HEATHKIT
CONTINUING
EDUCATION

Individual Learning Program

MICROPROCESSORS

Unit 2
MICROCOMPUTER BASICS

EE-3401

HEATH COMPANY
BENTON HARBOR, MICHIGAN 49022

CONTENTS

Unit 2

MICROPROCESSORS

INTRODUCTION

A microprocessor is a very complex electronic circuit. It consists of thousands of microscopic transistors squeezed onto a tiny chip of silicon that is often no more than one-eighth inch square. The chip is placed in a package containing up to about 40 leads.

The thousands of transistors that make up the microprocessor are arranged to form many different circuits within the chip. From the standpoint of learning how the microprocessor operates, the most important circuits on the chip are registers, counters, and decoders. In this unit, you will learn how these circuits can work together to perform simple but useful tasks.

UNIT OBJECTIVES

When you have completed this unit you will be able to:

1. Define the terms: microprocessor, microcomputer, input, output, I/O, I/O device, I/O port, instruction, program, stored program concept, word, byte, MPU, ALU, operand, memory, address, read, write, RAM, fetch, execute, MPU cycle, mnemonic, opcode, and bus.

2. Explain the purpose of the following circuits in a typical microprocessor: accumulator, program counter, instruction decoder, controller sequencer, data register, and address register.

3. Using a simplified block diagram of a hypothetical microprocessor, trace the data flow that takes place between the various circuits during the execution of a simple program.

4. Describe the difference between inherent, immediate, and direct addressing.

5. Write simple, straight-line programs that can be executed by the ET-3400 Microprocessor Trainer.

UNIT ACTIVITY GUIDE

	Completion Time
☐ Read Section on Terms and Conventions.	_____
☐ Complete Self-Test Review Questions 1 — 11.	_____
☐ Read Section on An Elementary Microcomputer.	_____
☐ Complete Self-Test Review Questions 12 — 30.	_____
☐ Read Section on Executing a Program.	_____
☐ Complete Self-Test Review Questions 31 — 40.	_____
☐ Read Section on Addressing Modes.	_____
☐ Complete Self-Test Review Questions 41 — 50.	_____
☐ Perform Experiment 3.	_____
☐ Complete Unit Examination.	_____
☐ Check Examination Answers.	_____

TERMS AND CONVENTIONS

A **microprocessor** is a logic device that is used in digital electronic systems. It is also being used by hobbyists, experimenters and low-budget research groups as a low-cost, general-purpose computer. But a distinction should be made between the microprocessor and the microcomputer.

The microprocessor unit, or MPU, is a complex logic element that performs arithmetic, logic, and control operations. The trend is to package it as a single integrated circuit.

A **microcomputer** contains a microprocessor, but it also contains other circuits such as memory devices to store information, interface adapters to connect it with the outside world, and a clock to act as a master timer for the system. Figure 2-1 shows a typical microcomputer in which these additional circuits are added. The arrows represent conductors over which binary information flows. The wide arrows represent several conductors connected in parallel. A group of parallel conductors which carry information is called a **bus**.

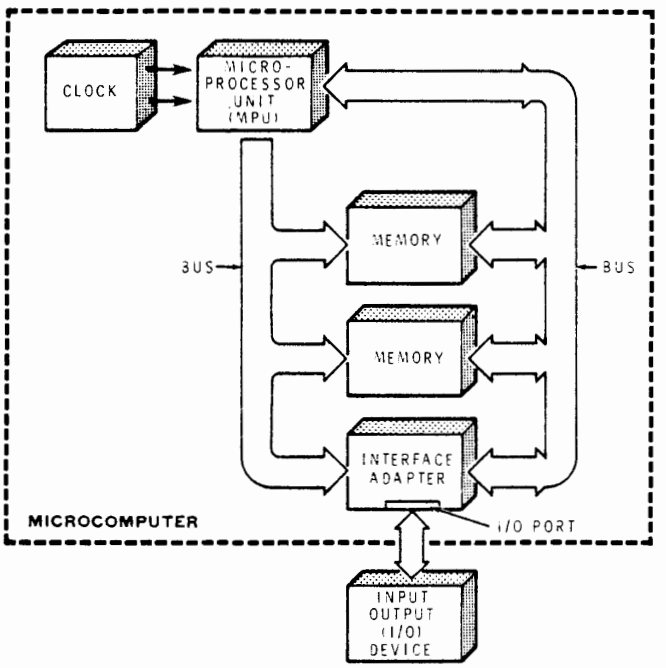


Figure 2-1
A Basic Microcomputer

The microcomputer is composed of everything inside the dotted line. Everything outside the dotted line is referred to as the **outside world**, and all microcomputers must have some means of communicating with it. Information received by the microcomputer from the outside world is referred to as **input** data. Information transmitted to the outside world from the microcomputer is referred to as **output** data.

Input information may come from devices like paper tape readers, typewriters, mechanical switches, keyboards, or even other computers. Output information may be sent to video displays, output typewriters, paper tape punches, or line printers. Some devices such as the teletypewriter can serve as both an input and an output device. These devices are referred to as **input/output** or **I/O** devices. The point at which the I/O device connects to the microcomputer is called an **I/O port**.

Stored Program Concept

A microcomputer is capable of performing many different operations. It can add and subtract numbers and it can perform logical operations. It can read information from an input device and transmit information to an output device. In fact, depending on the microprocessor used, there may be 100 or more different operations that the microcomputer can perform. Moreover, two or more individual operations can be combined to perform much more complex operations.

In spite of all its capabilities, the computer will do nothing on its own accord. It will do only what it is told to do, nothing more and nothing less. You must tell the computer exactly what operations to perform and the order in which it should perform them. The operations that the computer can be told to perform are called **instructions** A few typical instructions are ADD, SUBTRACT, LOAD INDEX REGISTER, STORE ACCUMULATOR, and HALT.

A group of instructions that allow the computer to perform a specific job is called a **program**. One who writes these instructions is called a **programmer**. To design with microprocessors, the engineer must become a programmer. To repair microprocessor-based equipment, the technician must understand programming. Generally, the length of the program is proportional to the complexity of the task that the computer is to perform. A program for adding a list of numbers may require only a dozen instructions. On the other hand, a program for controlling all the traffic lights in a small city may require thousands of instructions.

A computer is often compared to a calculator, which is told what to do by the operator via the keyboard. Even inexpensive calculators can perform several operations that can be compared to instructions in the computer. By depressing the right keys, you can instruct the calculator to add, subtract, multiply, divide, and clear the display. Of course, you must also enter the numbers that are to be added, subtracted, etc. With the calculator, you can add a list of numbers as quickly as you can enter the numbers and the instructions. That is, the operation is limited by the speed and accuracy of the operator.

From the start, computer designers recognized that it was the human operator that slowed the computation process. To overcome this, the **stored program concept** was developed. Using this approach, the program is stored in the computer's memory. Suppose, for example, that you have 20 numbers that are to be manipulated by a program that is composed of 100 instructions. Let's further suppose that 10 answers will be produced in the process.

Before any computation begins, the 100-instruction program plus the 20 numbers are loaded into the computer's memory. Furthermore, 10 memory locations are reserved for the 10 answers. Only then is the computer allowed to execute the program. The actual computation time might be less than one millisecond. Compare this to the time that it would take to manually enter the instructions and numbers, one at a time, while the computer is running. This automatic operation is one of the features that distinguishes the computer from the calculator.

Computer Words

In computer terminology, a **word** is a group of binary digits that can occupy a storage location. Although the word may be made up of several binary digits, the computer handles each word as if it were a single unit. Thus, the **word** is the fundamental unit of information used in the computer.

A word may be a binary number that is to be handled as data. Or, the word may be an instruction that tells the computer which operation it is to perform. It may be an ASCII character representing a letter of the alphabet. Finally, a word can be an "address" that tells the computer where a piece of data is located.

Word Length

In the past few years, a wide variety of microprocessors have been developed. Their cost and capabilities vary widely. One of the most important characteristics of any microprocessor is the word length it can handle. This refers to the length in bits of the most fundamental unit of information.

The most common word length for microprocessors is 8 bits. In these units; numbers, addresses, instructions, and data are represented by 8-bit binary numbers. The lowest 8-bit binary number is $0000\ 0000_2$ or 00_{16}. The highest is $1111\ 1111_2$ or FF_{16}. In decimal, this range is from 0 to 255_{10}. Thus, an 8-bit binary number can have any one of 256_{10} unique values.

An 8-bit word can specify positive numbers between 0 and 255_{10}. Or, if the 8-bit word is an instruction, it can specify any of 256_{10} possible operations. It is also entirely possible that the 8-bit word is an ASCII character. In this case, it can represent letters of the alphabet, punctuation marks, or numerals. As you can see, the 8-bit word can represent many different things, depending on how it is interpreted. The programmer must insure that an ASCII character or binary number is not interpreted as an instruction. Later, you will see the consequences of making this mistake.

While the 8-bit word length is the most popular, other word lengths are sometimes used. The earliest microprocessor used a 4-bit word length, and four-bit microprocessors are still used in many cases because of their low cost. A few 12-bit and 16-bit microprocessors have also been developed.

Longer word lengths allow us to work with larger numbers. For example, a 16-bit word can represent numbers up to $65{,}535_{10}$. However, this capability adds to the complexity and cost of the microprocessor. Because most microprocessors use 8-bit word lengths, we will restrict our discussion to units of this type.

It should be pointed out that just because the word length is 8-bits, it does not mean that we are restricted to numbers below 256_{10}. It simply means that you must use two or more words to represent larger numbers.

The 8-bit word length defines the size of many different components in the microprocessor system. For example, many of the important registers will have 8-bit capacity. Memory will be capable of holding a large number of 8-bit words. And, the bus which is used to transfer data words will consist of eight parallel conductors.

Even 16-bit microprocessors use 8-bit segments of data in many applications. For example, inputs from teletypewriters often consist of 8-bit ASCII characters. To distinguish these 8-bit segments of information from the 16-bit (or longer) word lengths, another term has come into general use: the term **byte**. A byte is a group of bits that are handled as a single unit. Generally, a byte is understood to consist of 8-bits. In the 8-bit microprocessor, each word consists of one byte. But in the 16-bit machines, each word contains two bytes. Figure 2-2 illustrates these points.

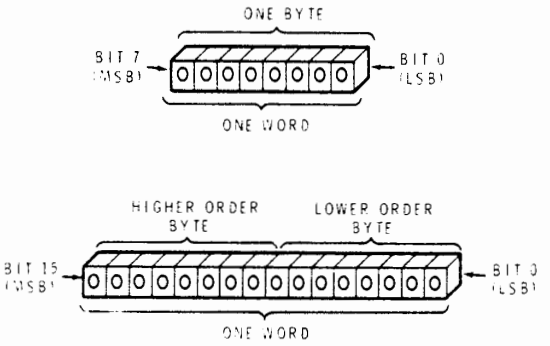


Figure 2-2
Words and Bytes.

Figure 2-2 also shows how the bits that make up the computer word are numbered. The least significant bit (LSB) is on the right while the most significant bit (MSB) is on the left. In the 8-bit word, the bits are numbered 0 through 7 from right to left. In the 16-bit word, the bits are numbered 0 through 15 as shown. The lower 8-bits are called the lower order byte while the upper 8-bits are called the higher order byte.

Self-Test Review

1. Explain the difference between a microprocessor and a microcomputer.

2. What is a bus?

3. Explain the difference between input and output data.

4. Define I/O.

5. The point at which data enters or leaves the computer is called an I/O_____.

6. The operations that the computer can be told to perform are called _____.

7. What is a program?

8. Explain what is meant by "stored program concept."

9. A byte generally consists of _____ bits.

10. A computer word may consist of one or more _____.

11. What is the largest number that can be represented by an 8-bit computer word? By a 16-bit word?

Answers

1. A microprocessor is a logic element that can perform a variety of arithmetic, control, and logic operations. A microcomputer is a system that consists of a microprocessor, memory, interface adapters, clock, etc.

2. A bus is a group of conductors over which information can be transmitted. The conductors may be wires in a cable, foil patterns on a printed circuit board, or microscopic metal deposits in a silicon chip.

3. Input data is information that is entered into the computer from the outside world. Output data is information that is transmitted from the computer to the outside world. The outside world is defined as anything outside the computer.

4. I/O is the abbreviation for input-output. Thus, a device that can send data to and accept data from the computer is called an I/O device.

5. Port.

6. Instructions.

7. A program is a group of instructions that tell the computer the operations to be performed and the sequence in which they are to be performed.

8. The stored program concept refers to the technique of storing the instruction to be performed in the memory section along with the data that is to be operated upon.

9. 8.

10. Bytes.

11. $1111\ 1111_2$ or 255_{10}. $1111\ 1111\ 1111\ 1111_2$ or $65{,}535_{10}$.

AN ELEMENTARY MICROCOMPUTER

One of the difficulties you may encounter in learning about a microcomputer for the first time is the complexity of its main component — the microprocessor. The microprocessor may have a dozen or more registers varying in size from 1 bit to 16 bits. It will have scores of instructions, most of which can be implemented several different ways. It will have data, address, and control buses. In short, it can be very intimidating to start out by considering a full-blown microprocessor.

A better approach is to start with a "stripped down" version. By initially omitting some of the processor's advanced features, we arrive at a device that can be readily understood and yet maintains the characteristics of an actual microprocessor. Strictly speaking, the microprocessor developed in this unit is hypothetical in nature. However, it is so close to the real thing that the programs we develop for it will actually run on the ET-3400 Microprocessor Trainer. Also, as you will see later, one of the most popular microprocessors in use today is a vastly advanced version of our elementary model.

A block diagram of a basic microcomputer is shown in Figure 2-3, which shows the microprocessor, the memory, and the I/O circuitry. For simplicity, we will ignore the I/O circuitry in this unit. We can do this by assuming that the program and data are already in memory and that the results of any computations will be held in a register or stored in memory. Ultimately of course, the program and data must come from the outside world and the results must be sent to the outside world. But we will save these procedures until a later unit. This will allow us to concentrate on the microprocessor unit and the memory.

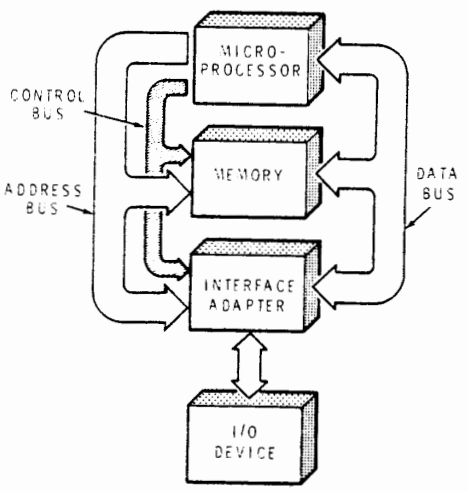


Figure 2-3
The Basic Microcomputer

The Microprocessor Unit (MPU)

The microprocessor unit is shown in greater detail in Figure 2-4. For simplicity, only the major registers and circuits are shown. In our elementary unit most of the counters, registers, and buses are 8-bits wide. That is, they can accomodate 8-bit words.

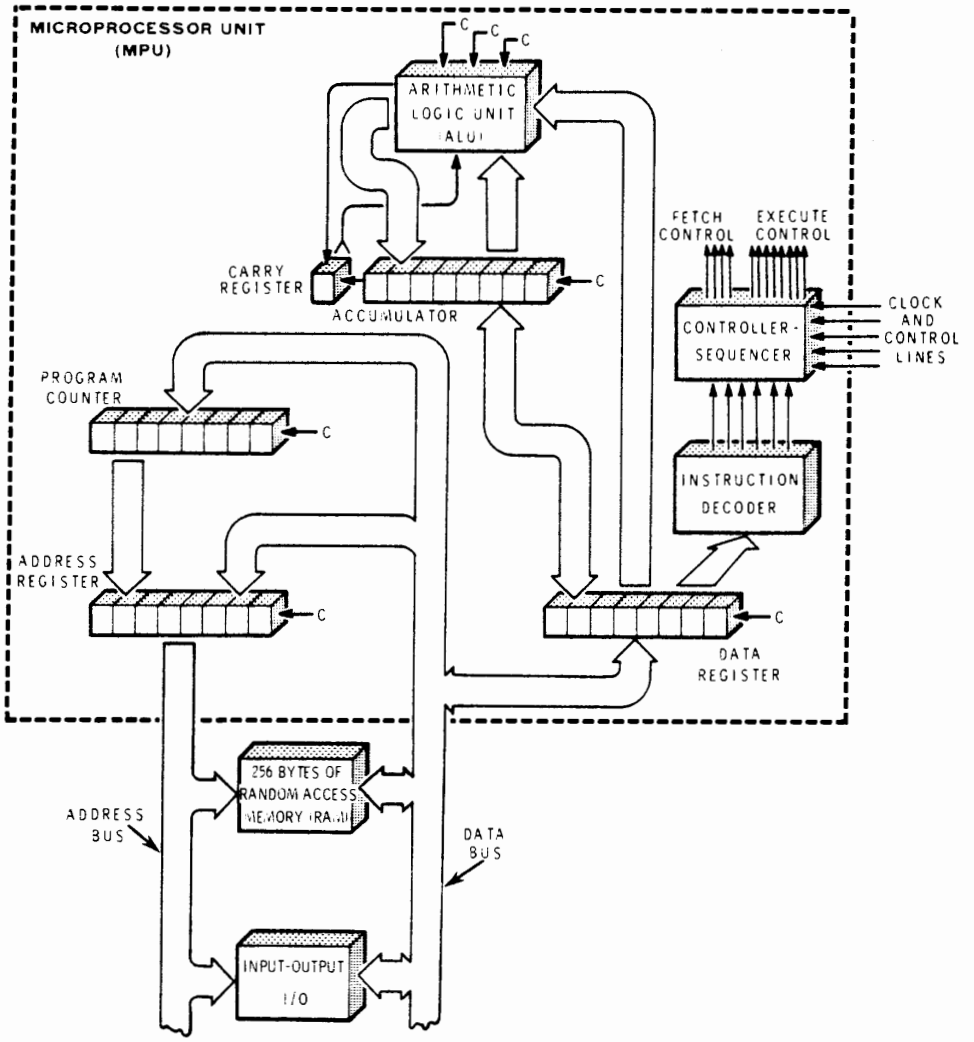


Figure 2-4
An Elementary Microprocessor.

One of the most important circuits in the microprocessor is the **arithmetic logic unit** (ALU). Its purpose is to perform arithmetic or logic operations on the data words that are delivered to it. The ALU has two main inputs. One comes from a register called the accumulator, and the other comes from the data register. The ALU can add the two input data words together, or it can subtract one from the other. It can also perform some logic operations which will be discussed in later units. The operation that the ALU performs is determined by signals on the various control lines (marked C on the block diagram).

Generally, the ALU receives two 8-bit binary numbers from the accumulator and the data register as shown in Figure 2-5A. Because some operation is performed on these data words, the two inputs are called **operands.**

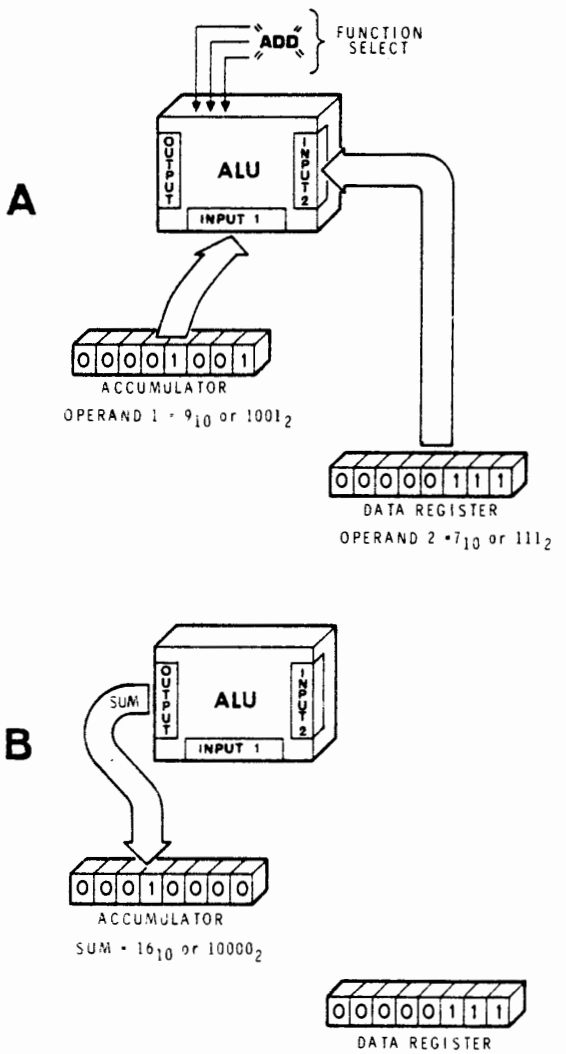


Figure 2-5
The Arithmetic Logic Unit.

The two operands may be added, subtracted, or compared in some way, and the result of the operation is stored back in the accumulator. For example, assume that two numbers (7 and 9) are to be added. Before the numbers can be added, one operand (9) is placed in the accumulator; the other (7) is placed in the data register. The proper control line is then activated to implement the add operation. The ALU adds the two numbers together, producing their sum (16_{10}) at the output. As shown in Figure 2-5B, the sum is stored in the accumulator, replacing the operand that was originally stored there. Notice that all the numbers involved are in binary form.

The **accumulator** is the most useful register in the microprocessor. During arithmetic and logic operations it performs a dual function. Before the operation, it holds one of the operands. After the operation, it holds the resulting sum, difference, or logical answer. The accumulator receives several instructions in every microprocessor. For example, the "load accumulator" instruction causes the contents of some specified memory location to be transferred to the accumulator. The "store accumulator" instruction causes the contents of the accumulator to be stored at some specified location in memory.

The **data register** is a temporary storage location for data going to or coming from the data bus. For example, it holds an instruction while the instruction is being decoded. Also, it holds a data byte while the word is being stored in memory.

The MPU also contains several other important registers and circuits: the address register, the program counter, the instruction decoder, and the controller-sequencer. These are shown in Figure 2-4.

The **address register** is another temporary storage location. It holds the address of the memory location or I/O device that is used in the operation presently being performed.

The **program counter** controls the sequence in which the instructions in a program are performed. Normally, it does this by counting in the sequence, 0, 1, 2, 3, 4, etc. At any given instant, the count indicates the location in memory from which the next byte of information is to be taken.

The **instruction decoder** does just what its name implies. After an instruction is pulled from memory and placed in the data register, the instruction is decoded by this circuit. The decoder examines the 8-bit code and decides which operation is to be performed.

The **controller-sequencer** produces a variety of control signals to carry out the instruction. Since each instruction is different, a different combination of control signals is produced for each instruction. This circuit determines the sequence of events necessary to complete the operation described by the instruction.

Later you will see how these various circuits work together to execute simple programs. But first, take a closer look at the memory for our microcomputer.

Memory

A simplified diagram of the 256- word, 8-bit read/write memory that is used in our hypothetical microcomputer is shown in Figure 2-6. The memory consists of 256_{10} locations, each of which can store an 8-bit word. This size memory is often referred to as 256 × 8. A read/write memory is one in which data can be written in and read out with equal ease.

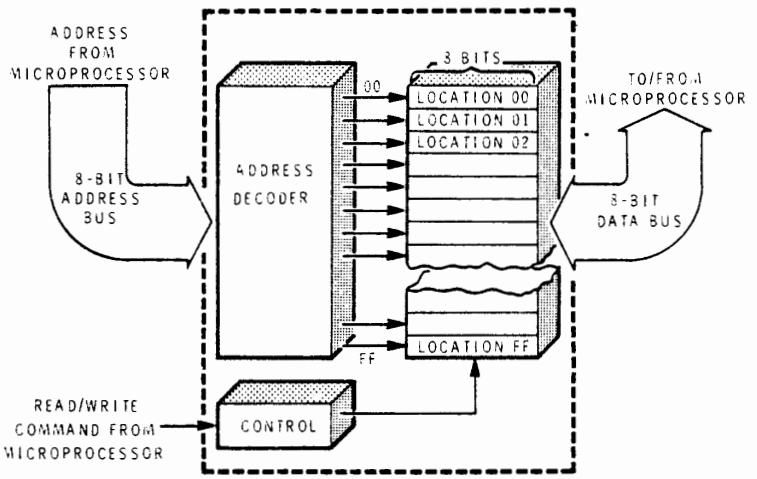


Figure 2-6
The Random Access Memory.

Two buses and a number of control lines connect the memory with the microprocessing unit. The address bus will carry an 8-bit binary number, which can specify 256_{10} locations, from the MPU to the memory address decoder. Each location is assigned a unique number called its address. The first location is given the address 0. The last location is given the address 255_{10}, which is 1111 1111 in binary and FF in hexadecimal. A specific location is selected by placing its 8-bit address on the address bus. The address decoder decodes the 8-bit number and selects the proper memory location.

The memory also receives a control signal from the MPU. This signal tells the memory the operation that is to be performed. A READ signal indicates that the selected location is to be read out. This means that the 8-bit number contained in the selected location is to be placed on the data bus where it can be transferred to the MPU.

The procedure is illustrated in Figure 2-7. Assume that the MPU is to read out the contents of memory location 04_{16}. Let's further assume that the number stored there is 97_{16}. First, the MPU places the address 04_{16} on the address bus. The decoder recognizes the address and selects the proper memory location. Second, the MPU sends a READ signal to the memory, indicating that the contents of the selected location are to be placed on the data bus. Third, the memory responds by placing the number 97_{16} on the data bus. The MPU can then pick up the number and use it as needed.

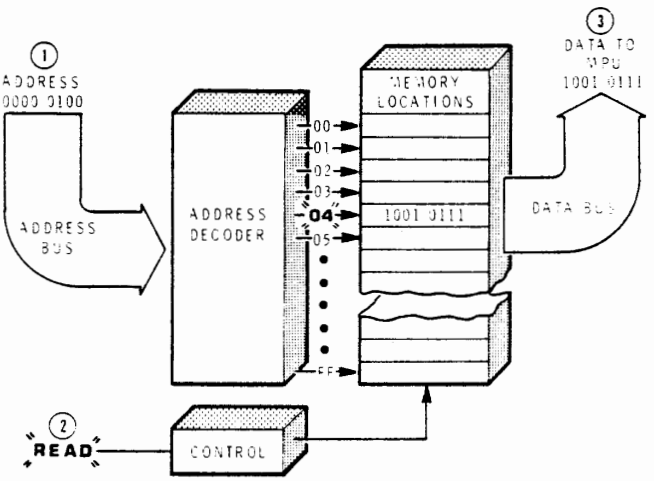


Figure 2-7
Reading from Memory.

It should be pointed out that the process of reading out a memory location does not disturb the contents of that location. That is, the number 97_{16} will still be present at memory location 04 after the read operation is finished. This characteristic is referred to as nondestructive readout (NDRO). It is an important feature because it allows us to read out the same data as many times as needed.

The MPU can also initiate a WRITE operation. This procedure is illustrated in Figure 2-8. During a WRITE operation, a data word is taken from the data bus and placed in the selected memory location. For example, let's see how the MPU can store the number 52_{16} at memory location 03. First, the MPU places the address 03 on the address bus. The decoder responds by selecting memory location 03. Second, the MPU places the number 52_{16} on the data bus. Third, the MPU sends the WRITE signal. The memory responds by storing the number contained on the data bus in the selected location. That is, 52_{16} is stored in location 03. The previous contents of the selected location are lost as the new number is written in that location.

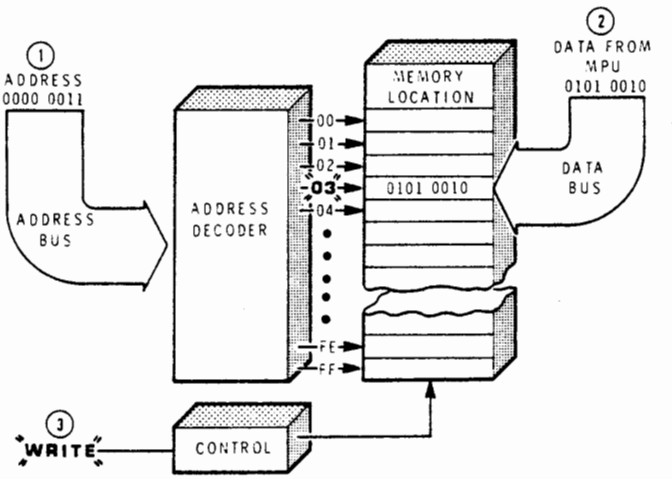


Figure 2-8
Writing into Memory.

The accepted name for a memory of this type is **Random Access Memory** (RAM). "Random access' means that all memory locations are equally accessible. However, in recent years RAM has come to mean a random access **read/write** memory. As you will see later, there is another type of memory called a read only memory (ROM). It is also randomly accessible, but it does not have a write capability. Today, the accepted definition of RAM is a random access **read/write** memory. A read only memory, although it is randomly accessible, is called a ROM and never a RAM.

Fetch-Execute Sequence

When the microcomputer is executing a program, it goes through a fundamental sequence that is repeated over and over again. Recall that a program consists of instructions that tell the microcomputer exactly what operations to perform. These instructions must be stored in an orderly manner in memory. Instructions must be fetched, one at a time, from memory by the MPU. The instruction is then executed by the MPU.

The operation of the microcomputer can be broken down into two phases, as shown in Figure 2-9. When the microprocessor is initially started, it enters the **fetch phase**. During the fetch phase, an instruction is taken from memory and decoded by the MPU. Once the instruction is decoded, the MPU switches to the **execute phase**. During this phase, the MPU carries out the operation dictated by the instruction.

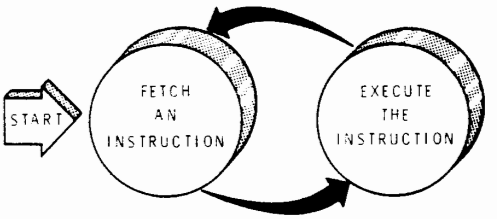


Figure 2-9
The Fetch-Execute Sequence.

The fetch phase always consists of the same series of operations. Thus, it always takes the same amount of time. However, the execute phase will consist of different sequences of events, depending on what type of instruction is being executed. Thus, the time of the execute phase may vary considerably from one instruction to the next.

A Sample Program

Now that you have a general idea of the registers and circuits found in a microcomputer, we will examine how all these circuits work together to execute a simple program. At this point, we are primarily interested in showing you how the microcomputer operates. Therefore, the program will be a very trivial one.

Let's see how the computer goes about solving a problem like 7 + 10 = ? While this seems like an incredibly easy problem, the computer, if left to its own resources, does not have the foggiest notion of how to solve it. You must tell the computer how to solve the problem right down to the smallest detail. You do this by writing a program.

Before you can write the program you must know what instructions are available to you and the computer. Every microprocessor comes with a listing of its instruction set. Assume that, after looking over the list, you decide that three instructions are necessary to solve the problem. These three instructions and a description of what they do are shown in Figure 2-10.

NAME	MNEMONIC	OPCODE	DESCRIPTION
Load Accumulator	LDA	$1000\ 0110_2$ or 86_{16}	Load the contents of the next memory location into the accumulator.
Add	ADD	$1000\ 1011_2$ or $8B_{16}$	Add the contents of the next memory location to the present contents of the accumulator. Place the sum in the accumulator.
Halt	HLT	$0011\ 1110_2$ or $3E_{16}$	Stop all operations.

Figure 2-10
Instructions Used in the Sample Program.

The first column in the table gives the name of the instruction. When writing programs, it is often inconvenient to write out the entire name. For this reason, each instruction is given an abbreviation or a memory aid called a **mnemonic**. The mnemonics are given in the second column. The third column is called the operation code or **opcode**. This is the binary number that the computer and the programmer use to represent the instruction. The opcode is given in both binary and hexadecimal form. The final column describes exactly what operation is performed when the instruction is executed. Study this table carefully; you will be using these instructions over and over again.

Assume that you wish to add 7 to 10_{10} and place the sum in the accumulator. The program is an elementary one. First you will load 7 into the accumulator with the LDA instruction. Next, you will add 10_{10} to the accumulator using the ADD instruction. Finally, you will stop the computer with the HLT instruction.

Using the mnemonics and the decimal representation of the numbers to be added, the program looks like this:

```
LDA 7
ADD 10
HLT
```

Unfortunately, the basic microcomputer cannot understand mnemonics or decimal numbers. It can interpret binary numbers and nothing else. Thus, you must write the program as a sequence of binary numbers. You can do this by replacing each mnemonic with its corresponding opcode and each decimal number with its binary counterpart.

That is:

LDA 7	becomes	1000 0110 opcode from Figure 2-10	0000 0111 binary representation for 7

And:

ADD 10	becomes	1000 1011 opcode from Figure 2-10	0000 1010 binary representation for 10_{10}

Finally,

HLT	becomes	0011 1110 opcode from Figure 2-10

Notice that the program consists of three instructions. The first two instructions have two parts: an 8-bit opcode followed by an 8-bit operand. The operands are the two numbers that are to be added (7 and 10_{10}).

Recall that the microprocessor and memory work with 8-bit words or bytes. Because the first two instructions consist of 16-bits of information, they must be broken into two 8-bit bytes before they can be stored in memory. Thus, when the program is stored in memory, it will look like this:

1st Instruction	1000	0110	Opcode for LDA
	0000	0111	Operand (7)
2nd Instruction	1000	1011	Opcode for ADD
	0000	1010	Operand (10_{10})
3rd Instruction	0011	1110	Opcode for HLT

Five bytes of memory are required. You can store this 5-byte program any place in memory you like. Assuming you store it at the first five memory locations, the memory can be diagrammed as shown in Figure 2-11.

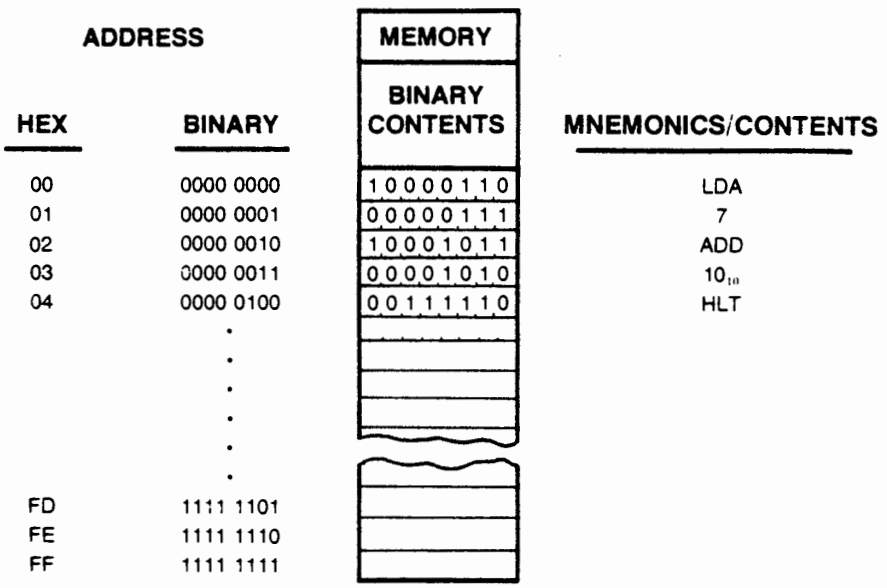


Figure 2-11
The Program in Memory.

Notice that each memory location has two 8-bit binary numbers associated with it. One is its address, the other is its contents. Be careful not to confuse these two numbers. The address is fixed. It is established when the microcomputer is built. However, the contents may be changed at any time by storing new data.

Before you see how this program is executed, let's review the material covered in this section.

Self-Test Review

12. The circuit in the microprocessor that performs arithmetic and logic operations is called the _____.

13. The numbers that are operated upon by the computer are called _____.

14. Where are the two operands held as they are transferred to the ALU?

15. Where is the result held after an arithmetic operation?

16. Which register in the MPU holds the instruction while the opcode is being decoded?

17. Which circuit in the MPU determines the memory location from which the next byte of information will be taken?

18. Which register holds the address while it is being decoded?

19. How many memory locations can be specified by an 8-bit address?

20. Explain the 3-step procedure for writing the number 34_{16} into memory location 9.

21. Define mnemonic.

22. Define opcode.

23. Define RAM.

24. An instruction is retrieved from memory and decoded during the _____ phase.

25. The operation indicated by the instruction is carried out during the _____ phase.

26. In our hypothetical microcomputer, the mnemonic for the load accumulator instruction is _____.

27. The opcode for the halt instruction is _____.

28. When the ADD instruction is executed, where is the SUM stored?

29. How many memory locations are required to store the following program?

```
LDA  13₁₀
ADD  17₁₀
ADD  10₁₀
HLT
```

30. What will be the contents of the accumulator after this program is executed?

Answers

12. Arithmetic logic unit (ALU).

13. Operands.

14. One is held in the accumulator; the other in the data register.

15. In the accumulator.

16. Data register.

17. The program counter.

18. The address register.

19. $2^8 = 256_{10}$.

20. Step 1. The address 9 is placed on the address bus.

    Step 2. The data 34_{16} is placed on the data bus.

    Step 3. The read/write line is switched to the write state.

21. A mnemonic is an abbreviation or memory aid.

22. An opcode is a binary number that tells the microprocessor which instruction to execute.

23. RAM has come to mean a random access read/write memory.

24. Fetch.

25. Execute.

26. LDA.

27. $0011\ 1110_2$.

28. In the accumulator.

29. Seven.

30. 40_{10} or 101000_2.

EXECUTING A PROGRAM

Before a program can be run, it must be placed in memory. Later, you will see how this is done. For now, assume that we have already loaded the program developed in the previous section.

The important registers of the microcomputer are shown in Figure 2-12. Notice that our 5-byte program for adding 7 and 10_{10} is shown in memory. The following paragraphs will take you through the step-by-step procedure by which the computer executes this program.

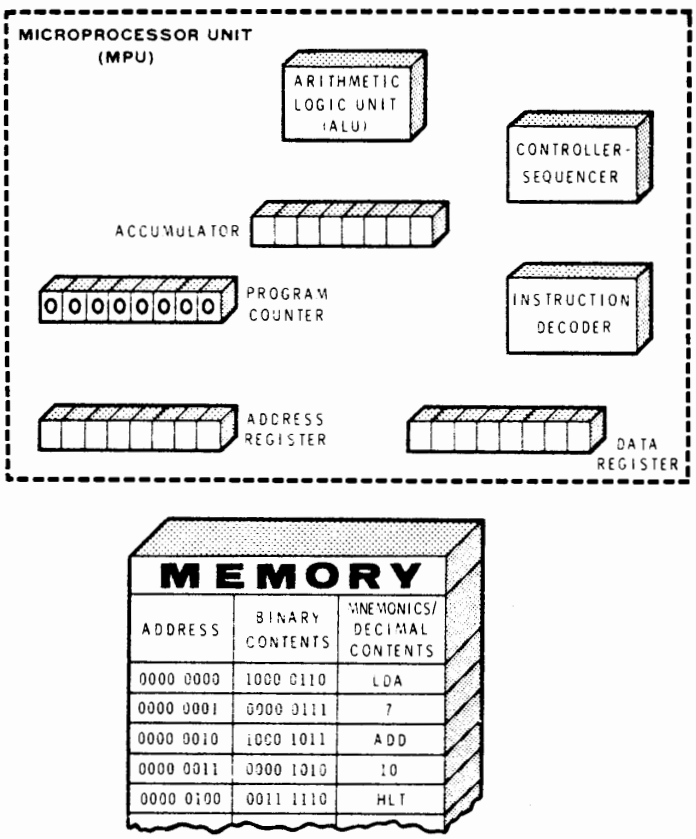


Figure 2-12
The Program Counter is Set to the Address of the First Instruction.

To begin executing the program, the program counter must be set to the address of the first instruction. In this case, the first instruction is in memory location 0000 0000, so the program counter is set accordingly. The procedure for setting the program counter to the proper address will be discussed later.

The Fetch Phase

The first step is to fetch the first instruction from memory. The sequence of events that happen during the fetch phase is controlled by the controller-sequencer. It produces a number of control signals which will cause the events illustrated in Figure 2-13 through 2-17 to occur.

First the contents of the program counter are transferred to the address register as shown in Figure 2-13. Recall that this is the address of the first instruction.

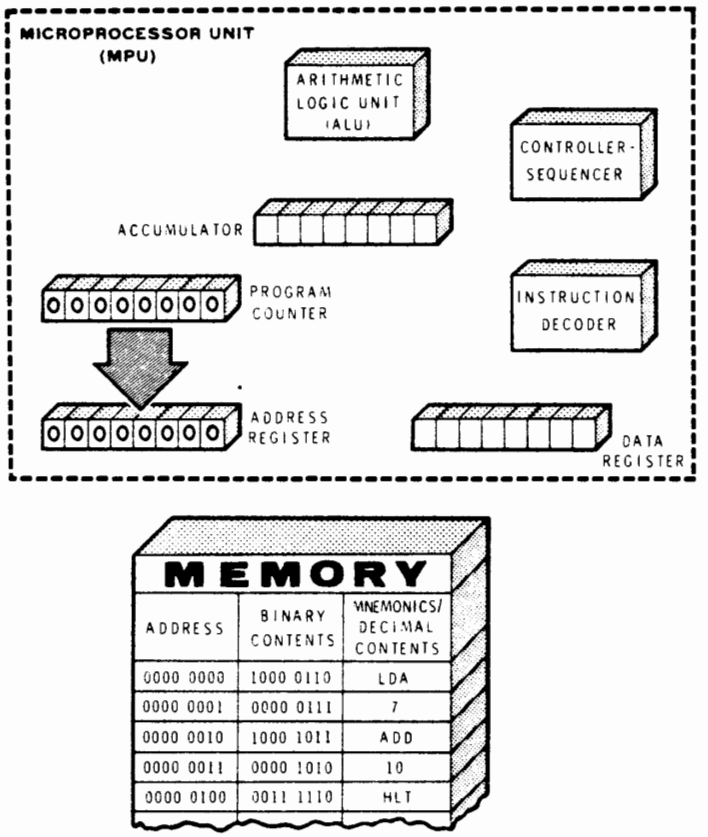


Figure 2-13
The Contents of the Program Counter are Transferred to the Address Register.

Once the address is safely in the address register, the program counter is incremented by one. That is, its contents change from 0000 0000 to 0000 0001 (Figure 2-14). Notice that this does not change the contents of the address register in any way.

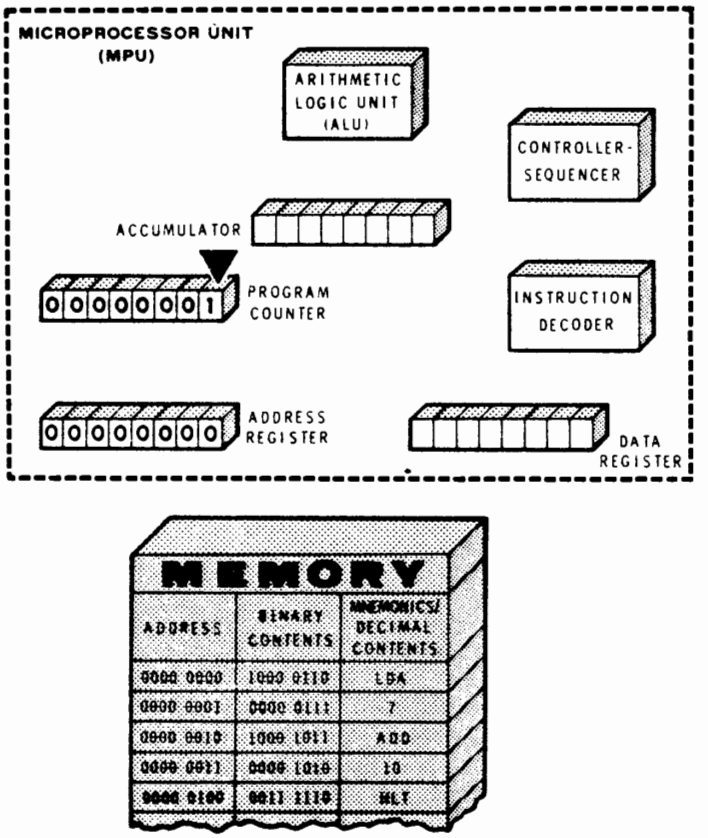


Figure 2-14
The Program Counter is Incremented.

Next, the contents of the address register (0000 0000) are placed on the address bus (Figure 2-15). The memory circuits decode the address and select memory location 0000 0000.

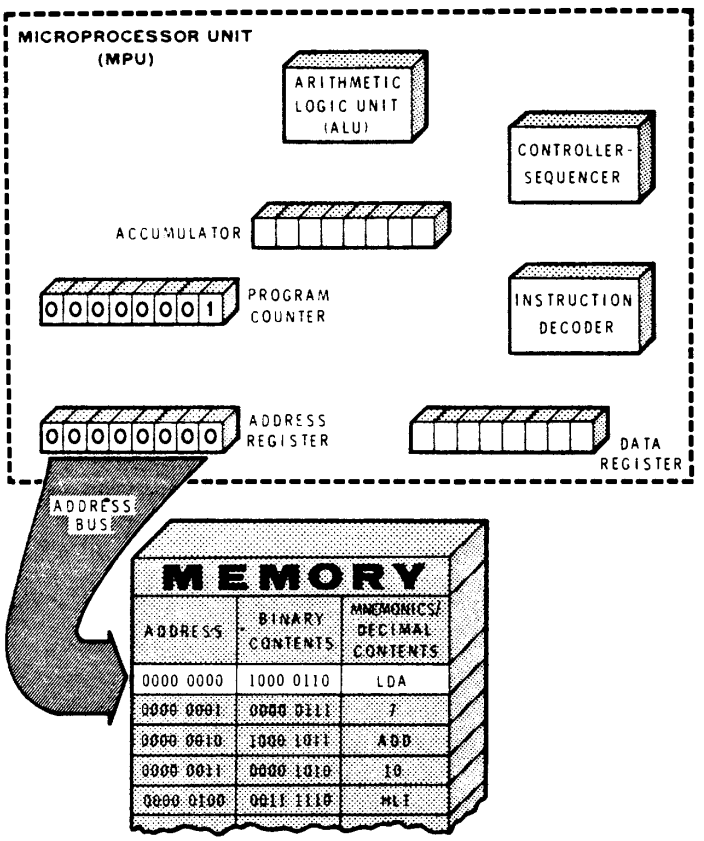


Figure 2-15
The Address of the First Instruction is Placed on the Address Bus.

The contents of the selected memory location are placed on the data bus and transferred to the data register in the MPU. After this operation, the opcode for the LDA instruction will be in the data register as shown in Figure 2-16.

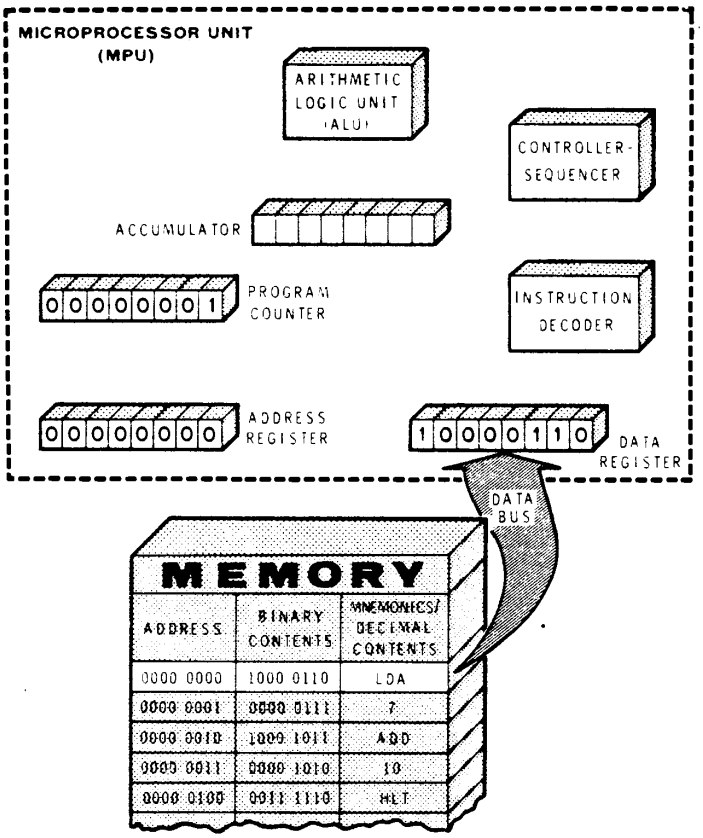


Figure 2-16
The Opcode of the First Instruction is Placed on the Data Bus.

The next step is to decode the instruction (Figure 2-17). The opcode is transferred to the instruction decoder. This circuit recognizes that the opcode is that of an LDA instruction. It informs the controller-sequencer of this fact and the sequencer produces the necessary control pulses to carry out the instruction. This completes the fetch phase of the first instruction.

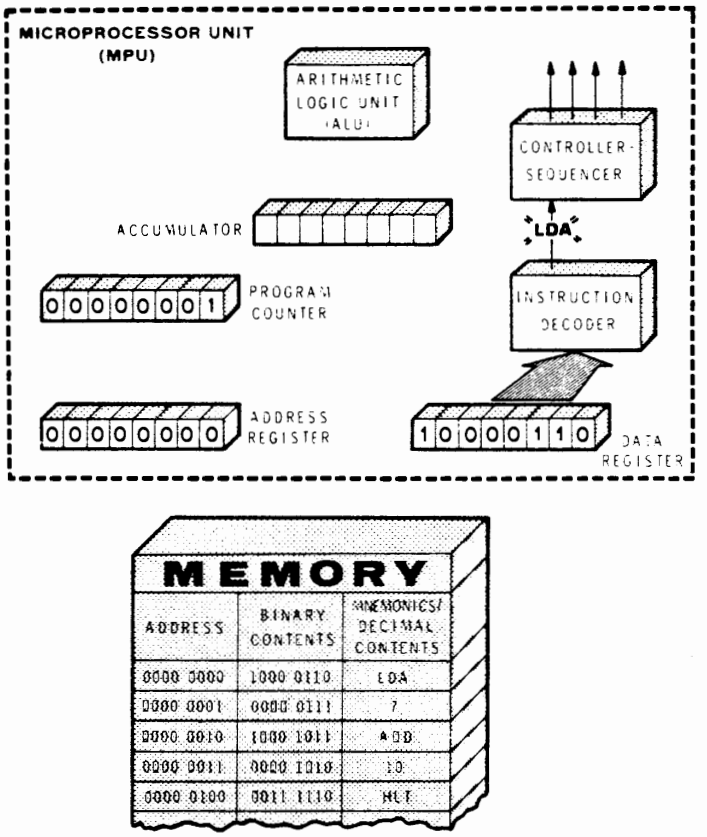
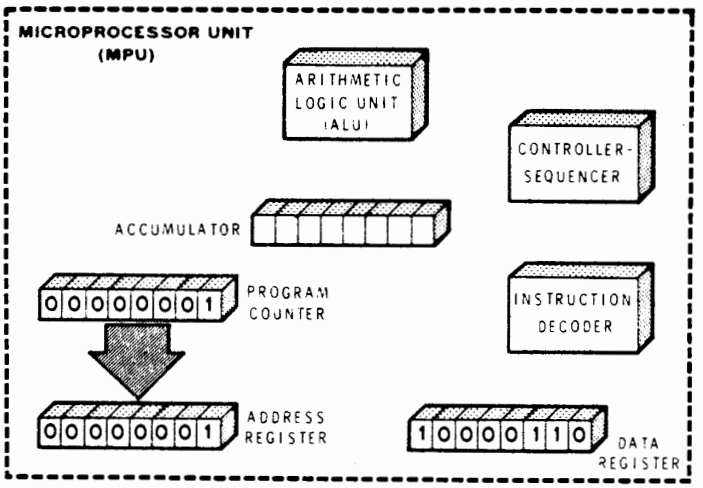


Figure 2-17
The Opcode is Decoded.

The Execute Phase

The first instruction was fetched from memory and decoded during the fetch phase. The MPU now knows that this is an LDA instruction. During the execute phase, it must carry out this instruction by reading out the next byte of memory and placing it in the accumulator.

The first step is to transfer the address of the next byte from the program counter to the address register (Figure 2-18). You will recall that the program counter was incremented to the proper address (0000 0001) during the previous fetch phase.



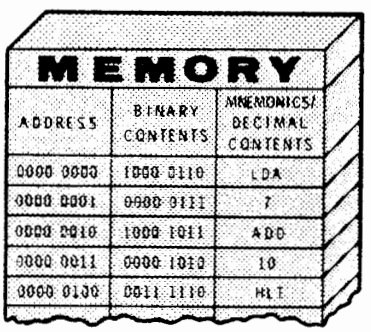


Figure 2-18
The Contents of the Program Counter are Transferred to the Address Register.

The next two operations are shown in Figure 2-19. First, the program counter is incremented to 0000 0010 in anticipation of the next fetch phase. Second, the contents of the address register (000 0001) are placed on the address bus.

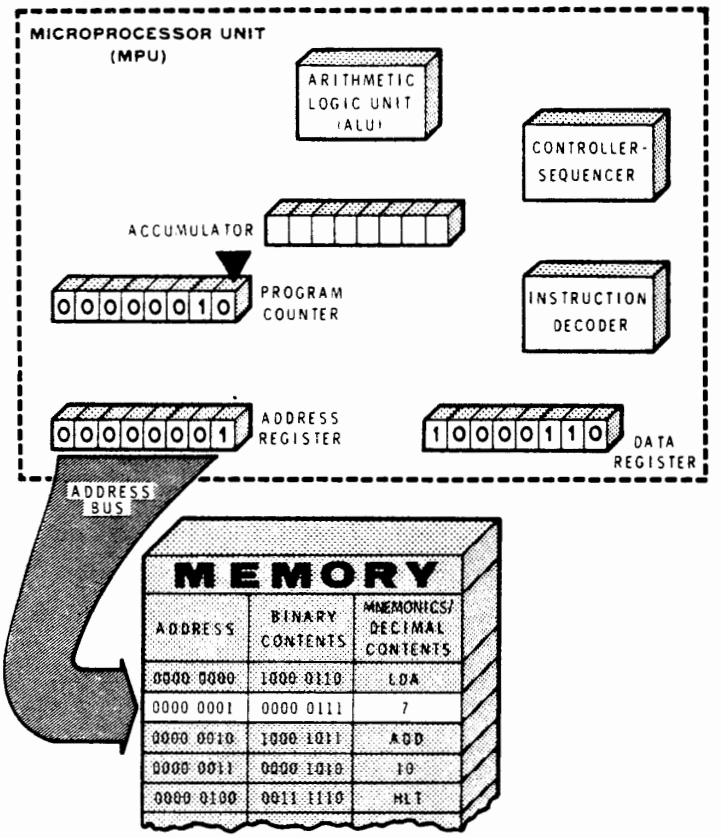


Figure 2-19
The Program Counter is Incremented; the Contents of the Address Register are Placed on the Address Bus.

The address is decoded and the contents of memory location 0000 0001 are loaded into the data register as shown in Figure 2-20. Recall that this is the number 7. An instant later, the number is transferred to the accumulator. Thus, the first execute phase ends with the number 7 in the accumulator.

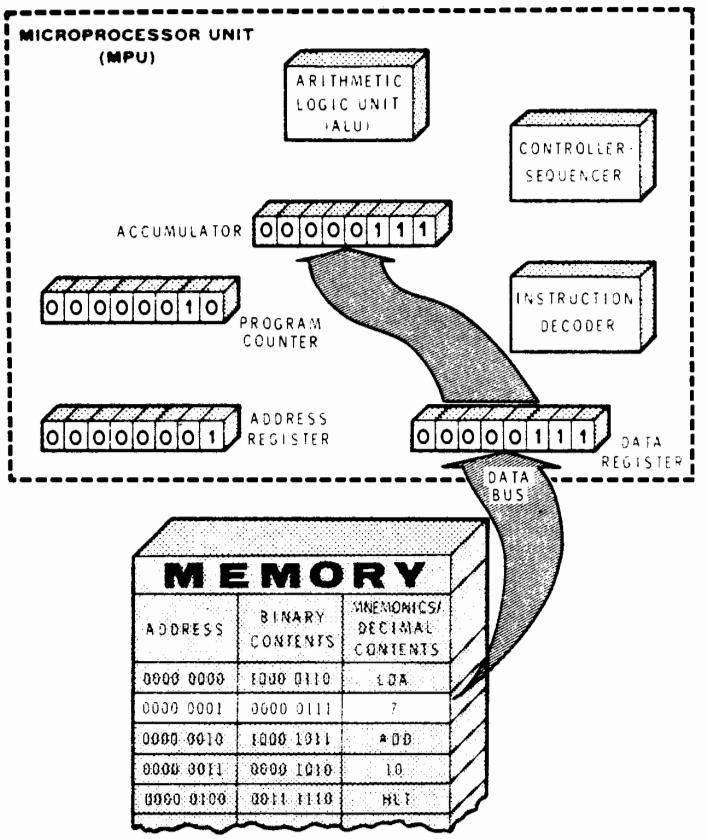


Figure 2-20
The First Operand is Transferred to the Accumulator Via the Data Register.

Fetching the Add Instruction

The next instruction in our program is the ADD instruction. It is fetched from memory using the same procedure outlined above. Figure 2-21 illustrates this five-step procedure:

1. The contents of the program counter (0000 0010) are transferred to the address register.

2. The program counter is incremented to 0000 0011.

3. The address is placed on the address bus.

4. The contents of the selected memory location are transferred to the data register.

5. The contents of the data register are decoded by the instruction decoder.

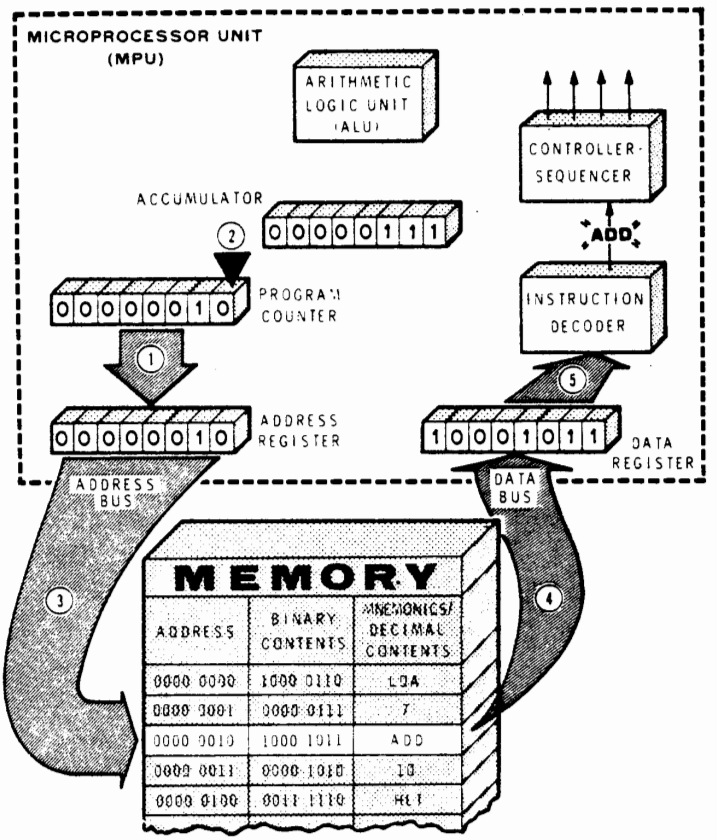


Figure 2-21
Fetching the ADD Instruction.

The data word fetched from memory is the opcode for the ADD instruction. Thus, the controller-sequencer produces the necessary control pulses to execute this instruction.

Executing the Add Instruction

The execution of the ADD instruction is a six-step procedure. This procedure is illustrated in Figure 2-22.

1. The contents of the program counter (0000 0011) are transferred to the address register.

2. The program counter is incremented to 0000 0100 in anticipation of the next fetch phase.

3. The address of the operand is placed on the address bus.

4. The operand (10_{10}) is transferred to the data register.

5A. The operand (10_{10}) is transferred into one input of the ALU.

5B. Simultaneously, the other operand (7) is transferred from the accumulator to the other input of the ALU.

6. The ALU adds the two operands. Their sum ($0001\ 0001_2$ or 17_{10}) is loaded into the accumulator, destroying the number (7) that was previously stored there.

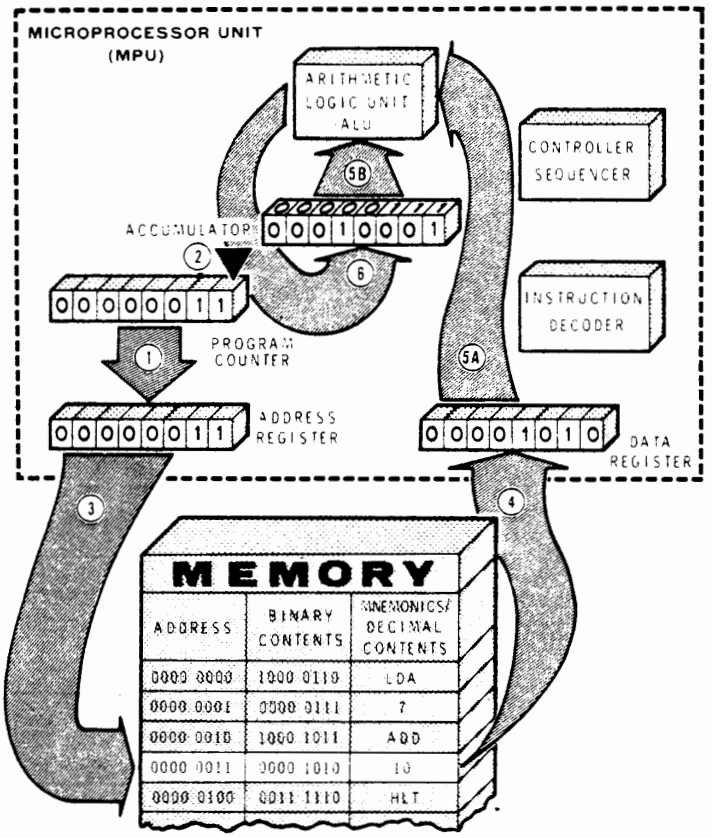


Figure 2-22
Executing the ADD Instruction.

The computation portion of our program ends with the sum of the two operands in the accumulator. However, the program is not finished until it tells the computer to stop executing instructions.

Fetching and Executing the HLT Instruction

The final instruction in the program is a HLT instruction. It is fetched using the same fetch procedure as before. The five steps are illustrated in Figure 2-23. The address comes from the program counter via the address register. When this address is placed on the address bus, memory location 0000 0100 is read out and the opcode for HLT is loaded into the data register. The opcode is decoded and the instruction is executed.

The execution of the HLT instruction is very simple. The controller-sequencer simply stops producing control signals. Consequently, all computer operations stop. Notice that the program has accomplished our objective of adding 7_{10} to 10_{10}. The resulting sum, 17_{10}, is in the accumulator.

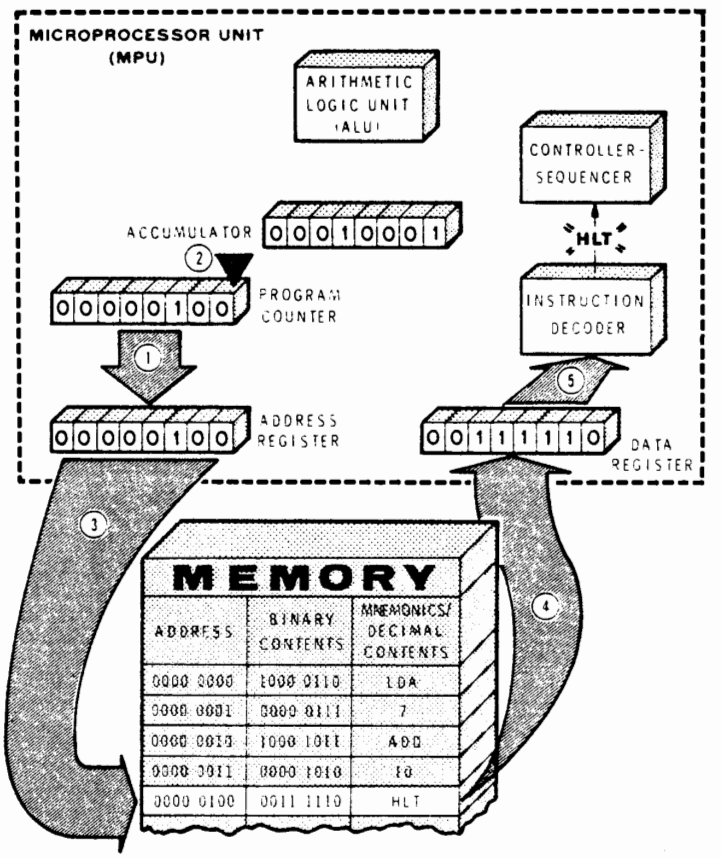


Figure 2-23
Fetching and Executing the HLT Instruction.

Self-Test Review

Examine this sample program carefully and answer the questions below:

```
LDA  8
ADD  7
ADD  4
HLT
```

31. During the first fetch phase, what binary number is loaded into the data register?

32. During the first execute phase the number $0000\ 1000_2$ is loaded into the _____.

33. During the second fetch phase, what binary number is loaded into the data register?

34. If the first byte of the program is placed in address 0000 0000, what is the address of the first ADD instruction?

35. How many bytes of memory are taken up by the program?

36. What number is in the accumulator during the third fetch phase?

37. When the program is finished running, what number is in the accumulator?

38. What is the final number in the program counter?

39. What are the final contents of the address register?

40. What are the final contents of the data register?

Answers

31. The opcode for the LDA instruction, $1000\ 0110_2$.

32. Accumulator.

33. The opcode for ADD, 1000 1011.

34. $0000\ 0010_2$.

35. Seven.

36. 15_{10} or $0000\ 1111_2$.

37. 19_{10} or $0001\ 0011_2$.

38. 7_{10} or $0000\ 0111_2$.

39. 6_{10} or $0000\ 0110_2$.

40. The opcode for the HLT instruction, $0011\ 1110_2$.

ADDRESSING MODES

If you examine the program discussed in the previous section, you will find that it uses two distinctly different types of instructions. One type of instruction requires an operand. LDA and ADD are examples of this type. These are two-byte instructions. The first byte is the opcode; the second is the operand.

Microprocessors also have single-byte instructions. HLT is a good example. This instruction requires no operand; thus, it can be implemented with a single byte.

Instructions can be classified in several different ways. One of the most basic distinctions is their addressing mode. The addressing mode refers to the method by which an instruction addresses its operand.

Inherent or Implied Addressing

Single-byte instructions have no operand or the operand is implied by the opcode itself. For example, the HLT instruction has no operand at all. However, some single-byte instructions may have an implied operand. An example is the "increment accumulator" instruction. The number that is operated upon (incremented) is the number in the accumulator. This instruction simply adds one to the contents of the accumulator. This type of addressing will be referred to in this course as **implied addressing** or **inherent addressing**. The operations performed during an instruction of this type are illustrated in Figure 2-24.

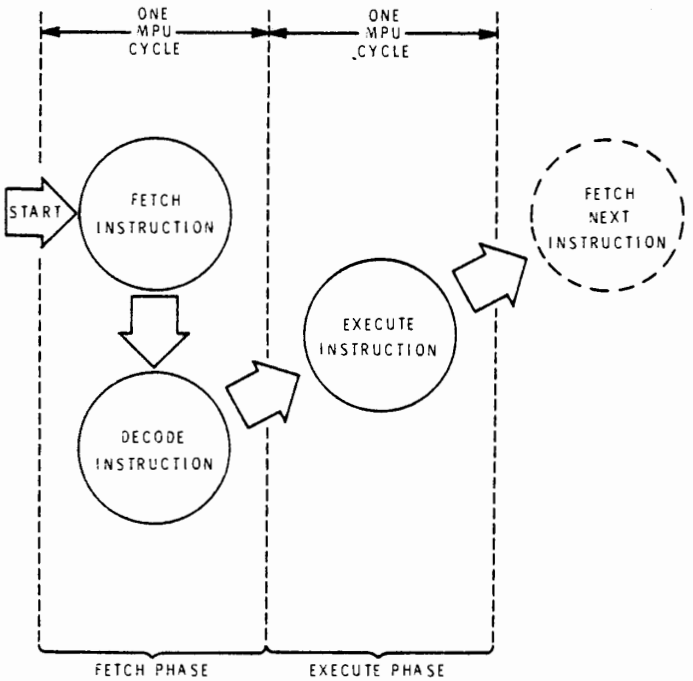


Figure 2-24
Operations Performed in the Inherent or Implied Addressing Mode.

Immediate Addressing

In our previous program, the two-byte instructions use the **immediate addressing** mode. In this mode, the operand is the byte immediately following the opcode. That is, the byte of data that is to be operated upon is the second byte of the instruction. When these two-byte instructions are stored in memory, the address of the operand is the memory location following the opcode. The operations performed during this type of instruction are illustrated in Figure 2-25.

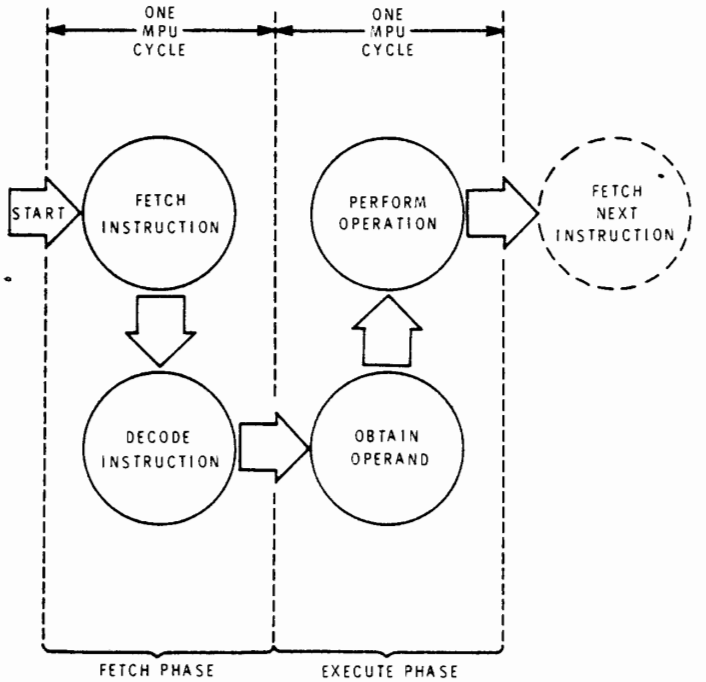


Figure 2-25
Operations Performed in the Immediate Addressing Mode.

The inherent and immediate addressing modes have two advantages. First, they require little memory space; one and two bytes respectively. This is important because memory locations cost money. Generally, the less memory space taken by a program, the better off we are. Second, these addressing modes require a minimum of execution time. The execution time can become important in long programs.

The time required for an instruction to be fetched and executed is often given in **MPU cycles**. We will define an MPU cycle as the minimum time required to fetch a data byte from memory. Thus, the fetch phase of an instruction requires one MPU cycle. In inherent and immediate addressing modes, the execute phase also requires one MPU cycle. Therefore, the **minimum** time required to fetch and execute any instruction is two MPU cycles. As you will see later, other addressing modes will require more MPU cycles.

Because of their fast execution time and their efficient use of memory, the inherent and immediate modes of addressing should be used wherever possible. However, there are many situations in which these addressing modes are simply not suitable. For this reason, every microprocessor will have instructions which use other addressing modes.

Direct Addressing

Most computer operations involve an operand. To realize its full potential, the computer must be able to manipulate the operand in many different ways. Immediate addressing of an operand is most useful when the operand is a constant that is used by only one instruction in the program. In this case, the operand can be placed immediately after that instruction's opcode.

However, there are many cases in which the operand is a variable which may be operated upon by many different instructions. In these cases, the immediate addressing mode is simply not practical and a more sophisticated form of addressing is necessary.

One way of solving this problem is to use the **direct addressing mode**. In this mode, an instruction requires two bytes of memory. The first byte is the opcode of the instruction just as before. However, the second byte is **not** the operand. Instead, the second byte is the **address** of the operand. The format of the instruction as it appears in memory is shown in Figure 2-26.

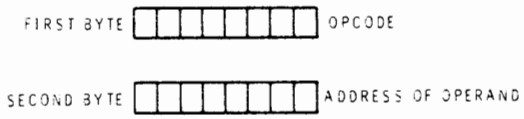


Figure 2-26
Format of an Instruction which uses the Direct Addressing Mode.

Three typical direct-addressing-mode instructions are listed in Figure 2-27. The first is the load accumulator instruction (LDA). Read the description carefully and note the difference between this instruction and the LDA immediate instruction discussed earlier. An example of each may help. In the **immediate addressing** mode, the instruction

LDA 50_{10}

means "load 50_{10} into the accumulator." But in the **direct addressing** mode, the same instruction means "load the number at memory location 50_{10} into the accumulator."

NAME	MNEMONIC	OPCODE	DESCRIPTION
Load Accumulator	LDA	$1001\ 0110_2$ or 96_{16}	Load the contents of the memory location whose address is given by the next byte into the accumulator.
Add	ADD	$1001\ 1011_2$ or $9B_{16}$	Add the contents of the memory location whose address is given by the next byte to the present contents of the accumulator. Place the sum in the accumulator.
Store Accumulator	STA	$1001\ 0111_2$ or 97_{16}	Store the contents of the accumulator in the memory location whose address is given by the next byte.

Figure 2-27
Direct Addressing Mode Instructions.

You may have noticed that this instruction has a different opcode from the LDA immediate instruction. This is necessary to tell the MPU the exact nature of the instruction. That is, the opcode tells the MPU the addressing mode as well as the operation that is to be performed.

The ADD instruction also has a slightly different meaning in the direct addressing mode. Recall that, in the **immediate** addressing mode,

ADD 10_{10}

means "add 10_{10} to the contents of the accumulator." However, in the **direct** addressing mode.

ADD 10_{10}

means "add the contents of memory location 10_{10} to the contents of the accumulator." Once again, the opcode tells the MPU the addressing mode. An opcode of $8B_{16}$ means ADD immediate whereas an opcode of $9B_{16}$ means ADD direct.

The last instruction shown is a store accumulator (STA) instruction. It tells the MPU to store the contents of the accumulator in the address indicated by the second byte of the instruction.

For example:

STA 20_{10}

means "store the contents of the accumulator in memory location 20_{10}." Because the second byte of the instruction must be an address, there is no STA immediate instruction.

The direct addressing mode instructions require one or more additional MPU cycles to execute. A typical fetch-execute sequence is illustrated in Figure 2-28. The instruction fetch is the same regardless of the addressing mode. However, the execution phase is extended since the MPU must first obtain the address of the operand from memory and then retrieve the operand itself from memory.

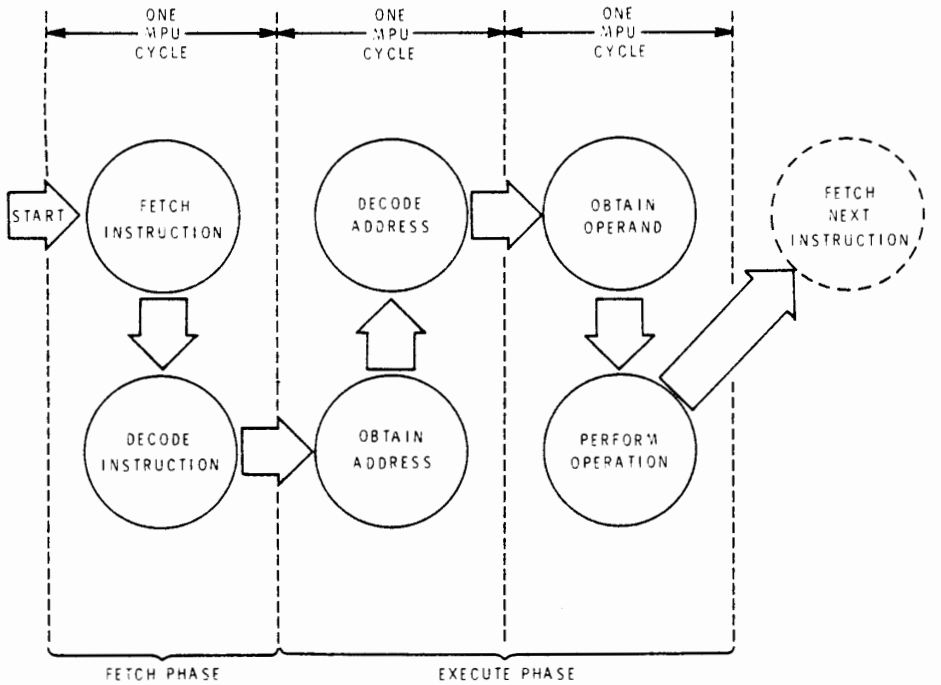


Figure 2-28
Most Direct Addressing Mode Instructions Require Three MPU Cycles.

Direct addressing generally requires more memory and longer execution times. However, there are many cases in which the added flexibility makes direct addressing worthwhile in spite of these disadvantages.

Sample Program Using Direct Addressing

The differences between direct and immediate addressing can be illustrated by a sample program. Earlier we examined a program which added two numbers (7 and 10). The sum was placed in the accumulator. Now let's look at the same program using direct addressing, only this time the sum will be stored in memory.

Figure 2-29 shows the program as it would look when stored in memory. Assume that we have arbitrarily stored the program starting at memory location or address $0001\ 0000_2$ (16_{10}). Addresses 16_{10} and 17_{10} contain the first instruction:

LDA 23_{10}

BINARY ADDRESS	BINARY CONTENTS	MNEMONICS/ CONTENTS
0001 0000	1001 0110	LDA
0001 0001	0001 0111	23_{10} } 1st Instruction
0001 0010	1001 1011	ADD
0001 0011	0001 1000	24_{10} } 2nd Instruction
0001 0100	1001 0111	STA
0001 0101	0001 1001	25_{10} } 3rd Instruction
0001 0110	0011 1110	HLT } 4th Instruction
0001 0111	0000 0111	7_{10}
0001 1000	0000 1010	10_{10} } Data
0001 1001	0000 0000	Reserved for sum

Figure 2-29
Sample Program Using Direct Addressing.

This instruction tells the MPU to load the contents of memory location 23_{10} into the accumulator. Looking down to address 23_{10} ($0001\ 0111_2$), you see that it contains the operand 7. Thus, the first instruction causes 7 to be loaded into the accumulator.

The second instruction is in memory locations 18_{10} and 19_{10}. It is:

ADD 24_{10}

This tells the MPU to add the number at address 24_{10} to the number in the accumulator. Address 24_{10} ($0001\ 1000_2$) contains the number 10_{10}. Therefore, the second instruction causes 10_{10} to be added to the contents of the accumulator. The sum (17_{10}) is placed back in the accumulator.

The third instruction, in locations 20_{10} and 21_{10}, is:

STA 25_{10}

This tells the MPU to store the contents of the accumulator in memory location 25_{10}. After this instruction is executed, the number 17_{10} will appear in location 25_{10}.

The final instruction tells the MPU to halt. The program illustrates the value of the HLT instruction. Let's assume that the HLT instruction is inadvertently omitted. In this case, the MPU would fetch the next byte in sequence and attempt to execute it as if it were an instruction. The next byte is the number 7_{10}. This is a data word and was never intended to be an instruction. Nevertheless, the MPU probably has an instruction with an opcode of 7_{10}. After executing this instruction, the MPU will continue to fetch and execute whatever it finds in the remaining memory locations. Without a HLT instruction, it has no way of knowing where the instructions end and data begins.

Executing the Sample Program

The data flow within the microcomputer is slightly different for the direct addressing mode. Figure 2-30 shows several of the data paths within the microcomputer. Using this type of diagram, let's examine the data manipulations that occur during the execution of our sample program.

Notice that our program is loaded in memory starting at address 16_{10}. The program counter is set to 16_{10}, so the MPU is ready to begin executing the program.

The first fetch phase is illustrated in Figure 2-30. During this phase:

1. The contents of the program counter are loaded into the address register.
2. The program counter is incremented to 17_{10}.
3. The contents of the address register are placed on the address bus.
4. The contents of the selected memory location are transferred via the data bus to the data register.
5. The contents of the data register are decoded.
6. The MPU recognizes that an LDA direct operation is indicated. This concludes the fetch phase.

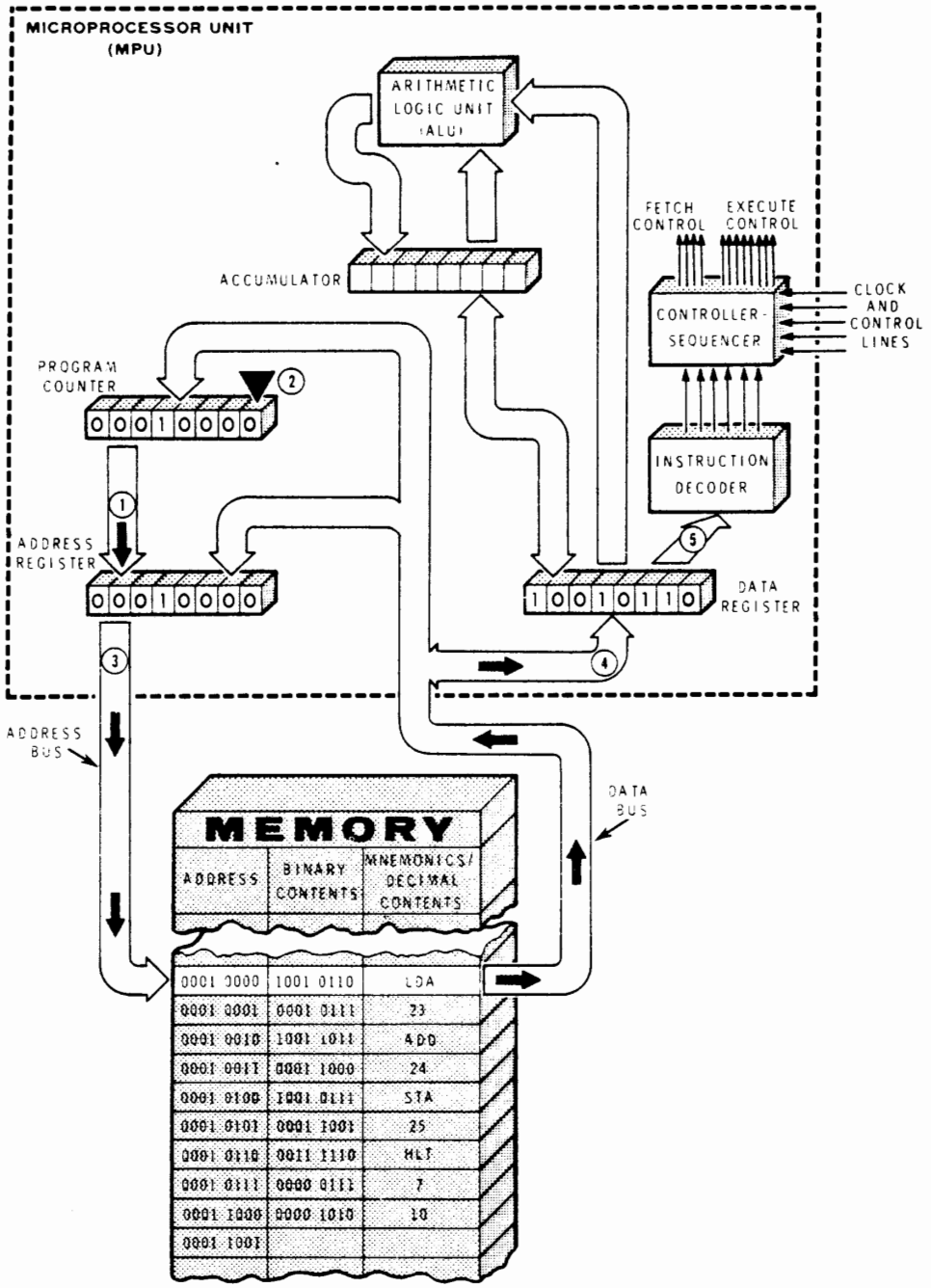


Figure 2-30
Fetching the Opcode of the First Instruction.

The execute phase has two parts when direct addressing is indicated. Figure 2-31 illustrates the first half of the execute phase. During this half:

1. The contents of the program counter are transferred to the address register. This number is the memory location that holds the address of the operand.

2. The program counter is incremented to 18_{10}.

3. The contents of the address register are placed on the address bus.

4. The contents of the selected memory location are placed on the data bus. However, in the direct addressing mode, this data is transferred to the address register. Thus, 23_{10} goes into the address register, replacing the previous contents. After this cycle, the address register will appear as shown in Figure 2-32.

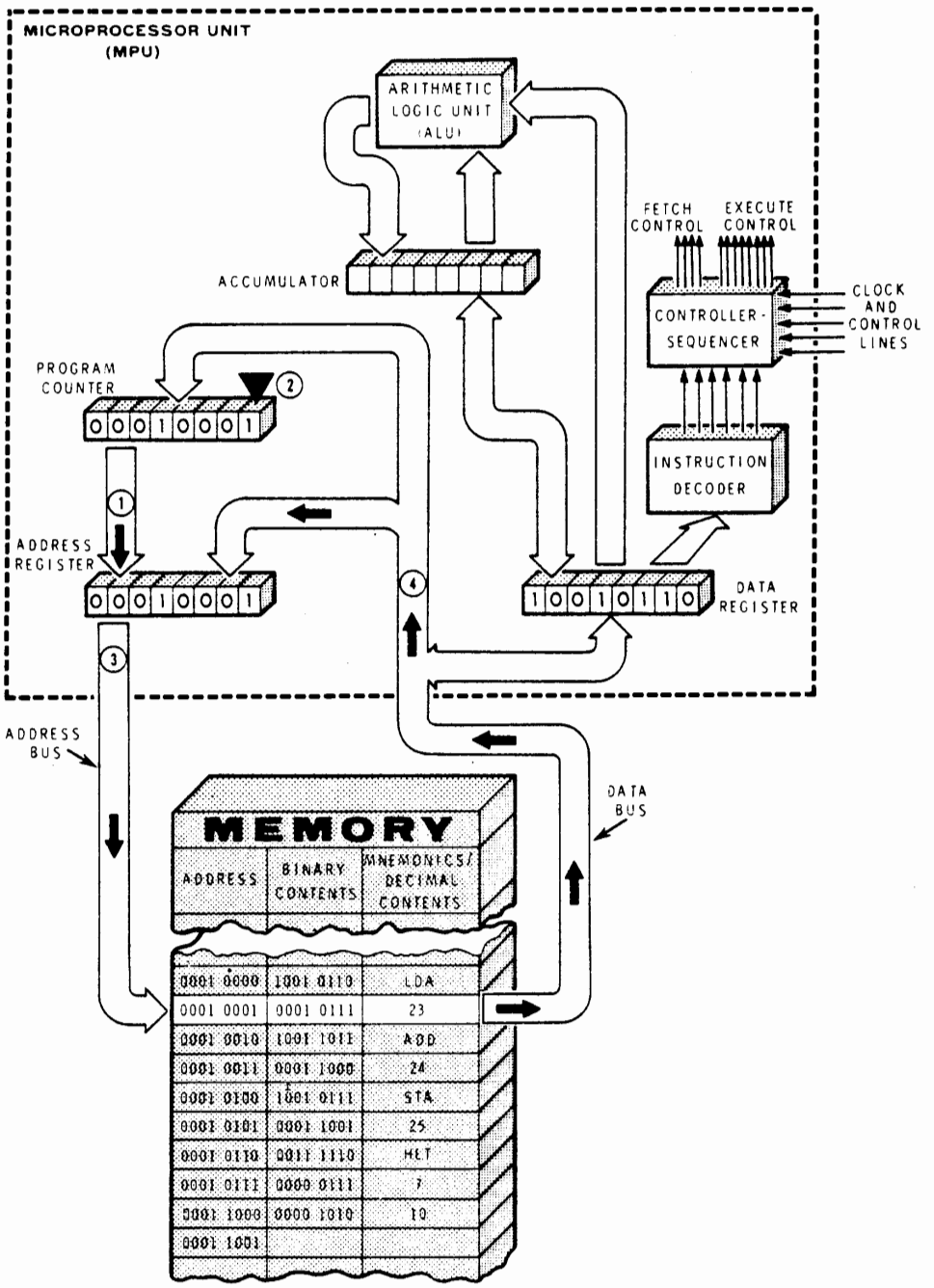


Figure 2-31
Fetching the Address of the First Operand

During the second half of the execute phase, the operand is loaded into the accumulator as shown in Figure 2-32. The procedure is:

1. The address of the operand which is in the address register is placed on the address bus.

2. The operand is read out of memory location 23_{10} and is transferred via the data bus to the data register.

3. The operand is transferred from the data register to the accumulator.

This completes the execution of the first instruction. Notice that the first operand (7_{10}) is now in the accumulator.

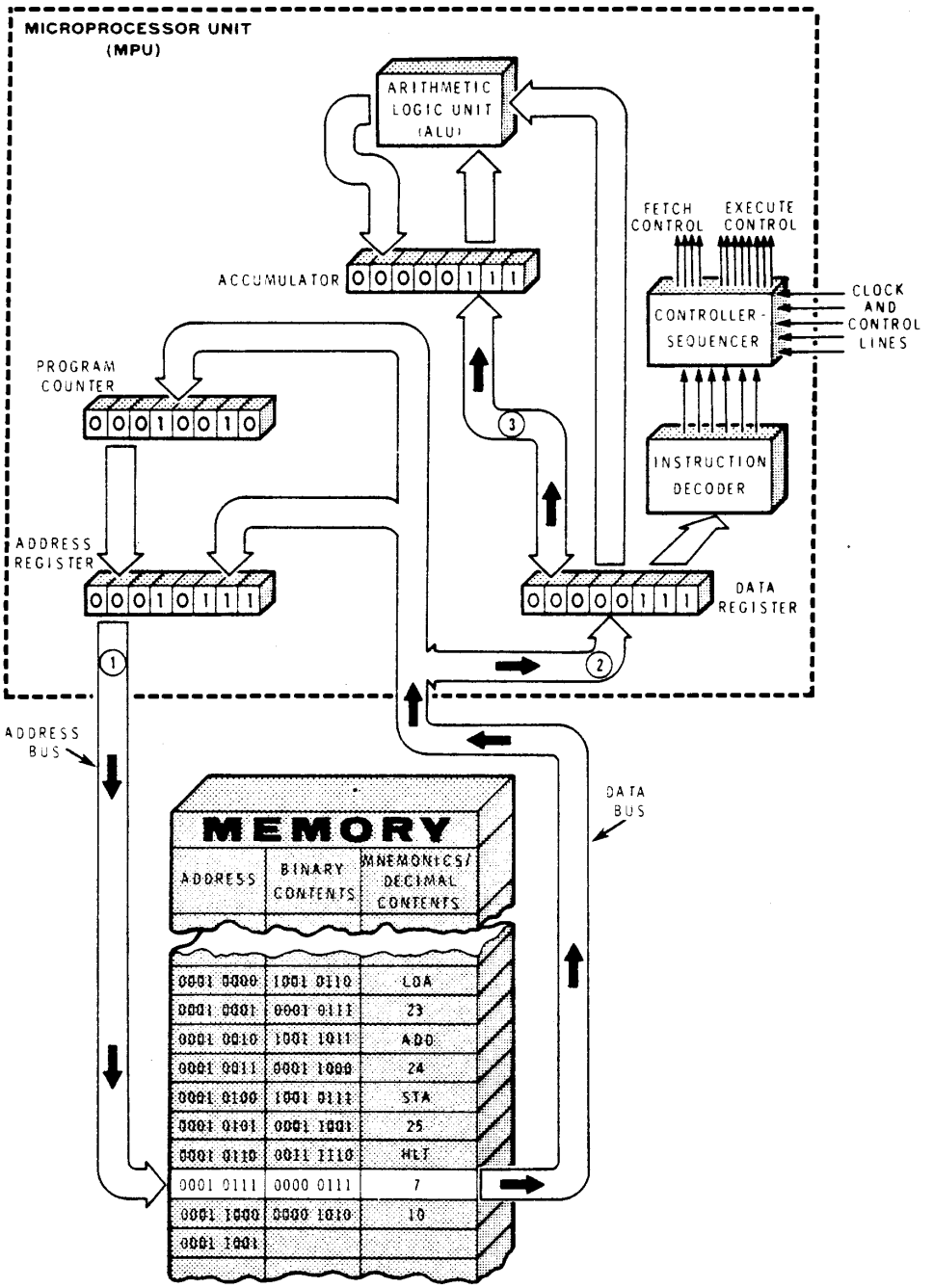


Figure 2-32
Fetching the First Operand.

The fetch phase for the second instruction is similar to that of the first. As shown in Figure 2-33, it causes the opcode of the ADD instruction to be read out of address 18_{10}. The opcode is transferred to the instruction decoder via the data bus and data register. In the process, the program counter is incremented to 19_{10}.

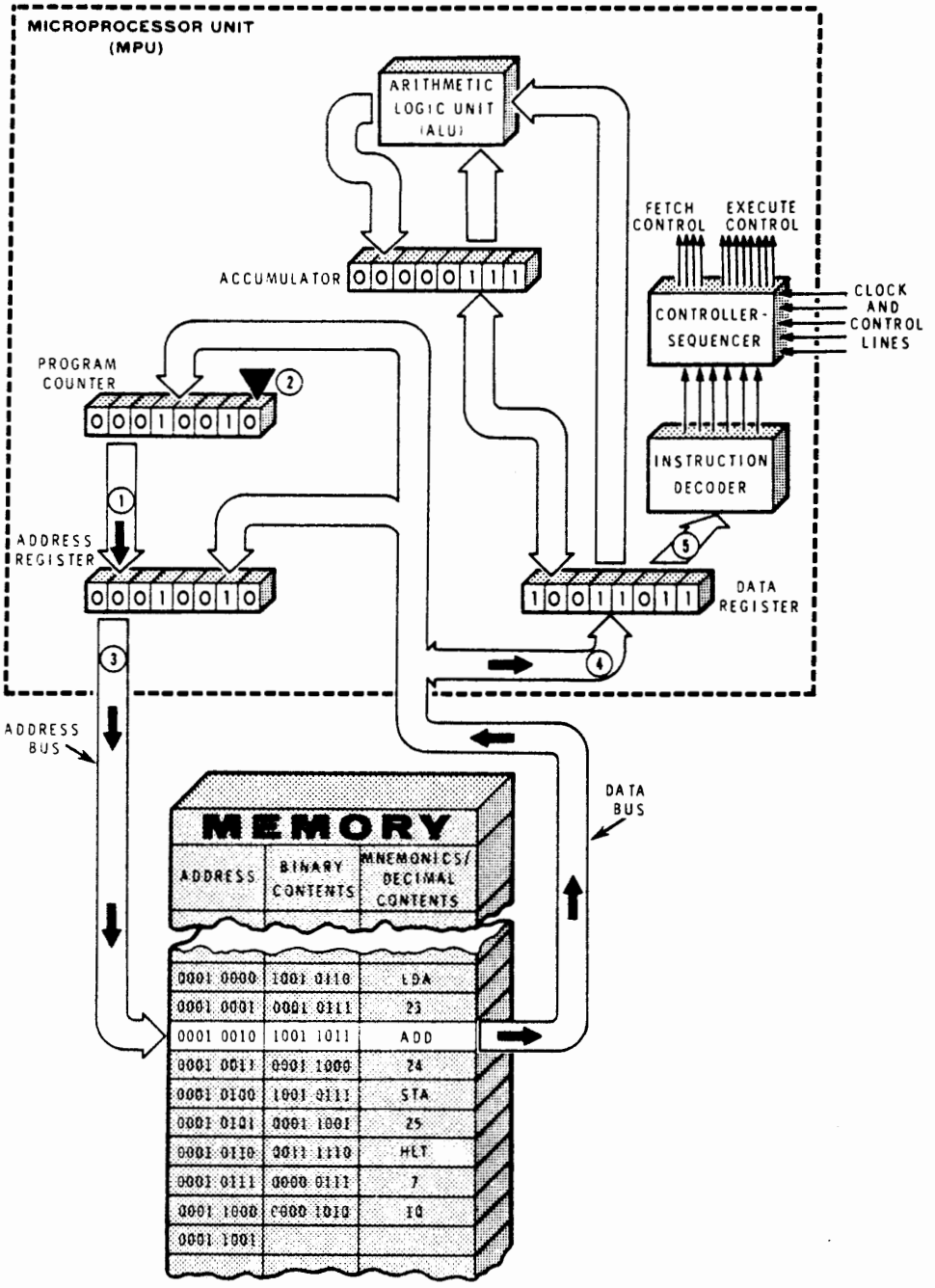


Figure 2-33
Fetching the Opcode of the Second Instruction.

The first half of the execute phase is illustrated in Figure 2-34. Here, the address of the second operand is read out of memory location 19_{10} and is placed in the address register.

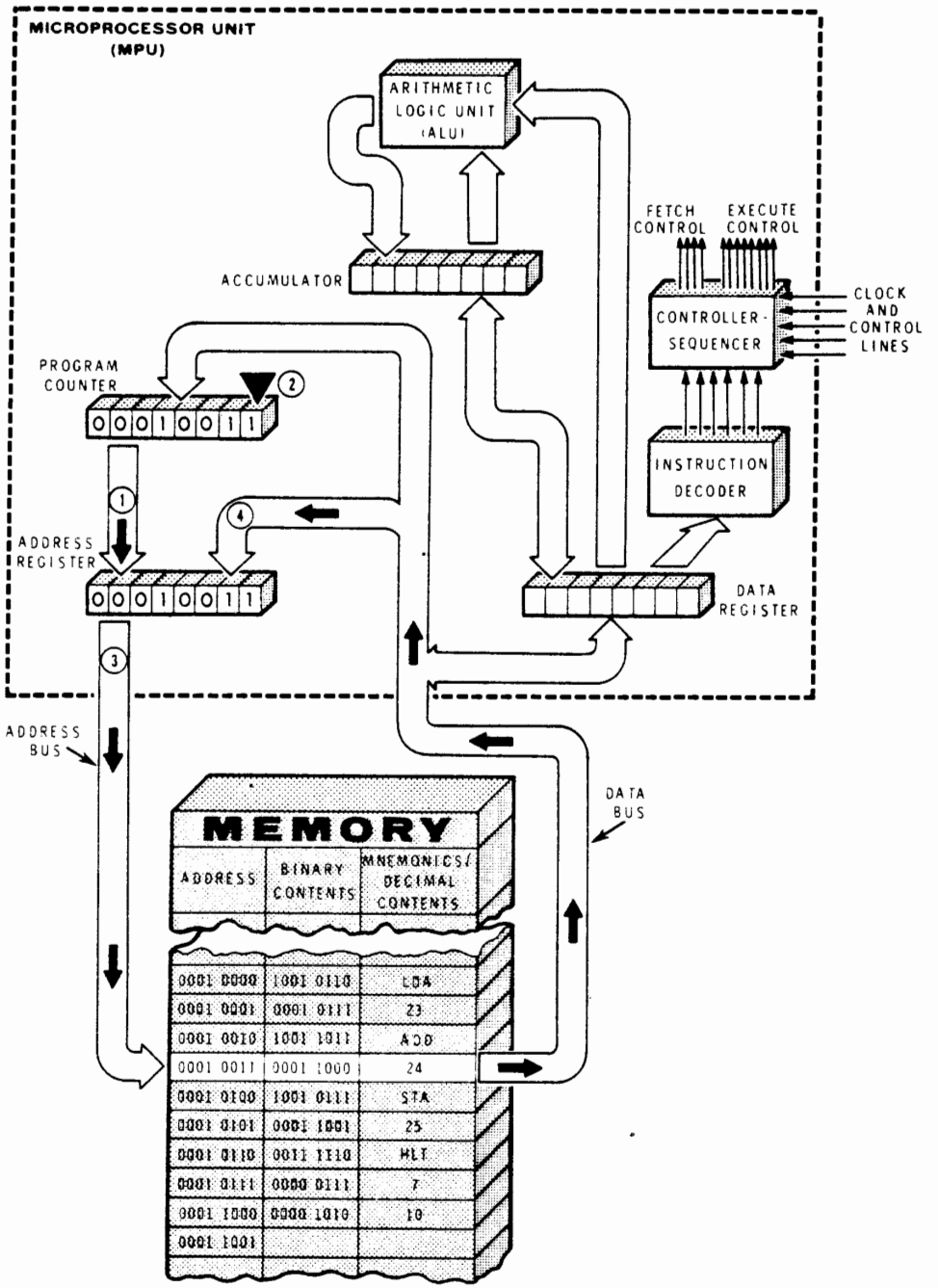


Figure 2-34
Fetching the Address of the Second Operand.

Figure 2-35 illustrates the second cycle of the execute phase. Here the address of the second operand is transferred from the address register to the address bus. The address is 24_{10}. Therefore, the contents of location 24_{10} are placed on the data bus and transferred to the data register. That is, the second operand 10_{10} is loaded into the data register. Then, the operand from the data register is made available at one input to the ALU. Simultaneously, the first operand which has been waiting in the accumulator is made available at the other input to the ALU. The ALU adds the two operands together, producing a result of 17_{10}. This sum is put back in the accumulator, replacing the previous number.

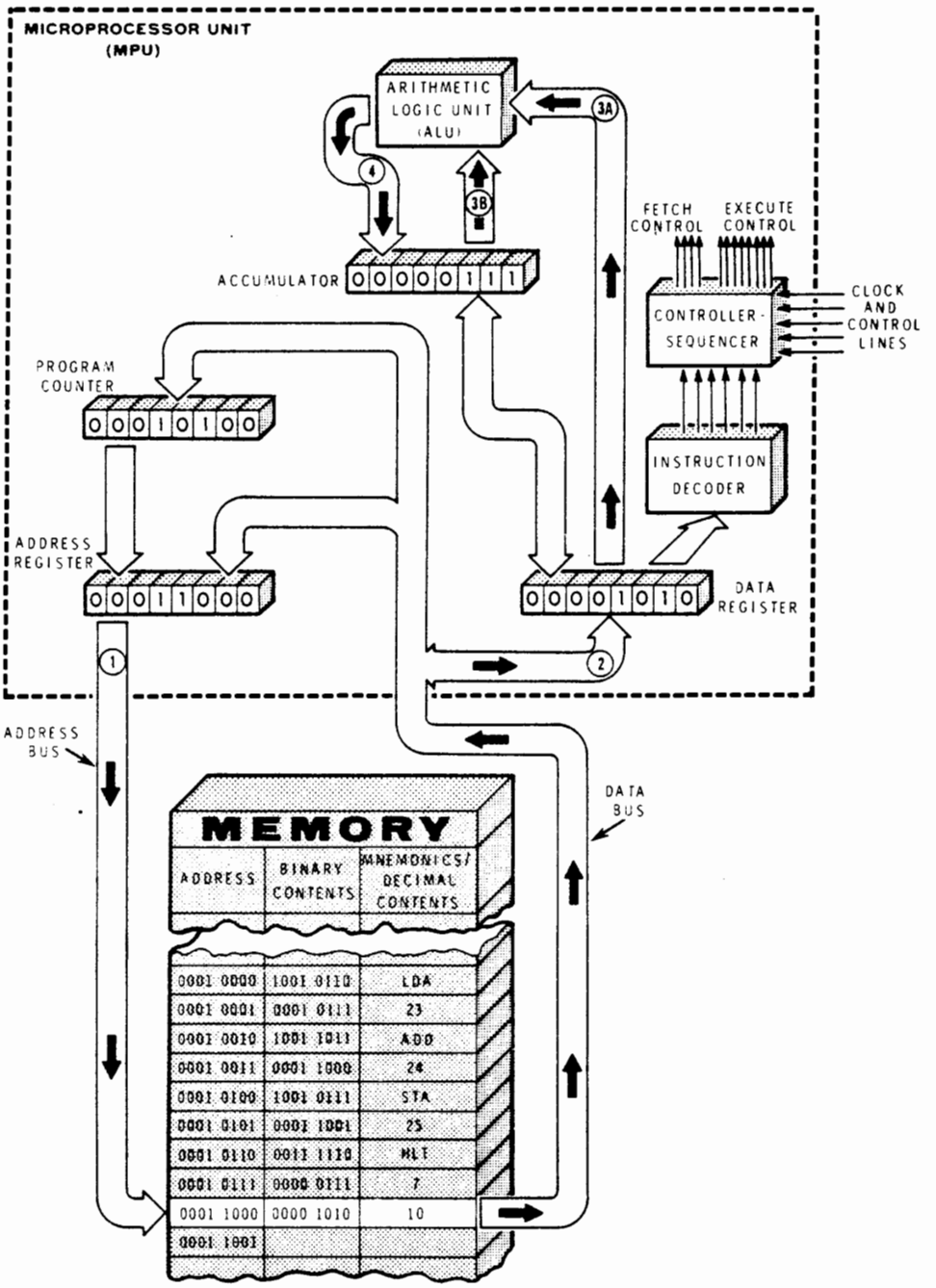


Figure 2-35
Adding the Two Operands.

Now all that remains is to place the sum in memory. This is done by the STA 25_{10} instruction. Since this is the next instruction in sequence, it will be fetched and executed next. The fetch phase is illustrated in Figure 2-36. It ends with the STA opcode being decoded.

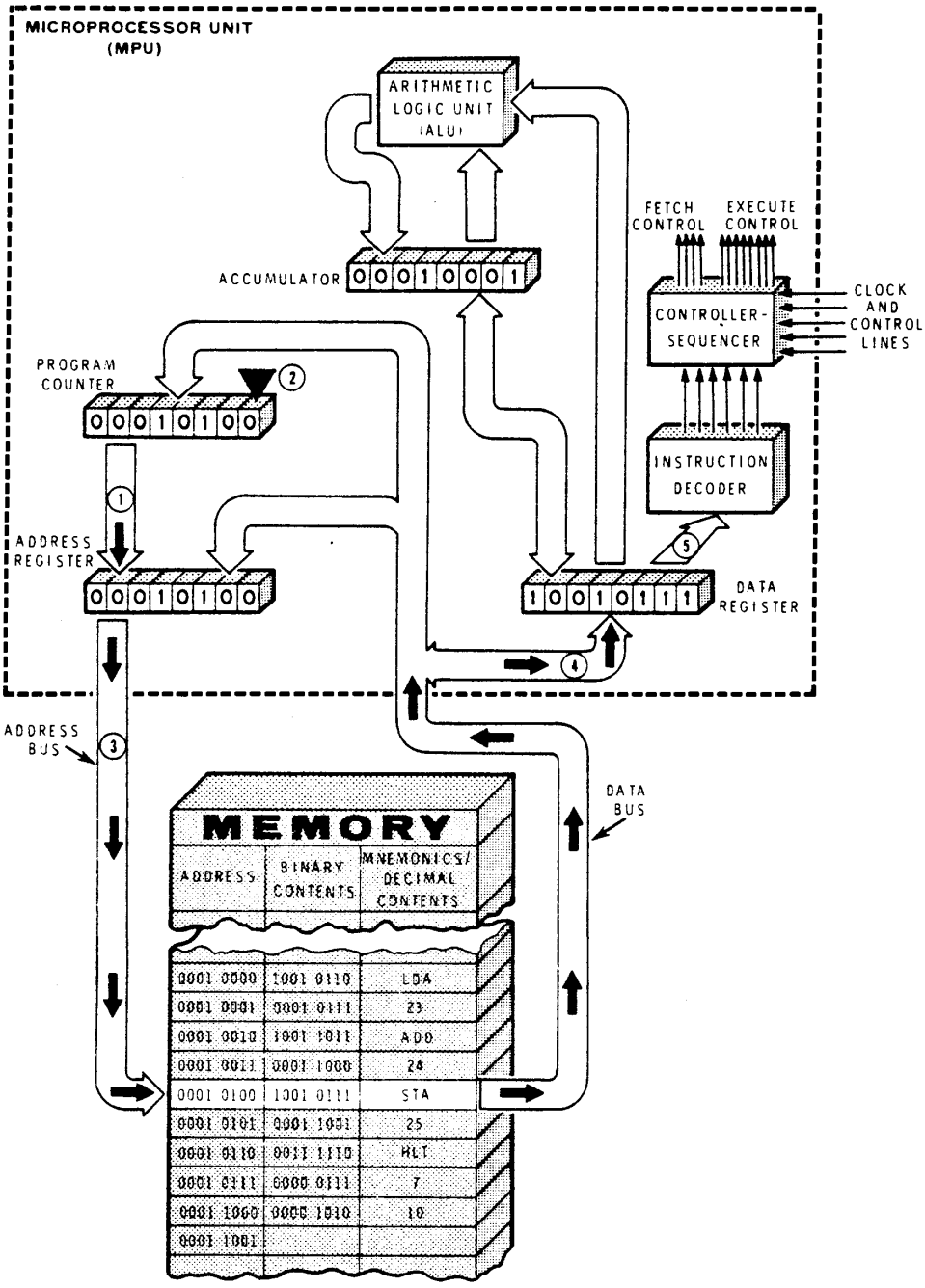


Figure 2-36
Fetching the Third Opcode.

The first half of the execution phase of the STA instruction involves loading the address of the storage location into the address register. Figure 2-37 illustrates that this four-step procedure is identical to that performed for the previous two instructions. It ends with the address 25_{10} in the address register.

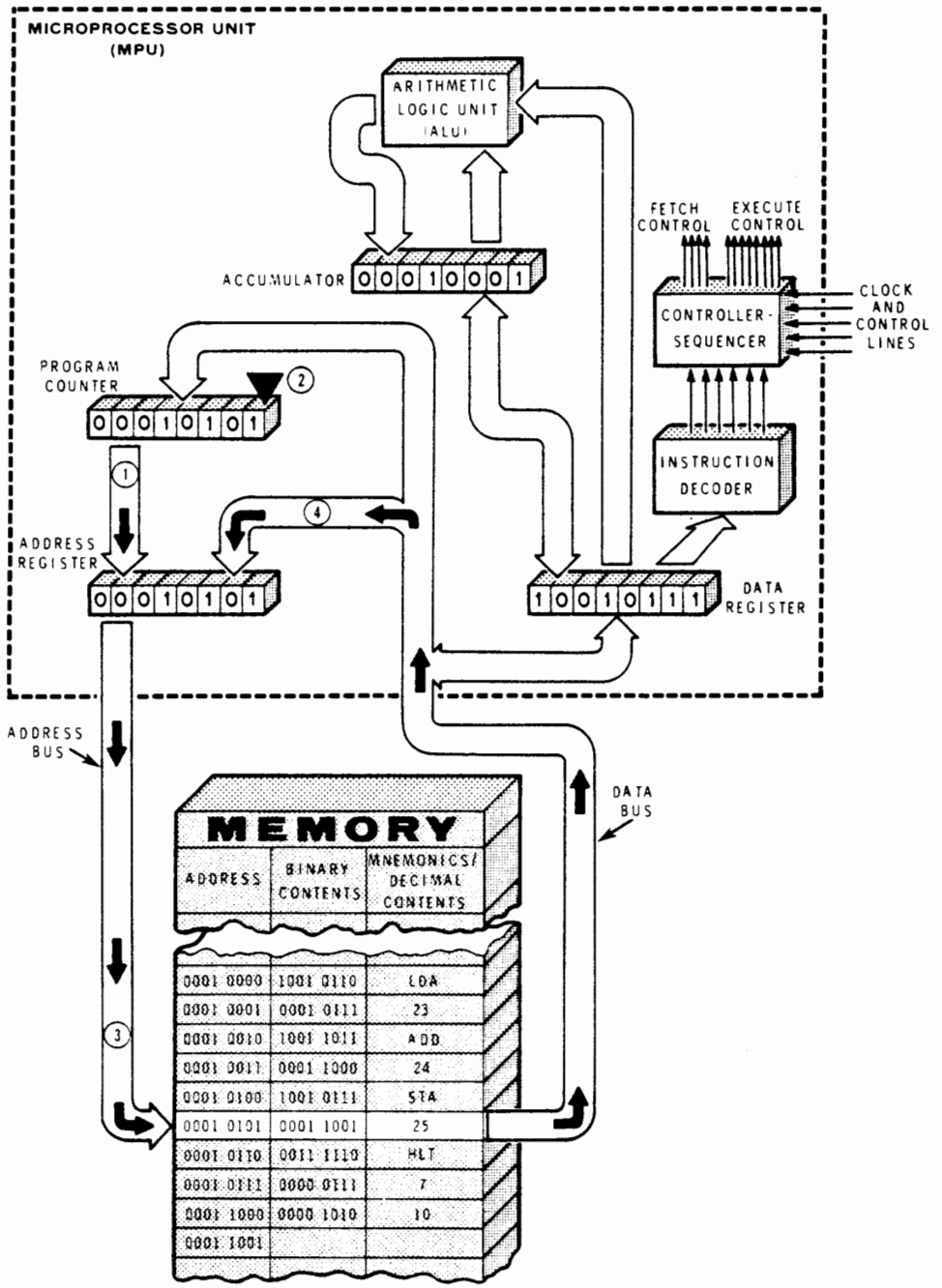


Figure 2-37
Fetching the Third Address.

During the final half of the execute phase, the contents of the accumulator are transferred to the data register and are then stored in the selected memory location. We have not yet discussed this operation in detail. Therefore, the step-by-step procedure is presented below. Refer to Figure 2-38 for the following steps:

1. The contents of the accumulator (17_{10}) are transferred to the data register. At this point, the number 17_{10} exists in both the accumulator and the data register.

2. The address at which this data is to be stored is placed on the address bus.

3. The contents of the data register are placed on the data bus.

4. The number on the data bus is written into the selected memory location. That is, 17_{10} is written into memory location 25_{10}.

Notice that, after this operation, the number 17_{10} appears at memory location 25_{10}, but it also appears in the accumulator. Thus, the number is merely "copied" into memory. It is also important to note that the previous contents of memory location 25_{10} are lost whenever you write new data into this location. For this reason, you must be certain that you do not write into a location that contains an instruction or some byte of data that you will need later.

The program has now accomplished its goal. It has added 10 to 7 and has stored the sum back in memory. The last step in the program is the HLT instruction. The MPU fetches and executes this instruction next. The fetch and execute sequence for this instruction were discussed earlier and need not be repeated here.

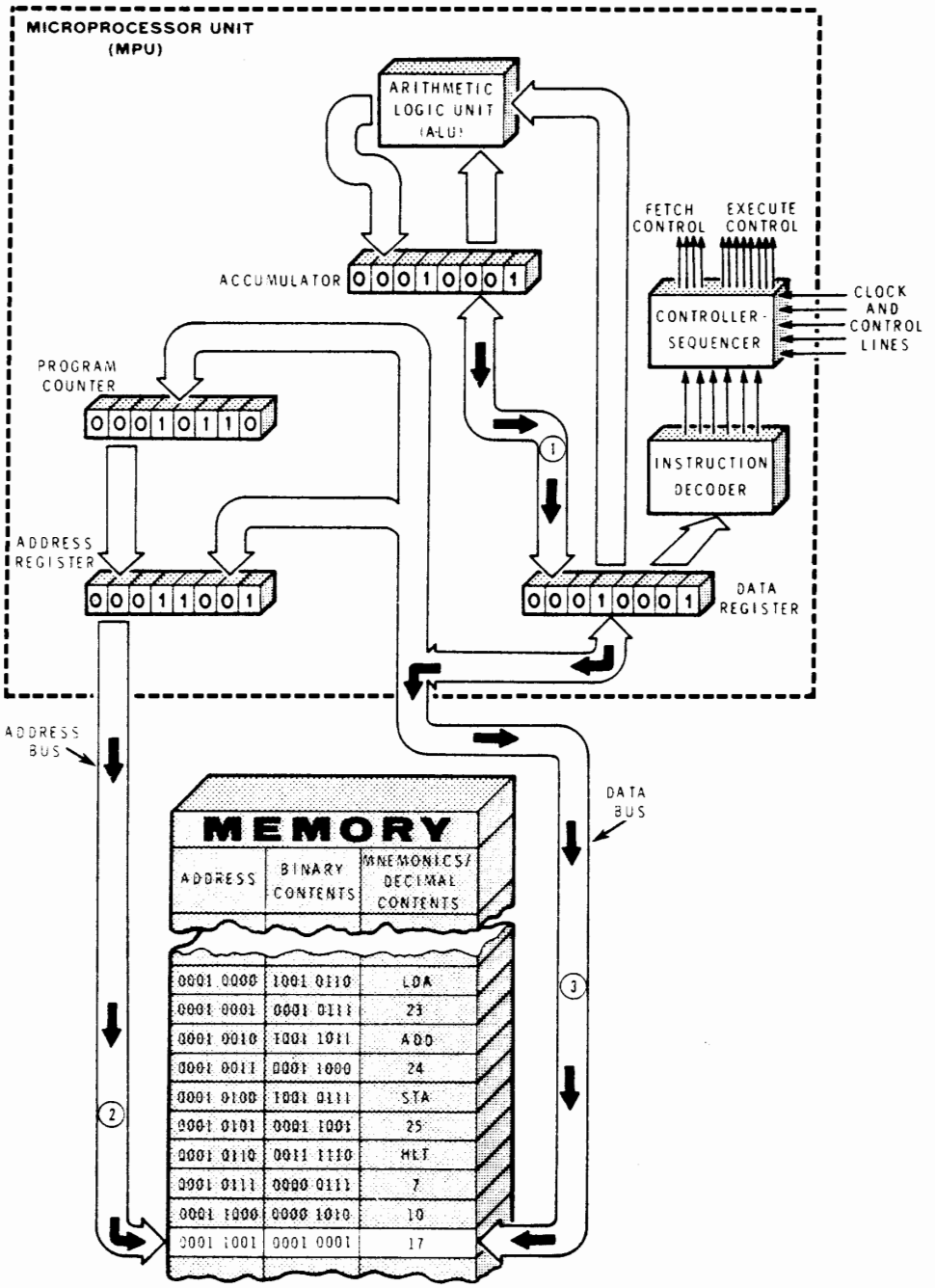


Figure 2-38
Storing the Sum.

Combining Addressing Modes

When writing programs, you can use the addressing mode that best suits your application. For example, the program that was just discussed can be shortened by using the immediate addressing mode with the first two instructions. Figure 2-39 compares two programs that do the same job.

Using direct addressing only, the program required ten bytes of memory. It's execution requires eleven MPU cycles. If you use immediate addressing for the first two instructions, the program requires eight bytes of memory. Furthermore, it can be executed in nine MPU cycles. Everything else being equal, the second approach would probably be preferred.

A. USING DIRECT ADDRESSING

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
00	96		Load accumulator direct with operand 1
01	07		which is stored at this address.
02	9B		Add to accumulator direct with operand 2
03	08		which is stored at this address.
04	97		Store the sum
05	09		at this address.
06	3E		Stop
07	21		Operand 1
08	17		Operand 2
09	—		Reserved for sum.

B. COMBINING ADDRESSING MODES

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS	COMMENTS
00	86		Load accumulator immediately with
01	21		Operand 1.
02	8B		Add to accumulator immediately with
03	17		Operand 2.
04	97		Store the sum
05	07		at this address.
06	3E		Stop
07	—		Reserved for sum.

Figure 2-39
By Combining the Addressing Modes, We Can Save Memory Space And Computer Time.

Self-Test Review

41. What addressing mode is used by single byte instructions?

42. In the immediate addressing mode, what is the second byte of the instruction?

43. In the direct addressing mode, what is the second byte of the instruction?

44. In all addressing modes, what is the first byte of the instruction?

45. Define MPU cycle.

46. Which of the three addressing modes discussed so far requires the longest execution time?

47. Refer to Figure 2-39A. What number is loaded into the accumulator by the first instruction?

48. What number is added to the accumulator by the second instruction?

49. When the computer halts, what number will be in memory location 09?

50. Refer to Figure 2-39B. When the computer halts, what number will be in memory location 07?

Answers

41. Inherent or implied.

42. The operand.

43. The address of the operand.

44. The opcode.

45. An MPU cycle is the time required to fetch a byte from memory.

46. The direct addressing mode.

47. 21_{16} or 33_{10}.

48. 17_{16} or 23_{10}.

49. 38_{16} or 56_{10}.

50. 38_{16} or 56_{10}.

EXPERIMENT 3

Perform Experiment 3 in Unit 9 of this course. After you finish the experiment, return to this unit and complete the final examination.

UNIT EXAMINATION

1. In microprocessor terminology, the number or piece of data that is operated upon is called the:

   A. Operand.
   B. Opcode.
   C. Address.
   D. Instruction.

2. The part of the instruction that tells the microprocessor what operation to perform is called the:

   A. Operand.
   B. Opcode.
   C. Address.
   D. Mnemonic.

3. The portion of the microcomputer in which instructions and data are stored is called the:

   A. ALU.
   B. MPU.
   C. RAM.
   D. Data bus.

4. An 8-bit byte in memory can represent an:

   A. Opcode.
   B. Operand.
   C. Address.
   D. All of the above.

5. During the fetch phase:

   A. The opcode is fetched from memory and is decoded.
   B. The address of the operand is fetched from memory and is decoded.
   C. The operand is fetched from memory and is operated upon.
   D. The program count is fetched from memory.

6. In what register is the result of an arithmetic operation normally placed?

   A. The data register.
   B. The address register.
   C. The arithmetic logic unit (ALU).
   D. The accumulator.

7. During the fetch and execute phases of the "load accumulator direct" instruction, the information on the data bus will be:

   A. The operand address followed by the operand.
   B. The program count, followed by the opcode, followed by the operand address, followed by the operand.
   C. The opcode, followed by the operand address, followed by the operand.
   D. The opcode, followed by the operand.

8. In the immediate addressing mode, the second byte of the instruction is the:

   A. Opcode of the instruction.
   B. Number that is to be operated upon.
   C. Address of the operand.
   D. Address of the opcode.

9. In the direct addressing mode, the second byte of the instruction is the:

   A. Opcode of the instruction.
   B. Number that is to be operated upon.
   C. Address of the operand.
   D. Address of the opcode.

10. Which of the following is normally a one-byte instruction?

    A. Halt.
    B. Add immediate.
    C. Load accumulator direct.
    D. Store accumulator direct.

11. At the start of the fetch phase, the program counter contains:

A. The address of the operand to be fetched.
B. The address of the opcode to be fetched.
C. The opcode of the instruction.
D. The operand.

12. Which register holds the opcode while it is being decoded?

A. The address register.
B. The accumulator.
C. The data register.
D. The program counter.

13. The program shown in Figure 2-40:

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS
00	86	LDA
01	00	00_{16}
02	97	STA
03	09	09_{16}
04	97	STA
05	0A	$0A_{16}$
06	97	STA
07	0B	$0B_{16}$
08	3E	HLT
09	—	—
0A	—	—
0B	—	—

Figure 2-40
Program for Question 13.

A. Adds the contents of memory location 09, 0A, and 0B.
B. Stores 00 in locations 09, 0A, 0B.
C. Stores 09 in location 03, 0A in location 05, and 0B in location 07.
D. Stores 0B in the accumulator.

14. The program shown in Figure 2-41:

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS
00	96	LDA
01	09	09_{16}
02	9B	ADD
03	09	09_{16}
04	9B	ADD
05	09	09_{16}
06	9B	ADD
07	09	09_{16}
08	3E	HLT
09	04	04_{16}

Figure 2-41
Program for Question 14.

A. Multiplies 4 times 4 and holds the product in the accumulator.
B. Multiplies 9 times 3 and holds the product in the accumulator.
C. Multiplies 4 times 3 and stores the product in the accumulator.
D. Multiplies 9 times 4 and holds the product in the accumulator.

15. The program shown in Figure 2-42:

A. Swaps the contents of memory location 0D and 0E.
B. Stores AA_{16} in locations 0D, 0E, and 0F.
C. Stores BB_{16} in locations 0D, 0E, and 0F.
D. Adds AA and BB, storing the sum at location 0F.

HEX ADDRESS	HEX CONTENTS	MNEMONICS/ CONTENTS
00	96	LDA
01	0D	$0D_{16}$
02	97	STA
03	0F	$0F_{16}$
04	96	LDA
05	0E	$0E_{16}$
06	97	STA
07	0D	$0D_{16}$
08	96	LDA
09	0F	$0F_{16}$
0A	97	STA
0B	0E	$0E_{16}$
0C	3E	HLT
0D	AA	AA_{16}
0E	BB	BB_{16}
0F	—	—

Figure 2-42
Program for Question 15.