Introduction to Minicomputers

Instruction Sets

1st Printing, June 1976
2nd Printing (Rev), October 1977
3rd Printing, August 1979

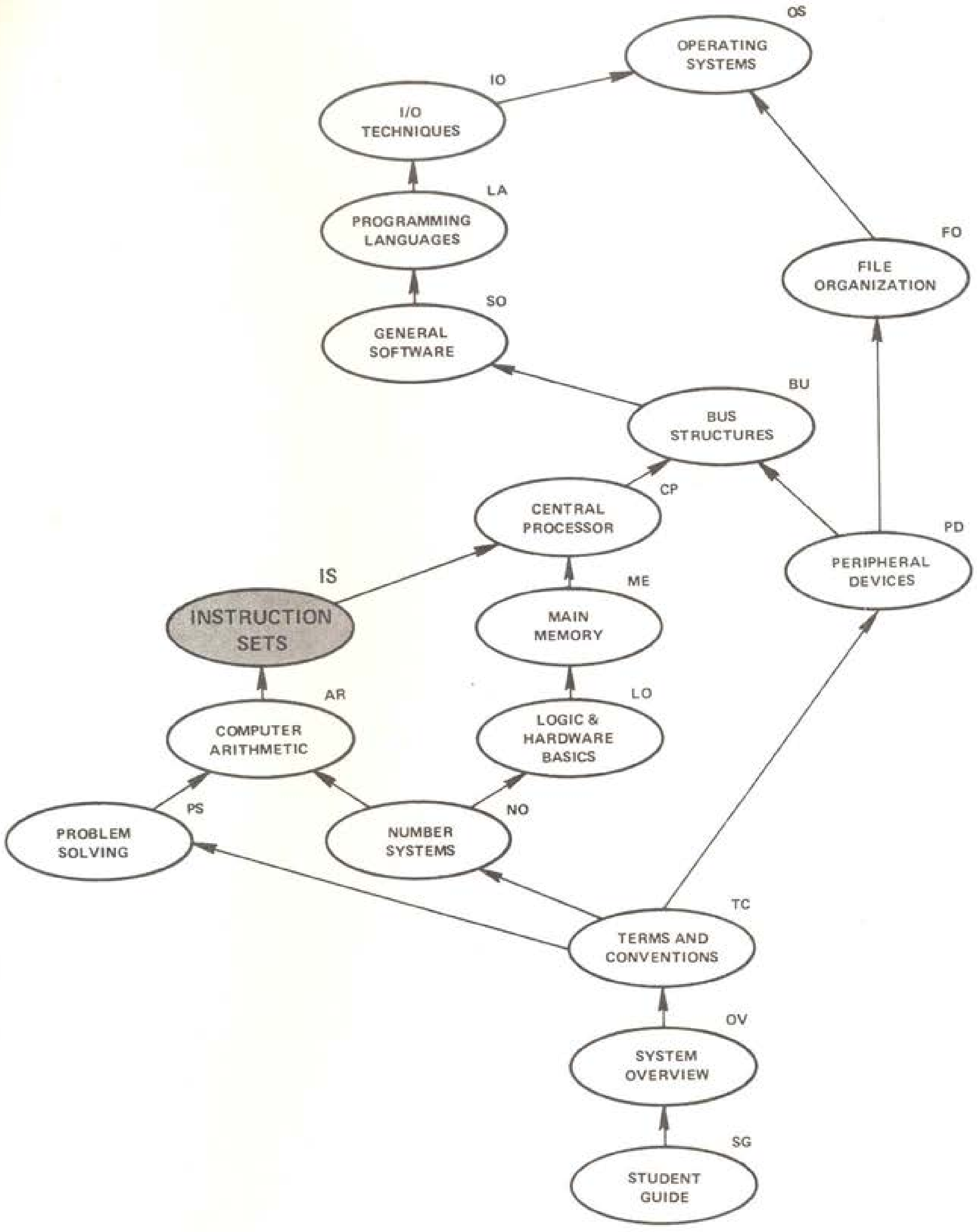
INTRODUCTION TO MINICOMPUTERS

Instruction Sets

Student Workbook

Audio-Visual Course by Digital Equipment Corporation

COURSE MAP


OS
OPERATING SYSTEMS
IO
I/O TECHNIQUES
LA
PROGRAMMING LANGUAGES
FO
FILE ORGANIZATION
SO
GENERAL SOFTWARE
BU
BUS STRUCTURES
CP
CENTRAL PROCESSOR
PD
PERIPHERAL DEVICES
IS
INSTRUCTION SETS
ME
MAIN MEMORY
AR
COMPUTER ARITHMETIC
LO
LOGIC & HARDWARE BASICS
PS
PROBLEM SOLVING
NO
NUMBER SYSTEMS
TC
TERMS AND CONVENTIONS
OV
SYSTEM OVERVIEW
SG
STUDENT GUIDE

CONTENTS

Introduction 1

Instruction Formats 3
- Objectives and Sample Test Items 3
- Operation Field and Op Codes 5
- Operand Field and Operands 6
- Instruction Mnemonics 6
- Variations in the Basic Instruction Format 7
- Exercises and Solutions 11

Addressing Data 15
- Objectives and Sample Test Items 15
- Direct Addressing 17
- Indirect Addressing 22
- Exercises and Solutions 25

Typical Instruction Set 31
- Objectives and Sample Test Items 31
- Program Examples 36
- Negative Numbers and Subtraction 44
- Program Loops, Tallys, and Branches 46
- Program Control Instructions 48
- Additional Program Examples 51
- Summary 56
- Exercises and Solutions 59

Instruction Sets

Introduction

In this module, you will have an opportunity to apply many of the basic concepts that you learned in the earlier units. In fact, when you complete this study unit, you will be able to write simple computer programs using a set of basic instructions.

As we have seen, a digital computer is completely dependent on the instructions that we supply it with. Any problem that the computer is required to solve must be specified correctly in every detail by the programmer.

Computer programmers frequently use *flowcharts* to help them plan the various steps, or operations, that the computer must perform. The flowcharts must then be converted into a sequence of *instructions* to which the computer can respond. A complete sequence of instructions for solving a particular problem is called a *program.*

After the program is written, the instructions and accompanying data are input to the computer and are stored in main memory. The central processor then retrieves and executes the instructions one-at-a-time in logical order to arrive at a solution to a given problem.

Each computer is designed to recognize and perform a specific group of instructions called an *instruction set.* Although instruction sets may vary significantly from one computer manufacturer and model to the next, there are certain fundamental instructions that are used in most general purpose digital computers.

In this module we will discuss the format and use of instruction sets. Our discussion is divided into three lessons. The first lesson describes the basic components of an instruction word and discusses variations in the basic instruction format. The second lesson explains how the instructions address data stored in main memory. Our third lesson defines a typical instruction set and develops some simple programs using these instructions.

Instruction Formats

OBJECTIVES

1. Given six field names and two field purposes, be able to select the two fields that constitute an instruction word, and match each of the word fields with its corresponding purpose.
2. Given the terms "operand," "op code," "instruction mnemonic," and "accumulator" and four definitions, be able to match each term with its definition.
3. Given four instruction formats and descriptions, be able to match each format with its description.

SAMPLE TEST ITEMS

1. In Part A below, circle the *two* letters that identify the names of instruction word fields.

   In Part B, write the letter of the field being defined in the space provided.

   Part A

   The two fields of an instruction word are:

   a. Instruction field
   b. Operand field
   c. Program field
   d. Word-size field
   e. Operation field
   f. Mnemonic field

   Part B

   a. The ________ tells the CPU where to find the data that are to be processed.

   b. The ________ holds a binary code that tells the CPU exactly what to perform next.

SAMPLE TEST ITEMS

2. Match each of the terms below with its definition.

Term	Definition
Operand	____
Op Code	____
Instruction Mnemonic	____
Accumulator	____

Definitions

a. A 3- or 4-letter abbreviation that programmers use in place of the binary operation code.

b. An item of data to be acted upon by an instruction.

c. A special storage area contained in the CPU.

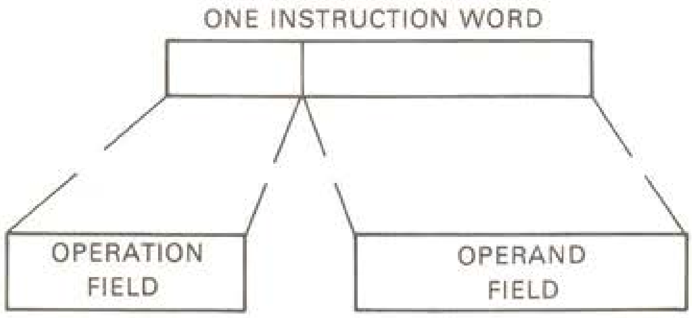
d. A predefined binary code that tells the CPU what operation it is to perform.

3. Match the following instruction formats with the appropriate description:

a. Three operand fields	( )	If the instruction calls for an addition, the sum is placed in a memory location formerly occupied by one of the operands.
b. Two operand fields		
c. Single operand field	( )	Memory is *not* referenced; the instruction operates directly on the contents of the AC.
d. No operand field		
	( )	Not widely used in minicomputers because it requires a large word size.
	( )	If the instruction calls for an addition, the contents of a memory location are added to the contents of the AC.

Mark your place in this workbook and view Lesson 1 in the A/V program, "Instruction Sets."

In the audio-visual portion of this lesson we defined the basic format of an instruction word. We noted that most computer instructions consist of two major elements, or fields – an *operation* field and an *operand* field.



Operation Field and Op Codes

The operation field holds a predefined binary code called an operation code or, simply, an *op code.* Each time the central processor (CPU) retrieves an instruction from main memory, the instruction op code tells the CPU exactly what operation to perform next. For example, an op code of 001 may specify an add operation, while an op code of 011 may designate a store operation.

At the time a computer is designed, a different op code is assigned to every instruction in the instruction set. Special circuits in the central processor are then used to interpret (decode) each op code so that the central processor can perform (execute) the operation that the op code represents.

The number of bits that make up the op code generally depends on the size of the computer's instruction set: the larger the instruction set, the more bits that are required to represent each instruction uniquely. For example, a 3-bit op code is sufficient if there are only 8 instructions in the set ($2^3 = 8$). However, if there are 32 different instructions, then a 5-bit op code is necessary ($2^5 = 32$). Stated in more general terms, *an op code of "n" bits can represent a maximum of 2^n different instructions.*

Operand Field and Operands

In addition to the op code, most instructions contain one or more operand fields (see Figure 1). Before the central processor can perform an operation, such as adding two numeric values, it must be told where the values are stored. This is the purpose of the operand field; it tells the CPU where to find the data (or operands) that are to be processed. Thus, an *operand* is simply an item of data to be acted upon by an instruction, and the *operand field* is a part of the instruction word that usually holds the memory address of the operand.

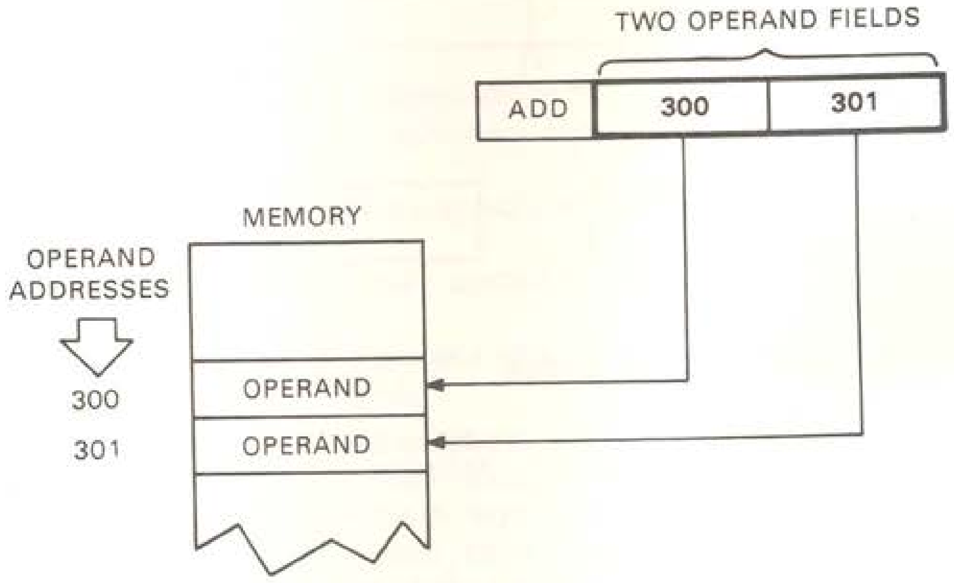


Figure 1 Instruction Referencing Operands Stored in Memory

Instruction Mnemonics

Although binary op codes are essential because the computer reads only 1s and 0s, they are very cumbersome for us to work with. Therefore, to simplify the task of writing or reading programs, each instruction is often represented by a simple 3- or 4-letter abbreviation. This abbreviation is called an *instruction mnemonic* (see Table 1). Since mnemonics suggest the meaning of each instruction, they are much easier for us to remember and to work with. For example, to specify an add instruction, we can use the mnemonic ADD in place of the op code 001. Or, to specify a store instruction, we can substitute the mnemonic STR for the op code 011.

Remember that the computer does not understand any language except binary numbers. Consequently, instruction mnemonics must still be converted into their equivalent binary op codes before the computer can use them.

Table 1 Typical Instruction Mnemonics

Typical Instruction	Instruction Mnemonic	Binary Op Code
Add	ADD	001
Store	STR	011
Jump	JMP	101

Variations in the Basic Instruction Format

As we noted in the audio-visual program, some instruction formats may use as many as *three* separate operand fields. At the other extreme, some instructions may *not* contain any operand field. The number of operand fields depends on factors such as:

- The design of the computer
- The word-size
- The type of operation called for by the instruction

Three Operand Fields

Figure 2 illustrates the use of an instruction format that contains three separate operand fields. Two of the fields specify the memory addresses of the two operands that are to be added together. The third operand field indicates the address of the memory location where the sum is to be stored. Note that this instruction format is not suitable for most computers – especially minicomputers – because it requires a large word size to accommodate all three operand fields.

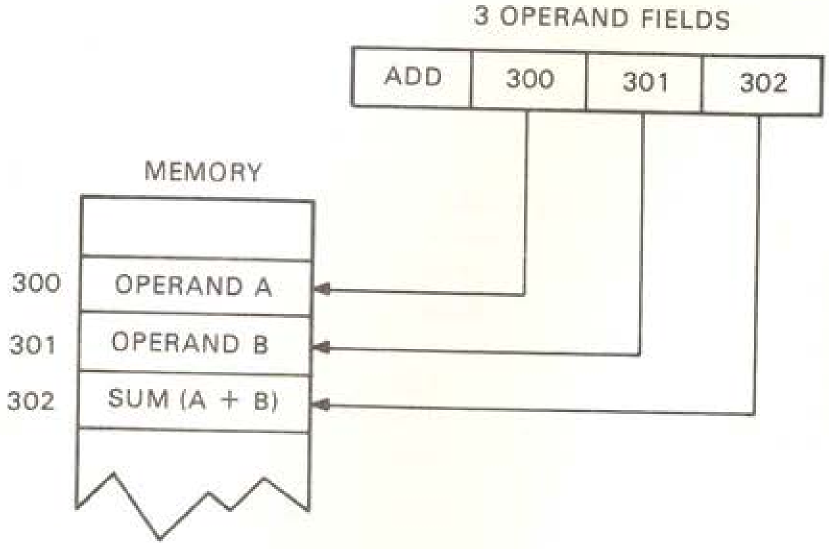


Figure 2 Typical Instruction with Three Operand Fields

Two Operand Fields

Instructions with two operand fields are used in some mini-computers (see Figure 3). This format still allows the central processor to address two operands stored in memory. However, after the two operands have been processed, the result must be stored in the same memory location that formerly held one of the operands, thereby destroying that operand.

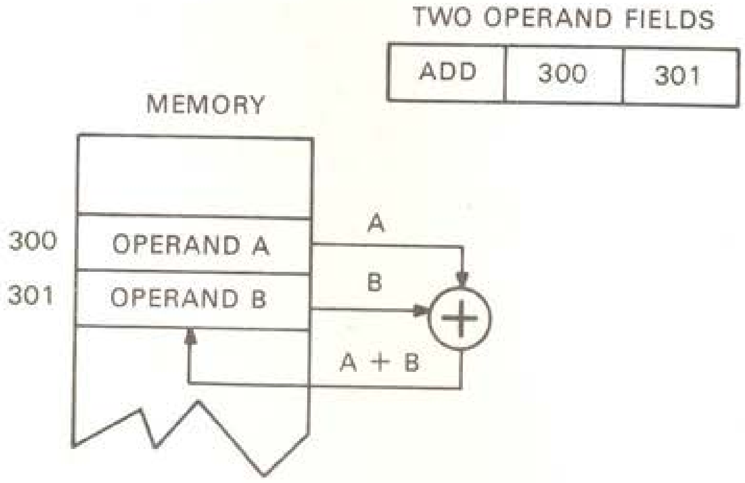


Figure 3 Typical Instruction with Two Operand Fields

Single Operand Field

A single-operand format is widely used in minicomputers because their smaller word sizes (12 to 16 bits) often cannot accommodate more than one operand field. When this format is used, the instruction can reference just one memory location. If a second operand is required (for example, in an addition), that operand is placed in a special storage element called an *accumulator*. Because the accumulator is part of the central processor, it is *not* necessary to address one of the operands (the operand is already in the CPU).

As Figure 4 shows, the CPU responds to the single-operand instruction by adding the contents of memory location 300 to a second operand held in the accumulator. After the addition is performed, the resulting sum is held in the accumulator where it is available for further processing. The operand stored in memory location 300 is not changed.

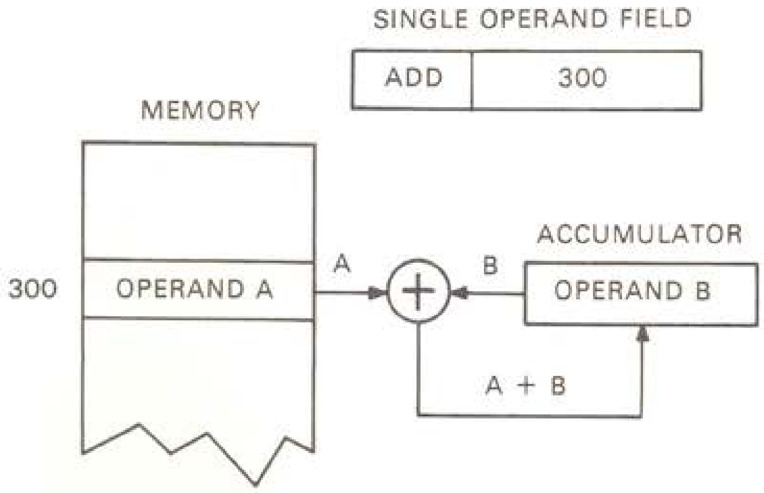


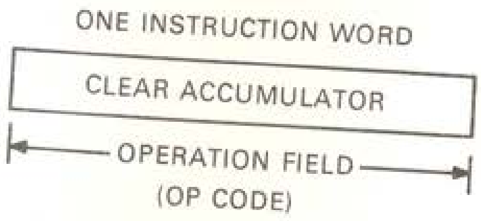
Figure 4 Typical Instruction with One Operand Field

Thus, the accumulator (or AC) serves as a working area for all computations. The central processor uses the accumulator to provide temporary storage for an operand that is to be used in a computation. After the computation is completed, the AC serves as a temporary holding area for the answer. Note that the accumulator is usually designed to hold just one word of data.

No Operand Field

Some types of instructions only operate on the contents of the accumulator. These instructions do not reference operands stored in memory and, therefore, do not contain an operand field.

A "clear accumulator" instruction is an example of an instruction with no operand field. This instruction is used whenever it is necessary to erase (or clear) the contents of the AC to all zeros. Because there is no operand field, the remaining bits in the instruction word are generally used as an extension of the operation field (op code).



EXERCISES

1. A complete sequence of instructions for solving a particular problem is called a __________________ .

2. The computer is designed to recognize and perform a specific group of instructions called a(an) ____________________ .

3. Write in the names of the two elements or fields that are contained in the following instruction word. Briefly explain the purpose of each element.

a.	b.

Purpose

a.

b.

4. Write a brief definition for each of the terms listed below.

   a. Operand

   b. Op Code

   c. Instruction Mnemonic

   d. Accumulator

SOLUTIONS

1. A complete sequence of instructions for solving a particular problem is called a <u>program.</u>

2. The computer is designed to recognize and perform a specific group of instructions called a(an) <u>instruction set.</u>

3. Write in the names of the two elements or fields that are contained in the following instruction word. Briefly explain the purpose of each element.

a. Operation Field	b. Operand Field

Purpose

a. The operation field holds a binary op code that tells the CPU exactly what operation to perform next.

b. The operand field tells the CPU where to find the data (operands) that are to be processed.

4. Write a brief definition for each of the terms listed below.

a. **Operand** – An item of data to be acted upon by an instruction.

b. **Op Code** – A predefined binary code that tells the CPU what operation it is to perform. The op code is held in the operation field of the instruction word.

c. **Instruction Mnemonic** – a 3- or 4-letter abbreviation that programmers use in place of the binary op code. Instruction mnemonics suggest the meaning of each instruction and are easier to work with.

d. **Accumulator** – A special storage area contained in the CPU. The AC serves as a working area for all computations and is usually designed to hold just one word of data.

EXERCISES

5. How many different instructions can be represented by a 4-bit op code?

6. If an instruction set consists of 64 different instructions, then how many bits are required in the op code to accommodate all 64 instructions?

7. Match the following instruction formats with the appropriate description:

a. Three operand fields	( )	If the instruction calls for an addition, the sum is placed in a *memory* location formerly occupied by one of the operands.
b. Two operand fields	( )	Memory is *not* referenced; the instruction operates directly on the contents of the AC.
c. Single operand field	( )	Not widely used in computers because it requires a large word size.
d. No operand field	( )	If the instruction calls for an addition, the contents of a memory location are added to the contents of the AC.

5. How many different instructions can be represented by a 4-bit op code?

   16 instructions

6. If an instruction set consists of 64 different instructions, then how many bits are required in the op code to accommodate all 64 instructions?

   6 bits

7. Match the following instruction formats with the appropriate description:

a. Three operand fields	(b) If the instruction calls for an addition, the sum is placed in a *memory* location formerly occupied by one of the operands.
b. Two operand fields	(d) Memory is *not* referenced; the instruction operates directly on the contents of the AC.
c. Single operand field	(a) Not widely used in computers because it requires a large word size.
d. No operand field	(c) If the instruction calls for an addition, the contents of a memory location are added to the contents of the AC.

Addressing Data

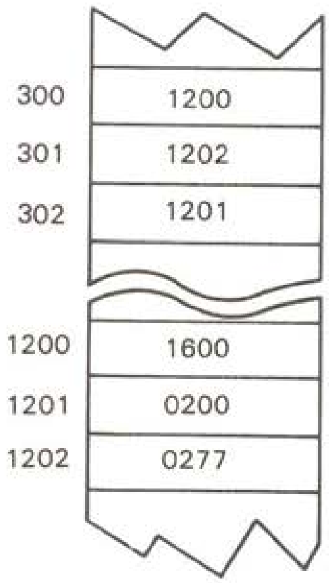
OBJECTIVES

1. Given a diagram specifying the content of various memory locations and a table listing instructions that reference the various memory locations, be able to complete the table by writing the addressing method, the operand address, and the operand for each of the instructions.

2. Given three techniques for overcoming addressing limitations and a list of descriptive phrases, be able to match each phrase with the technique it describes.

SAMPLE TEST ITEMS

1. The diagram at the right specifies the contents of various memory locations. The table below lists several instructions that reference these memory locations. For each instruction, specify the addressing method and the actual *operand address* and the accompanying *operand.*



Instruction	Addressing Method	Operand Address	Operand
ADD 300		300	1200
ADD I 300			
ADD I 301			
ADD 302			

SAMPLE TEST ITEMS

2. The computer word size places a limit on the maximum number of memory locations that can be directly addressed. Three techniques that may be used to overcome this addressing limitation are: multiple-word instructions (MW), special registers (SR), and memory pages (MP).

   Match each of the descriptions below with the technique it describes by writing the correct abbreviation in the space provided.

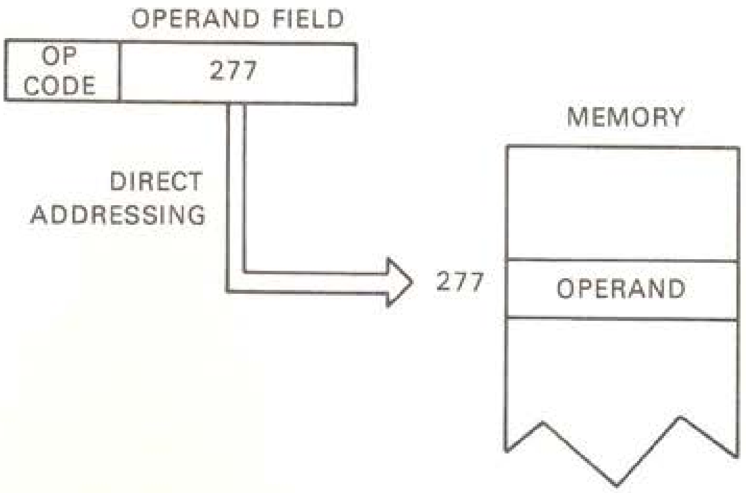
Description	Technique
Part of the CPU.	_____
Sometimes used for purposes other than addressing main memory.	_____
Segments of main memory.	_____
Used in place of single-word instructions.	_____
Size of each is chosen so that CPU can address any location by using the available address bit in the instruction.	_____

Mark your place in this workbook and view Lesson 2 in the A/V program, "Instruction Sets."

In the audio-visual portion of this lesson we described different methods of addressing operands stored in main memory. Let's review and amplify the major points associated with each of these addressing methods.

Direct Addressing

The following example illustrates the use of direct addressing.



This method is called *direct* addressing because the operand field of the instruction "points" *directly* to the memory location (277) that contains the operand.

The number of memory locations that can be addressed on a direct basis depends on the number of address bits that are used. We can define this relationship using a simple formula:

$$2^n = L.$$

In our formula, "n" represents the number of address bits and "L" represents the maximum number of locations that can be addressed. Thus, if an instruction word contains a 7-bit address, the central processor can reference up to 128 memory locations ($2^7 = 128$).

Most minicomputer systems contain at least 4K of memory (4096 words). To *directly* address all 4096 locations, we need a 12-bit address ($2^{12} = 4096$). However, if our minicomputer uses a 12-bit word, it cannot possibly accommodate all 12 address bits in one instruction word and still have room for the op code. Consequently, the computer word size places a limit on the number of memory locations that can be directly addressed. The following techniques can be used to overcome this addressing limitation:

- Use multiple-word instructions.
- Address memory using registers in the CPU.
- Divide memory into smaller segments called pages.

Multiple-Word Instructions

When the word size limits the number of address bits, one solution is to use two or three words to represent one complete instruction. This solves our addressing problem by providing enough address bits to directly reference any location in main memory. Figure 5 illustrates the use of a multiple-word instruction.

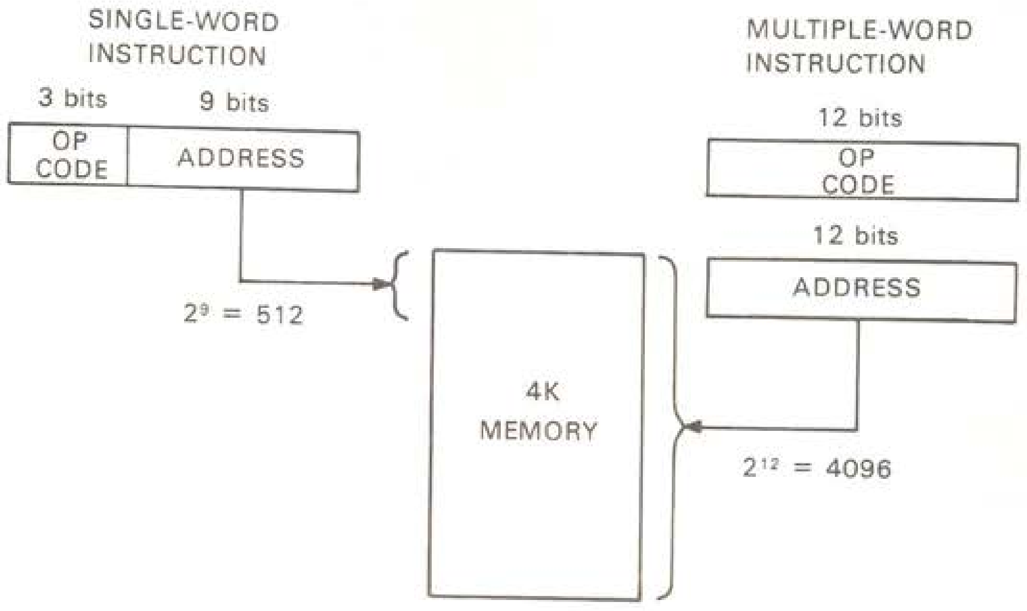


Figure 5 Single-Word Instruction Vs Multiple-Word Instruction

In this example, the single-word instruction can only reference 512 memory locations. On the other hand, the multiple-word instruction can reference all 4096 locations.

A major disadvantage in using multiple-word instructions is that main memory must be referenced two or more times to retrieve one complete instruction. This effectively reduces the operating speed of the CPU. Another obvious disadvantage is that multiple-word instructions use up more memory space (two or three memory locations compared to one location for a single-word instruction).

Using Registers to Address Memory

A *register* is a common storage element that is usually designed to hold just *one word* of information. In some minicomputers the central processor is equipped with one or more registers that are used for addressing operands stored in memory. This addressing technique is shown in Figure 6.

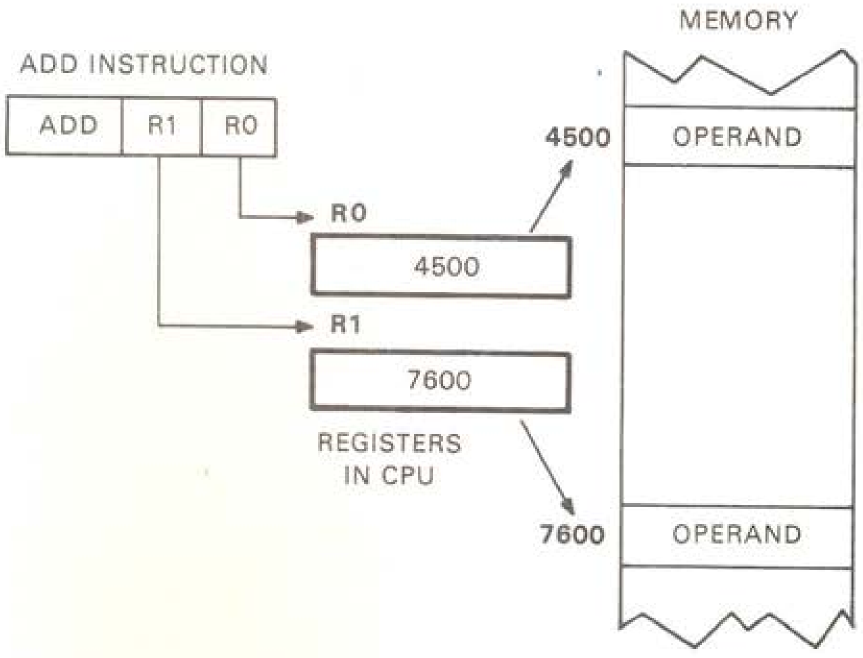


Figure 6 Addressing Memory via Registers in the CPU

In this example, the memory addresses of the two operands are placed in registers R0 and R1. Therefore, when the CPU retrieves and executes the ADD instruction, these two registers effectively "point" to the memory locations containing the operands. Because each register holds one computer word, it can usually accommodate enough address bits to directly address any location in main memory.

Some computers are designed so that these registers can be used for purposes other than to address operands. For example, each register may be used as an accumulator. This allows the central processor to operate directly on the contents of a register. The CPU may be directed to increment or complement a value held in one of the registers, or to add the contents of one register to the contents of a second register. When these registers are used for many different purposes, they are called *general purpose registers* (GPRs).

Memory Pages

Frequently, main memory is divided into smaller, more easily addressable segments called *pages* (see Figure 7). The size of each page is chosen so that the CPU can directly address any location within a given page. For example, if there are only 7 address bits in the operand field of the instruction, then the page size is 128 words ($2^7 = 128$). With a page size of 128 words, 4K of memory could be divided into 32 separate pages or ($4096 \div 128 = 32$).

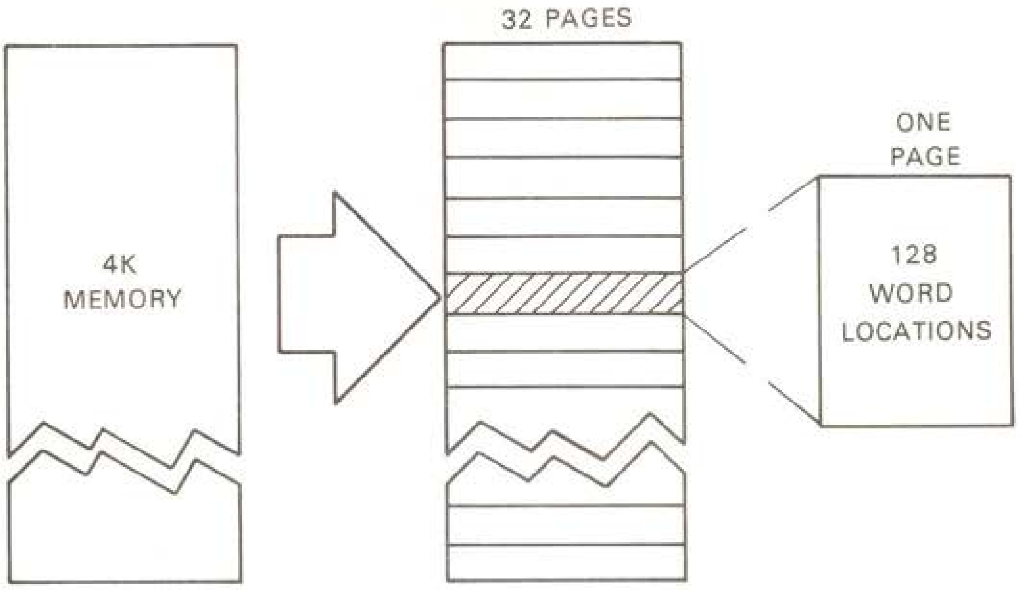


Figure 7 Memory Pages

Whenever memory pages are used, the CPU must have some way to determine which page is being referenced. One approach uses a special register (page register) to keep track of the page currently in use. This page register always contains the starting address of the current page. In the following example (Figure 8), the page register contains the starting address of page 1 (0200). To obtain the actual memory address of the operand, the CPU automatically adds the contents of the page register to the address contained in the operand field of the instruction.

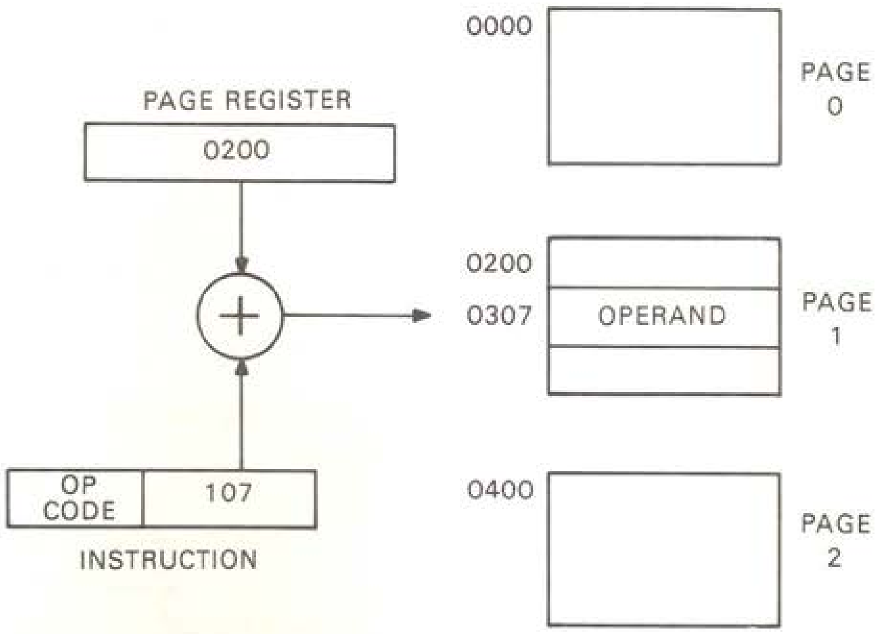
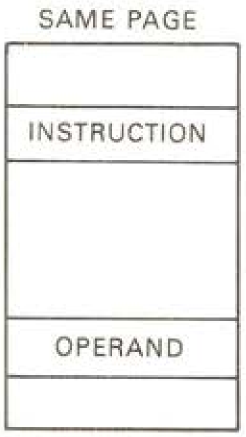


Figure 8 Use of a Page Register



If a page register is not used, another approach is to store the instruction and the operand that it is referencing in the *same* page. Of course, this means that only a small portion of memory can be directly addressed.

Indirect Addressing

Indirect addressing must be used when an instruction references an operand stored in a *different* page. Figure 9 compares direct addressing and indirect addressing. When *direct* addressing is used, the operand field of the instruction points *directly* to the memory location containing the operand. On the other hand, if *indirect* addressing is used, the operand field points to a memory location that contains the *address* of the operand. The CPU must first retrieve this address from memory. Then it uses the address to retrieve the operand. Thus, memory must be referenced at least twice when indirect addressing is used. Remember that indirect addressing is necessary whenever the instruction and the operand are stored in *different pages.*

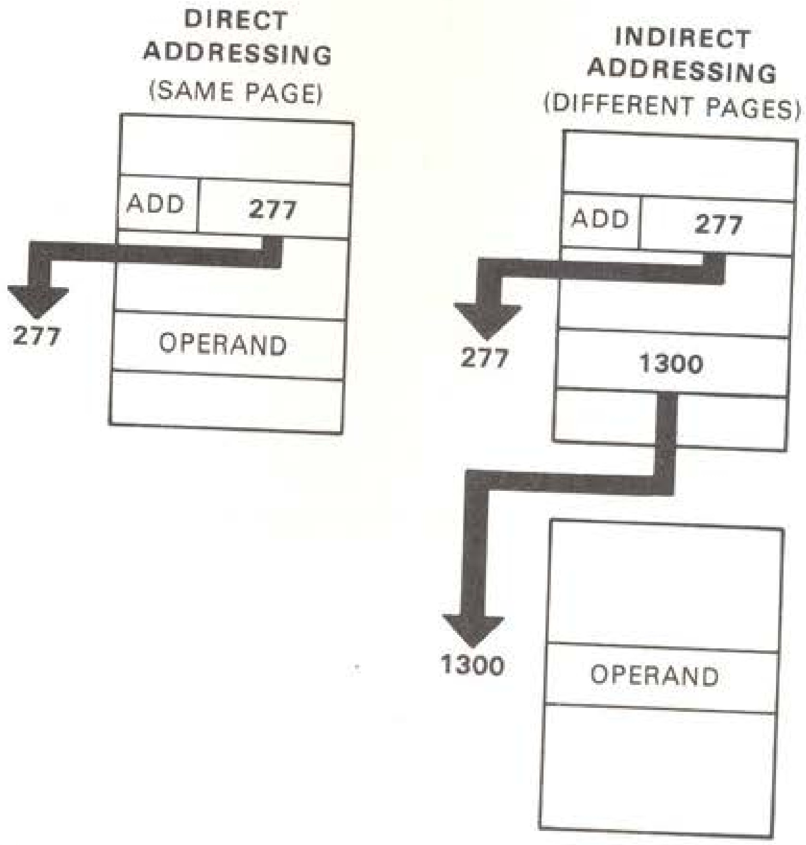


Figure 9 Direct Addressing Vs Indirect Addressing

The CPU must be told when it is to use indirect addressing to retrieve an operand. A special bit in the instruction word is usually reserved for this purpose. If the bit is a 1, it signifies indirect addressing. Conversely, if the bit is a 0, it signifies that direct addressing is to be used.

When we are writing programs, we will use the following symbol to designate indirect addressing:

INDIRECT
ADDRESSING

```
ADD   I   277.
```

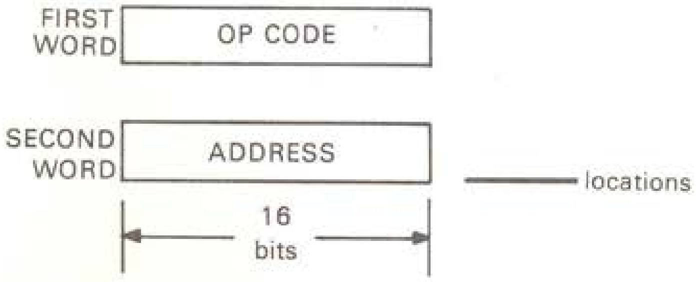
If the "I" is omitted, it signifies that direct addressing is to be used.

EXERCISES

1. How many memory locations can be directly addressed if the operand field of the instruction word holds 7 address bits?

2. If the size of the operand field is increased to 10 address bits, how many memory locations can be directly addressed?

3. Specify the maximum number of memory locations that can be directly addressed by the following multiple-word instruction:



4. In a minicomputer, the word size is typically 12 or 16 bits. This word size places a limit on the maximum number of memory locations that can be directly addressed. Describe three techniques that may be used to overcome this addressing limitation.

SOLUTIONS

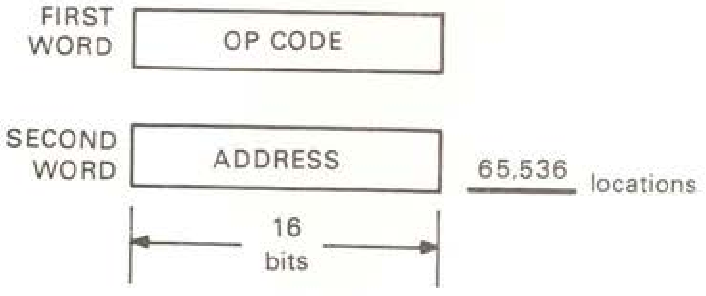
1. How many memory locations can be directly addressed if the operand field of the instruction word holds 7 address bits?

   128

2. If the size of the operand field is increased to 10 address bits, how many memory locations can be directly addressed?

   1024

3. Specify the maximum number of memory locations that can be directly addressed by the following multiple-word instruction:



65,536 locations

4. In a minicomputer, the word size is typically 12 or 16 bits. This word size places a limit on the maximum number of memory locations that can be directly addressed. Describe three techniques that may be used to overcome this addressing limitation.

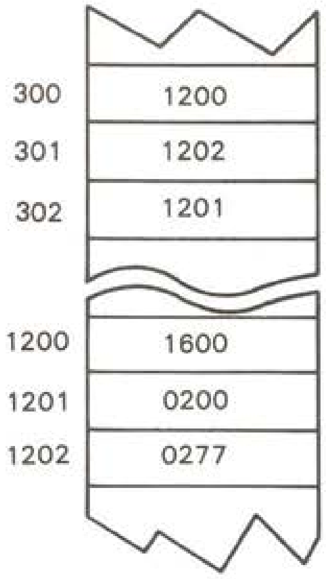
   a. Multiple-word instructions (2- or 3-word instructions) may be used instead of single-word instructions. This technique usually provides enough address bits in the operand field of the instruction so that any location in main memory can be directly addressed.

   b. Special registers in the CPU may be used for addressing memory. Because each register holds one computer word, it can usually accommodate enough address bits to directly address any location in main memory. Sometimes these registers are used for other purposes besides addressing memory. In such cases, they are usually called general purpose registers (GPRs).

   c. Main memory can be divided into smaller segments called pages. The size of each page is chosen so that the CPU can directly address any location in a given page simply by using the available address bits in the operand field of the instruction.

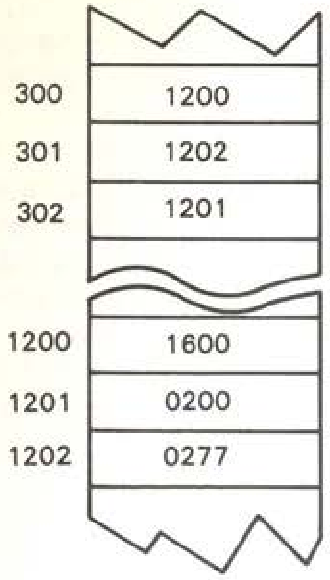
EXERCISES

5. The diagram at the right specifies the contents of various memory locations. The table below lists several instructions that reference these memory locations. Some instructions call for *direct* addressing; others call for *indirect* addressing. For each instruction, specify the *operand address* and the accompanying *operand*.



Instruction	Addressing Method	Operand Address	Operand
ADD 300	Direct	300	1200
ADD I 300	Indirect		
ADD I 301	Indirect		
ADD 302	Direct		
ADD 301	Direct		
ADD I 302	Indirect		

5. The diagram at the right specifies the contents of various memory locations. The table below lists several instructions that reference these memory locations. Some instructions call for *direct* addressing; others call for *indirect* addressing. For each instruction, specify the *operand address* and the accompanying *operand.*



Instruction	Addressing Method	Operand Address	Operand
ADD 300	Direct	300	1200
ADD I 300	Indirect	1200	1600
ADD I 301	Indirect	1202	0277
ADD 302	Direct	302	1201
ADD 301	Direct	301	1202
ADD I 302	Indirect	1201	0200

EXERCISES

6. Explain when it is necessary to use indirect addressing.

7. Match each term in the left-hand column with the appropriate description in the right-hand column. There may be more than one description for some terms.

a. Direct Addressing

b. Indirect Addressing

c. Multiple-Word Instruction

d. General Purpose Register

e. Page

f. Page Register

( ) A smaller segment of main memory that is directly addressable.

( ) Used to keep track of the page currently being referenced.

( ) The operand field of the instruction points to a memory location that contains the address of the operand.

( ) The operand field of the instruction points to a memory location that contains the operand.

( ) Used as an accumulator or for addressing operands stored in memory.

( ) Memory must be referenced two or more times to retrieve one complete instruction.

( ) ADD I 177.

6. Explain when it is necessary to use indirect addressing.

   Indirect addressing must be used whenever an instruction is stored in one memory page and the operand referenced by the instruction is stored in a different page.

7. Match each term in the left-hand column with the appropriate description in the right-hand column. There may be more than one description for some terms.

   a. Direct Addressing

   b. Indirect Addressing

   c. Multiple-Word Instruction

   d. General Purpose Register

   e. Page

   f. Page Register

   (e) A smaller segment of main memory that is directly addressable.

   (f) Used to keep track of the page currently being referenced.

   (b) The operand field of the instruction points to a memory location that contains the address of the operand.

   (a) The operand field of the instruction points to a memory location that contains the operand.

   (d) Used as an accumulator or for addressing operands stored in memory.

   (c) Memory must be referenced two or more times to retrieve one complete instruction.

   (b) ADD I 177.

Typical Instruction Set

OBJECTIVES

1. Given a simple mathematical expression and several known factors and restrictions, be able to write the steps of a simple program to solve the given expression. The program must include the known factors and abide by the restrictions given.

2. Given a simple mathematical expression and several known factors and restrictions, be able to write the steps of a simple program to solve the given expression. The program must include the known factors, a program loop, and must abide by the restrictions.

3. Given a complete program and five mathematical expressions, be able to select the expression that is solved by the program.

4. Given five program control statements, be able to label those statements that refer to conditional instructions and those that refer to unconditional instructions.

SAMPLE TEST ITEMS

1. Write a simple program that adds A, B, and C and then stores the answer (X) in memory location 333.

Known Factors:	A is stored in location 330. B is stored in location 331. C is stored in location 332.
Restrictions:	Use only the instructions defined in the lesson "Typical Instruction Set." Use 200 as the starting address of the program.

SAMPLE TEST ITEMS

2. Write a program that multiplies 73_8 by 26_8 and then stores the answer in memory location 214. Use a program loop in your solution.

Known Factors:	The operand 73_8 is stored in memory location 212; the operand 26_8 is stored in location 213.
Restrictions:	Do not use a multiply instruction; use only the instructions that are defined in this lesson. The starting address of your program should be 200.

3. Circle the letter of the mathematical expression that is solved by the following program.

```
200  CLA
201  ADD   216
202  CMA
203  IAC
204  STR   216
205  ADD   215
206  ISZ   216
207  JMP   205
210  STR   215
211  ADD   215
212  ADD   215
213  ADD   217
214  HLT
215  A
216  B
217  C
```

Answers

a. (2 + A * B) + C
b. (2 * A + C) + B
c. (2 * B * C) + A
d. (2 * A * B) + C
e. (2 * A + B) + C

SAMPLE TEST ITEMS

4. Indicate that each of the following statements refers to a conditional instruction (C) or an unconditional instruction (U) by writing the correct letter in the space provided.

Statement	Instruction Type
If y is negative, branch to 277.	________
Branch to location 215 if $x = 0$.	________
Skip the next instruction in the sequence if A is positive.	________
Skip the next instruction in the sequence.	________
Jump to location 307.	________

Mark your place in this workbook and view Lesson 3 in the A/V program, "Instruction Sets."

The audio-visual program defined a typical set of instructions and explained how the instructions are used to construct some simple computer programs. We began our discussion with the following instructions:

- Clear Accumulator
- Add
- Store
- Halt

Let's briefly review the functions of each of these fundamental instructions.

Clear Accumulator

CLA

The Clear Accumulator instruction is identified by the mnemonic CLA. This instruction causes the central processor to reset (or clear) the contents of its accumulator to all zeros. Because the CLA instruction operates *directly* on the contents of the accumulator, it does *not* reference memory and, therefore, does not contain an operand field.

Add Instruction

ADD	XXX

The Add instruction consists of two elements – the mnemonic ADD and the accompanying operand field. This instruction directs the CPU to retrieve the operand from memory location XXX and add the operand to the contents of the accumulator. After the addition is performed, the resulting sum appears in the accumulator – destroying the original contents. However, the operand stored in memory is not changed.

Store Instruction

STR	XXX

The Store instruction also consists of two elements – the mnemonic STR and the operand field. When the central processor executes this STR instruction, it transfers the contents of the accumulator into memory location XXX and then clears the accumulator to all zeros.

Halt Instruction

HLT

The Halt instruction is identified by the mnemonic HLT. Since the HLT instruction stops computer operations, it is usually the last instruction in a program. Note that the Halt instruction does not operate on data and, therefore, does not contain an operand field.

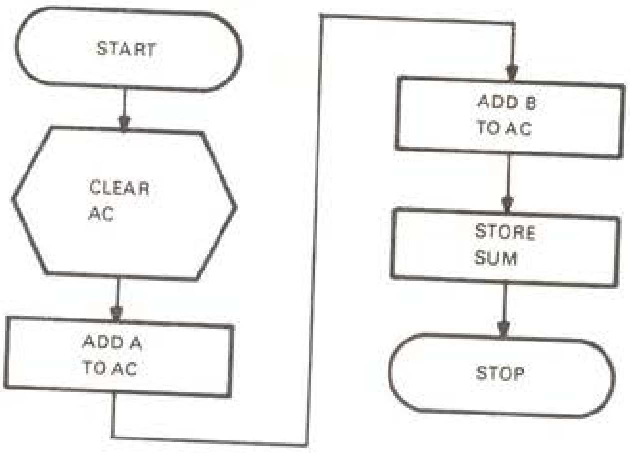
Program Examples

In the examples that follow, we will develop some simple programs using the instructions CLA, ADD, STR, and HLT. Take the necessary time to study and work through these sample programs. They should help to reinforce your understanding of instructions and basic programming techniques. Later in this workbook we will ask you to develop some simple programs using the same set of instructions.

Sample Program No. 1

Problem: The operands A and B are stored in memory locations 205 and 206. Our job is to write a short program that will allow the computer to solve the equation $X = A + B$ and store the answer in memory location 207.

a. First, we construct a flowchart.



b. Then we convert the flowchart into a computer program and assign memory locations to the instructions in the program.

Address	Instruction or Data	Explanation
200	CLA	Clear contents of AC.
201	ADD 205	Add A to contents of AC.
202	ADD 206	Add B to contents of AC.
203	STR 207	Store sum in location 207.
204	HLT	Stop operations.
205	A	
206	B	Data (operands)
207	X	

Sample Program No.2

Problem: Write a program to solve the equation Y = A + B + C and store the result in memory location 303.

a. In this example we have chosen not to construct a flowchart since the solution is similar to the one used in the preceding problem.

b. We begin by writing a basic program and assigning memory locations to the instructions.

```
200   CLA
201   ADD A
202   ADD B
203   ADD C
204   STR Y
205   HLT
```

c. Note that the operands A, B, and C must be stored in memory before our program can be executed. Therefore, we assign each operand to a separate memory location and modify the program to reflect these memory assignments. The resulting program is shown on the next page.

Address	Instruction or Data	Explanation
200	CLA	Clear contents of AC.
201	ADD 300	Add A to contents of AC.
202	ADD 301	Add B to contents of AC.
203	ADD 302	Add C to contents of AC.
204	STR 303	Store Y in location 303.
205	HLT	Stop operations.
300	A	Operands
301	B	
302	C	
303	Y	

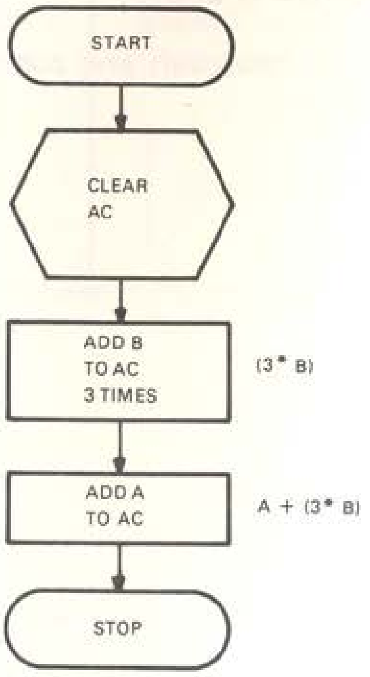
You've had an opportunity to study examples of two relatively simple computer programs. Now it's your turn to write a program that uses the CLA, ADD, STR, and HLT instructions.

Problem: Construct a flowchart and then write a program to solve the equation X = A + (3 * B). The operand A is stored in memory location 206; the operand B is stored in location 207. The starting address of your program is 200.

a. First construct a flowchart and compare it against the solution on the next page.

Solution: Construct a flowchart and then write a program to solve the equation X = A + (3 * B).

a. Flowchart



b. If your flowchart is correct, convert it into a program and then compare your program against the solution on page IS 40.

Solution: Write a program to solve the equation X = A + (3 * B).

b. Program

Address	Instruction or Data	Explanation
200	CLA	Clear contents of AC.
201	ADD 207	Add B to contents of AC 3 times (3 * B).
202	ADD 207	
203	ADD 207	
204	ADD 206	Add A to (3 * B).
205	HLT	Stop operations.
206	A	Operands
207	B	

We've now examined three simple computer programs using the instructions CLA, ADD, STR, and HLT. If your program is correct, you are ready to learn about some additional instructions. However, if you had difficulty with this program, consult your course manager.

Additional Instructions

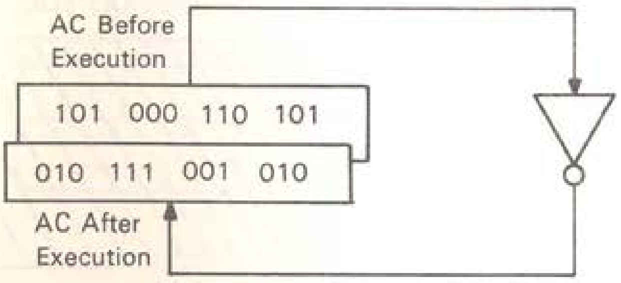
We are now going to add two more instructions to our basic instruction set:

- Complement Accumulator
- Increment Accumulator

Complement Accumulator

CMA

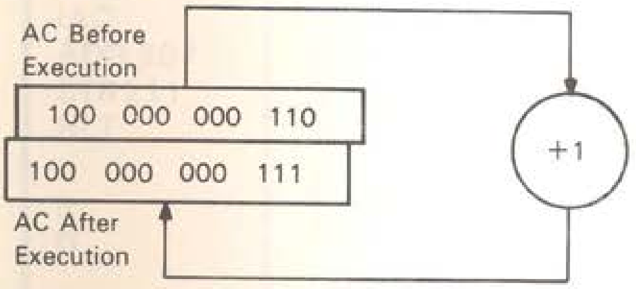
The Complement Accumulator instruction is identified by the mnemonic CMA. This instruction directs the central processor to replace the value held in the accumulator with its *ones complement.* In other words, each 1 is replaced with a 0, and each 0 is replaced by a 1. Since the CMA instruction does *not* reference an operand stored in memory, it does *not* contain an operand field.



Increment Accumulator

IAC

The Increment Accumulator instruction is assigned the mnemonic IAC. This instruction causes the CPU to increment (add one) to the contents of its accumulator. Note that the IAC instruction does *not* reference an operand stored in memory and, therefore, this instruction does *not* have an operand field.



Negative Numbers and Subtraction

The CMA and IAC instructions allow the computer to express a negative number as the *two's complement* of the positive value and to perform subtraction using *two's complement* addition. The following example illustrates how subtraction is implemented using the CMA and IAC instructions.

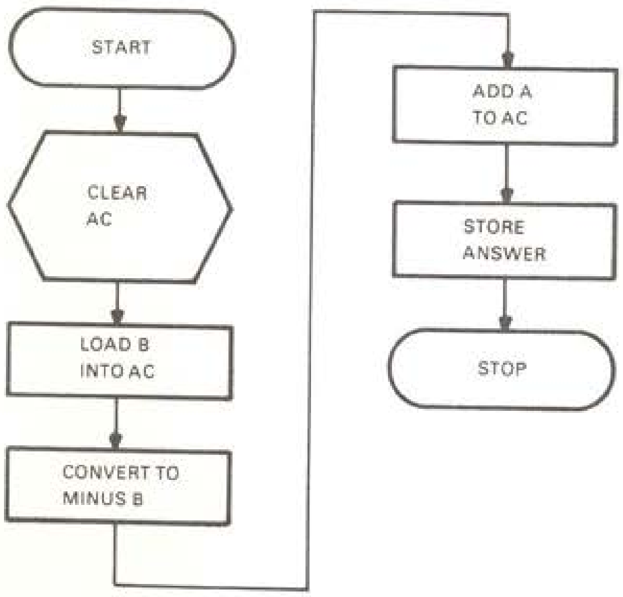
Address	Instruction or Data	Contents of AC After Instruction Is Executed	
		Binary	Octal
200	CLA	000 000 000 000	0000
201	ADD 301	000 000 000 011	0003
202	**CMA**	111 111 111 100	7774
203	**IAC**	111 111 111 101	7775
204	ADD 300	000 000 000 100	0004
•	•	▨	▨
•	•	▨	▨
•	•	▨	▨
300	0007	▨	▨
301	0003	▨	▨

In the above example the number to be subtracted (subtrahend) is first brought into the accumulator using the ADD 301 instruction. Next, the number in the accumulator is complemented (*one's complement*) and incremented by one to form the *two's complement*. The minuend (0007) is then added to the contents of the accumulator and the difference (0004) is obtained.

Sample Program No. 3 – Subtraction

Problem: Develop a program that will allow the computer to solve the expression D = A – B and store the answer in memory location 211. The operand A is stored in location 207; the operand B is stored in location 210.

a. Draw a flowchart.



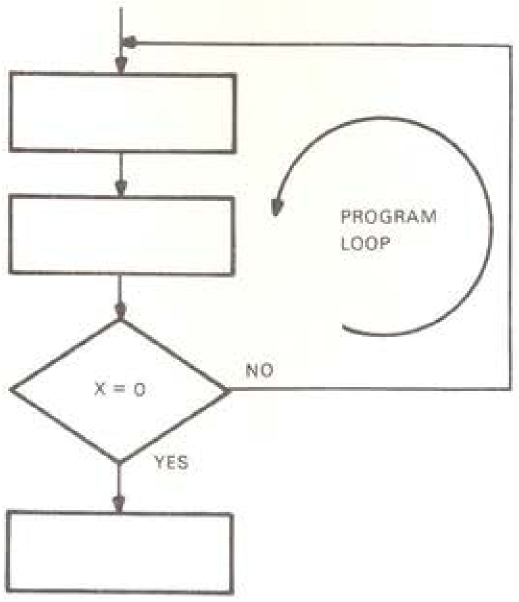
b. Convert the flowchart to a program and assign memory locations to the instructions

Address	Instruction or Data	Explanation
200	CLA	Clear contents of AC.
201	ADD 210	Load B into AC.
202	CMA	Form 1's complement of B.
203	IAC	Form 2's complement of B.
204	ADD 207	Add A to –B.
205	STR211	Store difference in location 211.
206	HLT	Stop operations.
207	A	} Operands
210	B	
211	D	

Program Loops, Tallys, and Branches

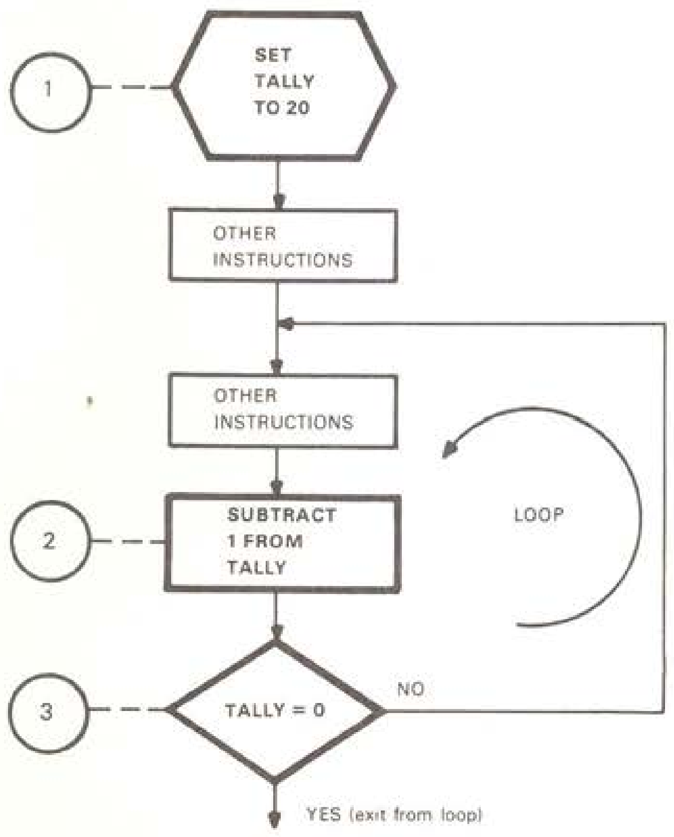
At some point in a program it may be necessary to repeat a sequence of instructions many times. Instead of writing the same instructions over and over again, we can use a very common programming technique called a *program loop.*

A program loop is constructed so that the central processor jumps back to an earlier part of the program and repeats the same sequence of instructions until a predefined condition is satisified. Once the predefined condition is met, the CPU simply exits from the loop and goes on to the next instruction in the program.

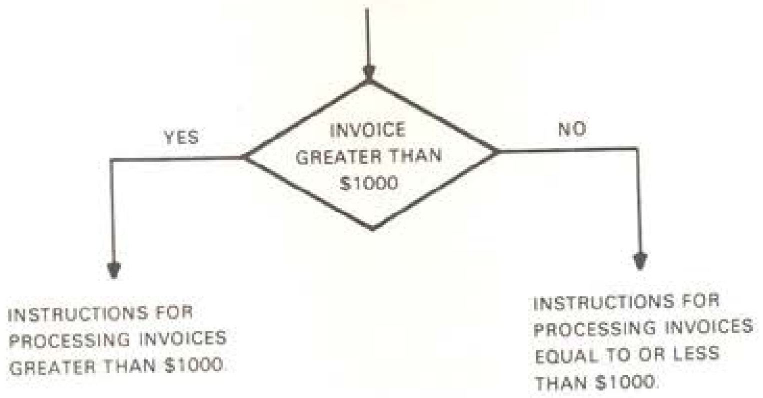


When we use a program loop, we must incorporate the following steps into our program:

1. Set up a tally (or count) that specifies the number of program loops to be performed.
2. Update the tally (add 1 or subtract 1) each time the central processor makes one pass through the program loop.
3. Test the updated tally so that the central processor can decide when it is time to exit from the program loop.



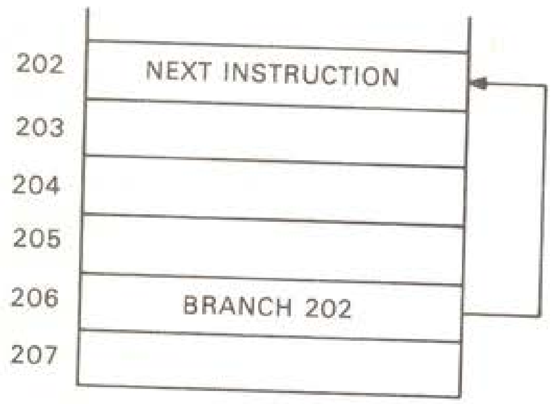
In addition to program loops, it is often necessary to include separate paths (branches) in our programs. For example, suppose that we are writing a program for processing customer invoices. If any invoice is greater than $1000, the central processor must execute one set of instructions. On the other hand, if an invoice is equal to or less than $1000, a different set of instructions is required. Our program must be written so that the CPU can decide which branch of instructions is to be executed. This decision is based on the dollar amount of the customer invoice. This programming technique is called *branching.*


YES
INVOICE GREATER THAN $1000
NO
INSTRUCTIONS FOR PROCESSING INVOICES GREATER THAN $1000
INSTRUCTIONS FOR PROCESSING INVOICES EQUAL TO OR LESS THAN $1000

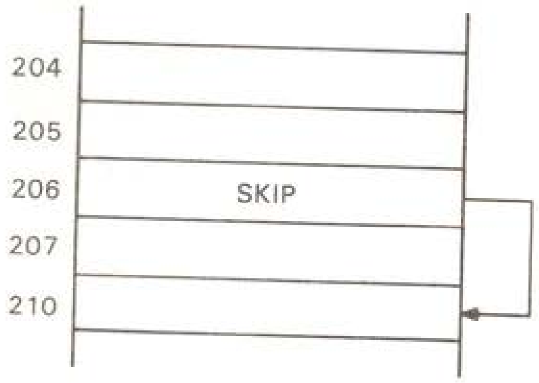
Program Control Instructions

The central processor normally executes instructions sequentially in the same order that they are stored in main memory. However, when we incorporate loops or branches into our programs, we must divert the CPU from the normal instruction sequence. This can be accomplished by using a special category of instructions called *program control instructions.*

The two most common forms of program control instructions are *branches* and *skips.* A branch instruction alters the normal instruction sequence by redirecting the CPU to another point in a program. In the example below, the branch instruction causes the CPU to get its next instruction from memory location 202.

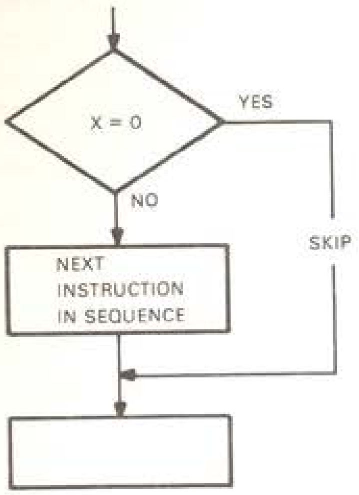


On the other hand, a skip instruction simply causes the CPU to skip the next instruction in the program.



Branch and skip instructions can be further divided into *conditional* and *unconditional* instructions. A *conditional* branch or skip instruction diverts the CPU from its normal program sequence only if a prescribed condition has been met.

In the example at the right, the CPU will skip the next instruction if X equals 0. However, if the condition is *not* satisfied (i.e., $X \neq 0$), then the CPU simply proceeds with the next instruction in the sequence.



Unconditional branch and skip instructions, on the other hand, always divert the CPU from its normal instruction sequence. There is no prescribed condition that has to be met before the branch or skip operation can take place.

In the audio-visual portion of this lesson we defined and used two program control instructions:

- Jump
- Increment and Skip If Zero

Jump Instruction

JMP	**XXX**

The Jump instruction consists of 2 elements – the mnemonic JMP and the operand field. This JMP instruction is nothing more than an *unconditional* branch instruction. It causes the central processor to retrieve its next instruction from memory location XXX.

Increment and Skip If Zero

ISZ	XXX

The Increment and Skip If Zero instruction also consists of 2 elements – the mnemonic ISZ and the operand field. This ISZ instruction causes the central processor to perform the following sequence of operations:

1. Increment (add 1) to the contents of memory location XXX.

2. Test the incremented value.

3. If the value is 0, skip the next instruction in the sequence. If the value is *not* 0, ignore the conditional skip and go to the next instruction.

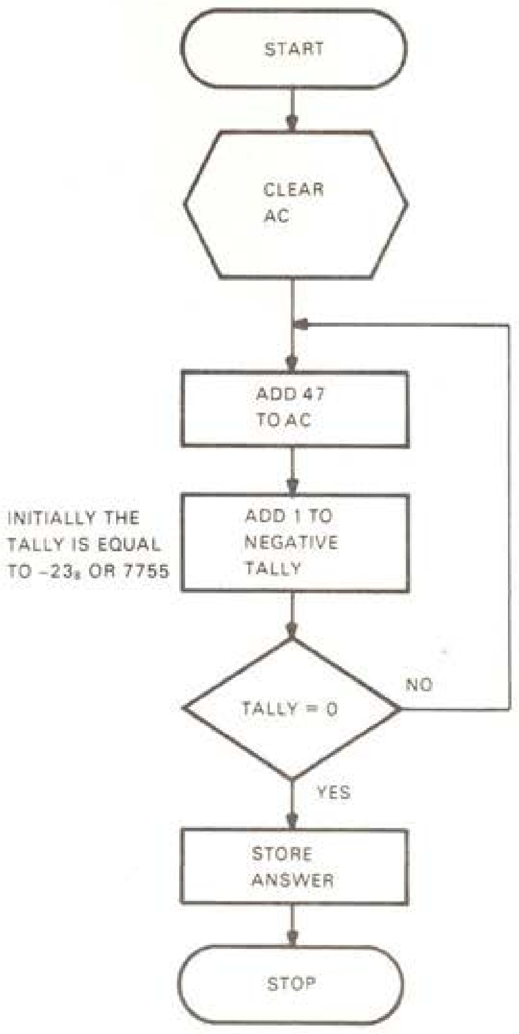
Additional Program Examples

In the examples that follow, we will write some simple programs using the program control instructions JMP and ISZ.

Sample Program No. 4

Problem: Write a program that will allow the CPU to multiply 47_8 times 23_8 and store the answer in memory location 352. Memory location 350 contains the operand 47; location 351 contains the operand 7755 (which is equivalent to -23_8).

a. Since we do not have a multiply instruction, the CPU must add 47_8 to the contents of its accumulator 23_8 times. The most efficient way of accomplishing this is to set up a tally and use the following program loop:



b. The flowchart is then converted to the following program:

Address	Instruction or Data	Explanation
200	CLA	Clear contents of AC.
201	ADD 350	Add 47 to contents of AC.
202	ISZ 351	Increment tally; skip if tally is 0.
203	JMP 201	Otherwise, go back to add instruction.
204	STR 352	Store answer in location 352.
205	HLT	Stop operations.
•	•	•
•	•	•
•	•	•
350	0047	Operand
351	7755 (–23)	Tally.
352	(Answer)	Product of 47_8 * 23_8.

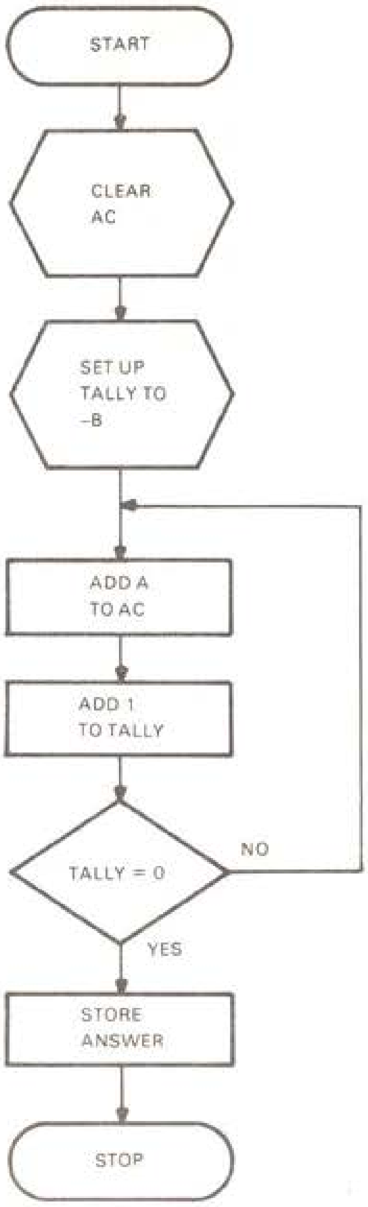
c. Additional comments:

- In the above example, the TALLY is –23 (or 7755 in *two's complement* notation).
- The ISZ instruction adds one to the TALLY each time the CPU makes one pass through the program loop (7756 . . . 7757 . . . 7760 . . . etc.).
- If the incremented TALLY is not 0, the JMP instruction sends the CPU back to the start of the program loop.
- Finally, when the TALLY is incremented to 0, it signifies that the required number of additions (loops) have been performed. As a consequence, the ISZ instruction redirects the CPU to the STR instruction so that the answer can be stored in memory.

Sample Program No. 5

Problem: Write a more general program that allows the CPU to multiply *any* two numbers (represented by the variables A and B) and then store the answer in location 302. Assume A is stored in memory location 300 and B is stored in location 301.

a. The following flowchart includes an additional initialization step for setting up a TALLY (–B):



b. The flowchart is then converted to the following program:

Address	Instruction or Data	Explanation
200	CLA	Clear contents of AC.
201	ADD301	Load B into AC.
202	CMA	Form 1's complement of B.
203	IAC	Form 2's complement of B.
204	STR 301	Store −B back into memory; clear AC.
205	ADD 300	Add A to contents of AC.
206	ISZ 301	Increment −B; skip if tally is 0.
207	JMP 205	Otherwise, go back to add instruction.
210	STR 302	Store answer.
211	HLT	Stop operations.
•	•	•
•	•	•
•	•	•
300	A	Operand.
301	B	Tally.
302	(Answer)	Product of A times B.

c. Additional comments:

- The purpose of the instructions ADD 301, IAC, and STR 301 is to set up a TALLY that is equal to −B (the *two's complement* of B).
- When this TALLY is incremented to zero, the ISZ instruction forces the central processor out of the program loop.

Summary

The following charts summarize all of the instructions that have been defined in this lesson.

Instructions with a Single Operand Field

Instruction	Symbolic Code	Octal Code	Description
Add	ADD XXX	1XXX	Add the contents of location XXX to the contents of the AC. Place the sum back into the AC.
Increment and Skip If Zero	ISZ XXX	2XXX	Increment the contents of location XXX and skip the next instruction if the incremented value is 0.
Store	STR XXX	3XXX	Store the contents of the AC in location XXX and then clear the AC.
Jump	JMP XXX	5XXX	Jump unconditionally to location XXX for the next instruction.

Instructions with No Operand Field

Instruction	Symbolic Code	Octal Code	Description
Increment AC	IAC	7001	Increment (add 1) to the contents of the AC.
Complement AC	CMA	7040	Complement the contents of the AC (1's complement).
Clear AC	CLA	7200	Clear the contents of the AC to all zeros.
Halt	HLT	7402	Stop all processing operations.

EXERCISES

1. Convert each statement in the left-hand column to the appropriate instruction. Use only the eight basic instructions that were defined in this lesson. Note that one of the statements requires two instructions.

a.	Clear the contents of the accumulator to all zeros.	CLA
b.	Add the operand stored in location 300 to the contents of the AC.	________
c.	Transfer the contents of the AC to memory location 301 and then clear the AC.	________
d.	Increment the tally stored in location 350, test if the updated tally is 0, and skip the next instruction if tally = 0.	________
e.	Jump unconditionally to the instruction stored at memory location 227.	________
f.	Convert the number held in the AC to its *one's complement.*	________
g.	Add 1 to the contents of the AC.	________
h.	Convert the number held in the AC to its *two's complement.*	________ ________
i.	Stop all processing operations.	________

SOLUTIONS

1. Convert each statement in the left-hand column to the appropriate instruction. Use only the eight basic instructions that were defined in this lesson. Note that one of the statements requires two instructions.

a. Clear the contents of the accumulator to all zeros.	CLA
b. Add the operand stored in location 300 to the contents of the AC.	ADD 300
c. Transfer the contents of the AC to memory location 301 and then clear the AC.	STR 301
d. Increment the tally stored in location 350, test if the updated tally is 0, and skip the next instruction if tally = 0.	ISZ 350
e. Jump unconditionally to the instruction stored at memory location 227.	JMP 227
f. Convert the number held in the AC to its *one's complement.*	CMA
g. Add 1 to the contents of the AC.	IAC
h. Convert the number held in the AC to its *two's complement.*	CMA IAC
i. Stop all processing operations.	HLT

EXERCISES

2. Write a simple program that adds 7 to 5 and then stores the answer in memory location 302.

   Known Factors: Value 5 is stored in location 300.
   Value 7 is stored in location 301.

   Restrictions: Use only the instructions that have been defined in this lesson. Use 200 as the starting address of your program.

Address	**Instruction or Data**

2. Write a simple program that adds 7 to 5 and then stores the answer in memory location 302.

Known Factors:	Value 5 is stored in location 300. Value 7 is stored in location 301.
Restrictions:	Use only the instructions that have been defined in this lesson. Use 200 as the starting address of your program.

Address	Instruction or Data	Explanation
200	CLA	Clear AC to all zeros.
201	ADD 300	Add 5 to contents of AC.
202	ADD 301	Add 7 to contents of AC.
203	STR 302	Store sum in location 302.
204	HLT	Stop all processing operations.
•	•	•
•	•	•
•	•	•
300	5	Operand.
301	7	Operand.
302		Sum.

EXERCISES

3. Write a program that subtracts 5 *from* 7 and then stores the difference in memory location 302.

Known Factors:	Value 5 is stored in location 300. Value 7 is stored in location 301.
Restrictions:	Use only the instructions that have been defined in this lesson. Use 200 as the starting address of your program.

3. Write a program that subtracts 5 *from* 7 and then stores the difference in memory location 302.

Known Factors: Value 5 is stored in location 300.
Value 7 is stored in location 301.

Restrictions: Use only the instructions that have been defined in this lesson. Use 200 as the starting address of your program.

Address	Instruction or Data	Explanation
200	CLA	Clear AC to all zeros.
201	ADD 300	Load 5 into AC.
202	CMA	Convert 5 to its 1's complement.
203	IAC	Convert 5 to its 2's complement.
204	ADD 301	Add 7 to −5.
205	STR 302	Store difference in location 302.
206	HLT	Stop all processing operations.
•	•	•
•	•	•
•	•	•
300	5	Operand.
301	7	Operand.
302		Difference.

EXERCISES

4. Write a program that solves the mathematical expression X = A + B − C and then stores the answer in memory location 213.

Known Factors:	Operands A, B, and C are stored in memory locations 210, 211 and 212, respectively.
Restrictions:	Confine your choice of instructions to those that have been defined in this lesson. The starting address of your program should be 200.

4. Write a program that solves the mathematical expression X = A + B − C and then stores the answer in memory location 213.

Known Factors: Operands A, B, and C are stored in memory locations 210, 211, and 212, respectively.

Restrictions: Confine your choice of instructions to those that have been defined in this lesson. The starting address of your program should be 200.

Address	Instruction or Data	Explanation
200	CLA	Clear AC to all zeros.
201	ADD 212	Load C into AC.
202	CMA	Convert C to its 1's complement.
203	IAC	Convert C to its 2's complement.
204	ADD 211	Add B to −C.
205	ADD 210	Add A to (B − C).
206	STR 213	Store answer in location 213.
207	HLT	Stop all processing operations.
210	A	Operand.
211	B	Operand.
212	C	Operand.
213	X	Answer (A + B − C).

EXERCISES

5. Refer to the statements below and indicate if they describe a *conditional* or an *unconditional* program control instruction by checking the appropriate box.

Conditional Instruction	Unconditional Instruction	
☐	☐	Branch to location 215 if X = 0.
☐	☐	Skip the next instruction in the sequence.
☐	☐	Skip the next instruction in the sequence if A is positive.
☐	☐	Jump to location 307.
☐	☐	If Y is negative, branch to location 277.

SOLUTIONS

5. Refer to the statements below and indicate if they describe a *conditional* or an *unconditional* program control instruction by checking the appropriate box.

Conditional Instruction	Unconditional Instruction	
☑	☐	Branch to location 215 if X = 0.
☐	☑	Skip the next instruction in the sequence.
☑	☐	Skip the next instruction in the sequence if A is positive.
☐	☑	Jump to location 307.
☑	☐	If Y is negative, branch to location 277.

EXERCISES

6. Circle the correct answer. The instruction ISZ 300 causes the central processor to:

   a. Skip to memory location 300 for its next instruction.

   b. Increment the value 300 to 301 and then skip the next instruction in the sequence.

   c. Increment the contents of location 300 and skip the next instruction in the sequence if the incremented value is 0.

   d. Increment the contents of location 300 and skip the next instruction in the sequence.

7. How many times will the central processor execute the following program loop?

```
 •           •
 •           •
 •           •
204         ADD   277  )
205         ISZ   300   }  Number of  =  ________.
206         JMP   204  )   loops
 •           •
 •           •
 •           •
277         0176
300         7767
```

SOLUTIONS

6. The instruction ISZ 300 causes the central processor to:

   a. Skip to memory location 300 for its next instruction.

   b. Increment the value 300 to 301 and then skip the next instruction in the sequence.

   [✓ c.] Increment the contents of location 300 and skip the next instruction in the sequence if the incremented value is 0.

   d. Increment the contents of location 300 and skip the next instruction in the sequence.

7. How many times will the central processor execute the following program loop?

```
 :          :
204        ADD   277  }
205        ISZ   300  }  Number of   =  9 TIMES .
206        JMP   204  }  loops
 :          :
277        0176
300        7767
```

EXERCISES

8. Write a program that multiplies 67_8 by 42_8 and then stores the answer in memory location 214. Use a program loop in your solution.

   Known Factors: The operand 67_8 is stored in memory location 212; the operand 42_8 is stored in location 213.

   Restrictions: Do *not* use a multiply instruction; use only the instructions that are defined in this lesson. The starting address of your program should be 200.

8. Write a program that multiplies 67_8 by 42_8 and then stores the answer in memory location 214. Use a program loop in your solution.

Known Factors:	The operand 67_8 is stored in memory location 212; the operand 42_8 is stored in location 213.
Restrictions:	Do *not* use a multiply instruction; use only the instructions that are defined in this lesson. The starting address of your program should be 200.

Address	Instruction or Data	Explanation
200	CLA	Clear AC to all zeros.
201	ADD 213	Load 42 into AC.
202	CMA	Convert 42 to its 1's complement.
203	IAC	Convert 42 to its 2's complement.
204	STR 213	Store −42 (7736) in location 213; clear AC.
205	ADD 212	Add 67 to contents of AC.
206	ISZ 213	Increment tally; skip next instruction if tally = 0.
207	JMP 205	Otherwise go back to the add instruction.
210	STR 214	Store answer in location 214.
211	HLT	Stop all processing operations.
212	0067	Operand.
213	0042	Tally.
214		Answer.

Take the test for this module before beginning another module.