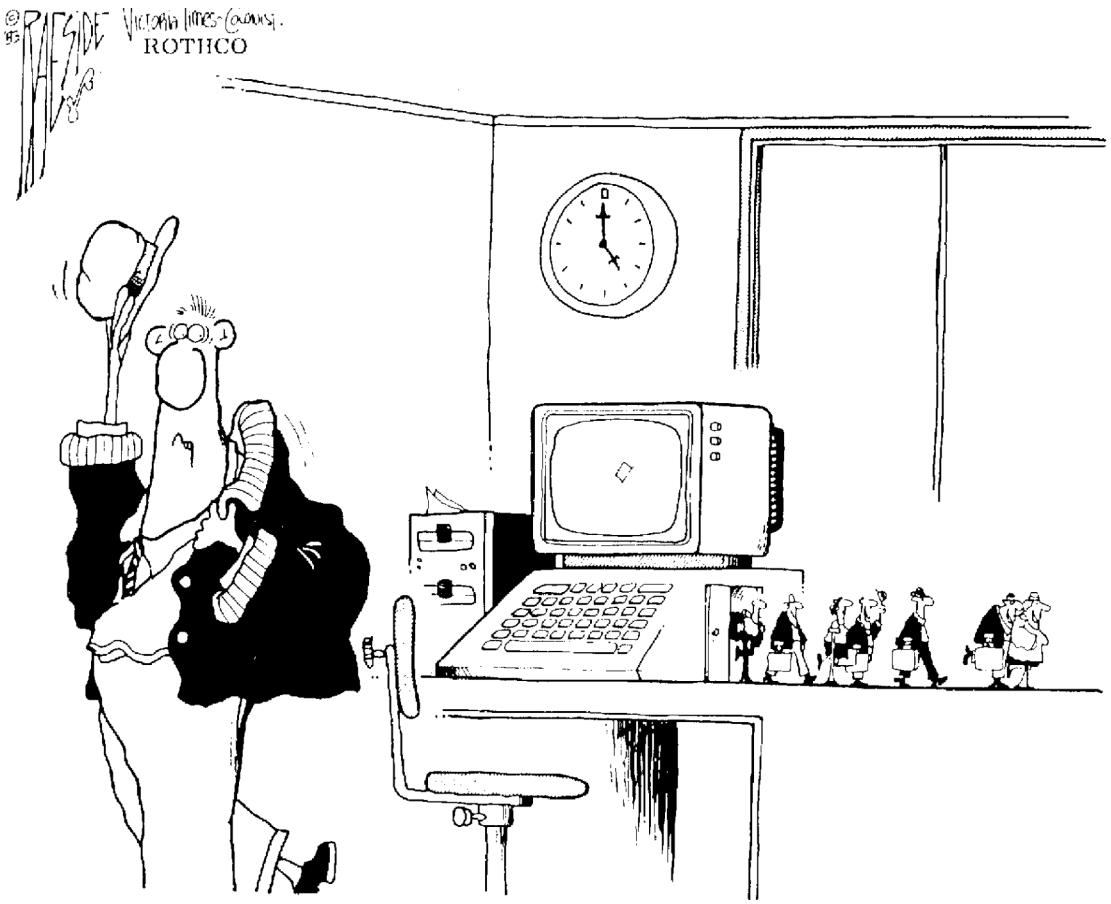
CHAPTER 6

THE LITTLE MAN COMPUTER



Raeside/Victoria Times-Colonist/Rothco

6.0 INTRODUCTION

The power of a computer does not arise from complexity. Instead, the computer has the ability to perform simple operations at an extremely high rate of speed. These operations can be combined to provide the computer capabilities that you are familiar with.

Consistent with this idea, the actual design of the computer is also simple, as you will see.

(The beauty of the design is that these simple operations can be used to solve extremely complex problems. The programmer's challenge, of course, is to produce the exact sequence of operations to perform a particular task correctly under all possible circumstances, since any error in selection or sequence of operations will result in a ''buggy'' program. With the large number of instructions required by modern programs, it is not surprising that few of today's programs are truly bug-free.)

In this chapter, we will begin to explore the operations that the computer is capable of performing and look at how those operations work together to provide the computer with its power. To simplify our exploration, we will begin by introducing a model of the computer; a model that operates in a very similar way to the real computer but that is easier to understand instinctively.

The model that we will use is called the **Little Man Computer (LMC).** The original LMC was created by Dr. Stuart Madnick at MIT in 1965. In 1979, Dr. Madnick produced a new version of the LMC, with a slightly modified instruction set; the later version is used in this book. It is a strength of the original model that it operates so similarly to a real computer that it is still an accurate representation of the way that computers work thirty-five years after its introduction.

Using this model we will introduce a simplified, but typical, set of instructions that a computer can perform. We will show you exactly how these instructions are executed in the Little Man Computer. Then we will demonstrate how these instructions are combined to form programs.

6.1 LAYOUT OF THE LITTLE MAN COMPUTER

We begin by describing the physical layout of the Little Man Computer. A diagram for the Little Man Computer appears in Figure 6.1.

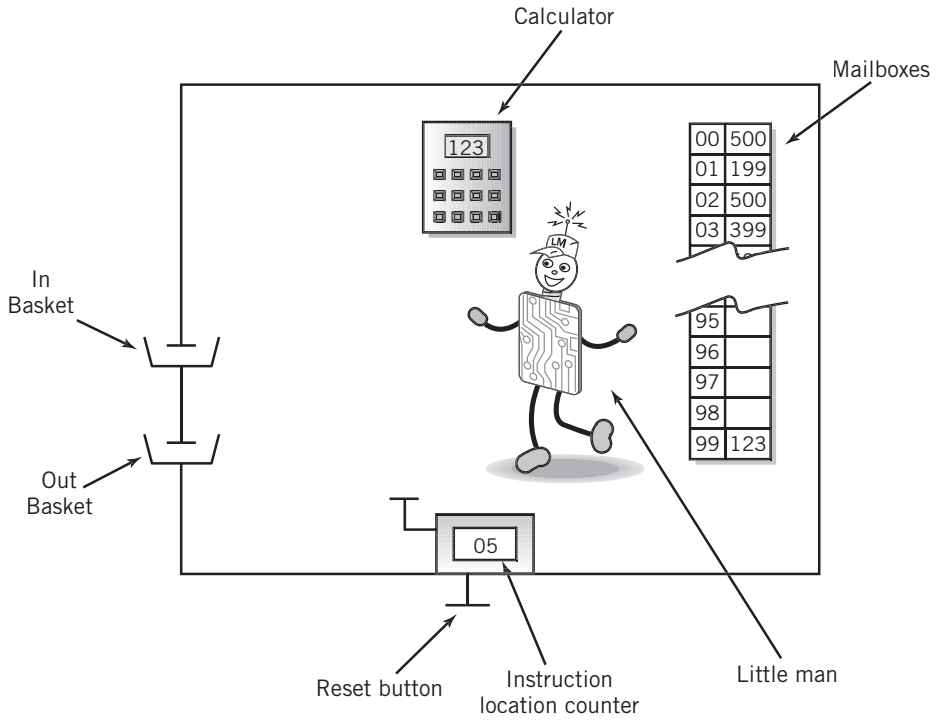
The LMC consists of a walled mailroom, represented by the dark line surrounding the model in the diagram. Inside the mailroom are several objects:

First, there is a series of one hundred *mailboxes*, each numbered with an address ranging from 00 to 99. This numbering system is chosen because each mailbox address can be represented by two digits, and this is the maximum number of mailboxes that can be represented by two decimal digits.

Each mailbox is designed to hold a single slip of paper, upon which is written a three-digit decimal number. Note carefully that the *contents* of a mailbox are not the same as the *address* of a mailbox. This idea is consistent with what you already know about your post office box: your post office box number identifies where you go

FIGURE 6.1

The Little Man Computer



to pick up your mail, but this has no relationship to the actual contents of the letters that you find in that mailbox.

Next, there is a *calculator* . . . basically a simple pocket calculator. The calculator can be used to enter and temporarily hold numbers, and also to add and subtract. The display on the calculator is three digits wide. At least for the purposes of this discussion, there is no provision made for negative numbers, or for numbers larger than three digits. As you are already aware, 10's complement arithmetic could be used for this purpose, but that is not of interest here.

Third, there is a two-digit *hand counter*, the type that you click to increment the count. The reset button for the hand counter is located outside the mailroom. We will call this counter an *instruction location counter*.

Finally, there is the Little Man. It will be his role to perform certain tasks that will be defined shortly.

Other than the reset switch on the hand counter, the only interaction between the Little Man Computer and the outside environment are an *in basket* and an *out basket*.

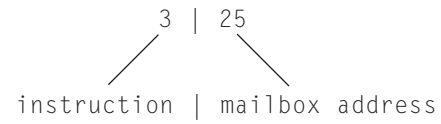
A user outside of the mailroom can communicate with the Little Man in the mailroom by putting a slip of paper with a three-digit number on it into the in basket, to be read by the Little Man at the appropriate time. Similarly, the Little Man can write a three-digit number on a slip of paper and leave it in the out basket, where it can be retrieved by the user.

Note that *all communication* between the Little Man Computer and the outside world takes place using three-digit numbers. Except for the reset button on the instruction location counter, no other form of communication is possible. The same is true within the mailroom: all instructions to the Little Man must be conveyed as three-digit numbers.

6.2 OPERATION OF THE LMC

We would like the Little Man to do some useful work. For this purpose we have invented a small group of instructions that he can perform. Each instruction will consist of a single digit. We will use the first digit of a three-digit number to tell the Little Man which operation to perform.

In some cases, the operation will require the Little Man to use a particular mailbox to store or retrieve data (in the form of three-digit numbers, of course!). Since the instruction only requires one digit, we can use the other two digits in a three-digit number to indicate the appropriate mailbox address to be used as a part of the instruction. Thus, using the three digits on a slip of paper we can describe an instruction to the Little Man according to the following diagram:



The instruction part of the three-digit code is also known as an "operation code," or **op code** for short. The op code number assigned to a particular instruction is arbitrary, selected by the computer designer based on various architectural and implementation factors. The op codes used by the author conform to the 1979 version of the Little Man Computer model.

Now let's define some instructions for the Little Man to perform:

LOAD instruction—op code 5

The Little Man walks over to the mailbox address specified in the instruction. He reads the three-digit number located in that mailbox, and then walks over to the calculator and punches that number into the calculator. The three-digit number in the mailbox is left unchanged, but of course the original number in the calculator is replaced by the new number.

STORE instruction—op code 3

This instruction is the reverse of the LOAD instruction. The Little Man walks over to the calculator and reads the number there. He writes that number on a slip of paper and puts it in the mailbox whose address was specified as the address part of the instruction. The number in the calculator is unchanged; the original number in the mailbox is replaced with the new value.

ADD instruction—op code 1

This instruction is very similar to the LOAD instruction. The Little Man walks over to the mailbox address specified in the instruction. He reads the three-digit number located in the mailbox and then walks over to the calculator and *adds it to the number already in the calculator*. The number in the mailbox is unchanged.

SUBTRACT instruction—op code 2

This instruction is the same as the ADD instruction, except that the Little Man subtracts the mailbox value from the value in the calculator. The result of a subtraction can leave a negative value in the calculator. Chapter 5 discussed the use of complements to implement negative values, but for simplicity, the LMC model ignores this solution. For the purposes of our LMC model, we will simply assume that the calculator holds and handles negative values correctly, and provides a minus sign as a flag to indicate that the value is negative. The Little Man cannot handle negative numbers outside of the calculator, however, because there is no provision in the model for storing the negative sign within the constraint of the three-digit number system used.

INPUT instruction (or read, if you prefer)—op code 9, "address" 01

The Little Man walks over to the in basket and picks up the slip of paper in the basket. He then walks over to the calculator and punches it into the calculator. The number is no longer in the in basket, and the original calculator value has been replaced by the new number. If there are multiple slips of paper in the basket, the Little Man picks them up in the order in which they were submitted, but each INPUT instruction handles only a single slip of paper; other input values must await the execution of subsequent INPUT instructions. Some authors use the concept of a conveyor belt in place of the in basket, to emphasize this point.

OUTPUT instruction (or print)—op code 9, "address" 02

The Little Man walks over to the calculator and writes down the number that he sees there on a slip of paper. He then walks over to the out basket and places the slip of paper there for the user outside the mailroom to retrieve. The original number in the calculator is unchanged. Each OUTPUT instruction places a single slip of paper in the out basket. Multiple outputs will require the use of multiple OUTPUT instructions.

Note that the INPUT and OUTPUT instructions do not use any mailboxes during execution, since the procedure for each only involves the transfer of data between an in or out basket and the calculator. Because this is true, the address part of the instruction can be used to extend the capability of the instruction set, by using the same op code with different "address" values to create a number of different instructions. In the LMC, 901 is the code for an INPUT instruction, while 902 is used for an OUTPUT instruction. In a real computer, for example, the instruction address might be used to specify the particular I/O device to be used for input or output.

COFFEE BREAK (or HALT) instruction—op code 0

The Little Man takes a rest. The Little Man will ignore the address portion of the instruction.

The instructions that we have defined so far fall into four categories:

- instructions that move data from one part of the LMC to another (LOAD, STORE)
- instructions that perform simple arithmetic (ADD, SUBTRACT)

- instructions that perform input and output (INPUT, OUTPUT)
- instructions that control the machine (COFFEE BREAK)

This is enough for now. We will discuss instructions 6, 7, and 8 later in this chapter.

6.3 A SIMPLE PROGRAM

Now let's see how we can combine these instructions into a program to have the Little Man do some useful work.

Before we do this, we need to store the instructions somewhere, and we need a method to tell the Little Man where to find the particular instruction that he is supposed to perform at a given time.

Without discussing how they got there, for now we will assume that the instructions are stored in the mailboxes, starting at mailbox number 00. The Little Man will perform instructions by looking at the value in the instruction location counter and executing the instruction found in the mailbox whose address has that value. Each time the Little Man completes an instruction, he will walk over to the instruction location counter and increment it. Again he will perform the instruction specified by the counter. Thus, the Little Man will execute the instructions in the mailboxes sequentially, starting from mailbox 00. Since the instruction location counter is reset from outside the mailroom, the user can restart the program simply by resetting the counter to 00.

Now that we have a method for guiding the Little Man through a program of instruction steps, let's consider a simple program that will allow the user outside the mailroom to use the Little Man Computer to add two numbers together. The user will place two numbers in the in basket. The sum of the two will appear as a result in the out basket. The question is what instructions we will need to provide to have the Little Man perform this operation.

INPUT 901

Since the Little Man must have access to the data, the first step, clearly, is to have the Little Man read the first number from the in basket to the calculator. This instruction leaves the first number to be added in the calculator.

STORE 99 399

Note that it is not possible for the Little Man to simply read another number into the calculator. To do so would destroy the first number. Instead, we must first save the first number somewhere.

Mailbox 99 was chosen simply because it is clearly out of the way of the program. Any other location that is beyond the end of the program is equally acceptable.

Storing the number at a location that is within the program would destroy the instruction at that location. This would mean that when the Little Man went to perform that instruction, it wouldn't be there.

More seriously, there is no way for the Little Man to distinguish between an instruction and a piece of data—both are made up of three-digit numbers. Thus, if we were to store data in a location that the Little Man is going to use as an instruction, the Little Man would simply attempt to perform the data as though it were an instruction. Since there is no way to predict what the data might contain, there is no way to predict what the program might do.

The concept that there is no way to distinguish between instructions and data except in the context of their use is a very important one in computing. For example, it allows a programmer to treat an instruction as data, to modify it, and then to execute the modified instruction.

INPUT 901

With the first number stored away, we are ready to have the Little Man read the second number into the calculator.

ADD 99 199

Note that there is no specific reason to save the second number. If we were going to perform some operation that required the reuse of the second number, it could be stored somewhere.

In this program, however, we have both numbers in place to perform the addition. The result is, of course, left in the calculator.

OUTPUT 902

All that remains is for us to have the Little Man output the result to the out basket.

COFFEE BREAK 000

The program is complete, so we allow the Little Man to take a rest.

These instructions are stored sequentially starting from mailbox 00, where the Little Man will retrieve and execute them one at a time, in order. The program is reshown in Figure 6.2.

Since we were careful to locate the data outside the program, this program can be rerun simply by telling the Little Man to begin again.

6.4 AN EXTENDED INSTRUCTION SET

The instructions that we have defined must always be executed in the exact sequence specified. Although this is sufficient for simple program segments that perform a sequence of operations, it does not provide any means for branching or looping, both constructs that you know are very important in programming. Let us extend the instruction set by adding three more instructions for this purpose:

FIGURE 6.2

Program to Add Two Numbers

Mailbox code		Instruction description
00	901	INPUT
01	399	STORE DATA
02	901	INPUT 2ND #
03	199	ADD 1ST # TO IT
04	902	OUTPUT RESULT
05	000	STOP
99		DATA

BRANCH UNCONDITIONALLY instruction (sometimes known as JUMP)—op code 6

This instruction tells the Little Man to walk over to the instruction location counter and actually *change* the counter to the location shown in the two address digits of the instruction. (Assume that the hand counter has thumbwheels for this purpose.) This means that the next instruction that the Little Man will execute is located at that mailbox address.

This instruction is very similar, conceptually, to the GOTO instruction in BASIC. Its execution will always result in a break in the sequence to another part of the program.

Note that this instruction also uses the address digits in an unusual way, since the Little Man does not use the data at the address specified. Indeed, the Little Man expects to find an instruction at that address, the next to be performed.

BRANCH ON ZERO instruction—op code 7

The Little Man will walk over to the calculator and will observe the number stored there. If its current value is zero, he will walk over to the instruction location counter and modify its value to correspond to the address specified within the instruction. The next instruction executed by the Little Man will be located at that address.

If the value in the calculator is not zero, he will simply proceed to the next instruction in sequence.

BRANCH ON POSITIVE instruction—op code 8

The Little Man will walk over to the calculator and will observe the number stored there. If its current value is positive, he will walk over to the instruction location counter and modify its value, to correspond to the address specified within the instruction. The next instruction executed by the Little Man will be located at that address.

If the value in the calculator is negative, he will simply proceed to the next instruction in sequence. Zero is considered to be a positive value.

Note that is it not necessary to provide BRANCH ON NEGATIVE or BRANCH ON NONZERO instructions. The instructions supplied can be used together to achieve equivalent results.

These three instructions make it possible to break from the normal sequential processing of instructions. Instructions of this type are used to perform branches and loops. As an example, consider the following WHILE-DO loop, common to many programming languages:

```
WHILE Value=0 DO
   Task;
NextStatement
```

This loop could be implemented using the Little Man BRANCH instruction as follows. Assume that these instructions are located starting at mailbox number 45 (comments are provided to the right of each line):

```
45  LDA 90     590  90 is assumed to contain value
46  BRZ 48     748  Branch if the value is zero
47  BR 60      660  Exit loop; Jump to NextStatement
48   :              This is where the task is located
59  BR 45      645  End to Task; loop to test again
60                  Next statement
```

EXAMPLE

Here is an example of a Little Man program that uses the BRANCH instructions to alter the flow of the program. This program finds the positive difference between two numbers (sometimes known as the absolute magnitude of the difference). For convenience, we are introducing a set of abbreviations for each instruction. These abbreviations are known as **mnemonics**

FIGURE 6.3

Little Man Mnemonic Instruction Codes with Their Corresponding OP Codes

LDA	5xx	Load
STO	3xx	Store
ADD	1xx	Add
SUB	2xx	Subtract
IN	901	Input
OUT	902	Output
COB or HLT	000	Coffee break (or Halt)
BRZ	7xx	Branch if zero
BRP	8xx	Branch if positive or zero
BR	6xx	Branch unconditional
DAT		Data storage location

(the first ''m'' is silent). Once you learn to read these mnemonics, you'll find that programs written with mnemonics are generally easy to read. It is more common to write programs this way. For a while, we will continue to print both the mnemonic and the code, but eventually, we will stop printing the code. Most programs are also written with *comments*, which help to clarify the code. The mnemonic instructions that we will use are shown in Figure 6.3. The DAT abbreviation is used to indicate that a particular mailbox will be used to store data. The data may be specified in advance, for example, to use as a constant, or it may be zero if the particular location is to be used to store the data later, during execution of the program.

The program, shown in Figure 6.4, works as follows: the first four instructions simply input and store the two numbers. The fifth instruction, in mailbox 04, subtracts the first

FIGURE 6.4

LMC Program to Find Positive Difference of Two Numbers

```
00   IN          901
01   STO   10    310
02   IN          901
03   STO   11    311
04   SUB   10    210
05   BRP   08    808   test
06   LDA   10    510   negative; reverse order
07   SUB   11    211
08   OUT         902   print result and
09   COB         000         stop.
10   DAT   00    000   used for data
11   DAT   00    000         "
```

number from the second. Instruction 05 tests the result. If the result is positive, all that's left to do is print out the answer. So, the instruction can be used to branch to the printout instruction. If the answer is negative, the subtraction is performed in the other order. Then the result is output, and the Little Man takes his break. Note that if the COB instruction is omitted (as in forgotten—this is a very common error!), the Little Man will attempt to execute the data stored in locations 10 and 11. Please study the example until you understand how it works in every detail.

The nine instructions that make up the instruction set that we have presented are sufficient to perform the steps of any computer program, although not necessarily in the most efficient way. It is important for you to realize that, although simplified, the Little Man instruction set is very similar to the instruction sets that appear in most real computers. In real computers, as in the Little Man Computer, most instruction steps are involved with the movement of data between the equivalent of mailbox locations and calculators, with very simple calculations, and with program branching.

The real computer differs mostly in the variations to these instructions that are provided, and with the addition of a few instructions that provide programming convenience, particularly multiplication and division instructions, and also instructions that shift the data in a word left or right. (Note that the traditional method of performing multiplication can be done in the computer using SHIFT and ADD instructions.)

We will discuss many of these variations when we look at the instruction sets in some real computers, in Chapters 7, 8, 11, and Supplementary Chapters 2 and 3.

6.5 THE INSTRUCTION CYCLE

We will refer to the steps that the Little Man takes to perform an instruction as the **instruction cycle**. This cycle, which is similar for all the instructions, can be broken into two parts:

1. The *fetch* portion of the cycle, in which the Little Man finds out what instruction he is to execute, and
2. The *execute* portion of the cycle, in which he actually performs the work specified in the instruction.

The fetch portion of the cycle is identical for every instruction. The Little Man walks to the location counter and reads its value. He then goes to the mailbox with the address that corresponds to that value and reads the three-digit number stored there. That three-digit number is the instruction to be performed. This is depicted in the drawings of Figure 6.5a.

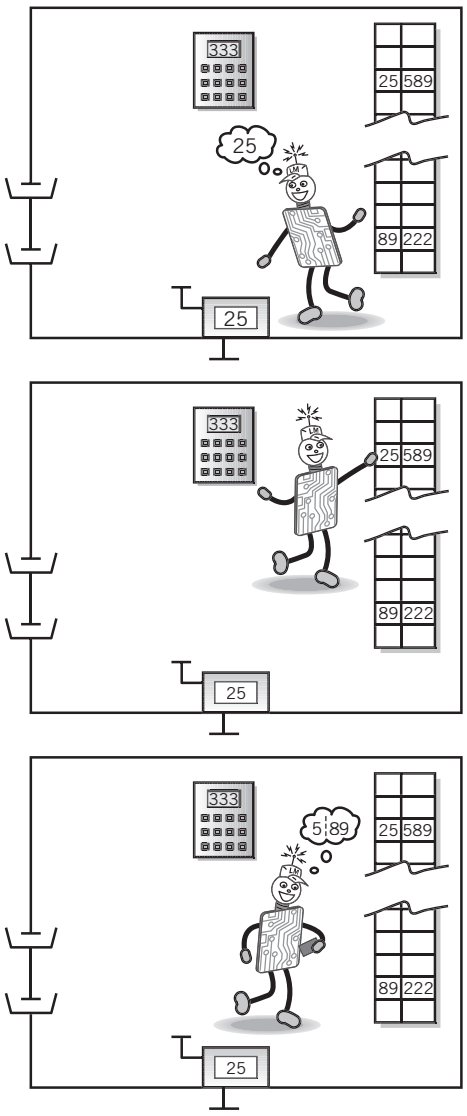
The fetch portion of the cycle has to occur first: until the Little Man has performed the fetch operation, he does not even know what instruction he will be executing!

The execute portion of each instruction is, of course, different for each instruction. But even here, there are many similarities. The first six instructions all require the Little Man to move data from one place in the mailroom to another. The first four instructions all involve the use of a second mailbox location for the data.

The LOAD instruction is typical. First, the Little Man fetches the instruction. To perform the execute phase of the LOAD instruction, the Little Man first looks at the mailbox with the address that is contained in the instruction. He reads the three-digit number on the slip

FIGURE 6.5(a)

The Fetch Portion of the Instruction Cycle

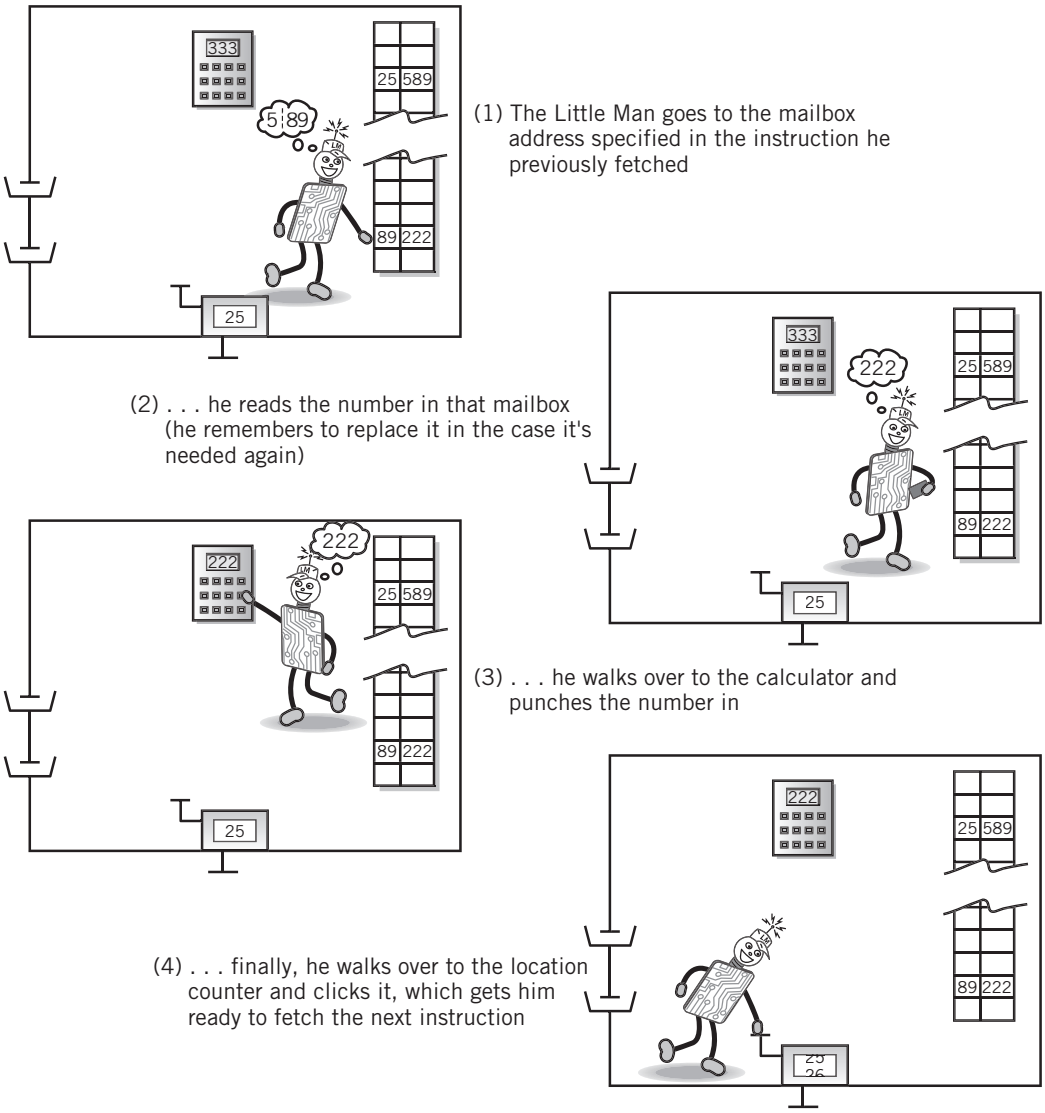


of paper in that mailbox and returns the slip of paper to its place. Then he walks over to the calculator and punches the number into the calculator. Finally, he walks over to the location counter and increments it. He has completed one instruction cycle and is ready to begin the next. These steps are shown in Figure 6.5b.

With the exception of the step in which the Little Man increments the location counter, the steps must be performed in the exact sequence shown. (The location counter can be incremented anytime after the fetch has occurred.) The fetch steps must occur before the

FIGURE 6.5(b)

The Execute Portion of the Instruction Cycle (LOAD Instruction)



execution steps; within the fetch, the Little Man must look at the location counter before he can pull the instruction from its mailbox.

Just as the sequence of instructions in a program is important—and you know that this is true for any language, Pascal, Little Man, or any other—so it is also true that the steps within each instruction must be performed in a particular order.

Notice that the ADD and SUBTRACT instructions are almost identical to the LOAD instruction. The only difference occurs during the execute step, when the Little Man enters

the number into the calculator. In the case of the arithmetic instructions, the Little Man adds or subtracts the number that he is carrying into the calculator, rather than simply entering it.

The other instructions are slightly different, although not any more difficult to trace through and understand. To improve your understanding, you should trace the steps of the Little Man through the remaining six instructions.

6.6 A NOTE REGARDING COMPUTER ARCHITECTURES

As we noted in Chapter 1, John von Neumann is usually considered to be the developer of the computer as we know it today. Between 1945 and 1951 von Neumann set down a series of guidelines that came to be known as the **von Neumann architecture** for computers. Although other experimental computer architectures have been developed and built, the von Neumann architecture continues to be the standard architecture for computers; no other architecture has had any commercial success to date. It is significant that, in a field where technological change occurs almost overnight, the architecture of computers is virtually unchanged since 1951.

The major guidelines that define a von Neumann architecture include:

- Memory holds both programs and data; this is known as the **stored program concept**. The stored program concept allows programs to be changed easily.
- Memory is addressed **linearly**; that is, there is a single sequential numeric address for each and every memory location.
- Memory is addressed by the location number without regard to the data contained within.

Instructions are executed sequentially unless an instruction or an outside event (such as the user resetting the location counter) causes a branch to occur.

In addition, von Neumann defined the functional organization of the computer to be made up of a control unit that executes instructions, an arithmetic/logic unit that performs arithmetic and logical calculations, and memory. The control unit and arithmetic/logic unit together make up the CPU, or central processing unit.

If you check over the guidelines just given, you will observe that the Little Man Computer is an example of a von Neumann architecture. In fact, we took care to point out features of the von Neumann architecture during our discussion of the Little Man Computer.

SUMMARY AND REVIEW

The workings of the computer can be simulated by a simple model. The Little Man Computer model consists of a Little Man in a mailroom with mailboxes, a calculator, and a counter. Input and output baskets provide communication to the outside world. The Little Man Computer meets all the qualifications of a von Neumann computer architecture.

The Little Man performs work by following simple instructions, which are described by three-digit numbers. The first digit specifies an operation. The last two digits are used for various purposes, but most commonly to point to an address. The instructions provide operations that can move data between the mail slots and the calculator, move data between the calculator and the input and output baskets, perform addition and subtraction, and allow the Little Man to stop working. There are also instructions that cause the Little Man to change the order in which instructions are executed, either unconditionally or based on the value in the calculator.

Both data and instructions are stored in individual mail slots. There is no differentiation between the two except in the context of the particular operation taking place. The Little Man normally executes instructions sequentially from the mail slots except when he encounters a branching instruction. In that case he notes the value in the calculator, if required, and resumes executing instructions from the appropriate location.

The exact steps performed by the Little Man are important because they reflect closely the steps performed in a real CPU in executing an instruction.

KEY CONCEPTS AND TERMS

instruction cycle
linear memory addressing
Little Man Computer (LMC)
mnemonics
op code
stored program concept
von Neumann architecture

READING REVIEW QUESTIONS

6.1 Without looking at the book, draw a Little Man Computer. Label each of the components in your drawing.

6.2 Instructions in the Little Man Computer are three digits, divided into two parts. Show the format of an LMC instruction.

6.3 Describe, step by step, what the Little Man does to execute a STORE instruction.

6.4 Describe, step by step, what the Little Man does to execute an INPUT instruction.

6.5 Extend the simple program shown in Section 6.3 to accept *three* inputs from a user, add them, and output the result.

6.6 If a user wants to enter two numbers, what must the Little Man program do before she enters the second number? Why?

6.7 Write a Little Man program that accepts two numbers as input and outputs the numbers in reverse order.

6.8 Write a Little Man program that accepts two numbers as input, subtracts the first from the second and outputs the result.

6.9 Explain carefully what the Little Man will do when he executes a JUMP instruction.

6.10 Explain carefully, step by step, what the Little Man will do when he executes a BRANCH ON ZERO instruction.

6.11 Why is the instruction cycle called a *cycle*?

6.12 Even if he runs out of instructions to execute, the Little Man only stops trying to execute instructions under one condition. What is that condition? What happens if the Little Man runs out of instructions and that condition is not met?

6.13 The instruction cycle is divided into two phases. Name each phase. The first phase is the same for every instruction. What is the purpose of the first phase that makes this true? Explain what the Little Man does during the first phase.

6.14 What does the Little Man do during the second phase of a COFFEE BREAK or HALT instruction?

EXERCISES

6.1 The steps that the Little Man performs are closely related to the way in which the CPU actually executes instructions. Draw a flow chart that carefully describes the steps that the Little Man follows to execute a branch instruction.

6.2 Repeat Exercise 6.1 for a subtract instruction.

6.3 Repeat Exercise 6.1 for a branch on positive instruction.

6.4 What are the criteria that define a von Neumann architecture? How does the example in this chapter in which we enter and add two numbers illustrate each of the criteria?

6.5 Consider the example in this chapter in which we enter and add two numbers. Suppose we had stored the first input entry in mailbox location 00. Would the program have produced the same result? What would have happened if the program were executed a second time? What characteristic of the computer makes this true?

6.6 Write a Little Man program that accepts three values as input and produces the largest of the three as output.

6.7 Write a Little Man program to accept an indefinite number of input values. The output value will be the largest of the input values. You should use the value 0 as a flag to indicate the end of input.

6.8 Write a Little Man program that accepts three values as input and outputs them in order of size, largest to smallest. (This is a more challenging variation on Exercise 6.6.)

6.9 Write a Little Man program that adds a column of input values and produces the sum as output. The first input value will contain the number of values that follow as input to be added.

6.10 Write a Little Man program that prints out the odd numbers from 1 to 99. No input is required.

6.11 Write a Little Man program that prints out the sums of the odd values from 1 to 39. The output will consist of 1, $1+3$, $1+3+5$, $1+3+5+7\ldots$. No input is required.

As an aside, do you notice anything interesting about the output results that are produced by this series? (Hint: This series is sometimes used as part of an algorithm for finding square roots of numbers.)

6.12 The following Little Man program is supposed to add two input numbers, subtract a third input number from the sum, and output the result, i.e.,

```
OUT = IN1 + IN2 -- IN3
```

mailbox	mnemonic code	numeric code
00	IN	901
01	STO 99	399
02	IN	901
03	ADD 99	199
04	STO 99	399
05	IN	901
06	SUB 99	299
07	OUT	902
08	COB	000

What is wrong with this program? Modify the program so that it produces the correct result.

6.13 Suppose we have a need to handle both negative and positive data beyond the simple test in the various conditional branch instructions. One way to do this would be to replace the subtract instruction with a 10's complement instruction. The COMP instruction complements the value in the calculator and leaves the value in the calculator.

a. How would subtraction be performed in this case?

b. Carefully trace the steps that the Little Man would perform to execute the new COMP instruction.

c. What is the new range of values possible with this modification, and how are these values represented in the Little Man Computer?

d. What would the Little Man do to execute a BRANCH ON POSITIVE instruction?

6.14 The programs that we have discussed in this chapter seem to have appeared in the mailboxes by magic. Consider a more realistic alternative:

Suppose a small program is permanently stored in the last few mailbox locations. A BRANCH instruction at location 0, also permanent, will start this program. This program will accept input values and will store them at consecutive mailbox locations, starting with mailbox 001. You may assume that these values represent the instructions and data of a user's program to be executed. When a 999 is received as input data, the program jumps to location 001 where it will proceed to execute the values just entered.

The small program described here is known as a *program loader*, or, under certain circumstances as a *bootstrap*. Write a Little Man program loader. (Hint: It may be useful to remember that instructions and data are indistinguishable. Thus, instructions could be treated as if they were data, if necessary.)

6.15 Show carefully how you would implement an IF-ELSE statement using Little Man instructions.

6.16 Show how you would implement a DO-WHILE statement using Little Man instructions.

6.17 The input data values in our problems have always been entered in the order that they were to be used. This is not always possible or convenient. Can you think of a simple way to accept input data in the wrong order and still use it correctly?

6.18 Suppose the Little Man Computer had been implemented as a 16-bit binary machine. Assume that the binary LMC provides the same instruction set, with the same op codes (in binary, of course), and the same instruction format (op code followed by address). How many bits would be required for the op code portion of the instruction? How many mailboxes could the binary machine accommodate? What is the range of 2's complement data that this machine could handle?

6.19 The original version of the Little Man Computer used op code 7 (i.e., instruction 700) for a COFFEE BREAK instruction instead of op code 0. What is the advantage of using 000 for the COB instruction instead of 700? (Hint: Consider what happens if the programmer forgets to put a COB instruction at the end of a program.)

6.20 When we discussed conditional branching we claimed that a BRANCH NEGATIVE instruction is not necessary. Show a sequence of BRANCH instructions that will cause a program to branch to location 50 if the value in the calculator is negative.

6.21 Show a sequence of instructions that will cause a program to branch to location 75 if the value in the calculator is *greater than* zero.