning of the fetch cycle, the address of the next instruction to be executed is in the program counter (PC); in this case, the address is 1100100. The first step is to move that address to the memory address register (MAR) because this is the only register connected to the address lines of the system bus. The second step is to bring in the instruction. The desired address (in the MAR) is placed on the bus, the control unit issues a READ command on the control bus, and the result appears on the data bus and is copied into the memory buffer register (MBR). We also need to increment the PC by the instruction length to get ready for the next instruction. Because these two actions (read word from memory, increment PC) do not interfere with each other, we can do them simultaneously to save time. The third step is to move the contents of the MBR to the instruction register (IR). This frees up the MBR for use during a possible indirect cycle.

Thus, the simple fetch cycle consists of three steps and four micro-operations. Each micro-operation involves the movement of data into or out of a register. So long as these movements do not interfere with one another, several of them can take place during one step, saving time. Symbolically, we can write this sequence of events as follows:

where I is the instruction length. We need to make several comments about this sequence. We assume that a clock is available for timing purposes and that it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit. The notation (t1, t2, t3) represents successive time units. In words, we have

First-time unit: Move contents of PC to MAR.

- **Second-time unit:** Move contents of the memory location specified by MAR to MBR. Increment by I the contents of the PC.
- **Third-time unit:** Move contents of MBR to IR.

Note that the second and third micro-operations both take place during the second time unit. The third micro-operation could have been grouped with the fourth without affecting the fetch operation:

The groupings of micro-operations must follow two simple rules:

The proper sequence of events must be followed. Thus (MAR - (PC)) must precede (MBR - Memory) because the memory read operation makes use of the address in the MAR. Conflicts must be avoided. One should not attempt to read to and write from the same register in a one-time unit, because the results would be unpredictable. For example, the

micro-operations (MBR Memory) and (IR E- MBR) should not occur during the same time unit.

A final point worth noting is that one of the micro-operations involves an addition. To avoid duplication of circuitry, this addition could be performed by the ALU. The use of the ALU may involve additional micro-operations, depending on the functionality of the ALU and the organization of the processor.

Whereas micro-operations are ignored in that figure, this discussion shows the micro-operations needed to perform the subcycles of the instruction cycle.

Once an instruction is fetched, the next step is to fetch source operands. Continuing our simple example, let us assume a one-address instruction format, with direct and indirect addressing allowed. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle.

The address field of the instruction is transferred to the MAR. This is then used to fetch the address of the operand. Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address.

The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle. We skip that cycle for a moment, to consider the interrupt cycle.

After the execute cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycle varies greatly from one machine to another. We have

t_I: MBR <-- (PC)

t_2: MAR <-- Save Address PC <-- Routine ddress

t_3: Memory <-- (MBR)

In the first step, the contents of the PC are transferred to the MBR, so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may each be a single micro-operation. However, because most processors provide multiple types and/or levels of interrupts, it may take one or more additional micro- operations to obtain the Save Address and the Routine Address before they can be transferred to the MAR and PC, respectively. In any case, once this is done, the final step is to store the MBR, which contains the old value of the PC, in memory. The processor is now ready to begin the next instruction cycle.

The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-operations and, in each case, the same micro-operations are repeated each time around.

This is not true of the execute cycle. Because of the variety of opcodes, there are various sequences of micro-operations that can occur. Let us consider several hypothetical examples.

First, consider an add instruction:
which adds the contents of location X to register R1. The following sequence of micro-operations might occur:
t_1: MAR <-- (IR(address))
t_2: MBR <-- Memory
t_3: R1 ~- (R1) + (MBR)
We begin with the IR containing the ADD instruction. In the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory location is read. Finally, the contents of R1 and MBR are added by the ALU. Again, this is a simplified example. Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers.
Let us look at two more complex examples. A common instruction is increment and skip if zero:
The content of location X is incremented by 1. If the result is 0, the next instruction is skipped. A possible sequence of micro-operations is
The new feature introduced here is the conditional action. The PC is incremented if (MBR) = 0. This test and action can be implemented as one micro-operation. Note also that this micro-operation can be performed during the same time unit during which the updated value in MBR is stored back in memory.

Finally, consider a subroutine call instruction. As an example, consider a branch and-save-address instruction:
BSA X
The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be used for return. This is a straightforward technique for providing subroutine calls. The following micro-operations suffice:
t 1 : MAR <-- (IR(address)) MBR <-- (PC)
t_z: PC ~_____ (IR(address)) Memory - (MBR)
t 3 : PC ~_ (PC) + I
The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in the IR. The latter address is also incremented to provide the address of the instruction for the next instruction cycle.

Self-Assessment Exercises 1

Fill in the gaps in the sentences below with the most suitable words:

1. _____ are the functional, or atomic, operations of a processor.

2. The fetch cycle consists of _____ steps and _____ micro-operations.

3. A _____ control unit uses fixed logic circuits while a _____ control unit stores control signals in memory.

THE INSTRUCTION CYCLE

We have seen that each phase of the instruction cycle can be decomposed into a sequence of elementary micro-operations. In our example, there is one sequence each for the fetch, indirect, and interrupt cycles, and, for the execute cycle, there is one sequence of micro-operations for each opcode. We assume a new 2-bit register called the *instruction cycle code (ICC).* The ICC designates the state of the processor in terms of which portion of the cycle it is in:

00: Fetch
01: Indirect
10: Execute
11: Interrupt

At the end of each of the four cycles, the ICC is set appropriately. The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle. For both the fetch and execute cycles, the next cycle depends on the state of the system.

Of course, this is a simplified example. The flowchart for an actual processor would be more complex. In any case, we have reached the point in our discussion in which the operation of the processor is defined as the performance of a sequence of micro-operations. We can now consider how the control unit causes this sequence to occur.

3.2 CONTROL OF THE PROCESSOR

As a result of our analysis in the preceding section, we have decomposed the behavior or functioning of the processor into elementary operations, called micro-operations. By reducing the operation of the processor to its most fundamental level, we can define exactly what it is that the control unit must cause to happen. Thus, we can define the *functional requirements* for the control unit: those functions that the control unit must perform. A definition of these functional requirements is the basis for the design and implementation of the control unit.

With the information at hand, the following three-step process leads to a characterization of the control unit:

1. Define the basic elements of the processor.
2. Describe the micro-operations that the processor performs.
3. Determine the functions that the control unit must perform to cause the micro-operations to be performed.

We have already performed steps 1 and 2. Let us summarize the results. First, the basic functional elements of the processor are the following:

- ALU
- Registers
- Internal data paths External data paths

- Control unit

Some thought should convince you that this is i complete list. The ALU is the functional essence of the computer. Registers are used to store data internally on the processor. Some registers contain status information needed to manage instruction sequencing (e.g., a program status word). Others contain data that go to or comes from the ALU, memory, and I/O modules. Internal data paths are used to move data between registers and between registers and ALU. External data paths link registers to memory and 1/O modules, often utilizing a system bus. The control unit causes operations to happen within the processor.

The execution of a program consists of operations involving these processor elements. As we have seen, these operations consist of a sequence of micro-operations. micro- operations fall into one of the following categories:

Transfer data from one register to another.

Transfer data from a register to an external interface (e.g., system bus).

Transfer data from an external interface to a register.

Perform an arithmetic or logic operation, using registers for input and output.

All of the micro-operations needed to perform one instruction cycle, including all of the micro-operations to execute every instruction in the instruction set, fall into one of these categories.

We can now be somewhat more explicit about how the control unit functions. The control unit performs two basic tasks:

➢ **Sequencing:** The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.

➢ **Execution:** The control unit causes each micro-operation to be performed.

The preceding is a functional description of what the control unit does. The key to how the control unit operates is the use of control signals.

Controls Signals

We have defined the elements that make up the processor (ALU, registers, data paths) and the micro-operations that are performed. For the control unit to perform its function, it must have inputs that allow it to determine the state of the system and outputs that allow it to control the behavior of the system. These are the external specifications of the control unit. Internally, the control unit must have the logic required to perform its sequencing and execution functions. The remainder of this section is concerned with the interaction between the control unit and the other elements of the processor.

The inputs are

✓ **Clock:** This is how the control unit "keeps time." The control unit causes one micro-operation (or a set of simultaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time.

✓ **Instruction registers:** The opcode and addressing mode of the current instruction are used to determine which micro-operations to perform during the execute cycle.

✓ **Flags**: These are needed by the control unit to determine the status of the processor and the outcome of previous ALU operations. For example, for the increment-and-skip-if-zero (ISZ) instruction, the control unit will increase the PC if the zero flag is set.

Control signals from the control bus: The control bus portion of the system bus provides signals to the control unit.

The outputs are as follows:

✓ Control signals within the processor: There are two types: those that cause data to be moved from one register to another, and those that activate specific ALU functions.

- Control signals to control bus: These are also of two types: control signals to memory, and control signals to the I/O modules.

Three types of control signals are used: those that activate an ALU function, those that activate a data path, and those that are signals on the external system bus or other external interface. All of these signals are ultimately applied directly as binary inputs to individual logic gates.

Let us consider again the fetch cycle to see how the control unit maintains control. The control unit keeps track of where it is in the instruction cycle. At a given point, it knows that the fetch cycle is to be performed next. The first step is to transfer the contents of the PC to the MAR. The control unit does this by activating the control signal that opens the gates between the bits of the PC and the bits of the MAR. The next step is to read a word from memory into the MBR and increment the PC. The control unit does this by sending the following control signals simultaneously:

A control signal that opens gates, allowing the contents of the MAR onto the address bus A memory read control signal on the control bus

A control signal opens the gates, allowing the contents of the data bus to be stored in the MBR

Control signals to logic that add 1 to the contents of the PC and store the result back to the PC.

Following this, the control unit sends a control signal that opens gates between the MBR and the IR.

This completes the fetch cycle except for one thing: The control unit must decide whether to perform an indirect cycle or an execute cycle next. To decide this, it examines the IR to see if an indirect memory reference is made.

The indirect and interrupt cycles work similarly. For the execute cycle, the control unit begins by examining the opcode and based on that, decides which sequence of micro-operations to perform for the execute cycle.

To illustrate the functioning of the control unit, let us examine a simple example. Figure

12 illustrates the example. This is a simple processor with a single

accumulator

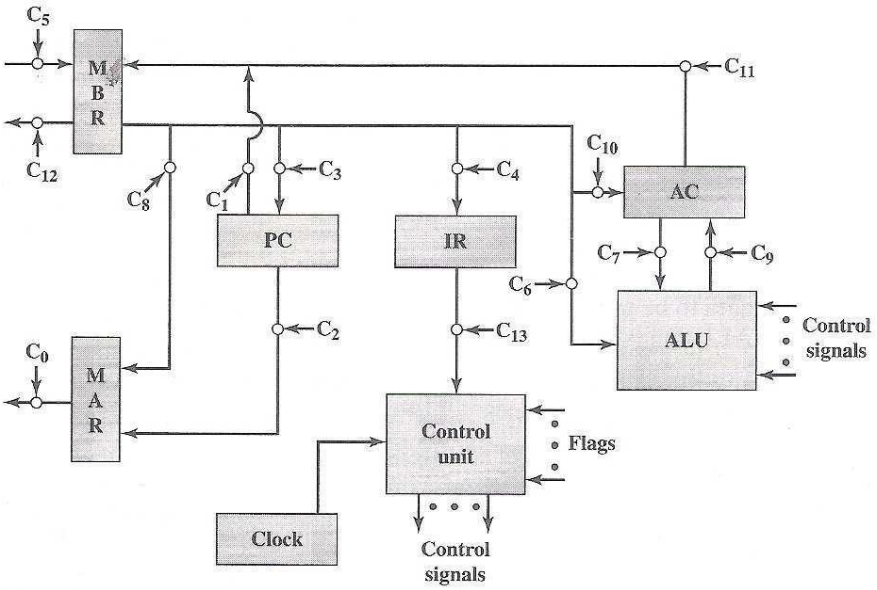


Figure 12. Data paths and control signals

(AC). The data paths between elements are indicated. The control paths for signals emanating from the control unit are not shown, but the terminations of control signals are labeled C_i and indicated by a circle. The control unit receives inputs from the clock, the instruction registers, and flags. With each **dock** cycle, the control unit reads all of its inputs and emits a set of control signals. Control signals go to three separate destinations:

Data paths: The control unit controls the internal flow of data. For example, on instruction fetch, the contents of the memory buffer register are transferred to the instruction register. For each path to be controlled, there is a switch (indicated by a circle in the figure). A control signal from the control unit temporarily opens the gate to let data pass.

ALU: The control unit controls the operation of the ALU by a set of control signals. These signals activate various logic circuits and gates within the ALU.

System bus: The control unit sends control signals out onto the control lines of the system bus (e.g., memory READ).

The control unit must maintain knowledge of where it is in the instruction cycle. Using this knowledge, and by reading all of its inputs, the control unit emits a sequence of control signals that cause micro-operations to occur. It uses the clock pulses to time the sequence of events, allowing time between events for signal levels to stabilize. For simplicity, the data and control paths for incrementing the PC and for loading the fixed addresses into the PC and MAR are not shown.

t is worth pondering the minimal nature of the control unit. The control

unit is the engine that runs the entire computer. It does this based only on knowing the instructions to be executed and the nature of the results of arithmetic and logical operations (e.g., positive, overflow, etc.). It never gets to specify the data being processed or the actual results produced. It controls everything with a few control signals to points within the processor and a few control signals to the system bus.

Figure 12 indicates the use of a variety of data paths. The complexity of this type of organization should be clear. Gates and control signals are provided for the movement of data onto and off the bus from each register. Additional control signals control data transfer to and from the system (external) bus and the operation of the ALU.

Two new registers, labeled Y and Z, have been added to the organization. These are needed for the proper operation of the ALU. When an operation involving two operands is performed, one can be obtained from the internal bus, but the other must be obtained from another source. The AC could be used for this purpose, but this limits the flexibility of the system and would not work with a processor with multiple general-purpose registers. Register Y provides temporary storage for the other input. The ALU is a combinatorial circuit with no internal storage. Thus, when control signals activate an ALU function, the input to the ALU is transformed into the output. Thus, the output of the ALU cannot be directly connected to the bus, because this

the output would feed back to the input. Register Z provides temporary output storage. With this arrangement, an operation to add a value from memory to the AC would have the following steps:

t_1: MAR ⟵ (IR (address))
t_2: MBR ⟵ Memory
t_3: Y ⟵ (MBR)
t_4: Z ⟵ (AC) + (Y)
t_5: AC ⟵ (Z)

Other organizations are possible, but, in general, some sort of internal bus or set of internal buses is used. The use of common data paths simplifies the interconnection layout and the control of the processor. Another practical reason for the use of an internal bus is to save space.

To illustrate some of the concepts introduced thus far in this unit, let us consider the Intel 8085. Its organization is shown in Figure 11. Several key components that may not be self-explanatory are:

- **Incremental decrementer address latch:** Logic that can add 1 to or subtract 1 from the contents of the stack pointer or program counter. This saves time by avoiding the use of the ALU for this purpose.
- **Interrupt control:** This module handles multiple levels of

interruption signals.

❖ **Serial UO control:** This module interfaces to devices that communicate 1 bit at a time.

Table 15.2 describes the external signals into and out of the 8085. These are linked to the external system bus. These signals are the interface between the 8085 processor and the rest of the system (Figure 12).

The control unit is identified as having two components labeled (1) instruction decoder and machine cycle encoding and (2) timing and control. A discussion of the first component is deferred until the next section. The essence of the control unit is the timing and control module. This module includes a clock and accepts as inputs the current instruction and some external control signals. Its output consists of control signals to the other components of the processor plus control signals to the external system bus.

The timing of processor operations is synchronized by the clock and controlled by the control unit with control signals. Each instruction cycle is divided into one to five *machine* cycles; each machine cycle is in turn divided into three to five *states.* Each state lasts one clock cycle. During a state, the processor performs one or a set of simultaneous micro-operations as determined by the control signals.

The number of machine cycles is fixed for a given instruction but varies from one instruction to accesses. Thus, the number of machine cycles for an instruction depends on t- lie number of times the processor must communicate with external devices. For example, if an instruction consists of two 8-bit portions, then two machine cycles are required to fetch the instruction. If that instruction involves a 1-byte memory or I/O operation, then a third machine cycle is required for execution.

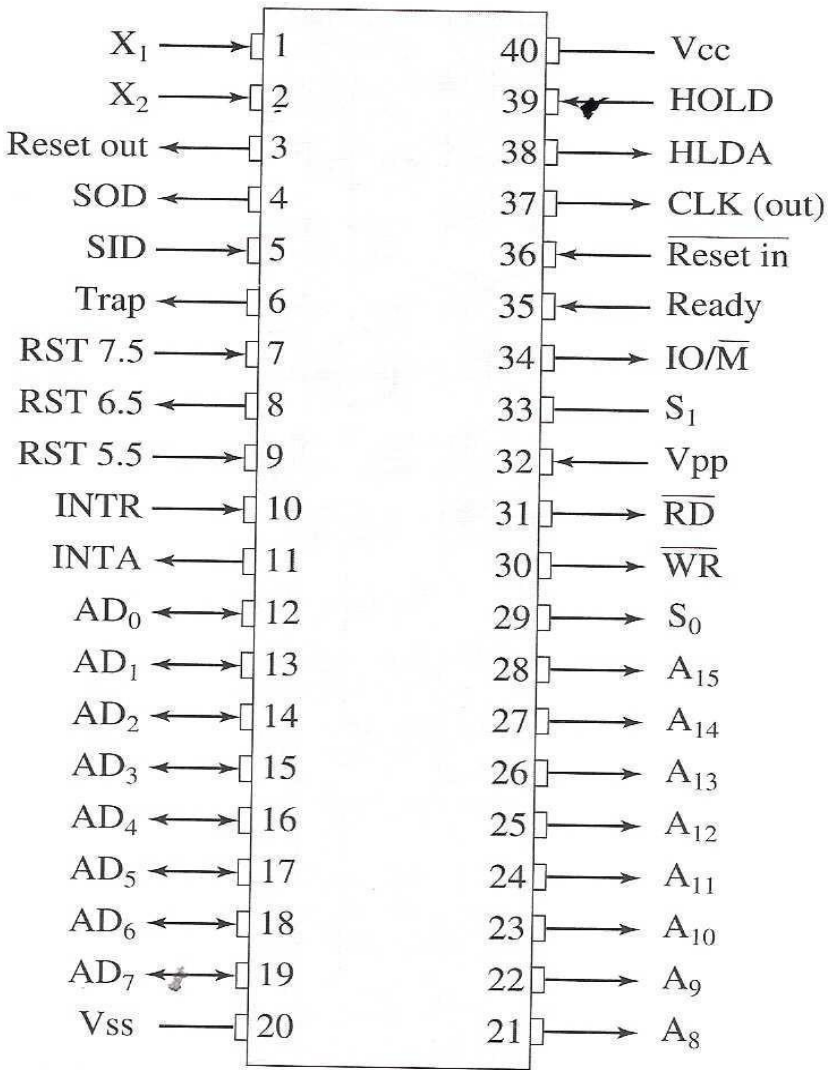


Figure 14. Intel 8085 External Control

Figure 14 gives an example of 8085 timing, showing the value of external control signals. Of course, at the same time, the control unit generates internal control signals that control internal data transfers. The diagram shows the instruction cycle for an OUT instruction. Three machine cycles (M_1, M_Z, M_3) are needed. During the first, the OUT instruction is fetched. The second machine cycle fetches the second half of the instruction, which contains the number of the I/O device selected for output. During the third cycle, the contents of the AC are written out to the selected device over the data bus.

Pulse signals the start of each machine cycle from the control unit. The ALE pulse alerts external circuits. During timing state T_l of machine cycle M_l, the control unit sets the IO/M signal to indicate that this is a memory operation. Also, the control unit causes the contents of the PC to be placed on the address bus (A_{ls} through A_s) and the address/data bus (AD_S through AD_O). With the falling edge of the ALE pulse, the other modules on the bus store the address. During timing state T_2, the addressed memory mole

places the contents of the addressed memory location on the address/data bus. The control unit sets the Read Control (RD) signal to indicate a read, but it waits until T_3 to copy the data from the bus. This gives the memory module time to put the data on the bus and for the signal levels to stabilize. The final state, T_4, is a bus *idle* state during which the processor decodes the instruction. The remaining machine cycles proceed similarly.

3.3 HARDWIRED CONTROL/ IMPLEMENTATION

In a hardwired implementation, the control unit is essentially a state machine circuit. Its input logic signals are transformed into a set of output logic signals, which are the control signals.

3.3.1 CONTROL UNIT INPUT

The key inputs are the instruction registers, the clock, flags, and control bus signals. In the case of the flags and control bus signals, each bit typically has some meaning (eg overflow). The other two inputs, however, are not directly useful to the control unit. First, consider the instruction register. The control unit makes use of the opcode and will perform different actions (issue a different combination of control signals) for different instructions. To simplify the control unit logic, there should be a unique logic input for each opcode. This function can be performed by a decoder, which takes an encoded input and produces a single output. The clock portion of the control unit issues a representative sequence of pulses. This is useful for measuring the duration of micro-operations. Essentially the period of the clock pulses must be long enough to allow the propagation of signals along data paths and through processor circuitry. However, the control unit emits different control signals at different time units within the same instruction cycle. Thus, we would like a counter as input to the control unit with a different control signal being used for T1, T2, and so forth. At the end of an instruction cycle, the control unit must feed back to the counter to reinitialize it at T1.

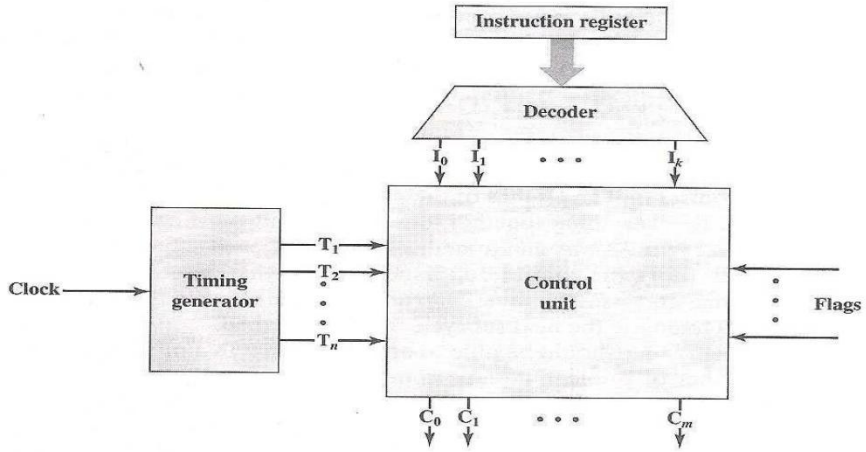


Figure 14. The control unit refirements

With these two refinements, the control unit can be depicted as in Figure 14.

To define the hardwired implementation of a control unit, all that remains is to discuss the internal logic of the control unit that produces output control signals as a function of its input signals.

Essentially, what must be done is, for each control signal, to derive a Boolean expression of that signal as a function of the inputs. This is best explained by example. Let us consider again our simple example illustrated in Figure 15.5. We saw in Table

15.1 the micro-operation sequences and control signals needed to control three of the four phases of the instruction cycle.

Let us consider a single control signal, C_5. This signal causes data to be read from the external data bus into the MBR. Let us define two new control signals, P and Q, that have the following interpretation:

PQ = 00 Fetch Cycle PQ = Ol Indirect Cycle PQ = 10 Execute Cycle PQ = 11 Interrupt Cycle

Then the following Boolean expression defines C5:

$C_5 = P \cdot Q \cdot T_2 + P.Q.T2$

That is, the control signal C5 will be asserted during the second time unit of both the fetch and indirect cycles.

This expression is not complete. C_5 is also needed during the execute cycle. For our simple example, let us assume that there are only three instructions that read from memory: LDA, ADD, and AND. Now we can define C_5 as

C_5 + P . Q . T_Z + P - Q - (LDA + ADD + AND)-T2

This same process could be repeated for every control signal generated by the processor. The result would be a set of Boolean equations that define the behavior of the control unit and hence of the processor.

To tie everything together, the control unit must control the state of the instruction cycle. As was mentioned, at the end of each sub-cycle (fetch, indirect, execute, interrupt), the control unit issues a signal that causes the timing generator to reinitialize and issue T_l. The control unit must also set the appropriate values of P and Q to define the next sub-cycle to be

performed.
The reader should be able to appreciate that in a modern complex processor, the number of Boolean equations needed to define the control unit is very large. The task of implementing a combinatorial circuit that satisfies all of these equations becomes extremely difficult. The result is that a far simpler approach, known as microprogramming, is usually used.

3.4 MICRO PROGRAMMED CONTROL

An alternative to a hardwired control unit is a microprogrammed control unit in which the logic of the control unit is specified by a microprogram. A micro program consists of a sequence of instructions in a microprogramming language. These are very simple instructions that specify micro-operations.

3.4.1 MICRO INSTRUCTIONS

Implement a control unit as n interconnection of basic logic elements is no easy task. The design must include logic for sequencing through micro-operations for executing micro-operations, interpreting opcodes, and for making decisions based on ALU flags. An alternative, which has been used in many CISC processors, is to implement a microprogrammed control unit.
In addition to the use of control signals, each micro-operation is described in symbolic notation. This notation looks suspiciously like a programming language. It is a language, known as a **microprogramming language**. Each line describes a set of micro-operations occurring at one time and is known as a **microinstruction**. A sequence of instructions is known as a **microprogram** or ***firmware.*** This latter term reflects the fact that a microprogram is midway between hardware and software. It is easier to design in firmware than **hardware**, but it is more difficult to write a firmware program than a software program.
How can we use the concept of microprogramming to implement a contra: unit? Consider that for each micro-operation, all that the control unit is allowed t o do is generate a set of control signals. Thus, for any micro-operation, each control link: emanating from the control unit is either on or off. This condition can, of course, be represented by a binary digit for each control line. So we could construct a contra *word* in which each bit represents one control line. Then each micro-operation would be represented by a different pattern of 1s and Os in the control word.
Suppose we string together a sequence of control words to represent the sequence of micro-operations performed by the control unit. Next, we must recognize that the sequence of micro-operations is not fixed. Sometimes we have an indirect cycle; sometimes we do not. So let us put our control words in a memory, with each word having a unique address. Now add an address field to each control word, indicating the location of the next control word to be executed if a certain condition is true (e.g., the indirect bit in a memory-reference instruction is 1). Also, add a few bits

to specify the condition.
The result is known as a horizontal microinstruction. The format of the microinstruction or control word is as follows. There is one bit for each internal processor control line and one bit for each system bus control line There is a condition field indicating the condition under which there should be a' branch, and there is a field with the address of the microinstruction to be executed next when a branch is taken. Such a microinstruction is interpreted as follows:

- To execute this microinstruction, turn on all the control lines indicated by a 1 bit; leave off all control lines indicated by a 0 bit. The resulting control signals will cause one or more micro-operations to be performed.
- If the condition indicated by the condition bits is false, execute the next microin- struction in sequence.
- If the condition indicated by the condition bits is true, the next microinstruction to be executed is indicated in the address field.

Figure 3.4.1b shows how these control words or microinstructions could be arranged in a control memory. The microinstructions in each routine are to be executed sequentially. Each routine ends with a branch or jump instruction indicating where to go next. There is a special execute cycle routine whose only purpose is to signify that one of the machine instruction routines (AND, ADD, and so on) is to be executed next, depending on the current opcode.
The control memory is a concise description of the complete operation of the control unit. It defines the sequence of micro-operations to be performed during each cycle (fetch, indirect, execute, interrupt), and it specifies the sequencing of these cycles. If nothing else, this notation would be a useful device for documenting the functioning of a control unit for a particular computer. But it is more than that. It is also a way of implementing the control unit.
The control memory contains a program that describes the behavior of the control unit. It follows that we could implement the control unit by simply executing that program. The set of micro instructions is stored in the *control memory*. The *control address register* contains the address of the next microinstruction to be read. When a microinstruction is read from the control memory, it is transferred to a control buffer register the left-hand portion of that register connects to the control lines emanating from the control unit. Thus, *reading* a microinstruction from the control memory is the same as *executing* that microinstruction. The third element shown in the figure is a sequencing unit that loads the control address register and issues a read command.
The control unit functions as follows:

1. To execute an instruction, the sequencing logic unit issues a READ command to the control memory.
2. The word whose address is Specified in the control address register is read into the control buffer register.

3. The content of the control buffer register generates control signals and next address information for the sequencing logic unit.
4. The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags.

All this happens during one clock pulse.

The last step just listed needs elaboration. After each microin- struction, the sequencing logic unit loads a new address into the control address register. Depending on the value of the ALU flags and the control buffer register, one of three decisions is made:

Depending on the value of the ALU flags and the control buffer register, one of three decisions is made:

- Get the next instruction: Add 1 to the control address register.
- Jump to a new routine based on a jump microinstruction: Load the address field of the control buffer register into the control address register.
- Jump to a machine instruction routine: Load the control address register based on the opcode in the IR.

3.4.2 ADVANTAGES AND DISADVANTAGES

The principal advantage of the use of micro-programming to implement a control unit is that it simplifies the design of the control unit. Thus it is both cheaper and less error-prone to implement. A hard-wired control unit must contain complex logic for sequencing through the many micro-operation s of the instructions cycle. On the other hand the decoders and sequencing logic unit of a microprogrammed control unit are very simple pieces of logic.

The principal disadvantage of a micro programmed unit is that it will be somewhat slower than a hardwired unit of comparable technology. Despite this, microprogramming is the dominant technique for implementing control units in pure CISC architecture due to its ease of implementation. RISC processors with their simpler instruction format, typically use hardwired control units

The two basic tasks performed by a microprogrammed control unit are as follows:

- Micro instruction sequencing: Get the next control signals needed to execute the micro instruction. In designing a control unit, these tasks must be considered together because both affect the format of the micro-instruction and the timing of the control unit.

Self-Assessment Exercises 2

Answer the following questions by choosing the most suitable option:

1. What is the main advantage of microprogrammed control units?
   A. They are faster than hardwired units
   B. They are easier to design and modify
   C. They use less power
   D. They are more reliable

2. Which type of processors typically use hardwired control units?
   A. CISC processors
   B. RISC processors
   C. Both CISC and RISC
   D. Neither CISC nor RISC

4.0 CONCLUSION

Micro- operations are the functional or atomic operations of a processor. The concepts of micro- operation serve as a guide to the design of the control unit.

5.0 SUMMARY

Each instruction cycle is made up of a set of micro-operations that generate control signals. Execution is accomplished by the effect of these control signals, emanating from the control unit to the ALU registers and system interconnection structure. Finally, an approach to the implementation of the control unit referred to as hard-wired implementation is presented. Furthermore, the concept of micro-operations leads to an elegant and powerful approach to control unit implementation, known as micro programming. Besides each instruction in the machine language of the processor is translated into a sequence of lower-level control unit instructions referred to as micro-instructions and the process of translation is referred to as microprogramming.

6.0 TUTOR- MARKED ASSIGNMENT

1. What is the relationship between instructions and micro operations?
2. Briefly what is meant by a hard-wired implementation of a control unit?
3. What are the basic tasks performed by a micro programmed control unit?
4. What is the difference between a hard-wired implementation and a micro-programmed implementation of a control unit?

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1

1. Micro-operations
2. Three, four
3. Hardwired, microprogrammed

Self-Assessment Exercise 2

1. B
2. B

7.0 REFERENCES/FURTHER READING

Carter J. Microprocessor Architecture and Microprogramming – Upper saddle River N. J Prentice HALL, 1996

Heath, S. (2014). Microprocessor Architectures: RISC, CISC and DSP. Elsevier.
Rafiquzzaman, M. (2005). Fundamentals of digital logic and microcomputer design. John Wiley & Sons.
Rafiquzzaman, M. (2021). Microprocessors and microcomputer-based system design. CRC press.
Chakraborty, P. (2020). Computer Organisation and Architecture: Evolutionary Concepts, Principles, and Designs. Chapman and Hall/CRC.
Null, L. (2023). Essentials of Computer Organization and Architecture. Jones & Bartlett Learning.

MODULE 3 CPU ORGANIZATION

UNIT 1: CPU Organization
UNIT 2: The Arithmetic and Logic Unit
UNIT 3: Control Unit

UNIT 1 CPU ORGANIZATION

1.0 INTRODUCTION

The full form of the CPU is the Central Processing Unit. It is the brain of the computer. All types of data processing operations and all important functions of a computer are performed by the CPU. It helps input and output devices communicate with each other and perform their respective operations. It also stores data that are input, intermediate results in between processing, and instructions. In this unit, we introduce the basic CPU organization and its instructions. This module also shows how a CPU is made, what's inside a CPU, how computer memory works, and how a CPU works.

A Central Processing Unit is the most important component of a computer system. A CPU is hardware that performs data input/output, processing and storage functions for a computer system. A CPU can be installed into a CPU socket. These sockets are generally located on the motherboard. CPU can perform various data processing operations. CPU can store data, instructions, programs, and intermediate results.

2.0 OBJECTIVES

At the end of the unit, you should be able to

- Recognize the history of Intel microprocessors
- Recall how a CPU is made from sand to chip
- List what's inside a CPU
- Demonstrate knowledge of computer memory integrating with a CPU

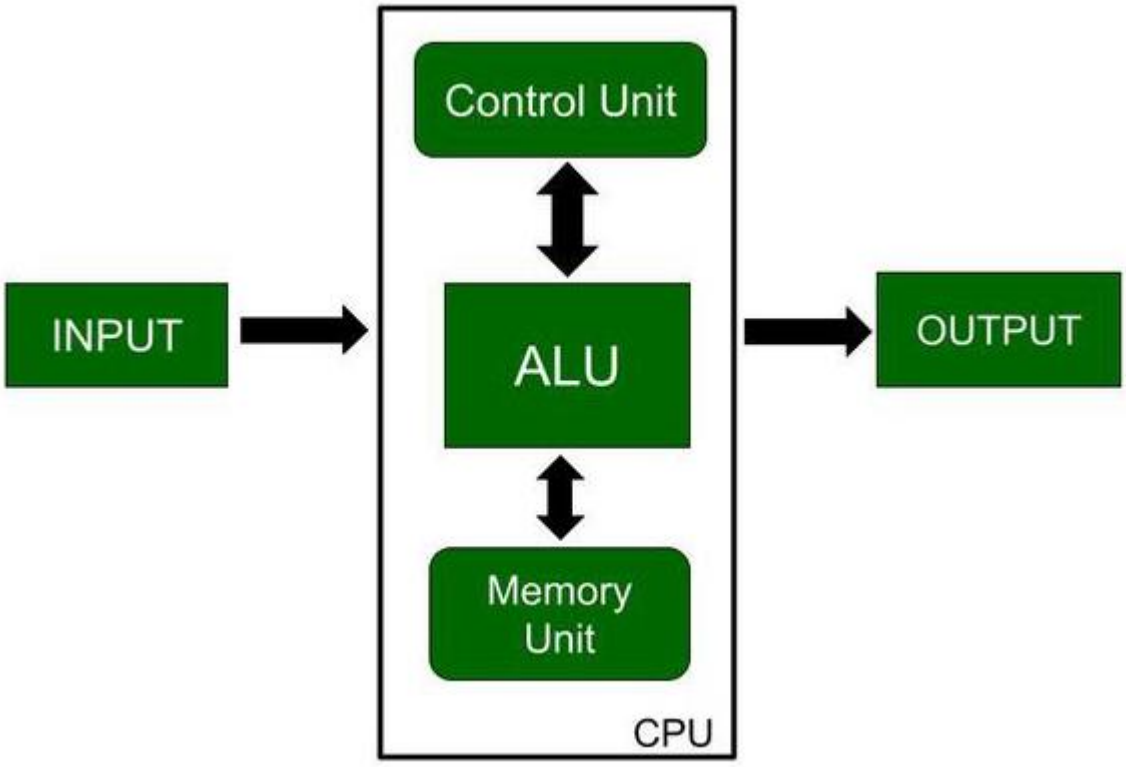
3.1 History of CPU

Since 1823, when Baron Jons Jakob Berzelius discovered silicon, which is still the primary component used in the manufacture of CPUs today, the history of the CPU has experienced numerous significant turning points.

The first transistor was created by John Bardeen, Walter Brattain, and William Shockley in December 1947. in 1958, the first working integrated circuit was built by Robert Noyce and Jack Kilby.
The Intel 4004 was the company's first microprocessor, which it unveiled in 1971. Ted Hoff's assistance was needed for this. When Intel released its 8008 CPU in 1972, Intel 8086 in 1976, and Intel 8088 in June 1979, it contributed to yet another win. The Motorola 68000, a 16/32-bit processor, was also released in 1979. The Sun also unveiled the SPARC CPU in 1987. AMD unveiled the AM386 CPU series in March 1991.
In January 1999, Intel introduced the Celeron 366 MHZ and 400 MHz processors. AMD back in April 2005 with its first dual-core processor. Intel also introduced the Core 2 Dual processor in 2006. Intel released the first Core i5 desktop processor with four cores in September 2009.
In January 2010, Intel released other processors like the Core 2 Quad processor Q9500, the first Core i3 and i5 mobile processors, first Core i3 and i5 desktop processors.
In June 2017, Intel released the Core i9 desktop processor, and Intel introduced its first Core i9 mobile processor In April 2018.
Different Parts of CPU
Now, the CPU consists of 3 major units, which are:

- Memory or Storage Unit
- Control Unit
- ALU (Arithmetic Logic Unit)

Let us now look at the block diagram of the computer:



Here, in this diagram, the three major components are also shown. So, let us discuss these major components:

Memory or Storage Unit

As the name suggests this unit can store instructions, data, and intermediate results. The memory unit is responsible for transferring information to other units of the computer when needed. It is also known as an internal storage unit or the main memory or the primary storage or Random Access Memory (RAM) as all these are storage devices.

Its size affects speed, power, and performance. There are two types of memory in the computer, which are primary memory and secondary memory. Some main functions of memory units are listed below:

Data and instructions are stored in memory units which are required for processing.

It also stores the intermediate results of any calculation or task when they are in process.

The final results of processing are stored in the memory units before these results are released to an output device for giving the output to the user.

All sorts of inputs and outputs are transmitted through the memory unit.

Control Unit

As the name suggests, a control unit controls the operations of all parts of the computer but it does not carry out any data processing operations. Executing already stored instructions, It instructs the computer by using the electrical signals to instruct the computer system. It takes instructions from the memory unit and then decodes the instructions after that it executes those instructions. So, it controls the functioning of the computer. Its main task is to maintain the flow of information across the processor. Some main functions of the control unit are listed below:

Controlling of data and transfer of data and instructions is done by the control unit among other parts of the computer.

The control unit is responsible for managing all the units of the computer.

The main task of the control unit is to obtain the instructions or data that is input from the memory unit, interpret them, and then direct the operation of the computer according to that.

The control unit is responsible for communication with Input and output devices for the transfer of data or results from memory.

The control unit is not responsible for the processing of data or storing data.

ALU (Arithmetic Logic Unit)

ALU (Arithmetic Logic Unit) is responsible for performing arithmetic and logical functions or operations. It consists of two subsections, which are:

Arithmetic Section

Logic Section

Now, let us know about these subsections:

Arithmetic Section: By arithmetic operations, we mean operations like addition, subtraction, multiplication, and division, and all these operation and functions are performed by ALU. Also, all the complex operations are done by making repetitive use of the mentioned operations by ALU.

Logic Section: By Logical operations, we mean operations or functions like selecting, comparing, matching, and merging the data, and all these are performed by ALU.
Note: CPU may contain more than one ALU and it can be used for maintaining timers that help run the computer system.
What Does a CPU Do?
The main function of a computer processor is to execute instruction and produce an output. CPU work are Fetch, Decode and Execute are the fundamental functions of the computer.
Fetch: the first CPU gets the instruction. That means binary numbers that are passed from RAM to CPU.
Decode: When the instruction is entered into the CPU, it needs to decode the instructions. with the help of ALU(Arithmetic Logic Unit) the process of decode begins.
Execute: After decode step the instructions are ready to execute
Store: After execute step the instructions are ready to store in the memory.
Types of CPU
We have three different types of CPU:
Single Core CPU: The oldest type of computer CPUs is single core CPU. These CPUs were used in the 1970s. these CPUs only have a single core that preform different operations. This means that the single core CPU can only process one operation at a single time. single core CPU is not suitable for multitasking.
Dual-Core CPU: Dual-Core CPUs contain a single Integrated Circuit with two cores. Each core has its cache and controller. These controllers and cache are work as a single unit. dual core CPUs can work faster than the single-core processors.
Quad-Core CPU: Quad-Core CPUs contain two dual-core processors present within a single integrated circuit (IC) or chip. A quad-core processor contains a chip with four independent cores. These cores read and execute various instructions provided by the CPU. Quad Core CPU increases the overall speed for programs. Without even boosting the overall clock speed it results in higher performance.
Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What are the three major components of a CPU?
   A. Input, Output, Storage
   B. Memory Unit, Control Unit, ALU
   C. Hardware, Software, Firmware
   D. Registers, Cache, Bus

2. Which CPU type has multiple independent cores?
   A. Single Core CPU

B. Dual-Core CPU
C. Quad-Core CPU
D. Both B and C

3. What does the CPU do during the "Fetch" step?
A. Executes the instruction
B. Stores the result
C. Gets the instruction from memory
D. Decodes the instruction

3.2 How the CPU work

Inside every computer is a central processing unit and inside every CPU are small components that carry out all the instructions for every program you run. These components include AND gates, OR gates, NOT gates, Clock, Multiplexer, ALU (arithmetic logic unit), etc. Data bus performs data transfer within a CPU and a computer. As shown in Fig. 8-1, the CPU is organized with a Program Counter (PC), Instruction Register (IR), Instruction Decoder, Control Unit, Arithmetic Logic Unit (ALU), Registers, and Buses. PC holds the address of the next instruction to be fetched from Memory. IR holds each instruction after it is fetched from Memory. Instruction Decoder decodes and interprets the contents of the IR, and splits a whole instruction into fields for the Control Unit to interpret. The Control Unit coordinates all activities within the CPU, has connections to all parts of the CPU, and includes a sophisticated timing circuit. ALU carries out arithmetic and logical operations, exemplified by addition, comparison, and Boolean AND/OR/NOT operations. Within ALU, input registers hold the input operands and output register holds the result of an ALU operation. Once completing ALU operation, the result is copied from the ALU output register to its final destination.

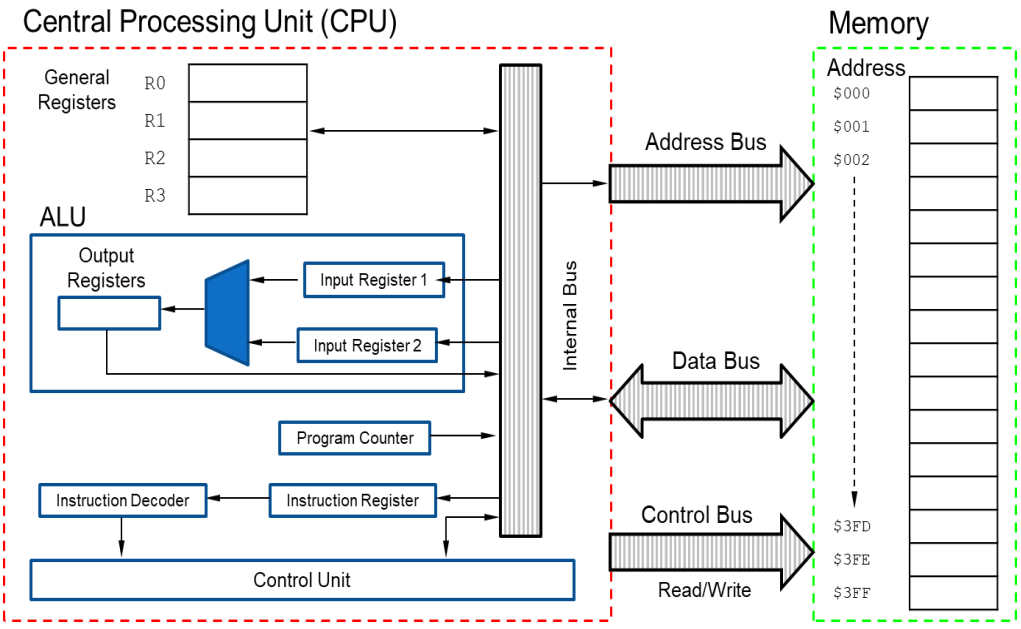


Figure 15. The CPU Organization

General-purpose registers are available for the programmer to use in their programs within the CPU. Typically, the programmer tries to maximize the use of these registers to speed program execution. Busses serve as communication highways for passing information on the computer.
The computer has memory which similarly memorizes data we remember past events. The register is the fastest memory which is located within the CPU of the computer.

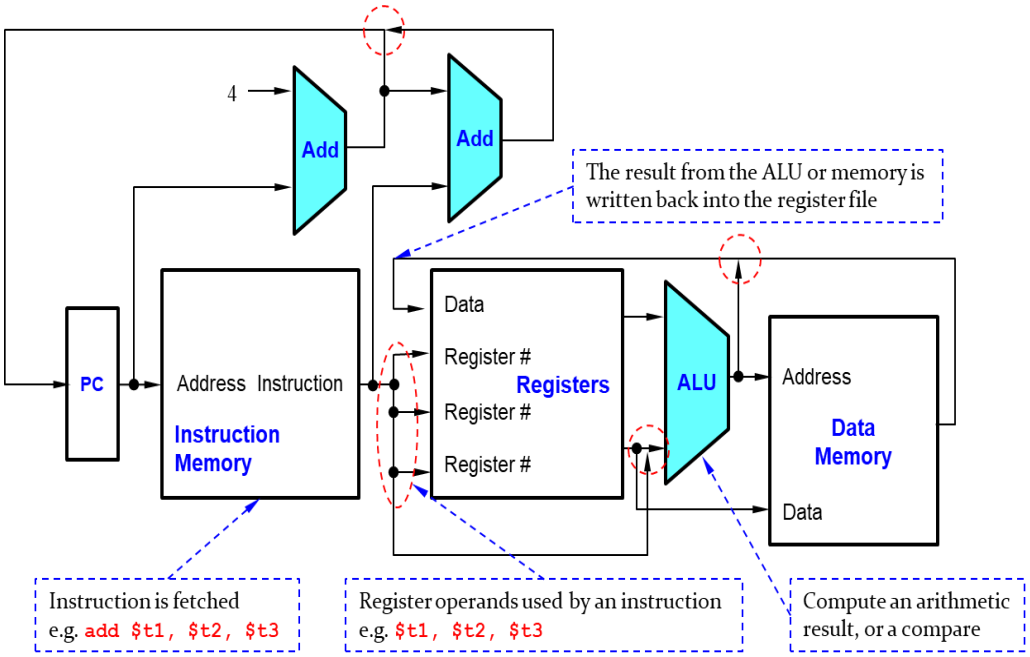


Figure 16. The CPU Overview

Figure 16 shows the CPU overview which consists of PC, instruction memory, registers, ALU, and Data memory. PC always holds the address of the next instruction to be fetched from Memory. Instruction, e.g. add $t1, $t2, $t3, is fetched into instruction memory. Register operands are used by an instruction in registers, where $t1 is the first source operand, $t2 is the second source operand, and $t3 is the storage of the result. ALU executes an arithmetic operation, e.g. Sum of $t1 and $t2. The result from the ALU or memory is written back into the register file ($t3). In the figure, ALU results and the output of data memory can't just join wires together. The red dash-dot line can be designed with the multiplexer to put the wires together.
The following figure shows CPU control with multiplexers. The first multiplexer controls what value replaces the PC (PC + 4 or the branch destination address), where the Mux is controlled by the AND gate with the Zero output of ALU and a control signal. The second multiplexer steers the output of the ALU or the output of the data memory. The third one determines whether the second ALU input is from the registers or

from the offset field of the instruction (for a load or store).

4.0 CONCLUSION

The Central Processing Unit (CPU) is often referred to as the brain of the computer. It executes instructions from computer programs by performing basic arithmetic, logical, control, and input/output (I/O) operations specified by the instructions. The CPU has a critical role in determining the speed and capability of a computer system.

5.0 SUMMARY

The Central Processing Unit (CPU) is the primary component of a computer responsible for interpreting and executing instructions. Often referred to as the computer's brain, it consists of the Arithmetic Logic Unit (ALU), which performs calculations and logical operations, and the Control Unit (CU), which directs all operations. The CPU fetches instructions from memory, decodes them, executes them, and writes back the results. Its performance is influenced by factors such as clock speed, number of cores, and cache size. Modern CPUs are designed for a range of devices, from high-performance servers to power-efficient mobile devices, continually advancing in power efficiency, integration, and parallel processing capabilities.

6.0 TUTOR- MARKED ASSIGNMENT

1. List and briefly explain parts of the CPU.
2. List the two most common types of control unit

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1

1. B
2. D
3. C

7.0 REFERENCES/FURTHER READING

Catanzaro B. Multiprocessor system Architecture Mountain View CA, Sun sift pres 1994

Null, L. (2023). Essentials of Computer Organization and Architecture. Jones & Bartlett Learning.

Liu, Z., Lin, Y., & Sun, M. (2023). Representation learning for natural language processing (p. 521). Springer Nature.

UNIT 2 THE ARITHMETIC AND LOGIC UNIT

1.0 Introduction

The Arithmetic Logic Unit (ALU) is a fundamental component of the Central Processing Unit (CPU) that is responsible for executing all arithmetic and logical operations within a computer. It performs essential arithmetic operations such as addition, subtraction, multiplication, and division, as well as logical operations including AND, OR, NOT, and XOR. Additionally, the ALU handles bitwise operations, which involve the manipulation of individual bits within a binary number. These operations are critical for various computational tasks, such as calculations, data manipulation, and decision-making processes. The ALU consists of input registers that store the operands, operational logic that performs the calculations, and result storage that temporarily holds the output before it is transferred to other CPU components or memory.

Beyond basic calculations, the ALU plays a crucial role in comparison operations, such as determining whether numbers are equal, greater than, or less than each other. This capability is essential for implementing control flow in programs, enabling the CPU to make decisions based on conditional statements and execute different instructions based on those conditions. The efficiency and speed of the ALU directly impact the overall performance of the CPU, as it processes the core computations required for running applications and executing instructions. By facilitating both arithmetic and logical operations, the ALU enables the CPU to perform complex tasks and drive the functionality of the computer.

2.0 Objectives

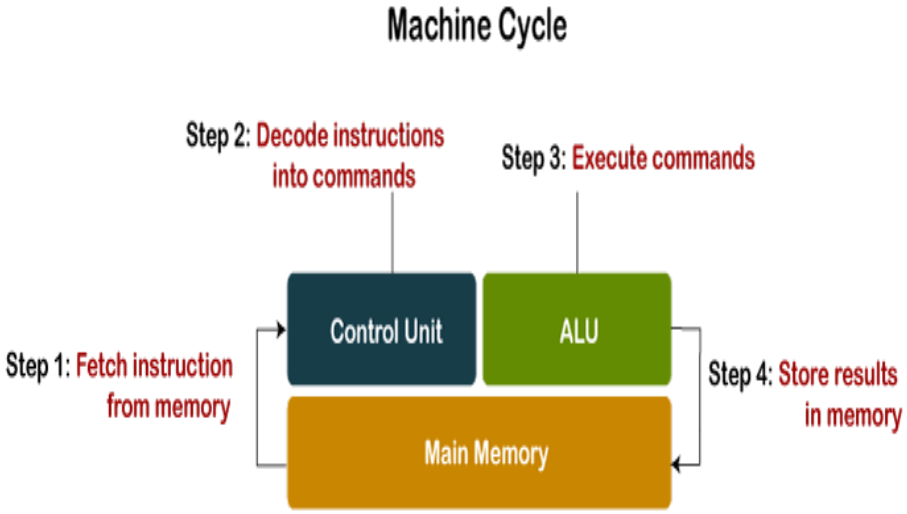
At the end of this unit, you should be able to

- Understand the general concepts of the Arithmetic and Logic Unit
- Explain the ALU of the computer system.

3.1 The General Concepts of CPU

In the computer system, ALU is a main component of the central processing unit, which stands for arithmetic logic unit and performs arithmetic and logic operations. It is also known as an integer unit (IU)

which is an integrated circuit within a CPU or GPU, which is the last component to perform calculations in the processor. It can perform all processes related to arithmetic and logic operations such as addition, subtraction, and shifting operations, including Boolean comparisons (XOR, OR, AND, and NOT operations). Also, binary numbers can accomplish mathematical and bitwise operations. The arithmetic logic unit is split into AU (arithmetic unit) and LU (logic unit). The operands and code used by the ALU tell it which operations have to be performed according to input data. When the ALU completes the processing of input, the information is sent to the computer's memory.



Except for performing calculations related to addition and subtraction, ALUs handle the multiplication of two integers as they are designed to execute integer calculations; hence, its result is also an integer. However, division operations commonly may not be performed by ALU as division operations may produce a result in a floating-point number. Instead, the floating-point unit (FPU) usually handles the division operations; other non-integer calculations can also be performed by FPU.

Additionally, engineers can design the ALU to perform any type of operation. However, ALU becomes costlier as the operations become more complex because ALU destroys more heat and takes up more space in the CPU. This is the reason for making powerful ALUs by engineers, which provides the surety that the CPU is fast and powerful as well.

The calculations needed by the CPU are handled by the arithmetic logic unit (ALU); most of the operations among them are logical. If the CPU is made more powerful, which is made based on how the ALU is designed. Then it creates more heat and takes more power or energy. Therefore, there must be moderation between how complex and powerful ALU is and not be costly. This is the main reason the faster CPUs are costlier; hence, they take up much power and destroy more heat. Arithmetic and

logic operations are the main operations that are performed by the ALU; it also perform bit-shifting operations.

Although the ALU is a major component in the processor, the ALU's design and function may be different in the different processors. For case, some ALUs are designed to perform only integer calculations, and some are for floating-point operations. Some processors include a single arithmetic logic unit to perform operations, and others may contain numerous ALUs to complete calculations. The operations performed by ALU are:

Logical Operations: The logical operations consist of NOR, NOT, AND, NAND, OR, XOR, and more.

Bit-Shifting Operations: It is responsible for displacement in the locations of the bits to the right or left by a certain number of places that is known as a multiplication operation.

Arithmetic Operations: Although it performs multiplication and division, this refers to bit addition and subtraction. But multiplication and division operations are more costly to make. In the place of multiplication, addition can be used as a substitute and subtraction for division.

Arithmetic Logic Unit (ALU) Signals

A variety of input and output electrical connections are contained by the ALU, which led to casting the digital signals between the external electronics and ALU.

ALU input gets signals from the external circuits, and in response, external electronics get outputs signals from ALU.

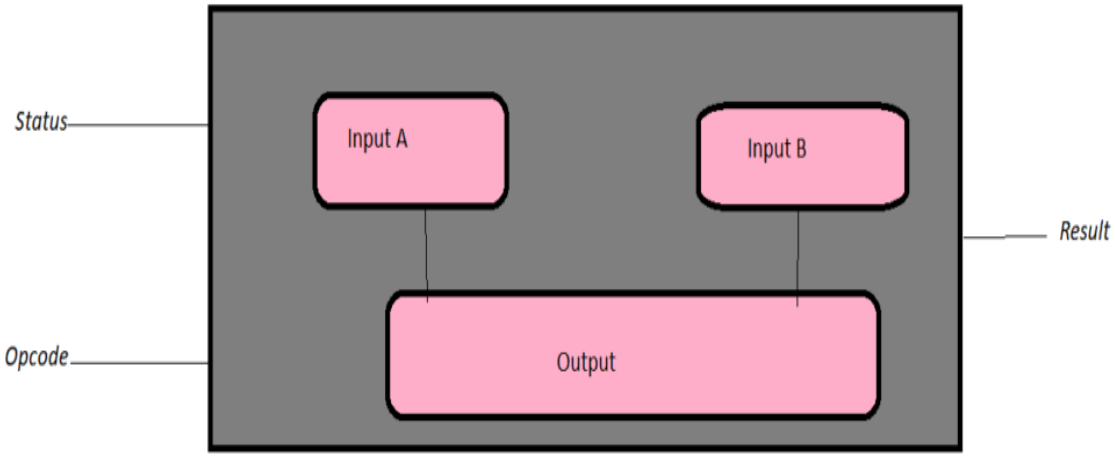
Data: Three parallel buses are contained by the ALU, which include two input and output operand. These three buses handle the number of signals, which are the same.

Opcode: When the ALU is going to operate, it is described by the operation selection code what type of operation an ALU is going to perform arithmetic or logic operation.

Status

Output: The results of the ALU operations are provided by the status outputs in the form of supplemental data as they are multiple signals. Usually, status signals like overflow, zero, carry out, negative, and more are contained by general ALUs. When the ALU completes each operation, the external registers contain the status output signals. These signals are stored in the external registers that led to making them available for future ALU operations.

Input: When ALU once operates, the status inputs allow ALU to access further information to complete the operation successfully. Furthermore, stored carry-out from a previous ALU operation is known as a single "carry-in" bit.



3.2 Configurations of the ALU

The description of how ALU interacts with the processor is given below. Every arithmetic logic unit includes the following configurations:

Instruction Set Architecture

Accumulator

Stack

Register to Register

Register Stack

Register Memory

Accumulator

The intermediate result of every operation is contained by the accumulator, which means Instruction Set Architecture (ISA) is not more complex because it is only required to hold one bit.

Generally, they are much faster and less complex but to make the Accumulator more stable; additional codes need to be written to fill it with proper values. Unluckily, with a single processor, it is very difficult to find Accumulators to execute parallelism. An example of an Accumulator is the desktop calculator.

Stack

Whenever the latest operations are performed, these are stored on the stack that holds programs in top-down order, which is a small register. When the new programs are added to execute, they push to put the old programs.

Register-Register Architecture

It includes a place for 1 destination instruction and 2 source instructions, also known as a 3-register operation machine. This Instruction Set Architecture must be longer for storing three operands, 1 destination, and 2 sources. After the end of the operations, writing the results back to the Registers would be difficult, and also the length of the word should be longer. However, it can cause more issues with synchronization if the write-back rule is followed at this place.

The MIPS component is an example of the register-to-register

Architecture. For input, it uses two operands, and for output, it uses a third distinct component. The storage space is hard to maintain as each needs a distinct memory; therefore, it has to be premium at all times. Moreover, it might be difficult to perform some operations.

Register - Stack Architecture

Generally, the combination of Register and Accumulator operations is known as Register-Stack Architecture. The operations that need to be performed in the register-stack Architecture are pushed onto the top of the stack. And its results are held at the top of the stack. With the help of using the Reverse polish method, more complex mathematical operations can be broken down. Some programmers, to represent operands, use the concept of a binary tree. It means that the reverse polish methodology can be easy for these programmers, whereas it can be difficult for other programmers. To carry out Push and Pop operations, there is a need to be new hardware created.

Self-Assessment Exercises 1

Fill in the gaps in the sentences below with the most suitable words:

1. The ALU is responsible for performing _____ and _____ operations.

2. ALU stands for _____ _____ Unit.

3. The three main types of operations performed by ALU are arithmetic operations, logical operations, and _____ operations.

Register and Memory

In this architecture, one operand comes from the register, and the other comes from the external memory as it is one of the most complicated architectures. The reason behind this is that every program might be very long as they require to be held in full memory space. Generally, this technology is integrated with Register-Register Register technology and practically cannot be used separately.

ALUs, in addition to doing addition and subtraction calculations, also handle the process of multiplication of two integers because they are designed to perform integer calculations; thus, the result is likewise an integer. Division operations, on the other hand, are frequently not done by ALU since division operations can result in a floating-point value. Instead, division operations are normally handled by the floating-point unit (FPU), which may also execute other non-integer calculations.

Engineers can also design the ALU to do any operation they choose. However, as the operations become more sophisticated, ALU becomes more expensive since it generates more heat as well as takes up more space on the CPU. Therefore, engineers create powerful ALUs, ensuring

that the CPU is both quick and powerful.
The ALU performs the computations required by the CPU; most of the operations are logical. If the CPU is built more powerful, it will be designed based on the ALU. Then it generates more heat and consumes more energy or power. As a result, there must be a balance between how intricate and strong ALU is and how much it costs. The primary reason why faster CPUs are more expensive is that they consume more power and generate more heat due to their ALUs. The ALU's major functions are arithmetic and logic operations, as well as bit-shifting operations.

3.3 Operations Performed by ALU

Although the ALU is a critical component of the CPU, the design and function of the ALU may vary amongst processors. Some ALUs, for example, are designed solely to conduct integer calculations, whereas others are built to perform floating-point computations. Some processors have a single arithmetic logic unit that performs operations, whereas others have many ALUs that conduct calculations. ALU's operations are as follows:

1. Arithmetic Operators: It refers to bit subtraction and addition, despite the fact that it does multiplication and division. Multiplication and division processes, on the other hand, are more expensive to do. Addition can be used in place of multiplication, while subtraction can be used in place of division.

2. Bit-Shifting Operators: It is responsible for a multiplication operation, which involves shifting the locations of a bit to the right or left by a particular number of places.

3. Logical Operations: These consist of NOR, AND, NOT, NAND, XOR, OR, and more.

ALU Signals

The ALU contains a variety of electrical input and output connections, which result in the digital signals being cast between the ALU and the external electronics. External circuits send signals to the ALU input, and the ALU sends signals to the external electronics.

Opcode: The operation selection code specifies whether the ALU will conduct arithmetic or a logic operation when it performs the operation.

Data: The ALU contains three parallel buses, each with two input and output operands. These three buses are in charge of the same amount of signals.

Advantages of ALU

ALU has various advantages, which are as follows:

- It supports parallel architecture and applications with high performance.
- It can get the desired output simultaneously and combine integer and floating-point variables.
- It has the capability of performing instructions on a very large set and has a high range of accuracy.
- Two arithmetic operations in the same code like addition and

multiplication or addition and subtraction, or any two operands can be combined by the ALU. For case, A+B*C.

- Throughout the whole program, they remain uniform, and they are spaced in a way that they cannot interrupt parts in between.
- In general, it is very fast; hence, it provides results quickly.
- There are no sensitivity issues and no memory wastage with ALU.
- They are less expensive and minimize the logic gate requirements.

Disadvantages of ALU

The disadvantages of ALU are discussed below:

- With the ALU, floating variables have more delays, and the designed controller is not easy to understand.
- The bugs would occur in our result if memory space were definite.
- It is difficult to understand amateurs as their circuit is complex; also, the concept of pipelining is complex to understand.
- A proven disadvantage of ALU is that there are irregularities in latencies.
- Another demerit is rounding off, which impacts accuracy.

Self-Assessment Exercises 2

Answer the following questions by choosing the most suitable option:

1. Which of the following is NOT an advantage of ALU?
   A. High processing speed
   B. Support for parallel architecture
   C. Unlimited memory capacity
   D. No sensitivity issues

2. What type of operations does ALU handle for floating-point numbers?
   A. All floating-point operations
   B. Limited floating-point operations
   C. No floating-point operations
   D. Only division operations

4.0 CONCLUSION

An arithmetic logic unit (ALU) is a key component of a computer's central processor unit. The ALU performs all arithmetic and logic operations that must be performed on instruction words. The ALU is split into two parts in some microprocessor architectures: the AU and the LU. ALU conducts arithmetic and logic operations. It is a major component of the CPU in a computer system. An integer unit (IU) is just an integrated circuit within a GPU or GPU that performs the last calculations in the processor. It can execute all arithmetic and logic operations, including Boolean comparisons, such as subtraction, addition, and shifting (XOR, OR, AND, and NOT operations). Binary numbers can also perform bitwise and mathematical operations. AU (arithmetic unit) and LU (logic unit) are two types of arithmetic logic units. The ALU's operands and

code instruct it on which operations to perform based on the incoming data. When the ALU has finished processing the data, it sends the result to the computer memory.

5.0 SUMMARY

The ALU is a crucial component of the CPU responsible for executing arithmetic operations like addition, subtraction, multiplication, and division, as well as logical operations. It also handles bitwise operations and comparisons, enabling the CPU to make decisions based on conditional statements. Comprising input registers for operands, operational logic for performing calculations, and result storage, the ALU's efficiency directly influences the CPU's overall performance. By facilitating essential computations and decision-making processes, the ALU plays a key role in the execution of programs and the overall functionality of the computer.

6.0 TUTOR MARKED ASSIGNMENT

1. What are some of the potential advantages of the ALU?
2. What are the chief characteristics of the ALU?

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. Arithmetic, logical
2. Arithmetic Logic
3. Bit-shifting

Self-Assessment Exercise 2
1. C
2. B

7.0 References/ Further reading

Herlihy, M., Shavit, N., Luchangco, V., & Spear, M. (2020). The art of multiprocessor programming. Newnes.

Jayanti, S. V. (2023). Simple, Fast, Scalable, and Reliable Multiprocessor Algorithms. Massachusetts Institute of Technology.

Brandenburg, B. B. (2022). Multiprocessor real-time locking protocols. In Handbook of Real-Time Computing (pp. 347-446). Singapore: Springer Nature Singapore.

UNIT 3 THE CONTROL UNIT

1.0 INTRODUCTION

The control unit (CU) is a critical component of a computer's central processing unit (CPU), responsible for directing the operation of the processor. It orchestrates the fetching, decoding, and execution of instructions by generating appropriate control signals to the various subsystems within the CPU. The CU ensures that the data flows correctly between the CPU, memory, and input/output devices, and it regulates the execution of instructions by controlling the timing and sequencing of operations. By interpreting the instructions in a program, the control unit determines which arithmetic, logic, or control operation is to be performed next and manages the data paths accordingly.

The control unit can be designed using either hardwired logic or microprogramming. A hardwired control unit uses fixed logic circuits to control signals, which makes it fast but less flexible and more difficult to modify or update. In contrast, a microprogrammed control unit stores control signals in a memory-based control store, allowing for easier modifications and updates at the cost of some performance. The CU plays a crucial role in the overall function and efficiency of the CPU, ensuring that all operations are performed correctly and in the proper sequence, thereby enabling the execution of complex computational tasks.

2.0 OBJECTIVES

At the end of this unit, you should be able to:

- Explain the control unit.
- Discuss the types of control units.
- Understand how the control unit works.

A **Central Processing Unit** is the most important component of a computer system. A control unit is a part of the CPU. A control unit controls the operations of all parts of the computer but it does not carry out any data processing operations.

What is a Control Unit?

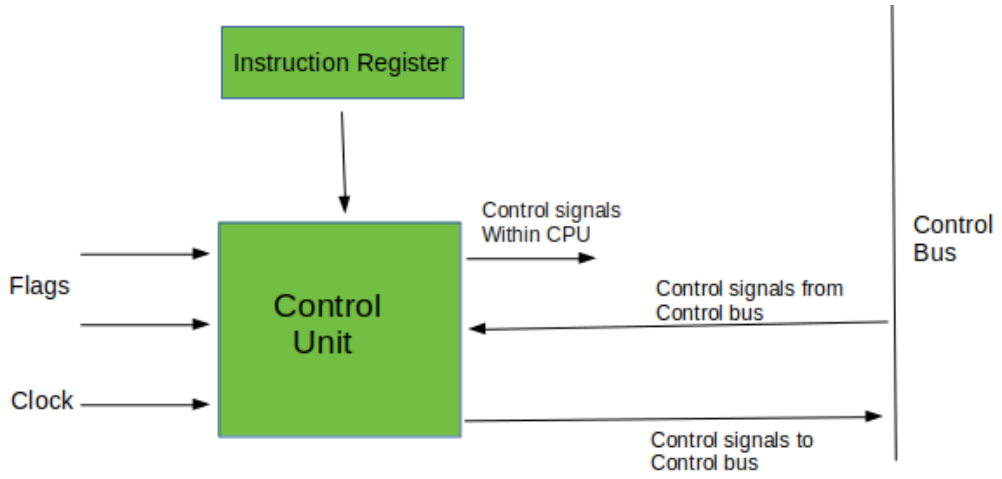
The Control Unit is the part of the computer’s central processing unit (CPU), which directs the operation of the processor. It was included as

part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the control unit to tell the computer's memory, arithmetic/logic unit, and input and output devices how to respond to the instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions. A control unit works by receiving input information which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of the CPU varies from manufacturer to manufacturer.

Examples of devices that require a CU are:

Control Processing Units(CPUs)

Graphics Processing Units(GPUs)



Block Diagram of the Control Unit

Functions of the Control Unit

- It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.
- It interprets instructions.
- It controls data flow inside the processor.
- It receives external instructions or commands which it converts to a sequence of control signals.
- It controls many execution units (i.e. ALU, data buffers, and registers) contained within a CPU.
- It also handles multiple tasks, such as fetching, decoding, execution handling, and storing results.

3.2 Types of Control Units

There are two types of control units:

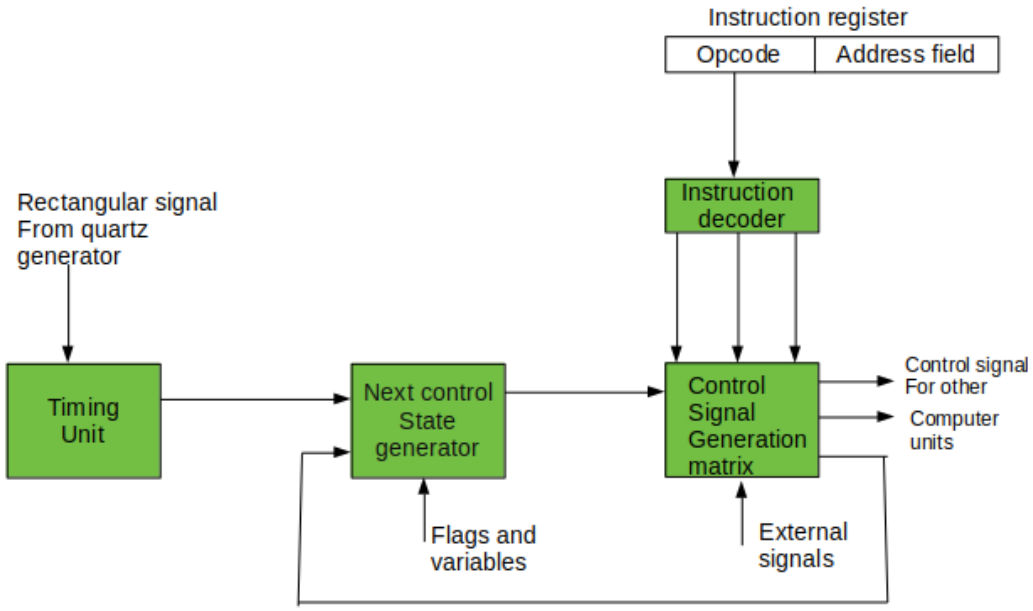
Hardwired

Micro programmable control unit.

Hardwired Control Unit

In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we cannot modify the signal generation method without a physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.

As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for execution units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.



Block diagram of a hardwired control unit of a computer

Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit. A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix.

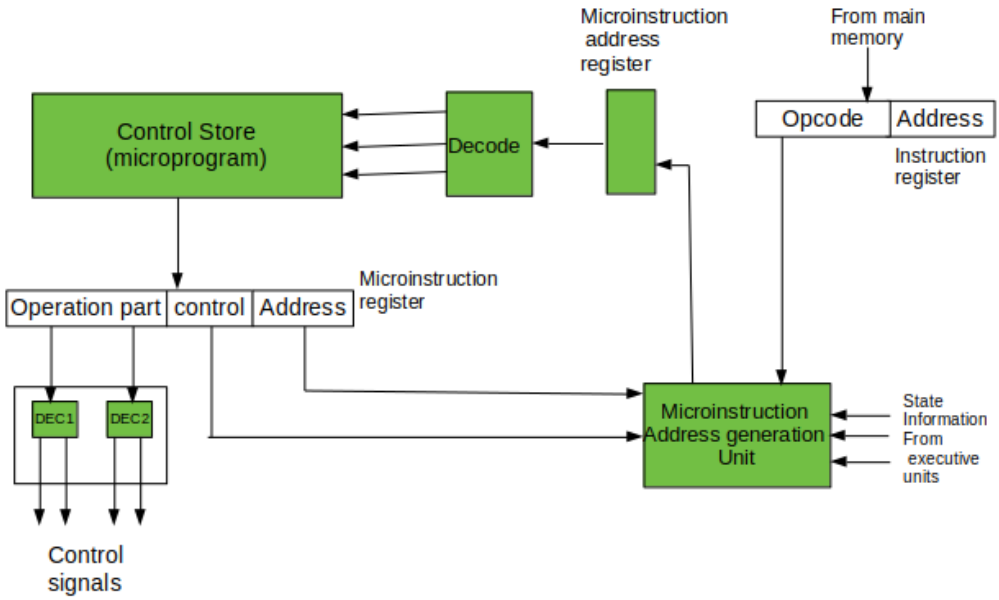
This matrix combines these signals with the timing signals, which are

generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered.

A change of any of the earlier mentioned signals stimulates the change of the control unit state. This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into the next control state which is the state concerned with the reaction to this external signal (e.g. interrupt processing).

The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle. The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of the computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for the next user directive.

Micro Programmable control unit

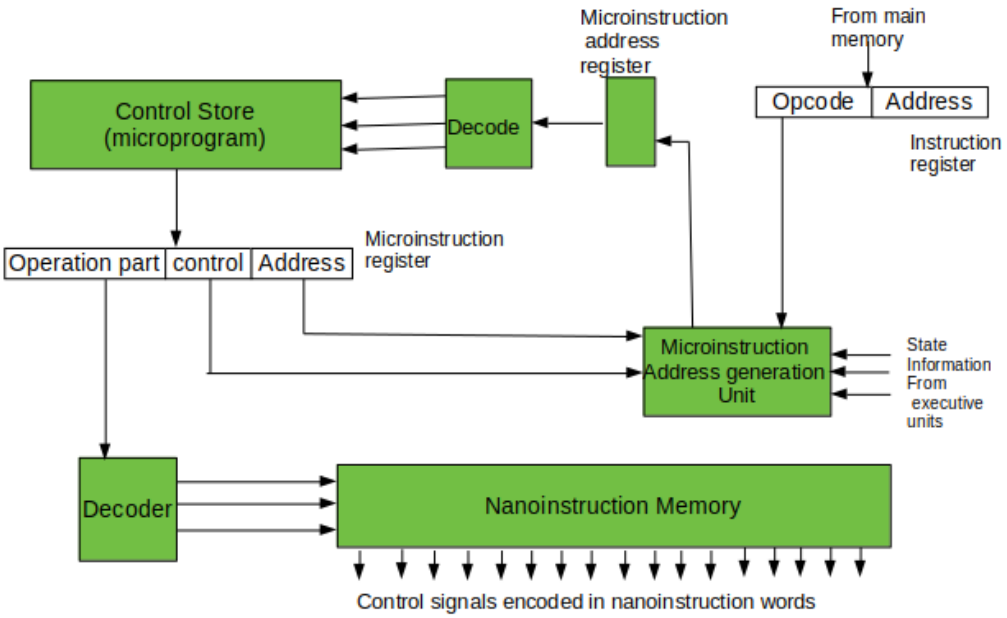


Microprogrammed control unit with a single level control store

The fundamental difference between these unit structures and the structure of the hardwired control unit is the existence of the control store that is used for storing words containing encoded control signals

mandatory for instruction execution. In microprogrammed control units, subsequent instruction words are fetched into the instruction register in a normal way. However, the operation code of each instruction is not directly decoded to enable immediate control signal generation but it comprises the initial address of a microprogram contained in the control store.

With a single-level control store: In this, the instruction opcode from the instruction register is sent to the control store address register. Based on this address, the first microinstruction of a microprogram that interprets the execution of this instruction is read to the microinstruction register. This microinstruction contains in its operation part encoded control signals, normally as few bit fields. In a set microinstruction field decoder, the fields are decoded. The microinstruction also contains the address of the next microinstruction of the given instruction microprogram and a control field used to control activities of the microinstruction address generator. The last mentioned field decides the addressing mode (addressing operation) to be applied to the address embedded in the ongoing microinstruction. In microinstructions along with conditional addressing mode, this address is refined by using the processor condition flags that represent the status of computations in the current program.



Microprogrammed control unit with a two-level control store

The last microinstruction in the instruction of the given microprogram is the microinstruction that fetches the next instruction from the main memory to the instruction register.

With a two-level control store: In this, in a control unit with a two-level control store, besides the control memory for microinstructions, a nano-

instruction memory is included. In such a control unit, microinstructions do not contain encoded control signals. The operation part of microinstructions contains the address of the word in the nano-instruction memory, which contains encoded control signals. The nano-instruction memory contains all combinations of control signals that appear in microprograms that interpret the complete instruction set of a given computer, written once in the form of nano-instructions. In this way, unnecessary storing of the same operation parts of microinstructions is avoided. In this case, microinstruction words can be much shorter than with the single-level control store. It gives a much smaller size in bits of the microinstruction memory and, as a result, a much smaller size of the entire control memory. The microinstruction memory contains the control for the selection of consecutive microinstructions, while those control signals are generated on the basis of nano-instructions. In nano-instructions, control signals are frequently encoded using a 1 bit/ 1 signal method that eliminates decoding.

Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What is the primary function of the Control Unit?
   A. To perform arithmetic calculations
   B. To store data permanently
   C. To control the operations of all parts of the computer
   D. To provide input/output interfaces

2. Which type of control unit is more flexible but slower?
   A. Hardwired Control Unit
   B. Micro-programmed Control Unit
   C. Both are equally flexible
   D. Neither is flexible

3. What type of computers typically use hardwired control units?
   A. CISC computers
   B. RISC computers
   C. Both CISC and RISC
   D. Mainframe computers only

Differences between Hardwired Control unit and Micro-programmed Control unit

There are differences between Micro-programmed CU and Hardwired CU, which are described as follows:

Hardwired Control Unit	Micro-programmed Control Unit
With the help of a hardware circuit, we can implement the hardwired control unit. In other words, we can say that it is a circuitry approach.	While with the help of programming, we can implement the micro-programmed control unit.
The hardwired control unit uses the logic circuit so that it can generate the control signals, which are required for the processor.	The micro-programmed CU uses microinstruction so that it can generate the control signals. Usually, control memory is used to store these microinstructions.
In this CU, the control signals are going to be generated in the form of hard wired. That's why it is very difficult to modify the hardwired control unit.	It is very easy to modify the micro-programmed control unit because the modifications are going to be performed only at the instruction level.
In the form of logic gates, everything has to be realized in the hardwired control unit. That's why this CU is more costly compared to the micro-programmed control unit.	The micro-programmed control unit is less costly compared to the hardwired CU because this control unit only requires the microinstruction to generate the control signals.
The complex instructions cannot be handled by a hardwired control unit because when we design a circuit for this instruction, it will become complex.	The micro-programmed control unit is able to handle the complex instructions.
Because of the hardware implementation, the hardwired control unit is able to use a limited number of instructions.	The micro-programmed control unit is able to generate control signals for many instructions.
The hardwired control unit is used in those types of computers that also use the RISC (Reduced instruction Set Computers).	The micro-programmed control unit is used in those types of computers that also use the CISC (Complex instruction Set Computers).
In the hardwired control unit, the	In this CU, the microinstructions

hardware is used to generate only the required control signals. That's why this control unit is faster compared to the micro-programmed control unit.	are used to generate control signals. That's why this CU is slower than the hardwired control unit.

Some Other differences between Micro-programmed control unit and Hardwire control unit

Now we will describe these differences on the basis of some parameters, such as speed, cost, modification, instruction decoder, control memory, etc. These differences are described as follows:

Speed

In the hardwired control unit, the speed of operations is very fast. In contrast, the micro-programmed control unit needs frequent memory access. So the speed of operation of a micro-programmed control unit is slow.

Modification

If we want to do some modifications to the Hardwired control unit, we have to redesign the entire unit. In contrast, if we want to do some modification in the micro-programmed control unit, we can do that just by changing the microinstructions in the control memory. In this case, the more flexible control unit is a micro-programmed control unit.

Cost

The implementation of a Hardwire control unit is very much compared to the Micro-programmed control unit. In this case, the micro-programmed control unit will save our money at the time of implementation.

Handling Complex Instructions

If we try to handle the complex instructions with the help of a hardwired control unit, it will be very difficult for us to handle them. But if we try to handle the complex instructions with the help of a micro-programmed control unit, it will be very easy for us to handle them. In this case, also, the Micro-programmed control unit is better.

Instruction decoding

In the hardwired control unit, if we want to perform instruction decoding, it will be very difficult. But if we do the same thing in a micro-programmed control unit, it will be very easy for us.

Instruction set size

A small instruction set is used by the hardwired CU. On the other hand, a large instruction set is used by the micro-programmed control unit.

Control Memory

The hardwired control unit does not use the control memory to generate the control signals, but the micro-programmed CU needs to use the control memory to generate the control signals.

Applications

The hardwired control unit is used in those types of processors that use a simple instruction set. This set is called a Reduced Instruction Set Computer. On the other hand, a micro-programmed control unit is used in those types of processors that basically use a complex instruction set. This set is called a Complex Instruction Set Computer.

Advantages of a Well-Designed Control Unit

Efficient instruction execution: A well-designed control unit can execute instructions more efficiently by optimizing the instruction pipeline and minimizing the number of clock cycles required for each instruction.

Improved performance: A well-designed control unit can improve the performance of the CPU by increasing the clock speed, reducing the latency, and improving the throughput.

Support for complex instructions: A well-designed control unit can support complex instructions that require multiple operations, reducing the number of instructions required to execute a program.

Improved reliability: A well-designed control unit can improve the reliability of the CPU by detecting and correcting errors, such as memory errors and pipeline stalls.

Lower power consumption: A well-designed control unit can reduce power consumption by optimizing the use of resources, such as registers and memory, and reducing the number of clock cycles required for each instruction.

Better branch prediction: A well-designed control unit can improve branch prediction accuracy, reducing the number of branch mispredictions and improving performance.

Improved scalability: A well-designed control unit can improve the scalability of the CPU, allowing it to handle larger and more complex workloads.

Better support for parallelism: A well-designed control unit can better support parallelism, allowing the CPU to execute multiple instructions simultaneously and improve overall performance.

Improved security: A well-designed control unit can improve the security of the CPU by implementing security features such as address space layout randomization and data execution prevention.

Lower cost: A well-designed control unit can reduce the cost of the CPU by minimizing the number of components required and improving manufacturing efficiency.

Disadvantages of a Poorly-Designed Control Unit

Reduced performance: A poorly designed control unit can reduce the performance of the CPU by introducing pipeline stalls, increasing the latency, and reducing the throughput.

Increased complexity: A poorly designed control unit can increase the complexity of the CPU, making it harder to design, test, and maintain.

Higher power consumption: A poorly designed control unit can

increase power consumption by inefficiently using resources, such as registers and memory, and requiring more clock cycles for each instruction.

Reduced reliability: A poorly designed control unit can reduce the reliability of the CPU by introducing errors, such as memory errors and pipeline stalls.

Limitations on instruction set: A poorly designed control unit may limit the instruction set of the CPU, making it harder to execute complex instructions and limiting the functionality of the CPU.

Inefficient use of resources: A poorly designed control unit may inefficiently use resources such as registers and memory, leading to wasted resources and reduced performance.

Limited scalability: A poorly designed control unit may limit the scalability of the CPU, making it harder to handle larger and more complex workloads.

Poor support for parallelism: A poorly designed control unit may limit the ability of the CPU to support parallelism, reducing the overall performance of the system.

Security vulnerabilities: A poorly designed control unit may introduce security vulnerabilities, such as buffer overflows or code injection attacks.

Higher cost: A poorly designed control unit may increase the cost of the CPU by requiring additional components or increasing the manufacturing complexity.

4.0 CONCLUSION

In the world of computer architecture, the Control Unit plays a pivotal role in ensuring the effective and efficient functioning of modern computing systems. Delving into the intricacies of this vital component allows you to gain insight into its core functions, applications, and different types. This article will explore the various aspects of the Control Unit, including its definition and key role in computer architecture, managing the data flow, and its relation to the Central Processing Unit (CPU). Moreover, the article will navigate the different types of Control Units, such as Hardwired and Microprogrammed Control Units, discussing their advantages, disadvantages, flexibility, and adaptability. You will also discover the crucial differences between these Control Unit types and understand how to choose the appropriate one for your computer system. Furthermore, it will examine the diverse applications of the Control Unit in various contexts of computer science, such as personal computers, laptops, modern devices, and the rapidly evolving Internet of Things (IoT). By understanding the importance and role of the Control Unit, you can appreciate its impact on shaping the future of computing technology.

5.0 SUMMARY

A control unit, or CU, is circuitry within a computer's processor that directs operations. It instructs the memory, logic unit, and both output and input devices of the computer on how to respond to the program's

instructions. CPUs and GPUs are examples of devices that use control units.

The Control Unit has a significant role within a computer system, which includes:

- Fetching instructions from memory
- Decoding instructions to determine what operation to perform
- Controlling and coordinating the execution of instructions
- Managing data flow between various units of the computer
- Monitoring and regulating the synchronization of input and output devices

6.0 TUTOR MARKED ASSIGNMENT

1. Explain Control Unit
2. List and briefly explain the types of control unit

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1

1. C
2. B
3. B

7.0 References/ further reading

Ellis, G. (2012). Control system design guide: using your computer to understand and diagnose feedback controllers. Butterworth-Heinemann.

Åström, K. J., & Wittenmark, B. (2013). Computer-controlled systems: theory and design. Courier Corporation.

Gopal, M. (2008). Control systems: principles and design. McGraw-Hill Science, Engineering & Mathematics.

Wolf, M. (2012). Computers as components: principles of embedded computing system design. Elsevier.

Clark, R. N. (1996). Control system dynamics. Cambridge University Press.

MODULE 4 INSTRUCTION SET ARCHITECTURE

UNIT 1 GENERAL OVERVIEW OF INSTRUCTION SET ARCHITECTURE

CONTENT

1.0 INTRODUCTION

An Instruction Set Architecture (ISA) is part of the abstract model of a computer that defines how the CPU is controlled by the software. The ISA acts as an interface between the hardware and the software, specifying both what the processor is capable of doing as well as how it gets done. The ISA provides the only way through which a user is able to interact with the hardware. It can be viewed as a programmer's manual because it's the portion of the machine that's visible to the assembly language programmer, the compiler writer, and the application programmer.

The ISA defines the supported data types, the registers, how the hardware manages main memory, key features (such as virtual memory), which instructions a microprocessor can execute, and the input/output model of multiple ISA implementations. The ISA can be extended by adding instructions or other capabilities, or by adding support for larger addresses and data values.

2.0 OBJECTIVES

At the end of this unit, you should be able to

Understand the importance of the instruction set architecture,

Discuss the features that need to be considered when designing the instruction set architecture.

3.1 Instruction Set Overview

We've already seen that the computer architecture course consists of two components – the instruction set architecture and the computer organization itself. The ISA specifies what the processor is capable of doing and the ISA, how it gets accomplished. So the instruction set architecture is the interface between your hardware and the software. The only way that you can interact with the hardware is the instruction set of the processor. To command the computer, you need to speak its language and the instructions are the words of a computer's language and the instruction set is basically its vocabulary. Unless you know the vocabulary and you have a very good vocabulary, you cannot gain good benefits out of the machine. ISA is the portion of the machine which is visible to either the assembly language programmer or a compiler writer or an application programmer. It is the only interface that you have, because the instruction set architecture is the specification of what the computer can do and the machine has to be fabricated in such a way that it will execute whatever has been specified in your ISA. The only way that you can talk to your machine is through the ISA. This gives you an idea of the interface between the hardware and software.

Let us assume you have a high-level program written in C which is independent of the architecture on which you want to work. This high-level program has to be translated into an assembly language program which is specific to a particular architecture. Let us say you find that this consists of a number of instructions like LOAD, STORE, ADD, etc., where, whatever you had written in terms of high-level language now have been translated into a set of instructions which are specific to the specific architecture. All these instructions that are being shown here are part of the instruction set architecture of the MIPS architecture. These are all English like and this is not understandable to the processor because the processor is after all made up of digital components which can understand only zeros and ones. So this assembly language will have to be finely translated into machine language, object code which consists of zeros and ones. So the translation from your high-level language to your assembly language and the binary code will have to be done with the compiler and the assembler.

We shall look at the instruction set features, and see what will go into the zeros and ones and how to interpret the zeros and ones, as data, instructions, or addresses. The ISA that is designed should last through many implementations, it should have portability, it should have compatibility, it should be used in many different ways so it should have generality and it should also provide convenient functionality to other levels. The taxonomy of ISA is given below.

3.2 Taxonomy

ISAs differ based on the internal storage in a processor. Accordingly, the ISA can be classified as follows, based on where the operands are stored and whether they are named explicitly or implicitly:

Single accumulator organization, which names one of the general purpose registers as the accumulator and uses it to necessarily store one of the operands. This indicates that one of the operands is implied to be in the accumulator and it is enough if the other operand is specified along with the instruction.
General register organization, which specifies all the operands explicitly. Depending on whether the operands are available in memory or registers, it can be further classified as
– Register – register, where registers are used for storing operands. Such architectures are also called load–store architectures, as only load and store instructions can have memory operands.
– Register – memory, where one operand is in a register and the other one in memory.
– Memory – memory, where all the operands are specified as memory operands.
Stack organization, where the operands are put into the stack and the operations are carried out on the top of the stack. The operands are implicitly specified here.
Let us assume you have to perform the operation A = B + C, where all three operands are memory operands. In the case of an accumulator-based ISA, where we assume that one of the general-purpose registers is being designated as an accumulator and one of the operands will always be available in the accumulator, you have to initially load one operand into the accumulator and the ADD instruction will only specify the operand's address. In the GPR-based ISA, you have three different classifications. In the register memory ISA, One operand has to be moved into any register and the other one can be a memory operand. In the register–register ISA, both operands will have to be moved to two registers and the ADD instruction will only work on registers. The memory–memory ISA permits both memory operands. So you can directly add. In a stack-based ISA, you'll have to first of all push both operands onto the stack and then simply give an add instruction which will add the top two elements of the stack and then store the result in the stack. So you can see from these examples that you have different ways of executing the same operation, and it obviously depends upon the ISA. Among all these ISAs, It is the register – register ISA that is very popular and used in all RISC architectures.

We shall now look at what are the different features that need to be considered when designing the instruction set architecture. They are:
Types of instructions (Operations in the Instruction set)
Types and sizes of operands
Addressing Modes
Addressing Memory
Encoding and Instruction Formats
Compiler-related issues

First of all, you have to decide on the types of instructions, i.e. what are the various instructions that you want to support in the ISA. The tasks carried out by a computer program consisting of a sequence of small steps, such as multiplying two numbers, moving data from a register to a memory location, testing for a particular condition like zero, reading a character from the input device or sending a character to be displayed to the output device, etc.. A computer must have the following types of instructions:

- Data transfer instructions
- Data manipulation instructions
- Program sequencing and control instructions
- Input and output instructions

Data transfer instructions perform data transfer between the various storage places in the computer system, viz. registers, memory, and I/O. Since, both the instructions as well as data are stored in memory, the processor needs to read the instructions and data from memory. After processing, the results must be stored in memory. Therefore, two basic operations involving the memory are needed, namely, Load (or Read or Fetch) and Store (or Write). The Load operation transfers a copy of the data from the memory to the processor and the Store operation moves the data from the processor to memory. Other data transfer instructions are needed to transfer data from one register to another or from/to I/O devices and the processor.

Data manipulation instructions perform operations on data and indicate the computational capabilities for the processor. These operations can be arithmetic operations, logical operations or shift operations. Arithmetic operations include addition (with and without carry), subtraction (with and without borrow), multiplication, division, increment, decrement and finding the complement of a number. The logical and bit manipulation instructions include AND, OR, XOR, Clear carry, set carry, etc. Similarly, you can perform different types of shift and rotate operations.

We generally assume a sequential flow of instructions. That is, instructions that are stored in consequent locations are executed one after the other. However, you have program sequencing and control instructions that help you change the flow of the program. This is best explained with an example. Consider the task of adding a list of n numbers. A possible sequence is given below.

```
Move DATA1, R0
Add DATA2, R0
Add DATA3, R0
Add DATAn, R0
Move R0, SUM
```

The addresses of the memory locations containing the n numbers are symbolically given as DATA1, DATA2, . . , DATAn, and a separate Add instruction is used to add each Databer to the contents of register R0. After

all the numbers have been added, the result is placed in memory location SUM. Instead of using a long list of Add instructions, it is possible to place a single Add instruction in a program loop, as shown below:

Move N, R1
Clear R0
LOOP Determine address of "Next" number and add "Next" number to R0
Decrement R1
Branch > 0, LOOP
Move R0, SUM

The loop is a straight-line sequence of instructions executed as many times as needed. It starts at location LOOP and ends at the instruction Branch>0. During each pass through this loop, the address of the next list entry is determined, and that entry is fetched and added to R0. The address of an operand can be specified in various ways, as will be described in the next section. For now, you need to know how to create and control a program loop. Assume that the number of entries in the list, n, is stored in memory location N. Register R1 is used as a counter to determine the number of times the loop is executed. Hence, the contents of location N are loaded into register R1 at the beginning of the program. Then, within the body of the loop, the instruction, Decrement R1 reduces the contents of R1 by 1 each time through the loop. The execution of the loop is repeated as long as the result of the decrement operation is greater than zero.

You should now be able to understand branch instructions. This type of instruction loads a new value into the program counter. As a result, the processor fetches and executes the instruction at this new address, called the branch target, instead of the instruction at the location that follows the branch instruction in sequential address order. The branch instruction can be conditional or unconditional. An unconditional branch instruction does a branch to the specified address irrespective of any condition. A conditional branch instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed. In the example above, the instruction Branch>0 LOOP (branch if greater than 0) is a conditional branch instruction that causes a branch to locate LOOP if the result of the immediately preceding instruction, which is the decremented value in register R1, is greater than zero.

This means that the loop is repeated as long as there are entries in the list that are yet to be added to R0. At the end of the nth pass through the loop, the Decrement instruction produces a value of zero, and, hence, branching does not occur. Instead, the Move instruction is fetched and executed. It moves the final result from R0 into memory location SUM. Some ISAs

refer to such instructions as Jumps. The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions. This is accomplished by recording the required information in individual bits, often called condition code flags. These flags are usually grouped together in a special processor register called the condition code register or status register. Individual condition code flags are set to 1 or cleared to 0, depending on the outcome of the operation performed. Some of the commonly used flags are: Sign, Zero, Overflow, and Carry.

The call and return instructions are used in conjunction with subroutines. A subroutine is a self-contained sequence of instructions that performs a given computational task. During the execution of a program, a subroutine may be called to perform its function many times at various points in the main program. Each time a subroutine is called, a branch is executed to the beginning of the subroutine to start executing its set of instructions. After the subroutine has been executed, a branch is made back to the main program, through the return instruction. Interrupts can also change the flow of a program. A program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internally generated request. Control returns to the original program after the service program is executed.

The interrupt procedure is, in principle, quite similar to a subroutine call except for three variations: (1) The interrupt is usually initiated by an internal or external signal apart from the execution of an instruction (2) the address of the interrupt service program is determined by the hardware or from some information from the interrupt signal or the instruction causing the interrupt; and (3) an interrupt procedure usually stores all the information necessary to define the state of the CPU rather than storing only the program counter. Therefore, when the processor is interrupted, it saves the current status of the processor, including the return address, the register contents and the status information called the Processor Status Word (PSW), and then jumps to the interrupt handler or the interrupt service routine. Upon completing this, it returns to the main program. Interrupts are handled in detail in the next unit on Input / Output.

Input and Output instructions are used for transferring information between the registers, memory, and the input/output devices. It is possible to use special instructions that exclusively perform I/O transfers, or use memory – related instructions itself to do I/O transfers.

Suppose you are designing an embedded processor that is meant to be performing a particular application, then definitely you will have to bring instructions that are specific to that particular application. When you're designing a general-purpose processor, you only look at including all general types of instructions. Examples of specialized instructions may be media and signal processing-related instructions, say vector type of instructions which try to exploit the data level parallelism, where the same

operation of addition or subtraction is going to be done on different data and then you may have to look at saturating arithmetic operations, multiply and accumulator instructions.
The data types and sizes indicate the various data types supported by the processor and their lengths. Common operand types –Half word (16 bits), Word (32 bits), Single Precision Floating Point (1 Word), Double Precision Floating Point (2 Words), Integers – two's complement binary numbers, Characters usually in ASCII, Floating point numbers following the IEEE Standard 754 and Packed and unpacked decimal numbers.
Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What does ISA stand for?
   A. Internal System Architecture
   B. Instruction Set Architecture
   C. Integrated Software Application
   D. Input/Storage/Access

2. Which ISA type is used in RISC architectures?
   A. Accumulator-based
   B. Stack-based
   C. Register-register (Load-store)
   D. Memory-memory

3. What are the main categories of instructions in an ISA?
   A. Data transfer, data manipulation, program control, I/O
   B. Fetch, decode, execute, store
   C. Read, write, calculate, display
   D. Input, process, output, feedback

3.3 Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data that is given straight away or stored in computer registers or memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is referenced. In this section, you will learn the most important addressing modes found in modern processors.
Computers use addressing mode techniques to accommodate one or both of the following:
1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.
2. To reduce the number of bits in the addressing field of the instruction.

When you write programs in a high-level language, you use constants, local and global variables, pointers, and arrays. When translating a high-level language program into assembly language, the compiler must be able to implement these constructs using the facilities provided in the instruction set of the computer in which the program will be run. The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes. Variables and constants are the simplest data types and are found in almost every computer program. In assembly language, a variable is represented by allocating a register or a memory location to hold its value.

Register mode — The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

Absolute mode — The operand is in a memory location; the address of this location is given explicitly in the instruction. This is also called Direct.

Address and data constants can be represented in assembly language using the Immediate mode.

Immediate mode — The operand is given explicitly in the instruction. For example, the instruction Move 200immediate, R0 places the value 200 in register R0. Clearly, the Immediate mode is only used to specify the value of a source operand. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand. Hence, we write the instruction above in the form Move #200, R0. Constant values are used frequently in high-level language programs. For example, the statement A = B + 6 contains the constant 6. Assuming that A and B have been declared earlier as variables and may be accessed using the Absolute mode, this statement may be compiled as follows:

Move B, R1
Add #6, R1
Move R1, A

Constants are also used in assembly language to increment a counter, test for some bit pattern, and so on.

Indirect mode — In the addressing modes that follow, the instruction does not give the operand or its address explicitly. Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of the operand. In this mode, the effective address of the operand is the contents of a register or memory location whose address appears in the instruction. We denote indirection by placing the name of the register or the memory address given in the instruction in parentheses. For example, consider the instruction, Add (R1), R0. To execute the Add instruction, the processor uses the value in register R1 as the effective address of the operand. It requests a read operation from the memory to read the contents of this location. The value read is the desired operand, which the processor adds

to the contents of register R0. Indirect addressing through a memory location is also possible as indicated in the instruction Add (A), R0. In this case, the processor first reads the contents of memory location A, then requests a second read operation using this value as an address to obtain the operand. The register or memory location that contains the address of an operand is called a pointer. Indirection and the use of pointers are important and powerful concepts in programming. Changing the contents of location A in the example fetches different operands to add to register R0.

Index mode — The next addressing mode you learn provides a different kind of flexibility for accessing operands. It is useful in dealing with lists and arrays. In this mode, the effective address of the operand is generated by adding a constant value (displacement) to the contents of a register. The register used may be either a special register provided for this purpose, or may be any one of the general-purpose registers in the processor. In either case, it is referred to as an index register. We indicate the Index mode symbolically as X(Ri), where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by EA = X + [Ri]. The contents of the index register are not changed in the process of generating the effective address. In an assembly language program, the constant X may be given either as an explicit number or as a symbolic name representing a numerical value. When the instruction is translated into machine code, the constant X is given as a part of the instruction and is usually represented by fewer bits than the word length of the computer. Since X is a signed integer, it must be sign-extended to the register length before being added to the contents of the register.

Relative mode — The above discussion defined the Index mode using general-purpose processor registers. A useful version of this mode is obtained if the program counter, PC, is used instead of a general-purpose register. Then, X (PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter. Since the addressed location is identified as “relative” to the program counter, which always identifies the current execution point in a program, the name Relative mode is associated with this type of addressing. In this case, the effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri. This addressing mode is generally used with control flow instructions.

Though this mode can be used to access data operands. But, its most common use is to specify the target address in branch instructions. An instruction such as Branch > 0 LOOP, which we discussed earlier, causes program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied. This location can be computed by specifying it as an offset from the current value of the program counter. Since the branch target may be either before or after the branch instruction, the offset is given as a signed number. Recall that

during the execution of an instruction, the processor increments the PC to point to the next instruction. Most computers use this updated value in computing the effective address in the Relative mode.

The two modes described next are useful for accessing data items in successive locations in the memory.

Autoincrement mode — The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. We denote the Autoincrement mode by putting the specified register in parentheses, to show that the contents of the register are used as the effective address, followed by a plus sign to indicate that these contents are to be incremented after the operand is accessed. Thus, the Autoincrement mode is written as (Ri)+.

Autodecrement mode — As a companion for the Autoincrement mode, another useful mode accesses the items of a list in the reverse order. In the autodecrement mode, the contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand. We denote the Autodecrement mode by putting the specified register in parentheses, preceded by a minus sign to indicate that the contents of the register are to be decremented before being used as the effective address. Thus, we write – (Ri). In this mode, operands are accessed in descending address order. You may wonder why the address is decremented before it is used in the Autodecrement mode and incremented after it is used in the Autoincrement mode. The main reason for this is that these two modes can be used together to implement a stack.

3.4 Instruction Formats

The previous sections have shown you that the processor can execute different types of instructions and there are different ways of specifying the operands. Once all this is decided, this information has to be presented to the processor in the form of an instruction format. The number of bits in the instruction is divided into groups called fields. The most common fields found in instruction formats are

1. An operation code field that specifies the operation to be performed. The number of bits will indicate the number of operations that can be performed.
2. An address field that designates a memory address or a processor register. The number of bits depends on the size of memory or the number of registers.
3. A mode field that specifies the way the operand or the effective address is determined. This depends on the number of addressing modes supported by the processor.

The number of address fields may be three, two or one depending on the type of ISA used. Also, observe that, based on the number of operands that are supported and the size of the various fields, the length of the instructions will vary. Some processors fit all the instructions into a single

sized format, whereas others make use of formats of varying sizes. Accordingly, you have a fixed format or a variable format.

Interpreting memory addresses – you basically have two types of interpretation of the memory addresses – Big endian arrangement and the little endian arrangement. Memories are normally arranged as bytes and a unique address of a memory location is capable of storing 8 bits of information. But when you look at the word length of the processor, the word length of the processor may be more than one byte. Suppose you look at a 32-bit processor, it is made up of four bytes. These four bytes span over four memory locations. When you specify the address of a word how you would specify the address of the word – are you going to specify the address of the most significant byte as the address of the word (big end) or specify the address of the least significant byte (little end) as the address of the word. That distinguishes between a big endian arrangement and a little endian arrangement. IBM, Motorola, HP follow the big endian arrangement and Intel follows the little endian arrangement. Also, when a data spans over different memory locations, and if you try to access a word which is aligned with the word boundary, we say there is an alignment. If you try to access the words not starting at a word boundary, you can still access, but they are not aligned. Whether there is support to access data that is misaligned is a design issue. Even if you're allowed to access data that is misaligned, it normally takes more number of memory cycles to access the data.

Finally looking at the role of compilers the compiler has a lot of role to play when you're defining the instruction set architecture. Gone are the days where people thought that compilers and architectures are going to be independent of each other. Only when the compiler knows the internal architecture of the processor it'll be able to produce optimised code. So the architecture will have to expose itself to the compiler and the compiler will have to make use of whatever hardware is exposed. The ISA should be compiler friendly. The basic ways in which the ISA can help the compiler are regularity, orthogonality and the ability to weigh different options.

Finally, all the features of an ISA are discussed with respect to the 80×86 and MIPS.

1. Class of ISA: Nearly all ISAs today are classified as general-purpose register architectures, where the operands are either registers or memory locations. The 80×86 has 16 general-purpose registers and 16 that can hold floating point data, while MIPS has 32 general-purpose and 32 floating-point registers. The two popular versions of this class are register-memory ISAs such as the 80×86, which can access memory as part of many instructions, and load-store ISAs such as MIPS, which can access memory only with load or store instructions. All recent ISAs are load-store.
2. Memory addressing: Virtually all desktop and server computers, including the 80×86 and MIPS, use byte addressing to access memory

operands. Some architectures, like MIPS, require that objects must be aligned. An access to an object of size s bytes at byte address A is aligned if A mod s = 0. The 80×86 does not require alignment, but accesses are generally faster if operands are aligned.

3. Addressing modes: In addition to specifying registers and constant operands, addressing modes specify the address of a memory object. MIPS addressing modes are Register, Immediate (for constants), and Displacement, where a constant offset is added to a register to form the memory address. The 80×86 supports those three plus three variations of displacement: no register (absolute), two registers (based indexed with displacement), two registers where one register is multiplied by the size of the operand in bytes (based with scaled index and displacement). It has more like the last three, minus the displacement field: register indirect, indexed, and based with scaled index.

4. Types and sizes of operands: Like most ISAs, MIPS and 80×86 support operand sizes of 8-bit (ASCII character), 16-bit (Unicode character or half word), 32-bit (integer or word), 64-bit (double word or long integer), and IEEE 754 floating point in 32-bit (single precision) and 64-bit (double precision). The 80×86 also supports 80-bit floating point (extended double precision).

5. Operations: The general categories of operations are data transfer, arithmetic logical, control, and floating point. MIPS is a simple and easy-to-pipeline instruction set architecture, and it is representative of the RISC architectures being used in 2006. The 80×86 has a much richer and larger set of operations.

6. Control flow instructions: Virtually all ISAs, including 80×86 and MIPS, support conditional branches, unconditional jumps, procedure calls, and returns. Both use PC-relative addressing, where the branch address is specified by an address field that is added to the PC. There are some small differences. MIPS conditional branches (BE, BNE, etc.) test the contents of registers, while the 80×86 branches (JE, JNE, etc.) test condition code bits set as side effects of arithmetic/logic operations. MIPS procedure call (JAL) places the return address in a register, while the 80×86 call (CALLF) places the return address on a stack in memory.

7. Encoding an ISA : There are two basic choices for encoding: fixed length and variable length. All MIPS instructions are 32 bits long, which simplifies instruction decoding (shown below). The 80×86 encoding is variable length, ranging from 1 to 18 bytes. Variable-length instructions can take less space than fixed-length instructions, so a program compiled for the 80×86 is usually smaller than the same program compiled for MIPS. Note that the choices mentioned above will affect how the instructions are encoded into a binary representation. For example, the number of registers and the number of addressing modes both have a significant impact on the size of instructions, as the register field and addressing mode field can appear many times in a single instruction.

some types of instruction sets?

The various types of instruction sets include the following:
Complex instruction set computer. CISC processors have an additional microcode or microprogramming layer where instructions act as small programs. Programmable instructions are stored in fast memory and can be updated. More instructions are included in CISC instruction sets than in other types of instruction sets. A single instruction can initiate multiple actions by the computer, such as a single add command launching multiple memory access load and store instructions.
Reduced instruction set computer. RISC architecture has hard-wired control. It does not require a microcode but has a greater base instruction set. RISC also uses a smaller and more compact instruction set with a fixed instruction format. RISC processors are designed to process faster and more efficiently.
Enhancement instruction sets. These instruction types are more familiar because they are often used in marketing CPUs. Examples of this go back to the 166-megahertz Intel Pentium with Multimedia Extensions (MMX) technologies. It was introduced in 1996 and marketed with enhanced Intel CPU multimedia performance. MMX refers to the extended instruction set.

Self-Assessment Exercises 2

Fill in the gaps in the sentences below with the most suitable words:

1. The _____ mode specifies that the operand is given explicitly in the instruction.

2. In _____ mode, the effective address is the contents of a register specified in the instruction.

3. The _____ addressing mode uses the program counter to address memory locations relative to the current instruction.

4.0 CONCLUSION

Basically means that an **ISA** describes the **design of a Computer** in terms of the **basic operations** it must support. The ISA is not concerned with the implementation-specific details of a computer. It is only concerned with the set or collection of basic operations the computer must support. For example, the AMD Athlon and the Core 2 Duo processors have entirely different implementations but they support more or less the same set of basic operations as defined in the x86 Instruction Set.

5.0 SUMMARY

An instruction set is a group of commands for a CPU in machine language. The term can refer to all possible instructions for a CPU or a subset of instructions to enhance its performance in certain situations. To

summarize, we have looked at the taxonomy of ISAs and the various features that need to be decided while designing the ISA. We also looked at example ISAs, the MIPS ISA and the 80×86 ISA.

6.0 Tutor marked assignment

1. What is a instruction set?

6.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. B
2. C
3. A

Self-Assessment Exercise 2
1. Immediate
2. Register
3. Relative

7.0 REFERENCES/ FURTHER READING

Computer Architecture – A Quantitative Approach , John L. Hennessy and David A. Patterson, 5th.Edition, Morgan Kaufmann, Elsevier, 2011.
Computer Organization and Design – The Hardware / Software Interface, David A. Patterson and John L. Hennessy, 4th.Edition, Morgan Kaufmann, Elsevier, 2009.
Computer Organization, Carl Hamacher, Zvonko Vranesic and Safwat Zaky, 5th.Edition, McGraw-Hill Higher Education, 2011.

UNIT 2 INSTRUCTION CYCLE

CONTENT

1.0 INTRODUCTION

The instruction cycle is a basic computer system that deals with the central processor unit's core operations. It is also known as the fetch-decode-execute cycle, and is a fundamental concept in computer architecture and microprocessor operation. It represents the series of steps that a computer's central processing unit (CPU) goes through to execute a single-machine instruction.

2.0 OBJECTIVES

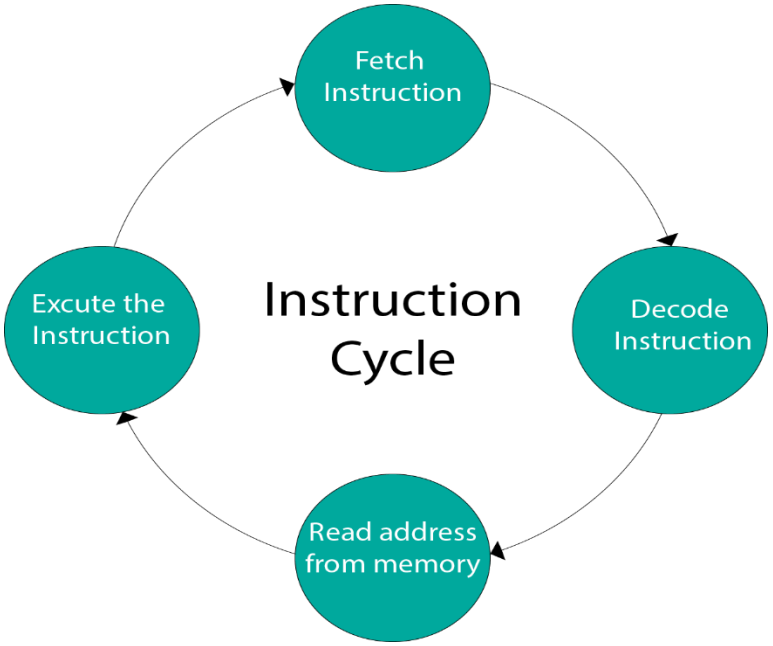
At the end of this unit, you should be able to
understand the instruction cycle.

3.1 Instruction Cycle

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction. An instruction cycle, also known as the fetch-decode-execute cycle is the basic operational process of a computer. This process is repeated continuously by the CPU from boot up to shut down of the computer.

In a basic computer, each instruction cycle consists of the following phases:

Fetch instruction from memory.
Decode the instructions.
Read the effective address from memory.
Execute the instruction.



During this phase, the computer system boots up and the Operating System loads into the central processing unit's main memory. It begins when the computer system starts.

Following are the steps that occur during an instruction cycle:

1. Fetch the Instruction

The first phase is instruction retrieval. Each instruction executed in a central processing unit uses the fetch instruction. During this phase, the central processing unit sends the PC to MAR and then the READ instruction to a control bus. After sending a read instruction on the data bus, the memory returns the instruction that was stored at that exact address in the memory. The CPU then copies data from the data bus into MBR, which it then copies to registers. The pointer is incremented to the next memory location, allowing the next instruction to be fetched from memory. The instruction is fetched from memory address that is stored in PC (Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

2. Decode the Instruction

The second phase is instruction decoding. During this step, the CPU determines which instruction should be fetched from the instruction and what action should be taken on the instruction. The instruction's opcode is also retrieved from memory, and it decodes the related operation that must be performed for the instruction. The instruction in the IR is executed by the decoder.

3. Read the Effective Address

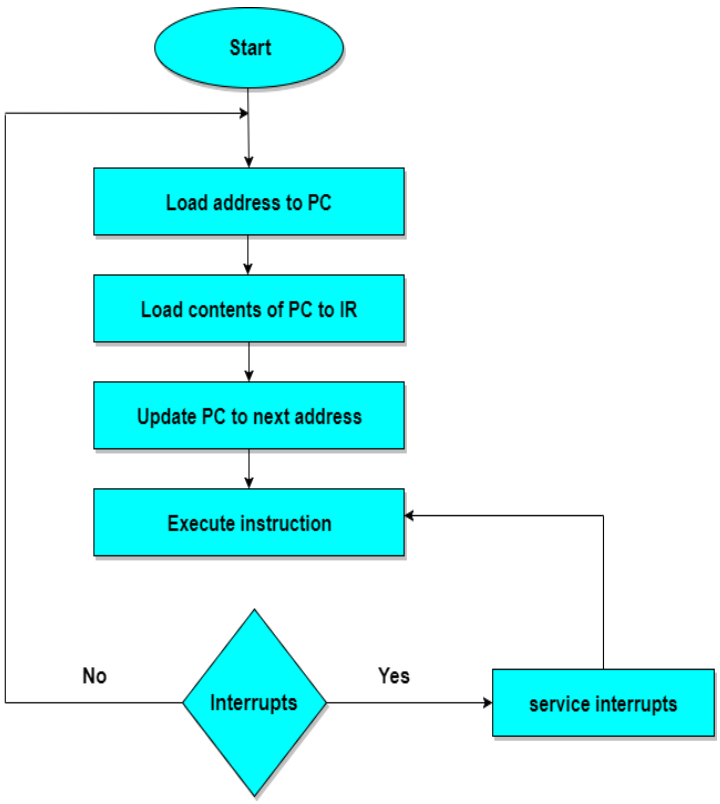
The third phase is the reading of an effective address. The operation's decision is made during this phase. Any memory-type operation or non-memory-type operation can be used. Direct memory instruction and

indirect memory instruction are the two types of memory instruction available. If the instruction has an indirect address, the effective address is read from the memory. Otherwise, operands are directly read in case of immediate operand instruction.

4. Execute the Instruction

The last step is to carry out the instructions. The instruction is finally carried out at this stage. The instruction is carried out, and the result is saved in the register. The CPU gets prepared for the execution of the next instruction after the completion of each instruction. The execution time of each instruction is calculated, and this information is used to determine the processor's processing speed. The Control Unit passes the information in the form of control signals to the functional unit of the CPU. The result generated is stored in the main memory or sent to an output device.

The cycle is then repeated by fetching the next instruction. Thus in this way, the instruction cycle is repeated continuously.



The sequence of operations performed by the CPU during its execution of instructions is presented in the figure. As long as there are instructions to execute, the next instruction is fetched from the main memory. The instruction is executed based on the operation specified in the opcode field of the instruction. After the instruction execution, a test is made to determine whether an interrupt has occurred. An interrupt handling routine needs to be invoked in case of an interrupt.

3.2 Different Instruction Cycles

The concept of instruction cycles is integral to understanding how a computer's central processing unit (CPU) executes instructions. Here's an explanation of each of these cycles:

Fetch Cycle

Description: The fetch cycle is the initial stage of the instruction cycle. It involves retrieving the next instruction from memory.

Operation: The CPU uses the program counter (PC) to access the memory location where the next instruction is stored. The instruction is fetched and placed in the instruction register (IR).

Purpose: This cycle ensures that the CPU has the next instruction ready for decoding and execution.

Indirect Cycle

Description: The indirect cycle is sometimes required when instructions involve accessing memory locations that contain addresses or pointers to the actual data.

Operation: During this cycle, the CPU may use an address obtained from the previous instruction to access another memory location, which holds the data or another address to be used in the next cycle.

Purpose: The indirect cycle enables the CPU to follow memory references and retrieve the actual data required for execution.

Execute Cycle

Description: The execute cycle is where the central processing unit performs the operation specified by the decoded instruction.

Operation: The CPU carries out arithmetic computations, logical operations, data transfers, or any other actions as dictated by the instruction. This may involve accessing data from registers or memory, performing calculations, and updating registers or memory locations.

Purpose: The execution stage accomplishes the intended operation and is where the actual work of the instruction takes place.

Interrupt Cycle

Description: The interrupt cycle comes into play when an external event or condition triggers an interrupt, causing the CPU to temporarily suspend its current execution to handle the interrupt request.

Operation: The CPU saves its current state (program counter and other relevant information) before jumping to an interrupt service routine (ISR). After servicing the interrupt, the CPU may restore its state and continue execution.

Purpose: Interrupt cycles enable a CPU to respond to external events or asynchronous inputs promptly without losing important data or program context.

Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. How many basic phases does an instruction cycle consist of?

A. Two (fetch and execute)
B. Three (fetch, decode, execute)
C. Four (fetch, decode, execute, store)
D. Five (fetch, decode, read, execute, store)

2. What happens during the decode phase?
A. The instruction is retrieved from memory
B. The CPU determines what operation to perform
C. The instruction is executed
D. The result is stored in memory

3. Which cycle handles external events that interrupt normal processing?
A. Fetch Cycle
B. Execute Cycle
C. Indirect Cycle
D. Interrupt Cycle

3.3 Uses of Different Instruction Cycles

The different instruction cycles (fetch, indirect, execute, and interrupt) in a computer's operation have different purposes and applications, ensuring efficient and responsive processing. Here are the uses of each instruction cycle:

Fetch Cycle

Use: Retrieving the next instruction from memory.

Application: Essential for the sequential execution of program instructions, ensuring the CPU has the next instruction ready for decoding and execution.

Example: Fetching the opcode of the next instruction from memory to be decoded and executed.

Indirect Cycle

Use: Handling instructions that involve accessing memory locations containing addresses or pointers.

Application: Facilitates memory referencing, allowing the CPU to navigate through multiple levels of indirection to access the actual data or instructions.

Example: Accessing data through a memory location that contains a pointer to the actual data's location.

Execute Cycle

Use: Performing the operation specified by the decoded instruction.

Application: Where the actual computation or data manipulation occurs, making it the heart of instruction execution.

Example: Carrying out arithmetic calculations, logical operations, data transfers, or any actions dictated by the instruction.

Interrupt Cycle

Use: Handling external events or requests for interrupting the CPU's current execution.

Application: Ensures prompt response to hardware or software events such as hardware interrupts, system calls, or exceptions, allowing the CPU to temporarily switch tasks.
Example: Responding to a keyboard input interrupt, saving the CPU's current state, and invoking an interrupt service routine (ISR).
Why do we Need an Instruction Cycle?
The instruction cycle of a computer system is necessary for understanding the flow of instructions and the execution of an instruction in a computer processor.
It is responsible for the complete flow of instructions from the start of the computer system through its shutdown. The instruction cycle helps to understand the internal flow of the central processing unit, allowing any faults to be immediately resolved.
It deals with a computer processor's basic operations and demands a detailed understanding of the many steps involved.
All instructions for the computer processor system follow the fetch-decode-execute cycle.
Importance of Instruction Cycle
The instructions are the basic activities conducted in the main memory of the central processing unit. That is why they are so crucial to the processor system.
It's a set of stages that helps us to understand how instruction flows. The instruction cycle allows the computer processor to see the sequence of instructions from start to finish.
It is common for all instruction sets to require a thorough understanding to perform all operations efficiently.
The processing time of a programme can be easily calculated using the instruction cycle, which aids in determining the processor's speed.
The processor's speed determines how many instructions can be executed simultaneously in the central processing unit.
Advantages of Instruction Cycle
Efficiency: The fetch-decode-execute cycle, consisting of instruction cycles, allows CPUs to execute instructions sequentially and efficiently, ensuring that each instruction is processed in a well-defined manner.
Flexibility: CPUs can handle a wide range of instructions, from arithmetic operations to data transfers, by following the execution cycle for each instruction type.
Control Flow: The instruction cycle controls the flow of program execution, advancing to the next instruction after each cycle, allowing for precise execution and program control.
Responsiveness: CPUs can quickly respond to external events and handle interrupts or exceptions using the interrupt cycle, making them versatile and suitable for various tasks.
Disadvantages of Instruction Cycle
Clock Speed: The speed of instruction execution is often constrained by the system's clock speed, limiting the number of instructions that can be

executed in a given period.

Pipeline Stalls: In pipelined architectures, where multiple instructions are processed simultaneously, issues like pipeline stalls can lead to inefficiencies if instructions depend on one another.

Resource Limitations: CPU execution is subject to resource limitations, such as the availability of registers, memory access times, and cache sizes, which can affect performance.

Instruction Set Limitations: CPUs are limited by their instruction set architectures (ISAs), which may not include certain specialized instructions or features required for specific applications.

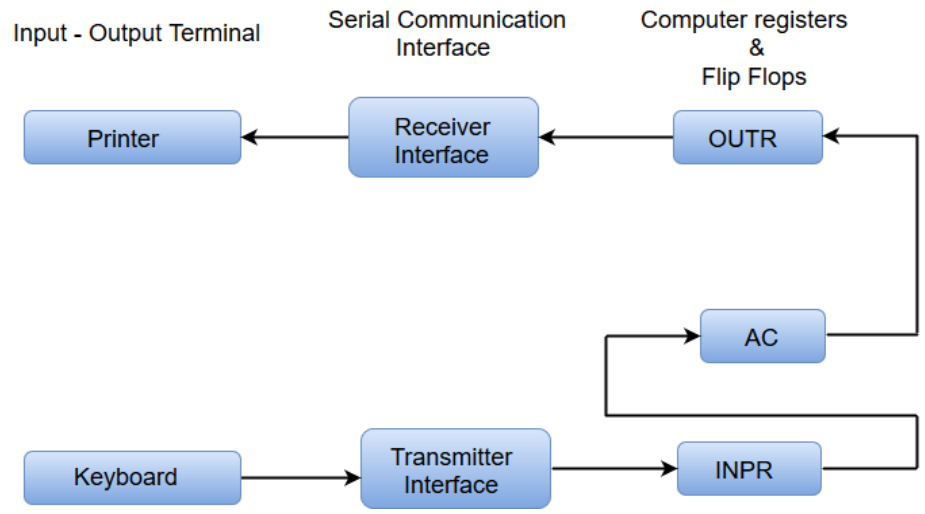
Complexity: The fetch-decode-execute cycle is an intricate process, and the complexity of instruction execution can lead to design challenges and potential errors in the processor's microarchitecture.

Input-Output Configuration

In computer architecture, input-output devices act as an interface between the machine and the user.

Instructions and data stored in the memory must come from some input device. The results are displayed to the user through some output device. The following block diagram shows the input-output configuration for a basic computer.

Input - Output Configuration:



- The input-output terminals send and receive information.
- The amount of information transferred will always have eight bits of an alphanumeric code.
- The information generated through the keyboard is shifted into an input register 'INPR'.
- The information for the printer is stored in the output register 'OUTR'.
- Registers INPR and OUTR communicate with a communication

interface serially and with the AC in parallel.

- The transmitter interface receives information from the keyboard and transmits it to INPR.
- The receiver interface receives information from OUTR and sends it to the printer serially.

4.0 CONCLUSION

we have thoroughly discussed on Instruction Cycle in Computer Architecture. We also learned about different instruction cycles, the uses of different instruction cycles, their needs, and their importance. Later in the end we discussed the advantages and disadvantages of the instruction Cycle in Computer Architecture.

5.0 SUMMARY

In computer organization, an instruction cycle, also known as a fetch-decode-execute cycle, is the basic operation performed by a CPU to execute an instruction. The instruction cycle consists of several steps, each of which performs a specific function in the execution of the instruction. The major steps in the instruction cycle are:

Fetch: In the fetch cycle, the CPU retrieves the instruction from memory. The instruction is typically stored at the address specified by the program counter (PC). The PC is then incremented to point to the next instruction in memory.

Decode: In the decode cycle, the CPU interprets the instruction and determines what operation needs to be performed. This involves identifying the opcode and any operands that are needed to execute the instruction.

Execute: In the execute cycle, the CPU performs the operation specified by the instruction. This may involve reading or writing data from or to memory, performing arithmetic or logic operations on data, or manipulating the control flow of the program.

Some additional steps may be performed during the instruction cycle, depending on the CPU architecture and instruction set:

Fetch operands: In some CPUs, the operands needed for an instruction are fetched during a separate cycle before the execute cycle. This is called the fetch operands cycle.

Store results: In some CPUs, the results of an instruction are stored during a separate cycle after the execute cycle. This is called the store results cycle.

Interrupt handling: In some CPUs, interrupt handling may occur during any cycle of the instruction cycle. An interrupt is a signal that the CPU receives from an external device or software that requires immediate attention. When an interrupt occurs, the CPU suspends the current instruction and executes an interrupt handler to service the interrupt.

6.0 Tutor marked assignment

1. What is the Instruction cycle?
2. What is five stage instruction cycle?
3. Why is instruction cycle important?
4. What are the steps of the instructional cycle?

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. C
2. B
3. D

7.0 REFERENCES/ FURTHER READING

Computer Architecture – A Quantitative Approach , John L. Hennessy and David A. Patterson, 5th.Edition, Morgan Kaufmann, Elsevier, 2011.
Computer Organization and Design – The Hardware / Software Interface, David A. Patterson and John L. Hennessy, 4th.Edition, Morgan Kaufmann, Elsevier, 2009.
Computer Organization, Carl Hamacher, Zvonko Vranesic and Safwat Zaky, 5th.Edition, McGraw-Hill Higher Education, 2011.

MODULE 5 THE MEMORY SYSTEMS

UNIT 1 COMPUTER MEMORY

1.0 INTRODUCTION

A computer is an electronic device and that accepts data, processes that data, and gives the desired output. It performs programmed computation with accuracy and speed. In other words, the computer takes data as input and stores the data/instructions in the memory (use them when required). After processing the data, it converts into information. Finally, gives the output. Here, input refers to the raw data that we want the machine to process and return to us as a result, output refers to the response that the machine provides in response to the raw data entered and the processing of data may involve analyzing, searching, distributing, storing data, etc. Thus, we can also call a computer data processing system.

2.0 OBJECTIVES

At the end of this unit, you should be able to

- Understand the memory characteristics and organization
- Explain the types of memory

3.1 Memory Characteristics and Organization

Memory is one of the important subsystems in a Computer. It is a volatile storage system that stores Instructions and Data. Unless the program gets loaded in memory in executable form, the CPU cannot execute it. CPU Interacts closely with memory for execution.

There are many other storage systems in a computer that share the characteristics of memory. So why have so many storage systems? Everyone desires to have very large, super fast, and cheap storage. Storage cost varies depending on the type of storage. Memory devices are hierarchically connected to design a cost-effective memory. When we say memory, we refer to the main memory, commonly referred to as RAM.

Memory (Storage Device) Characteristics

Although Memory and Storage devices share many characteristics, there is uniqueness in each one of them. Some of the most important characteristics are as below:

Access Time - The access time depends on the physical nature of the storage medium and the access mechanisms used. Refer to Figure 1. At the bottom is access time in Milliseconds, while at the top of the triangle, it is less than 10 ns.

For memory, the access time can be calculated as the time difference between the request to the memory and the service by memory.

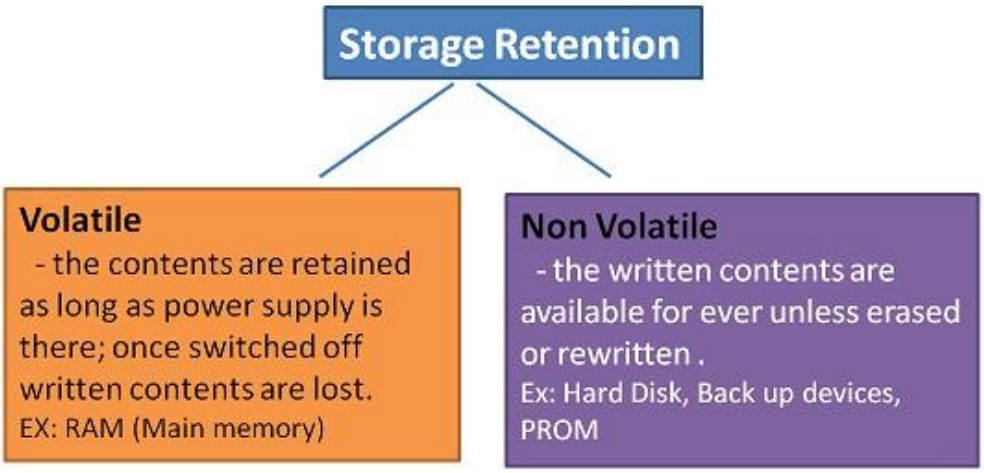
Access Mode - Access mode is a function of both memory organization and the inherent characteristics of the storage technology of the device. Access mode has relevance to the access time. There are three types of access methods.

Random Access: If storage locations can be accessed in any order then access time is independent of the storage location being accessed. Ex: Semiconductor memory.

Serial Access: Memory whose storage locations can be accessed only in a certain predetermined sequence. Ex: Magnetic tape

Semi Random: The access is partly random and there apart serial. Ex: Hard disk, CD drives. It is random to locate the tracks and access within the track is serial.

Retention - This is the characteristic of memory relating to the availability of written data for reading at a later time. Retention is a very important characteristic in the design of a system.



Cycle Time - Is defined as the minimum time between two consecutive access operations. This is greater than the access time. Generally, when once access is over, there is a time gap required to start the next access, although minimal. Cycle time = Access time + defined time delay. Ex: You ask the shop keeper of what is the speed of the memory strip.

Capacity - Measured in Units of Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Petabytes. In figure 1, the bottom of the triangle

has a larger capacity and the ones at the top have the far lesser capacity. Ex: the Memory strip as 2GB, 4GB, Hard disk as 1TB, GPRs are 128 words.

Cost Per bit – Factors of cost per bit are Access time, Cycle time, Storage capacity, the purchase cost of the device and the hardware to use the device (controller). We don't have much choice on this; designers care for this.

Reliability – It is related to the lifetime of the device. Measured as Mean Time Between Failure (MTBF), in the units of days/years. Ex: Think of how frequently you replace your Hard disk while the CPU is still usable.

There is a capacity/performance/price gap between each pair of adjacent levels of storage types (Refer figure 1). The objective of multilevel memory organisation is to achieve a good trade-off between cost, storage capacity and performance for the memory system as a whole.

Multilevel hierarchical memory is based on the principle of Locality of Reference i.e. the address generated by a program tend to be localised to successive address locations and therefore predictable. In figure 1, the unit of data movement between successive levels is also inscribed.

CPU Memory Interface

Level 0 to Level 3 of the storage devices are volatile memory subsystems which are accessed by CPU directly. The Level 4 and level 5 are storage devices which are classified as I/O devices and will be dealt with later as a separate category. So let us see about the CPU Memory Interface basics. The CPU interacts with memory for two operations i.e READ or WRITE. READ is for getting either instructions or Data (Operands). Write is generally for writing results upon instruction execution. To access memory, the address of the memory location is required. This address is always loaded in the Memory Address Register (MAR) by the CPU. READ or WRITE operation is always carried out on the location specified by MAR. In the case of READ, the memory returns the data to the CPU while in the case of WRITE the data to be written onto the memory location is given by CPU. The data exchange happens via the Memory Data Register (MDR). The CPU communicates to the memory about the READ or WRITE activity as control signals. Also, some more signals to time the validity of information on the Address bus and Data bus are part of Control Signals.

The communication about the address and data and the associated Control signals happen in the bus. A bus is a set of physical connections between two entities used for communication using electrical signals. This external bus has three components namely,(i) Address bus, (ii) Data bus, and (iii) Control Signals. Memory Address Register (MAR) and the Memory Data Register (MDR) play an important role in communication. The control signals are generated by the Control Unit. For more clarity refer to figure 16.

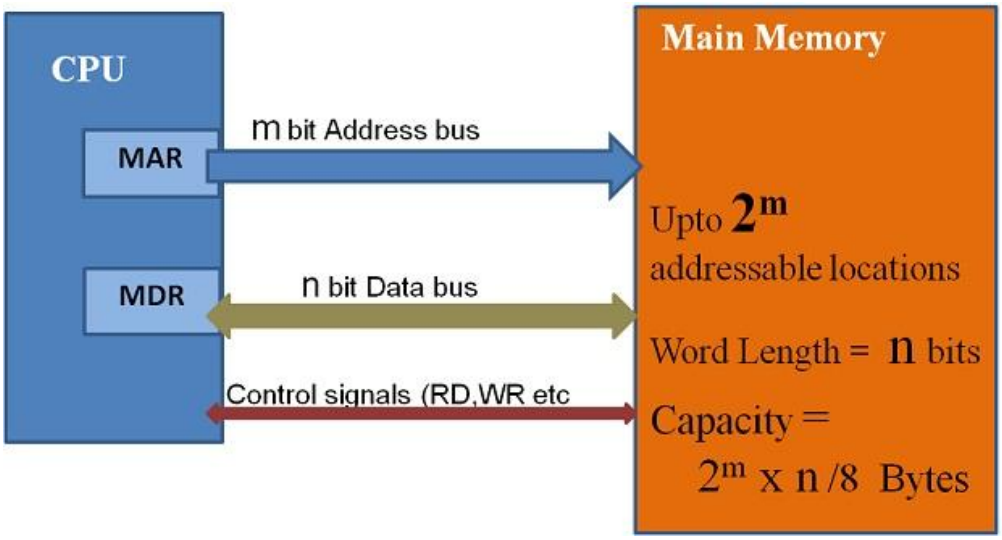


Figure 16: CPU Memory Communication Interface

Please note that the address bus is unidirectional and the data bus is bidirectional for obvious reasons discussed above. The control bus is also bidirectional. Further, the width of the address bus and data bus have critical meaning. The CPU can READ or WRITE data equal to the width of the data bus in one access. Generally, the width of the data bus equals the CPU word width. The width or the number of bits in the address bus has a bearing on the maximum number of locations that can be addressed or accessed by CPU. The signals on the bus are synchronised with the CPU clock.

Data transfer rate or bandwidth is one of the measures of the performance of the external bus between CPU and Memory. The maximum amount of information that can be transferred to or from the memory per unit time is the data transfer rate or bandwidth and is measured in bits or words per second.

Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. Which type of memory is volatile?
   A. ROM
   B. RAM
   C. Flash memory
   D. Hard disk

2. What does DRAM stand for?
   A. Direct Random Access Memory
   B. Dynamic Random Access Memory
   C. Dual Random Access Memory
   D. Digital Random Access Memory

3. Which characteristic describes how long it takes to access data in memory?

A. Capacity
B. Access Time
C. Cycle Time
D. Retention

Memory Capacity Integration

Memory is often available in standard capacity strips or modules. More often we need to integrate these modules to meet our requirement. When more than a strip is assembled, how do the expansion and chaos-free access happen is a curiosity. We will see now.

A typical memory module has the interface as shown in figure 17. This is in line with the signals on the external bus. A mention is required on RD/WR' and CS'. RD/WR' is a signal for READ or WRITE operation in mutual exclusion. When the signal is logical HIGH it is READ operation and when Logical LOW, WRITE is enabled on the Memory Module. CS' is Chip Select and active LOW i.e when this signal is logical LOW, only then the module is enabled and any operation can be done on this module. This Chip Select signal is useful in memory expansion. When RD is active, DataOUT comes from the module, while WR' is active the direction of data is DATA-IN.

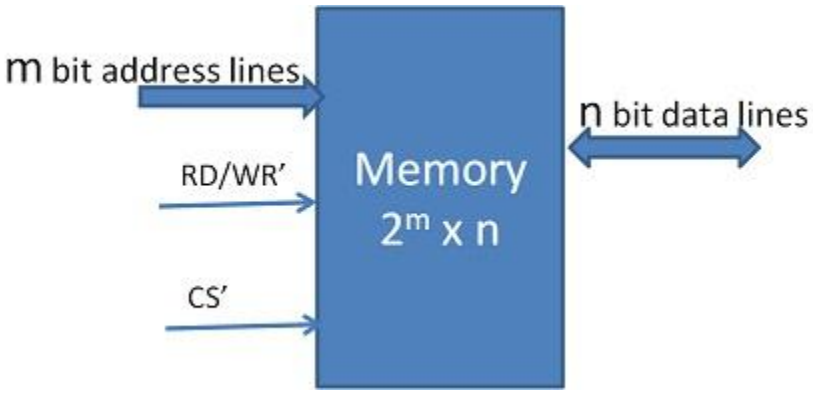


Figure 17: Typical Memory Module Interface

Memory expansion to the desired capacity is achieved by two means:
Increasing the word width by a factor (Refer figure 18)

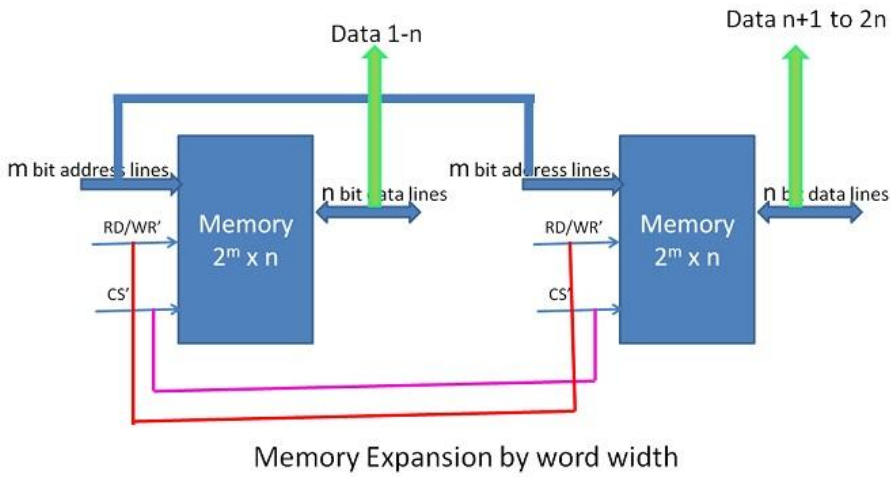


Figure 18: Memory expansion by word width

Increasing the Number of Words (address) by a Factor (Refer figure 19)

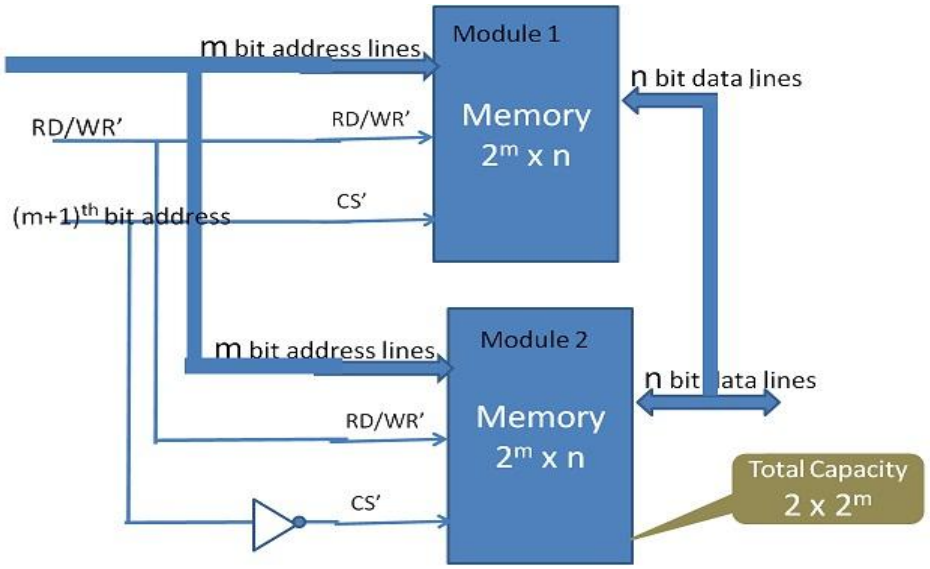


Figure 19: Memory expansion by address range

When the capacity is expanded to increase the addressable range, the CS signal plays a role in selecting the correct block. The MSB bit(s) of the address is(are) decoded and connected to each module as CS' enable. In figure 19, a simple inverter (NOT logic) is used on the MSB line as there are only 2 modules. If there are more modules then a decoder is required. This kind of extrapolation is feasible to any capacity in multiples of the basic module.

3.2 Types of Computer Memory

In general, computer memory is of three types:
Primary memory
Secondary memory
Cache memory
Now we discuss each type of memory one by one in detail:
1. Primary Memory
It is also known as the main memory of the computer system. It is used to store data and programs or instructions during computer operations. It uses semiconductor technology and hence is commonly called semiconductor memory. Primary memory is of two types:
RAM (Random Access Memory): It is a volatile memory. Volatile memory stores information based on the power supply. If the power supply fails/is interrupted/stopped, all the data and information on this memory will be lost. RAM is used for booting up or starting the computer. It temporarily stores programs/data which has to be executed by the processor. RAM is of two types:
S RAM (Static RAM): S RAM uses transistors and the circuits of this memory are capable of retaining their state as long as the power is applied. This memory consists of the number of flip flops with each flip flop storing 1 bit. It has less access time and hence, it is faster.
DRAM (Dynamic RAM): D RAM uses capacitors and transistors and stores the data as a charge on the capacitors. They contain thousands of memory cells. It needs refreshing of charge on the capacitor after a few milliseconds. This memory is slower than S RAM.
ROM (Read Only Memory): It is a non-volatile memory. Non-volatile memory stores information even when there is a power supply failed/ interrupted/stopped. ROM is used to store information that is used to operate the system. As its name refers to read-only memory, we can only read the programs and data that is stored on it. It contains some electronic fuses that can be programmed for a piece of specific information. The information stored in the ROM in binary format. It is also known as permanent memory. ROM is of four types:
MROM(Masked ROM): Hard-wired devices with a pre-programmed collection of data or instructions were the first ROMs. Masked ROMs are a type of low-cost ROM that works in this way.
PROM (Programmable Read Only Memory): This read-only memory is modifiable once by the user. The user purchases a blank PROM and uses a PROM program to put the required contents into the PROM. Its content can’t be erased once written.
EPROM (Erasable Programmable Read Only Memory): EPROM is an extension to PROM where you can erase the content of ROM by exposing it to Ultraviolet rays for nearly 40 minutes.
EEPROM (Electrically Erasable Programmable Read Only Memory): Here the written contents can be erased electrically. You can delete and reprogramme EEPROM up to 10,000 times. Erasing and

programming take very little time, i.e., nearly 4 -10 ms(milliseconds). Any area in an EEPROM can be wiped and programmed selectively.

2. Secondary Memory

It is also known as auxiliary memory and backup memory. It is a non-volatile memory and used to store a large amount of data or information. The data or information stored in secondary memory is permanent, and it is slower than primary memory. A CPU cannot access secondary memory directly. The data/information from the auxiliary memory is first transferred to the main memory, and then the CPU can access it.

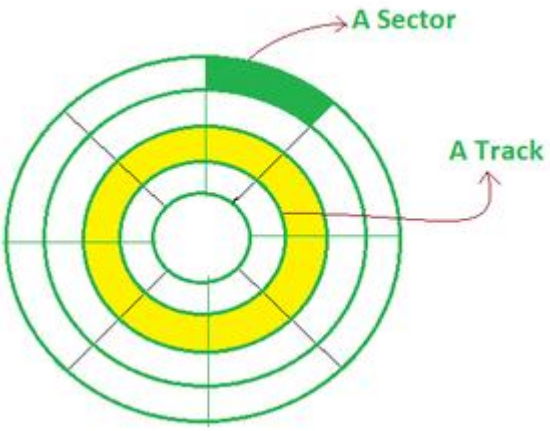
Characteristics of Secondary Memory

It is a slow memory but reusable.
It is a reliable and non-volatile memory.
It is cheaper than primary memory.
The storage capacity of secondary memory is large.
A computer system can run without secondary memory.
In secondary memory, data is stored permanently even when the power is off.

Types of Secondary Memory

1. Magnetic Tapes: Magnetic tape is a long, narrow strip of plastic film with a thin, magnetic coating on it that is used for magnetic recording. Bits are recorded on tape as magnetic patches called RECORDS that run along many tracks. Typically, 7 or 9 bits are recorded concurrently. Each track has one read/write head, which allows data to be recorded and read as a sequence of characters. It can be stopped, started moving forward or backward or rewound.

2. Magnetic Disks: A magnetic disk is a circular metal or a plastic plate and these plates are coated with magnetic material. The disc is used on both sides. Bits are stored in magnetized surfaces in locations called tracks that run in concentric rings. Sectors are typically used to break tracks into pieces.



Hard discs are discs that are permanently attached and cannot be removed by a single user.

3. Optical Disks: It's a laser-based storage medium that can be written to and read. It is reasonably priced and has a long lifespan. The optical disc can be taken out of the computer by occasional users.

Types of Optical Disks

CD – ROM

It's called a compact disk. Only read from memory.

Information is written to the disc by using a controlled laser beam to burn pits on the disc surface.

It has a highly reflecting surface, which is usually aluminium.

The diameter of the disc is 5.25 inches.

16000 tracks per inch is the track density.

The capacity of a CD-ROM is 600 MB, with each sector storing 2048 bytes of data.

The data transfer rate is about 4800KB/sec. & the new access time is around 80 milliseconds.

WORM-(WRITE ONCE READ MANY)

A user can only write data once.

The information is written on the disc using a laser beam.

It is possible to read the written data as many times as desired.

They keep lasting records of information but access time is high.

It is possible to rewrite updated or new data to another part of the disc.

Data that has already been written cannot be changed.

Usual size – 5.25 inch or 3.5 inch diameter.

The usual capacity of 5.25 inch disk is 650 MB,5.2GB etc.

DVDs

The term "DVD" stands for "Digital Versatile/Video Disc," and there are two sorts of DVDs:

DVDR (writable)

DVDRW (Re-Writable)

DVD-ROMS (Digital Versatile Discs): These are read-only memory (ROM) discs that can be used in a variety of ways. When compared to CD-ROMs, they can store a lot more data. It has a thick polycarbonate plastic layer that serves as a foundation for the other layers. It's an optical memory that can read and write data.

DVD-R: DVD-R is a writable optical disc that can be used just once. It's a DVD that can be recorded. It's a lot like WORM. DVD-ROMs have capacities ranging from 4.7 to 17 GB. The capacity of 3.5 inch disk is 1.3 GB.

3. Cache Memory

It is a type of high-speed semiconductor memory that can help the CPU run faster. Between the CPU and the main memory, it serves as a buffer. It is used to store the data and programs that the CPU uses the most frequently.

Advantages of Cache Memory
It is faster than the main memory.
When compared to the main memory, it takes less time to access it.
It keeps the programs that can be run in a short amount of time.
It stores data in temporary use.
Disadvantages of Cache Memory
Because of the semiconductors used, it is very expensive.
The size of the cache (amount of data it can store) is usually small.

Self-Assessment Exercises 2

Fill in the gaps in the sentences below with the most suitable words:

1. _____ memory is non-volatile and stores information even when power is off.

2. The three main types of computer memory are primary memory, secondary memory, and _____ memory.

2. Memory expansion can be achieved by increasing the _____ width or increasing the number of _____.

4.0 CONCLUSION

A physical device that stores data or information temporarily or permanently in it is called memory. It's a device where data is stored and processed. In common, a computer has primary and secondary memories. Auxiliary (secondary) memory stores data and programs for long-term storage or until the time a user wants to keep them in memory, while main memory stores instructions and data during programme execution; hence, any programme or file that is currently running or executing on a computer is stored in primary memory.

5.0 SUMMARY

Computer memory is a crucial component of a computer system responsible for storing and accessing data and instructions necessary for processing tasks. It is broadly categorized into two types: volatile memory (such as RAM) and non-volatile memory (such as ROM and storage devices like SSDs and HDDs). Volatile memory, like RAM, temporarily holds data and instructions that the CPU needs while performing tasks, ensuring quick access and efficient processing. Non-volatile memory, on the other hand, retains data even when the computer is powered off, storing essential firmware, system software, and user data. The interplay between these types of memory enables a computer to function efficiently, balancing speed and storage capacity to handle various computing tasks.

6.0 Tutor marked assignment

1. What is memory?
2. List and briefly define the types of memory

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1

1. B
2. B
3. B

Self-Assessment Exercise 2

1. ROM
2. Cache
3. Word, addresses

7.0 References/ Further reading

Read, Jennifer (5 November 2020). "DDR5 Era To Officially Begin In 2021, With DRAM Market Currently Transitioning Between Generations, Says TrendForce". EMSNow. Retrieved 2 November 2022.
Jump up to:a b Hemmendinger, David (February 15, 2016). "Computer memory". Encyclopedia Britannica. Retrieved 16 October 2019.
A.M. Turing and R.A. Brooker (1952). Programmer's Handbook for Manchester Electronic Computer Mark II Archived 2014-01-02 at the Wayback Machine. University of Manchester.
"The MOS Memory Market" (PDF). Integrated Circuit Engineering Corporation. Smithsonian Institution. 1997. Archived (PDF) from the original on 2003-07-25. Retrieved 16 October 2019.
"MOS Memory Market Trends" (PDF). Integrated Circuit Engineering Corporation. Smithsonian Institution. 1998. Archived (PDF) from the original on 2019-10-16. Retrieved 16 October 2019.

UNIT 2 MEMORY HIERARCHY

1.0 INTRODUCTION

In the Computer System Design, memory hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references.

1.0 OBJECTIVES

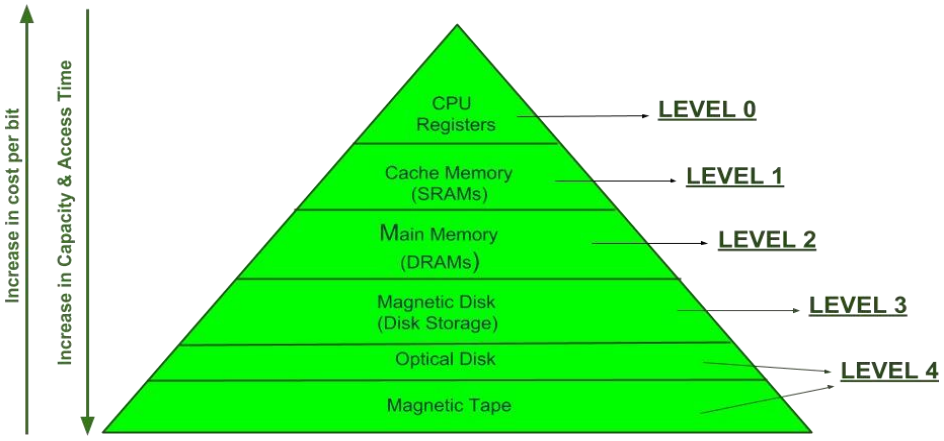
At the end of this unit, you should be able to

- Memory hierarchy
- List and discuss levels of memory hierarchy

3.1 Memory Hierarchy Design

In computer architecture, the memory hierarchy separates computer storage into a hierarchy based on response time. Since response time, complexity, and capacity are related, the levels may also be distinguished by their performance and control technologies. RAM (Random Access Memory) is an internal memory device which temporarily holds data and instructions while processing is happening. If the CPU is the “brain” of the computer, then RAM is the “working memory” or "thinking memory" used to store data just for the programs and applications being used at that time.

A typical memory hierarchy starts with a small, expensive, and relatively fast unit, called the cache, followed by a larger, less expensive, and relatively slow main memory unit. Cache and main memory are built using solid-state semiconductor material (typically CMOS transistors). It is customary to call the fast memory level the primary memory. The solid-state memory is followed by larger, less expensive, and far slower magnetic memories that consist typically of the (hard) disk and the tape. It is customary to call the disk the secondary memory, while the tape is conventionally called the tertiary memory. The objective behind designing a memory hierarchy is to have a memory system that performs as if it consists entirely of the fastest unit and whose cost is dominated by the cost of the slowest unit.



MEMORY HIERARCHY DESIGN

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references. The figure below clearly demonstrates the different levels of memory hierarchy :

This Memory Hierarchy Design is divided into 2 main types:

External Memory or Secondary Memory: Comprising Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.

Internal Memory or Primary Memory –Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

Thus, a memory system can be considered to consist of three groups of memories. These are:

3.2 Internal Processor Memories

These consist of a small set of high-speed registers that are internal to a processor and are used as temporary locations where actual processing is done.

Primary Memory or Main Memory

It is a large memory which is fast but not as fast as internal processor memory. This memory is accessed directly by the processor. It is mainly based on integrated circuits (IC).

Secondary Memory/Auxiliary Memory/Backing Store:

Auxiliary memory is much larger than main memory but is slower than main memory. It normally stores system programs (programs which are used by system to perform various operational functions), other instructions, programs and data files. Secondary memory can also he used as an overflow memory in case the main memory capacity has been exceeded. Secondary memories cannot be accessed directly by a processor.

First the information of these memories is transferred to the main memory

and then the information can be accessed as the information of main memory. There is another kind of memory which is increasingly being used in modern computers, this is called Cache memory. It is logically positioned between the internal memory (registers) and main memory. It stores or catches some of the content of the main memory which is currently in use of the processor. We will discuss about this memory in greater details in a subsequent section of this unit.

3.3 Characteristics Terms for Various Memory Devices

The memory hierarchy can be characterized by a number of parameters. Among these parameters are the access type, capacity, cycle time, latency, bandwidth, and cost.

The term access: refers to the action that physically takes place during a read or writes operation.

The capacity: of a memory level is usually measured in bytes.

The cycle time: is defined as the time elapsed from the start of a read operation to the start of a subsequent read.

The latency: is defined as the time interval between the request for information and the access to the first bit of that information.

The bandwidth: provides a measure of the number of bits per second that can be accessed.

The cost: of a memory level is usually specified as dollars per megabytes.

Figure 1 depicts a typical memory hierarchy. Table 1 provides typical values of the memory hierarchy parameters.

The term random access: refers to the fact that any access to any memory location takes the same fixed amount of time regardless of the actual memory location and/or the sequence of accesses that takes place. For example, if a write operation to memory location 100 takes 15 ns and if this operation is followed by a read operation to memory location 3000, then the latter operation will also take 15 ns. This is to be compared to sequential access in which if access to location 100 takes 500 ns, and if a consecutive access to location 101 takes 505 ns, then it is expected that an access to location 300 may take 1500 ns. This is because the memory has to cycle through locations 100 to 300, with each location requiring 5 ns.

The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information. This principle is possible due to a phenomenon called locality of reference; that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly. There exist two forms of locality: spatial and temporal locality.

RAM and ROM architecture.

Read-only memory, or ROM, is a form of data storage in computers and other electronic devices that cannot be easily altered or reprogrammed. RAM is referred to as volatile memory and is lost when the power is turned off whereas ROM in non-volatile and the contents are retained

even after the power is switched off.

Types of ROM: Semiconductor-Based

Classic mask-programmed ROM chips are integrated circuits that physically encode the data to be stored, and thus it is impossible to change their contents after fabrication. Other types of non-volatile solid-state memory permit some degree of modification:

Programmable read-only memory (PROM), or one-time programmable ROM (OTP), can be written to or programmed via a special device called a PROM programmer. Typically, this device uses high voltages to permanently destroy or create internal links (fuses or antifuses) within the chip. Consequently, a PROM can only be programmed once.

Erasable programmable read-only memory (EPROM) can be erased by exposure to strong ultraviolet light (typically for 10 minutes or longer), then rewritten with a process that again needs higher than usual voltage applied. Repeated exposure to UV light will eventually wear out an EPROM, but the endurance of most EPROM chips exceeds 1000 cycles of erasing and reprogramming. EPROM chip packages can often be identified by the prominent quartz "window" which allows UV light to enter. After programming, the window is typically covered with a label to prevent accidental erasure. Some EPROM chips are factory-erased before they are packaged, and include no window; these are effectively PROM.

Electrically erasable programmable read-only memory (EEPROM) is based on a similar semiconductor structure to EPROM, but allows its entire contents (or selected banks) to be electrically erased, then rewritten electrically, so that they need not be removed from the computer (whether general-purpose or an embedded computer in a camera, MP3 player, etc.). Writing or flashing an EEPROM is much slower (milliseconds per bit) than reading from a ROM or writing to a RAM (nanoseconds in both cases).

Electrically alterable read-only memory (EAROM) is a type of EEPROM that can be modified one bit at a time. Writing is a very slow process and again needs higher voltage (usually around 12 V) than is used for read access. EAROMs are intended for applications that require infrequent and only partial rewriting. EAROM may be used as non-volatile storage for critical system setup information; in many applications, EAROM has been supplanted by CMOS RAM supplied by mains power and backed-up with a lithium battery.

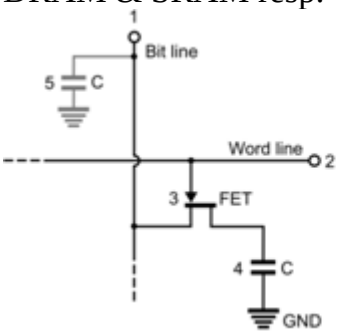
Flash memory (or simply flash) is a modern type of EEPROM invented in 1984. Flash memory can be erased and rewritten faster than ordinary EEPROM, and newer designs feature very high endurance (exceeding 1,000,000 cycles). Modern NAND flash makes efficient use of silicon chip area, resulting in individual ICs with a capacity as high as 32 GB as of 2007; this feature, along with its endurance and physical durability, has allowed NAND flash to replace magnetic in some applications (such as USB flash drives). Flash memory is sometimes called flash ROM or flash EEPROM when used as a replacement for older ROM types, but not in

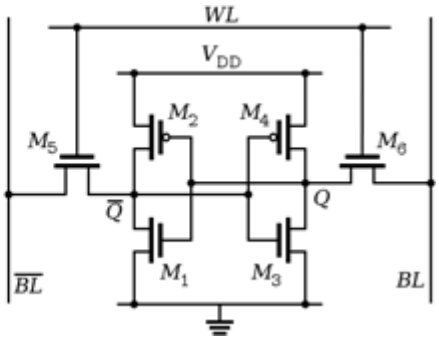
applications that take advantage of its ability to be modified quickly and frequently.

Random-access memory, or RAM, is a form of data storage that can be accessed randomly at any time, in any order and from any physical location in contrast to other storage devices, such as hard drives, where the physical location

of the data determines the time taken to retrieve it. RAM is measured in megabytes and the speed is measured in nanoseconds and RAM chips can read data faster than ROM.

Types of RAM: The two widely used forms of modern RAM are static RAM (SRAM) and dynamic RAM (DRAM). In SRAM, a bit of data is stored using the state of a six transistor memory cell. This form of RAM is more expensive to produce, but is generally faster and requires less dynamic power than DRAM. In modern computers, SRAM is often used as cache memory for the CPU. DRAM stores a bit of data using a transistor and capacitor pair, which together comprise a DRAM cell. The capacitor holds a high or low charge (1 or 0, respectively), and the transistor acts as a switch that lets the control circuitry on the chip read the capacitor's state of charge or change it. As this form of memory is less expensive to produce than static RAM, it is the predominant form of computer memory used in modern computers. The figure below shows DRAM & SRAM resp.





Both static and dynamic RAM are considered volatile, as their state is lost or reset when power is removed from the system. By contrast, read-only memory (ROM) stores data by permanently enabling or disabling selected transistors, such that the memory cannot be altered. Writeable variants of ROM (such as EEPROM and flash memory) share properties of both ROM and RAM, enabling data to persist without power and to be updated without requiring special equipment. These persistent forms of semiconductor ROM include USB flash drives, memory cards for cameras and portable devices, and solid-state drives. ECC memory (which can be either SRAM or DRAM) includes special circuitry to detect and/or correct random faults (memory errors) in the stored data, using parity bits or error correction codes.

In general, the term RAM refers solely to solid-state memory devices (either DRAM or SRAM), and more specifically the main memory in most computers. In optical storage, the term DVD-RAM is somewhat of a misnomer since, unlike CD- RW or DVD-RW it does not need to be

erased before reuse. Nevertheless, a DVD- RAM behaves much like a hard disc drive if somewhat slower.

Difference between Static Ram And Dynamic Ram

Static RAM	Dynamic RAM
➢ SRAM uses transistor to store a single bit of data	➢ DRAM uses a separate capacitor to store each bit of data
➢ SRAM does not need periodic refreshment to maintain data	➢ DRAM needs periodic refreshment to maintain the charge in the capacitors for data
➢ SRAM's structure is complex than DRAM	➢ DRAM's structure is simplex than SRAM
➢ SRAM are expensive as compared to DRAM	➢ DRAM's are less expensive as compared to SRAM
➢ SRAM are faster than DRAM	➢ DRAM's are slower than SRAM
➢ SRAM are used in Cache memory	➢ DRAM are used in Main memory

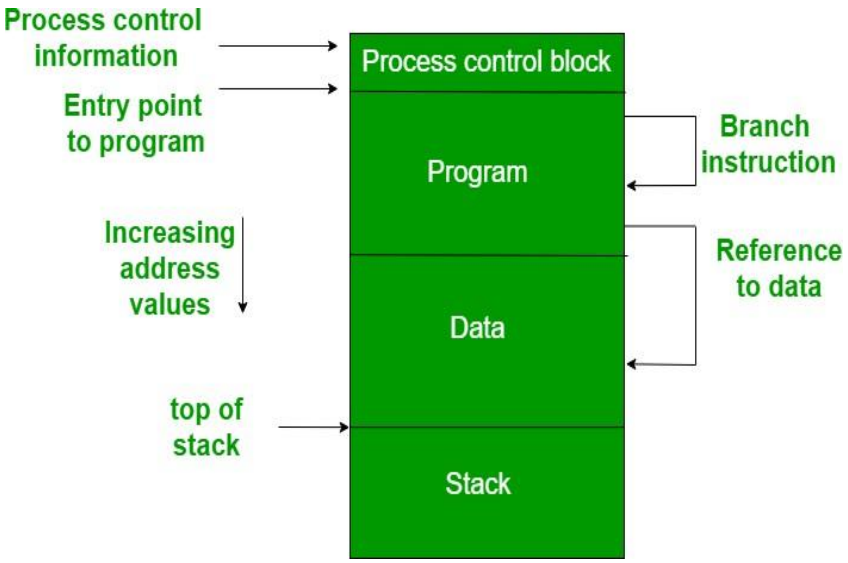
Requirements of Memory Management System

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed. Memory management meant to satisfy some requirements that we should keep in mind.

These Requirements of memory management are:

Relocation – The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

When a program gets swapped out to disk memory, then it is not always possible that when it is swapped back into main memory it occupies the previous memory location, since the location may still be occupied by another process. We may need to relocate the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The figure depicts a process image. The process image occupies a continuous region of the main memory. The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

After loading the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of the byte or word of data referenced.

Protection – There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirements a trade-off occurs as the satisfaction of
relocation requirement increases the difficulty of satisfying the protection requirement.

Prediction of the location of a program in main memory is not possible, that's why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.

Sharing – A protection mechanism must have to allow several processes to access the same portion of main memory. Allowing each processes access to the same copy of the program rather than have their own separate copy has an advantage.

For example, multiple processes may use the same system file and it is natural to load one copy of the file in main memory and let it shared by those processes. It is the task of Memory management to allow controlled access to the shared areas of memory without compromising the protection. Mechanisms are used to support relocation supported sharing capabilities.

Logical organization – Main memory is organized as linear or it can be a one- dimensional address space which consists of a sequence of bytes or words. Most of the programs can be organized into modules, some of those are unmodifiable (read-only, execute only) and some of those contain data that can be modified. To effectively deal with a user program, the operating system and computer hardware must support a basic module to provide the required protection and sharing. It has the following advantages:

Modules are written and compiled independently and all the references

from one module to another module are resolved by `the system at run time.
Different modules are provided with different degrees of protection.
There are mechanisms by which modules can be shared among processes. Sharing can be provided on a module level that lets the user specify the sharing that is desired.
Physical organization – The structure of computer memory has two levels referred to as main memory and secondary memory. Main memory is relatively very fast and costly as compared to the secondary memory. Main memory is volatile. Thus secondary memory is provided for storage of data on a long-term basis while the main memory holds currently used programs. The major system concern between main memory and secondary memory is the flow of information and it is impractical for programmers to understand this for two reasons:
The programmer may engage in a practice known as overlaying when the main memory available for a program and its data may be insufficient. It allows different modules to be assigned to the same region of memory. One disadvantage is that it is time-consuming for the programmer.
In a multiprogramming environment, the programmer does not know how much space will be available at the time of coding and where that space will be located inside the memory.
Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What is the primary purpose of memory hierarchy?
   A. To increase memory capacity
   B. To minimize access time while managing cost
   C. To improve data security
   D. To reduce power consumption

2. Which memory level is fastest but most expensive?
   A. Secondary memory
   B. Main memory
   C. Cache memory
   D. Virtual memory

3. What principle makes memory hierarchy effective?
   A. Locality of reference
   B. Random access patterns
   C. Sequential processing
   D. Parallel execution

4.0 CONCLUSION

The computer memory can be divided into 5 major hierarchies that are based on use as well as speed. A processor can easily move from any one level to some other on the basis of its requirements. These five hierarchies

in a system's memory are register, cache memory, main memory, magnetic disc, and magnetic tape.

5.0 SUMMARY

The memory hierarchy in computer systems is a structured arrangement of various types of memory based on speed, cost, and size, designed to optimize performance and efficiency. At the top of the hierarchy are the fastest and most expensive memory types, such as CPU registers and cache, which provide quick access to frequently used data. Below these are main memory, or RAM, which is slower and less costly but has higher capacity. Further down are secondary storage devices like SSDs and HDDs, which offer large storage capacities at lower speeds and costs. At the bottom, tertiary storage includes external drives and cloud storage, used for long-term data retention with the slowest access speeds. This hierarchical arrangement ensures that the most critical data is accessed rapidly while providing cost-effective solutions for large-scale data storage needs.

6.0 Tutor marked assignment

1. What do you mean by Memory Hierarchy?
2. Explain the types of Memory Hierarchy

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1

1. B
2. C
3. A

7.0 References/ Further reading

Przybylski, S. A. (1990). Cache and memory hierarchy design: a performance directed approach. Morgan Kaufmann.

Milenkovic, A., Milenkovic, M., & Barnes, N. (2003, March). A performance evaluation of memory hierarchy in embedded systems. In Proceedings of the 35th Southeastern Symposium on System Theory, 2003. (pp. 427-431). IEEE.

Przybylski, S. A. (1988). Performance directed memory hierarchy design. Stanford University.

UNIT 3 VIRTUAL MEMORY

1.0 Introduction

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites and program-generated addresses are translated automatically to the corresponding machine addresses. A memory hierarchy, consisting of a computer system's memory and a disk, that enables a process to operate with only some portions of its address space in memory. A virtual memory is what its name indicates- it is an illusion of a memory that is larger than the real memory. We refer to the software component of virtual memory as a virtual memory manager. The basis of virtual memory is the noncontiguous memory allocation model. The virtual memory manager removes some components from memory to make room for other components. The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations.

2.0 objectives

At the end of this unit, you should be able to
Discuss the concept of virtual memory and
Discuss the various implementations of virtual memory.

The objectives of this module are to
Discuss the concept of virtual memory and
Discuss the various implementations of virtual memory.

3.1 The Virtual Memory

All of us are aware of the fact that our program needs to be available in main memory for the processor to execute it. Assume that your computer has something like 32 or 64 MB RAM available for the CPU to use. Unfortunately, that amount of RAM is not enough to run all of the programs that most users expect to run at once. For example, if you load the operating system, an e-mail program, a Web browser and word processor into RAM simultaneously, 32 MB is not enough to hold all of them. If there were no such thing as virtual memory, then you will not be able to run your programs, unless some program is closed. With virtual memory, we do not view the program as one single piece. We divide it into pieces, and only the one part that is currently being referenced by the processor need to be available in main memory. The entire program is

available in the hard disk. As the copying between the hard disk and main memory happens automatically, you don't even know it is happening, and it makes your computer feel like is has unlimited RAM space even though it only has 32 MB installed. Because hard disk space is so much cheaper than RAM chips, it also has an economic benefit.

Techniques that automatically move program and data blocks into the physical main memory when they are required for execution are called virtual memory techniques. Programs, and hence the processor, reference an instruction and data space that is independent of the available physical main memory space. The binary addresses that the processor issues for either instructions or data are called virtual or logical addresses. These addresses are translated into physical addresses by a combination of hardware and software components. If a virtual address refers to a part of the program or data space that is currently in the physical memory, then the contents of the appropriate location in the main memory are accessed immediately.

On the other hand, if the referenced address is not in the main memory, its contents must be brought into a suitable location in the memory before they can be used. Therefore, an address used by a programmer will be called a virtual address, and the set of such addresses the address space. An address in main memory is called a location or physical address. The set of such locations is called the memory space, which consists of the actual main memory locations directly addressable for processing. As an example, consider a computer with a main-memory capacity of 32M words. Twenty-five bits are needed to specify a physical address in memory since 32 M = 225. Suppose that the computer has available auxiliary memory for storing 235, that is, 32G words. Thus, the auxiliary memory has a capacity for storing information equivalent to the capacity of 1024 main memories. Denoting the address space by N and the memory space by M, we then have for this example N = 32 Giga words and M = 32 Mega words.

The portion of the program that is shifted between main memory and secondary storage can be of fixed size (pages) or of variable size (segments). Virtual memory also permits a program's memory to be physically noncontiguous , so that every portion can be allocated wherever space is available. This facilitates process relocation. Virtual memory, apart from overcoming the main memory size limitation, allows sharing of main memory among processes. Thus, the virtual memory model provides decoupling of addresses used by the program (virtual) and the memory addresses (physical). Therefore, the definition of virtual memory can be stated as, " The conceptual separation of user logical memory from physical memory in order to have large virtual memory on a small physical memory". It gives an illusion of infinite storage, though the memory size is limited to the size of the virtual address.

Even though the programs generate virtual addresses, these addresses cannot be used to access the physical memory. Therefore, the virtual to

physical address translation has to be done. This is done by the memory management unit (MMU). The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by the CPU. This concept is depicted diagrammatically in Figures 20 and 21. Figure 20 gives a general overview of the mapping between the logical addresses and physical addresses. Figure 21 shows how four different pages A, B, C and D are mapped. Note that, even though they are contiguous pages in the virtual space, they are not so in the physical space. Pages A, B and C are available in physical memory at non-contiguous locations, whereas, page D is not available in physical storage.

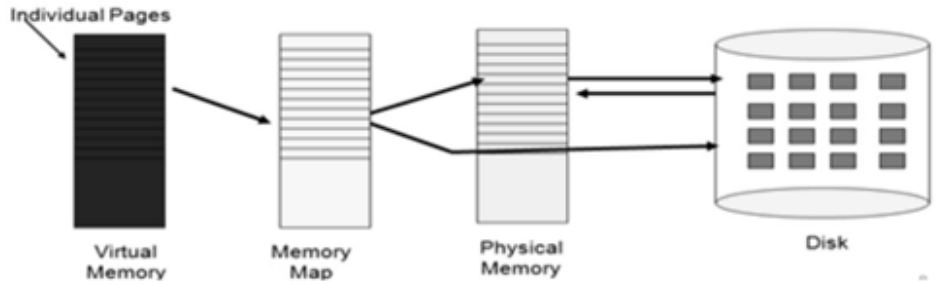


Figure 20. Overview of the mapping between logical and physical addresses

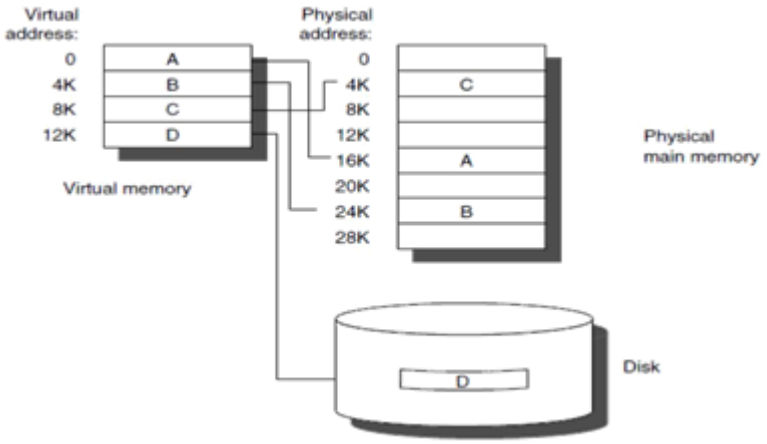


Figure 21. Four various mapping pages

3.2 Types of Virtual Memory

Address mapping using Paging: The address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size. The physical memory is broken down into groups of equal size called page frames and the logical memory is divided into pages of the same size. The programs are also considered to be split into pages. Pages commonly range from 2K to 16K bytes in length. They constitute the basic unit of information that is moved between the main memory and the disk whenever the translation mechanism determines that a move is required. Pages should not be too small, because the access time of a magnetic disk is much longer than the access time of the main memory. The reason for this is that it takes a considerable amount of time to locate the data on the disk, but once located, the data can be transferred at a rate of several megabytes per second. On the other hand, if pages are too large it is possible that a substantial portion of a page may not be used, yet this unnecessary data will occupy valuable space in the main memory. If you consider a computer with an address space of 1M and a memory space of 64K, and if you split each into groups of 2K words, you will obtain 29 (512) pages and thirty-two page frames. At any given time, up to thirty-two pages of address space may reside in main memory in anyone of the thirty-two blocks.

In order to do the mapping, the virtual address is represented by two numbers: a page number and an offset or line address within the page. In a computer with 2 p words per page, p bits are used to specify an offset and the remaining high-order bits of the virtual address specify the page number. In the example above, we considered a virtual address of 20 bits. Since each page consists of 211 = 2K words, the high order nine bits of the virtual address will specify one of the 512 pages and the low-order 11 bits give the offset within the page. Note that the line address in address space and memory space is the same; the only mapping required is from a page number to a block number.

The mapping information between the pages and the page frames is available in a page table. The page table consists of as many pages that a virtual address can support. The base address of the page table is stored in a register called the Page Table Base Register (PTBR). Each process can have one or more of its own page tables and the operating system switches from one page table to another on a context switch, by loading a different address into the PTBR. The page number, which is part of the virtual address, is used to index into the appropriate page table entry. The page table entry contains the physical page frame address, if the page is available in main memory. Otherwise, it specifies wherein secondary storage, the page is available. This generates a page fault and the operating system brings the requested page from secondary storage to main storage. Along with this address information, the page table entry also provides information about the privilege level associated with the page and the

access rights of the page. This helps in p roviding protection to the page. The mapping process is indicated in Figure 22. Figure 23 shows a typical page table entry. The dirty or modified bit indicates whether the page was modified during the cache residency period.

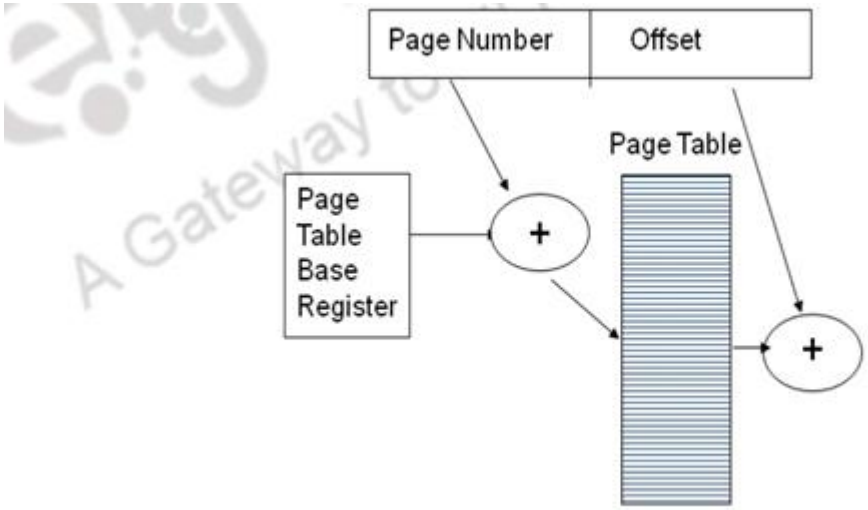


Figure 22. The Mapping Process

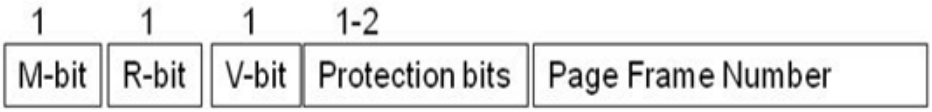


Figure 23. Example of Page Table entry
M – indicates whether the page has been written (dirty)
R – indicates whether the page has been referenced (useful for replacement)
V – Valid bit
Protection bits – indicate what operations are allowed on this page
Page Frame Number says where in memory is the page
A virtual memory system is thus a combination of hardware and software tech-niques. The memory management software system handles all the software operations for the efficient utilization of memory space. It must decide the answers to the usual four questions in a hierarchical memory system:
Q1: Where can a block be placed in the upper level?
Q2: How is a block found if it is in the upper level?
Q3: Which block should be replaced on a miss?
Q4: What happens on a write?

The hardware mapping mechanism and the memory management software together constitute the architecture of a virtual memory and

answer all these questions .
Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What is the main purpose of virtual memory?
   A. To increase processing speed
   B. To provide the illusion of larger memory than physically available
   C. To improve data security
   D. To reduce power consumption

2. What unit is used to transfer data between main memory and secondary storage in virtual memory systems?
   A. Bytes
   B. Words
   C. Pages
   D. Sectors

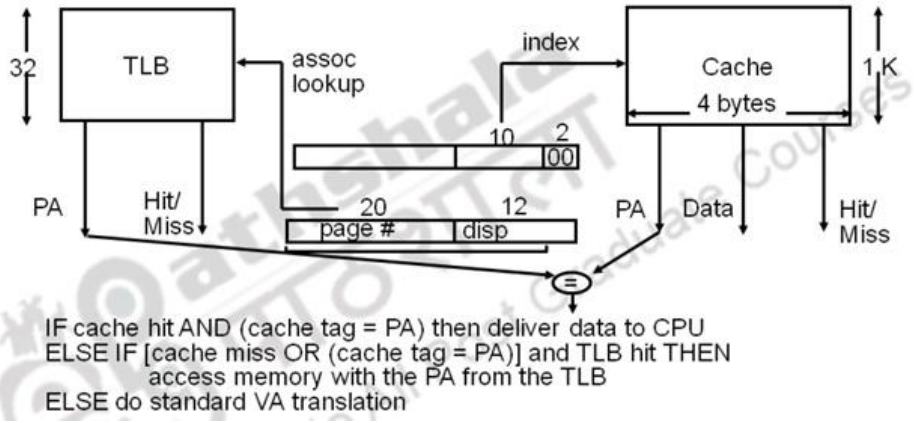
3. What happens when a program references a page not in main memory?
   A. System crash
   B. Page fault
   C. Memory overflow
   D. Cache miss

When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position. Thus, the page table entries help in identifying a page. The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called a page fault. When a page fault occurs, the execution of the present program is suspended until the required page is brought into main memory. Since loading a page from auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the I/O processor. In the meantime, control is transferred to the next program in memory that is waiting to be processed in the CPU. Later, when the memory block has been assigned and the transfer completed, the original program can resume its operation. It should be noted that it is always a write back policy that is adopted, because of the long access times associated with the disk access.
Also, when a page fault is serviced, the memory may already be full. In this case, as we discussed for caches, a replacement has to be done. The replacement policies are again FIFO and LRU. The FIFO replacement policy has the advantage of being easy to implement. !t has the disadvantage that under certain circumstances pages are removed and loaded from memory too frequently. The LRU policy is more difficult to implement but has been more attractive on the assumption that the least recently used page is a better candidate for removal than the least recently

loaded page as in FIFO. The LRU algorithm can be implemented by associating a counter with every page that is in main memory. When a page is referenced, its associated counter is set to zero. At fixed intervals of time, the counters associated with all pages presently in memory are incremented by 1. The least recently used page is the page with the highest count. The counters are often called aging registers, as their count indicates their age, that is, how long ago their associated pages have been referenced.

Drawback of Virtual memory: So far we have assumed that the page tables are stored in memory. Since, the page table information is used by the MMU, which does the virtual to physical address translation, for every read and write access, every memory access by a program can take at least twice as long: one memory access to obtain the physical address and a second access to get the data. So, ideally, the page table should be situated within the MMU. Unfortunately, the page table may be rather large, and since the MMU is normally implemented as part of the processor chip, it is impossible to include a complete page table on this chip. Therefore, the page table is kept in the main memory. However, a copy of a small portion of the page table can be accommodated within the MMU. This portion consists of the page table entries that correspond to the most recently accessed pages. A small cache, usually called the Translation Lookaside Buffer (TLB) is incorporated into the MMU for this purpose. The TLB stores the most recent logical to physical address translations. The operation of the TLB with respect to the page table in the main memory is essentially the same as the operation we have discussed in conjunction with the cache memory.
An essential requirement is that the contents of the TLB be coherent with the contents of page tables in the memory. When the operating system changes the contents of page tables, it must simultaneously invalidate the corresponding entries in the TLB. The valid bit in the TLB is provided for this purpose. When an entry is invalidated, the TLB will acquire the new information as part of the MMU's normal response to access misses.

With the introduction of the TLB, the address translation proceeds as follows. Given a virtual address, the MMU looks in the TLB for the referenced page. If the page table entry for this page is found in the TLB, the physical address is obtained immediately. If there is a miss in the TLB, then the required entry is obtained from the page table in the main memory and the TLB is updated.
Recall that the caches need a physical address, unless we use virtual caches. As discussed with respect to cache optimizations, machines with TLBs go one step further to reduce the number of cycles/cache access. They overlap the cache access with the TLB access. That is, the high order bits of the virtual address are used to look in the TLB while the low order bits are used as index into the cache. The flow is as shown below.



The overlapped access only works as long as the address bits used to index into the cache do not change as the result of VA translation. This usually limits things to small caches, large page sizes, or high n-way set associative caches if you want a large cache.

Advantages of Virtual Memory

More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.

A process may be larger than all of the main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in the main memory as required.

It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

It has twice the capacity for addresses as main memory.

It makes it possible to run more applications at once.

Users are spared from having to add memory modules when RAM space runs out, and applications are liberated from shared memory management.

When only a portion of a program is required for execution, speed has increased.

Memory isolation has increased security.

It makes it possible for several larger applications to run at once.

Memory allocation is comparatively cheap.

It doesn't require outside fragmentation.

It is efficient to manage logical partition workloads using the CPU.

Automatic data movement is possible.

Disadvantages of Virtual Memory

It can slow down the system performance, as data needs to be constantly transferred between the physical memory and the hard disk.

It can increase the risk of data loss or corruption, as data can be lost if the hard disk fails or if there is a power outage while data is being transferred

to or from the hard disk.
It can increase the complexity of the memory management system, as the operating system needs to manage both physical and virtual memory.
Self-Assessment Exercises 2

Fill in the gaps in the sentences below with the most suitable words:

1. The _____ management unit (MMU) performs virtual to physical address translation.

2. Virtual memory allows _____ allocation of memory and supports sharing of main memory among processes.

3. The _____ policy determines which page to remove when memory is full.

4.0 Conclusion

In the ever-evolving world of computer science, the concept of virtual memory has become increasingly important for both computer architecture and organisation. This in-depth guide will provide an overview of what virtual memory is, along with its benefits and drawbacks. Delving into the role of virtual memory in the overall structure of computer systems, you will gain an understanding of how it interacts with primary memory and enhances system performance. Furthermore, the discussion will encompass topics such as the purpose and functionality of virtual memory, its role in memory management and allocation, as well as addressing common issues and challenges associated with its implementation. So, let's embark on a journey through the fascinating realm of virtual memory and uncover its implications for modern computer science.

5.0 Summary

To summarize, we have looked at the need for the concept of virtual memory. Virtual memory is a concept implemented using hardware and software. The restriction placed on the program size is not based on the RAM size but based on the virtual memory size. There are three different ways of implementing virtual memory. The MMU does the logical to physical address translation. Paging uses fixed-size pages to move between main memory and secondary storage. Paging uses page tables to map the logical addresses to physical addresses. Thus, virtual memory helps in dynamic allocation of the required data, sharing of data, and providing protection. The TLB is used to store the most recent logical to physical address translations.

6.0 Tutor Marked Assignment

1. What are the differences among various mapping
2. What is a virtual memory?
3. State five (5) advantages of virtual memory.

7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. B
2. C
3. B

Self-Assessment Exercise 2
1. Memory
2. Dynamic
3. Replacement

7.0 References/Further reading

Adamck, J. Foundation of coding New York Wiley 1991
Smith,a CACHE MEMORIES ACM computing surveys September 1992

UNIT 4 CACHE MEMORY

1.0 Introduction

A small but fast cache memory, in which the contents of the most commonly accessed locations are maintained, can be placed between the main memory and the CPU. When a program executes, the cache memory is searched first, and the referenced word is accessed in the cache if the word is present. If the referenced word is not in the cache, then a free location is created in the cache, and the referenced word is brought into the cache from the main memory. In general most future access to main memory by the processor will likely be to locations recently accessed. So the cache memory automatically retains a copy of some of the recently used words from the dynamic random-access memory (DRAM)

2.0 Objectives

At the end of this unit, you should be able to

- Explain the principles and elements of cache design\understood Pentium 4 cache organization
- Discuss ARM cache organization

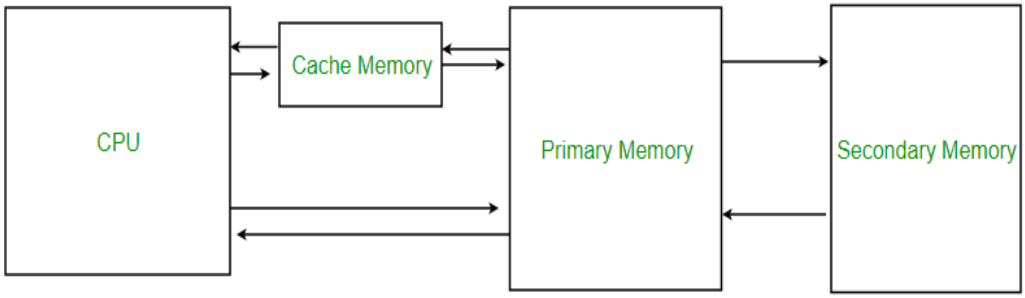
3.1 Cache memory principles
3.2 Replacement Policies in Associative Mapped Caches
3.3 Cache Performance

3.0 MAIN CONTENTS

3.1 Cache Principle

Cache Memory is a special very high-speed memory. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data. The most important use of cache memory is that it is used to reduce the average time to access data from the main memory. The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time. Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory, then the CPU moves into the main memory. Cache memory is placed between the CPU and the main memory.

Characteristics of Cache Memory
Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.
Cache Memory holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
Cache memory is costlier than main memory or disk memory but more economical than CPU registers.
Cache Memory is used to speed up and synchronize with a high-speed CPU.



Levels of Memory
Level 1 or Register: It is a type of memory in which data is stored and accepted that are immediately stored in the CPU. The most commonly used register is Accumulator, Program counter, Address Register, etc.
Level 2 or Cache memory: It is the fastest memory that has faster access time where data is temporarily stored for faster access.
Level 3 or Main Memory: It is the memory on which the computer works currently. It is small in size and once power is off data no longer stays in this memory.
Level 4 or Secondary Memory: It is external memory that is not as fast as the main memory but data stays permanently in this memory.
The speed of the main memory is very low in comparison with the speed of modern processors. For good performance, the processor cannot spend much of its time waiting to access instructions and data in main memory. Hence, it is important to devise a scheme that reduces the time needed to access the necessary information. Since the speed of the main memory unit is limited by electronic and packaging constraints, the solution must be sought in a different architectural arrangement. An efficient solution is to use a fast cache memory, which essentially makes the main memory appear to the processor to be faster than it is. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are to cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory.
The effectiveness of the cache mechanism is based on a property of

computer programs called locality of reference. Analysis of programs shows that most of their execution time is spent on routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other. The actual detailed pattern of instruction sequencing is not important – the point is that many instructions in localized areas of the program are executed repeatedly during some time, and the remainder of the program is accessed relatively infrequently. This is referred to as the locality of reference. It manifests itself in two ways: temporal and spatial. The first means that a recently executed instruction is likely to be executed again very soon. The spatial aspect means that instructions in close proximity to a recently executed instruction (with respect to the instructions' addresses) are also likely to be executed soon.

If the active segments of a program can be placed in a fast cache memory, then the total execution time can be reduced significantly. Conceptually, operation of a cache memory is very simple. The memory control circuitry is designed to take advantage of the property of locality of reference. The temporal aspect of the locality of reference suggests that whenever an information item (instruction or data) is first needed, this item should be brought into the cache where it will hopefully remain until it is needed again. The spatial aspect suggests that instead of fetching just one item from the main memory to the cache, it is useful to fetch several items that reside at adjacent addresses as well. We will use the term block to refer to a set of contiguous address locations of some size. Another term that is often used to refer to a cache block is cache line.

The cache memory that is included in the memory hierarchy can be split or unified/dual. A split cache is one where we have a separate data cache and a separate instruction cache. Here, the two caches work in parallel, one transferring data and the other transferring instructions. A dual or unified cache is wherein the data and the instructions are stored in the same cache. A combined cache with a total size equal to the sum of the two split caches will usually have a better hit rate. This higher rate occurs because the combined cache does not rigidly divide the number of entries that may be used by instructions from those that may be used by data. Nonetheless, many processors use a split instruction and data cache to increase cache bandwidth.

When a Read request is received from the processor, the contents of a block of memory words containing the location specified are transferred into the cache. Subsequently, when the program references any of the locations in this block, the desired contents are read directly from the cache. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory. The correspondence between the main memory blocks and those in the cache is specified by a mapping function. When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must

decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the replacement algorithm.

Therefore, the three main issues to be handled in cache memory are

Cache placement – where do you place a block in the cache?

Cache identification – how do you identify whether the requested information is available in the cache or not?

Cache replacement – which block will be replaced in the cache, making way for an incoming block?

These questions are answered and explained with an example main memory size of 1MB (the main memory address is 20 bits), a cache memory of size 2KB and a block size of 64 bytes. Since the block size is 64 bytes, you can immediately identify that the main memory has 214 blocks and the cache has 25 blocks. That is, the 16K blocks of main memory have to be mapped to the 32 blocks of cache. There are three different mapping policies – direct mapping, fully associative mapping and n-way set associative mapping that are used.

The word is then accessed in the cache. Although this process takes longer than accessing main memory directly, the overall performance can be improved if a high proportion of memory accesses are satisfied by the cache. Modern memory systems may have several levels of cache, referred to as Level 1 (L1), Level 2 (L2), and even, in some cases, Level 3 (L3). In most instances the

L1 cache is implemented right on the CPU chip. Both the Intel Pentium and the IBM-Motorola PowerPC G3 processors have 32 Kbytes of L1 cache on the CPU chip.

A cache memory is faster than main memory for a number of reasons. Faster electronics can be used, which also results in a greater expense in terms of money, size, and power requirements. Since the cache is small, this increase in cost is relatively small. A cache memory has fewer locations than a main memory, and as a result it has a shallow decoding tree, which reduces the access time.

The cache is placed both physically closer and logically closer to the CPU than the main memory, and this placement avoids communication delays over a shared bus. A typical situation is shown in Figure 24. A simple computer without a cache memory is shown in the left side of the figure.

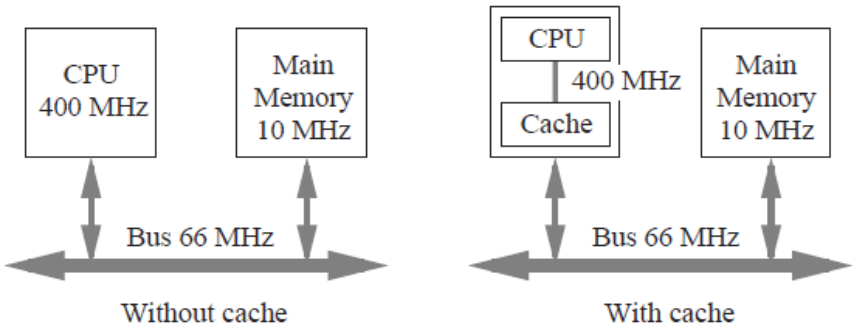


Figure 24: Placement of Cache in a Computer System

This cache-less computer contains a CPU that has a clock speed of 400 MHz, but communicates over a 66 MHz bus to a main memory that supports a lower clock speed of 10 MHz. A few bus cycles are normally needed to synchronize the CPU with the bus, and thus the difference in speed between main memory and the CPU can be as large as a factor of ten or more. A cache memory can be positioned closer to the CPU as shown in the right side of Figure 2, so that the CPU sees fast accesses over a 400 MHz direct path to the cache.

3.2 Replacement Policies in Associative Mapped Caches

When a new block needs to be placed in an associative mapped cache, an available slot must be identified. If there are unused slots, such as when a program begins execution, then the first slot with a valid bit of 0 can simply be used.

When all of the valid bits for all cache slots are 1, however, then one of the active slots must be freed for the new block. Four replacement policies that are commonly used are: least recently used (LRU), first-in first-out (FIFO), least frequently used (LFU), and random. A fifth policy that is used for analysis purposes only, is optimal.

For the LRU policy, a time stamp is added to each slot, which is updated when any slot is accessed. When a slot must be freed for a new block, the contents of the least recently used slot, as identified by the age of the corresponding time stamp, are discarded and the new block is written to that slot. The LFU policy works similarly, except that only one slot is updated at a time by incrementing a frequency counter that is attached to each slot. When a slot is needed for a new block, the least frequently used slot is freed.

The FIFO policy replaces slots in round-robin fashion, one after the next in the order of their physical locations in the cache. The random replacement policy simply chooses a slot at random. The optimal replacement policy is not practical, but is used for comparison purposes to determine how effective other replacement policies are to the best

possible.

That is, the optimal replacement policy is determined only after a program has already executed, and so it is of little help to a running program. Studies have shown that the LFU policy is only slightly better than the random policy. The LRU policy can be implemented efficiently, and is sometimes preferred over the others for that reason.

Advantages and Disadvantages of the Associative Mapped Cache
The associative mapped cache has the advantage that any main memory block can be placed into any cache slot. This means that regardless of how irregular the data and program references are, if a slot is available for the block, it can be stored in the cache. This results in considerable hardware overhead needed for cache bookkeeping. Each slot must have a 27-bit tag that identifies its location in main memory, and each tag must be searched in parallel. This means that in the example above the tag memory must be 27 x 214 bits in size, and as described above, there must be a mechanism for searching the tag memory in parallel. Memories that can be searched for their contents, in parallel, are referred to as associative, or content-addressable memories. By restricting where each main memory block can be placed in the cache, we can eliminate the need for an associative memory. This kind of cache is referred to as a direct mapped cache, which is discussed in the next section.
Self-Assessment Exercises 1

Answer the following questions by choosing the most suitable option:

1. What is the primary purpose of cache memory?
   A. To store large amounts of data permanently
   B. To provide faster access to frequently used data
   C. To backup important files
   D. To connect to external devices

2. Which replacement policy removes the least recently used item?
   A. FIFO
   B. LRU
   C. Random
   D. Optimal

3. What are the two forms of locality of reference?
   A. Spatial and temporal
   B. Physical and logical
   C. Static and dynamic
   D. Sequential and random

Direct Mapped Cache

Figure 24 shows a direct mapping scheme for a 232 word memory. As before, the memory is divided into 227 blocks of 25 = 32 words per block, and the cache consists of 214 slots. There are more main memory blocks than there are cache slots, and a total of 227/214 = 213 main memory blocks can be mapped onto each cache slot. In order to keep track of which of the 213 possible blocks is in each slot, a 13-bit tag field is added to each slot which holds an identifier in the range from 0 to 213 – 1.

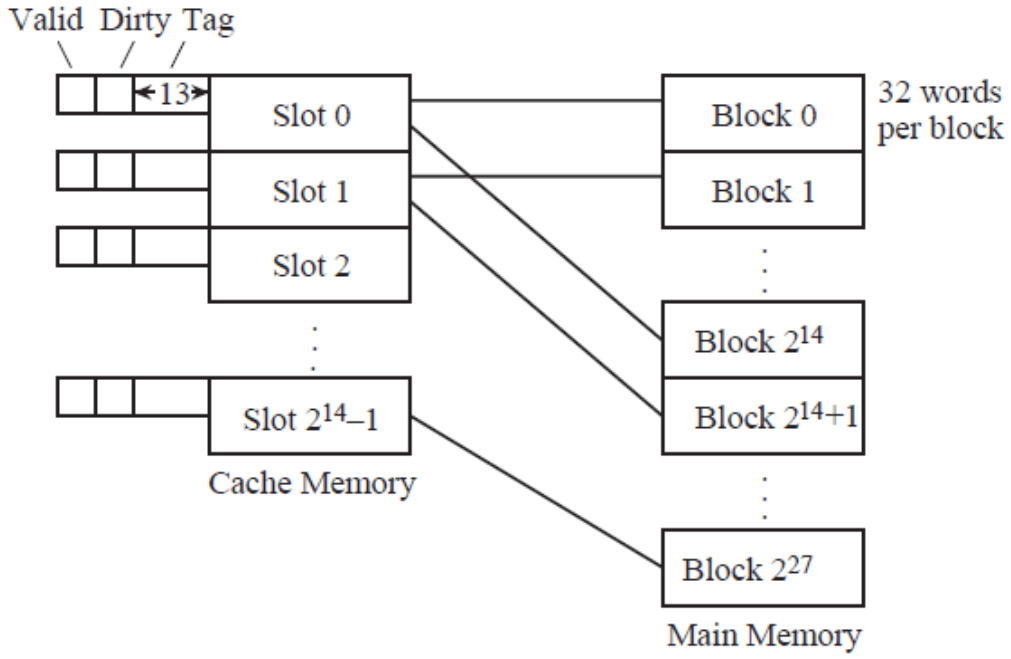


Figure 24: A Direct Mapping Scheme for Cache Memory

This scheme is called “direct mapping” because each cache slot corresponds to an explicit set of main memory blocks. For a direct mapped cache, each main memory block can be mapped to only one slot, but each slot can receive more than one block. The mapping from main memory blocks to cache slots is performed by partitioning an address into fields for the tag, the slot, and the word as shown below:

The 32-bit main memory address is partitioned into a 13-bit tag field, followed by a 14-bit slot field, followed by a five-bit word field. When a reference is made to a main memory address, the slot field identifies in which of the 214 slots the block will be found if it is in the cache. If the valid bit is 1, then the tag field of the referenced address is compared with the tag field of the slot. If the tag fields are the same, then the word is taken from the position in the slot specified by the word field. If the valid bit is 1 but the tag fields are not the same, then the slot is written back to main memory if the dirty bit is set, and the corresponding main memory block is then read into the slot. For a program that has just started execution, the valid bit will be 0, and so the block is simply written to the slot. The valid bit for the block is then set to 1, and the program resumes execution.

Tag	Slot	Word
13 bits	14 bits	5 bits

Advantages and Disadvantages of the Direct Mapped Cache
The direct mapped cache is a relatively simple scheme to implement. The tag memory in the example above is only 13 x 214 bits in size, less than half of the associative mapped cache. Furthermore, there is no need for an associative search, since the slot field of the main memory address from the CPU is used to "direct" the comparison to the single slot where the block will be if it is indeed in the cache.
This simplicity comes at a cost. Consider what happens when a program references locations that are 219 words apart, which is the size of the cache. This pattern can arise naturally if a matrix is stored in memory by rows and is accessed by columns. Every memory reference will result in a miss, which will cause an entire block to be read into the cache even though only a single word is used.
Worse still, only a small fraction of the available cache memory will actually be used. Now it may seem that any programmer who writes a program this way deserves the resulting poor performance, but in fact, fast matrix calculations use power-of-two dimensions (which allows shift operations to replace costly multiplications and divisions for array indexing), and so the worst-case scenario of accessing memory locations that are 219 addresses apart is not all that unlikely.
To avoid this situation without paying the high implementation price of a fully associative cache memory, the set associative mapping scheme can be used, which combines aspects of both direct mapping and associative mapping.

3.3 Cache Performance

Notice that we can readily replace the cache direct mapping hardware with associative or set associative mapping hardware, without making any other changes to the computer or the software. Only the runtime performance will change between methods. Runtime performance is the purpose behind using a cache memory, and there are a number of issues that need to be addressed as to what triggers a word or block to be moved between the cache and the main memory.
Cache read and write policies are summarized in Figure 25. The policies depend upon whether or not the requested word is in the cache. If a cache read operation is taking place, and the referenced data is in the cache, then there is a "cache hit" and the referenced data is immediately forwarded to the CPU. When a cache miss occurs, then the entire block that contains the referenced word is read into the cache.
In some cache organizations, the word that causes the miss is immediately forwarded to the CPU as soon as it is read into the cache, rather than waiting for the remainder of the cache slot to be filled, which is known as a load-through operation. For a non-interleaved main memory, if the word

occurs in the last position of the block, then no performance gain is realized since the entire slot is brought in before load-through can take place. For an interleaved main memory, the order of accesses can be organized so that a load-through operation will always result in a performance gain.

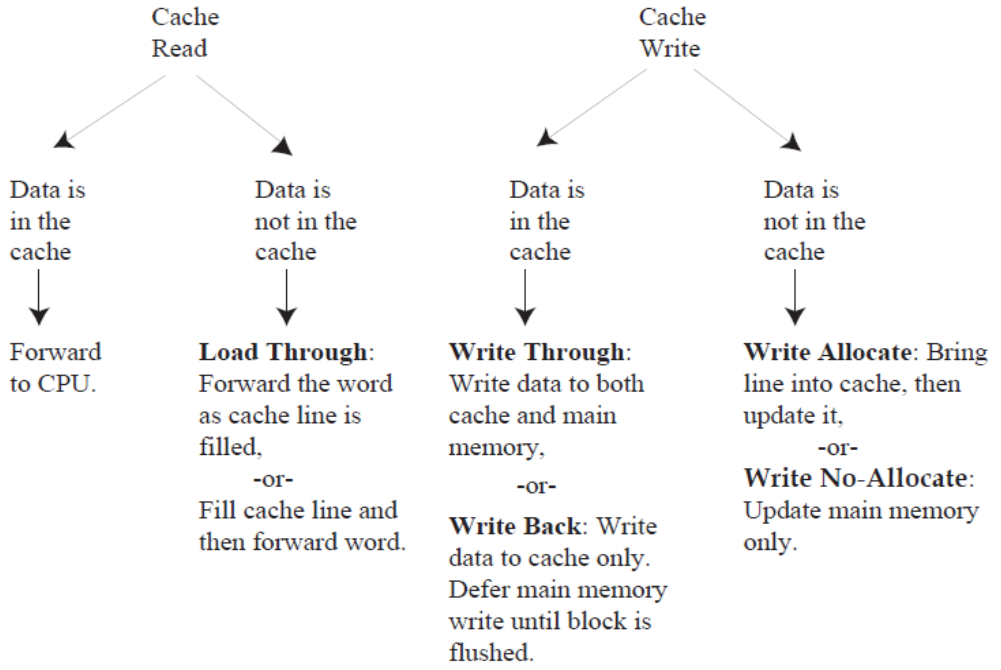


Figure 25: Cache Read and Write Policies

For write operations, if the word is in the cache, then there may be two copies of the word, one in the cache, and one in main memory. If both are updated simultaneously, this is referred to as write-through. If the write is deferred until the cache line is flushed from the cache, this is referred to as write-back.

Even if the data item is not in the cache when the write occurs, there is the choice of bringing the block containing the word into the cache and then updating it, known as write-allocate, or to update it in main memory without involving the cache, known as write-no-allocate. Some computers have separate caches for instructions and data, which is a variation of a configuration known as the Harvard architecture (also known as a split cache), in which instructions and data are stored in separate sections of memory.

Since instruction slots can never be dirty (unless we write self-modifying code, which is rare these days), an instruction cache is simpler than a data cache. In support of this configuration, observations have shown that most of the memory traffic moves away from main memory rather than toward it.

Statistically, there is only one write to memory for every four read operations from memory. One reason for this is that instructions in an executing program are only read from the main memory, and are never

written to the memory except by the system loader. Another reason is that operations on data typically involve reading two operands and storing a single result, which means there are two read operations for every write operation.

A cache that only handles reads, while sending writes directly to main memory can thus also be effective, although not necessarily as effective as a fully functional cache. As to which cache read and write policies are best, there is no simple answer. The organization of a cache is optimized for each computer architecture and the mix of programs that the computer executes. Cache organization and cache sizes are normally determined by the results of simulation runs that expose the nature of memory traffic.

4.0 Summary

In this unit, you have learnt that:

Cache memory, also called CPU memory, is high-speed static random access memory (SRAM) that a computer microprocessor can access more quickly than it can access regular random access memory (RAM).

A cache memory is faster than main memory and has fewer locations than a main memory.

A cache is placed both physically closer and logically closer to the CPU than the main memory

The physical memory is smaller than the size of the program, but is larger than any single routine.

Self-Assessment Exercises 2

Fill in the gaps in the sentences below with the most suitable words:

1. In a direct mapped cache, each main memory block can be mapped to only _____ slot.

2. Cache _____ occurs when the requested data is found in the cache.

3. The _____ policy determines whether data is written to both cache and main memory simultaneously.

4.0 Conclusion

If the cache is designed properly then most of the time the processor will request memory words that are already in the cache. *Cache* is memory placed in between the processor and main memory. Cache is responsible for holding copies of main memory data for faster retrieval by the processor.Cache memory consists of a collection of blocks. Each block can hold an entry from the main memory.

5.0 SUMMARY

Cache memory, also called CPU memory, is high-speed static random access memory (SRAM) that a computer microprocessor can access more quickly than it can access regular random access memory (RAM).

A cache memory is faster than main memory and has fewer locations than a main memory.

A cache is placed both physically closer and logically closer to the CPU than the main memory
The physical memory is smaller than the size of the program, but is larger than any single routine.

6.0 Tutor marked assignment

For a direct mapped cache a main memory address is viewed as consisting of two fields list and define the two fields.
7.1 Possible Answers to Self-Assessment Exercises

Self-Assessment Exercise 1
1. B
2. B
3. A

Self-Assessment Exercise 2
1. One
2. Hit
3. Write-through

7.0 Reference and further reading

Computer Organization and Design – The Hardware / Software Interface, David A. Patterson and John L. Hennessy, 4th Edition, Morgan Kaufmann, Elsevier, 2009.
Computer Architecture – A Quantitative Approach , John L. Hen nessy and David A.Patterson, 5th Edition, Morgan Kaufmann, Elsevier, 2011.
Computer Organization, Carl Hamacher, Zvonko Vranesic and Safwat Zaky, 5th.Edition, McGraw- Hill Higher Education, 2011