C-800

digital SMALL COMPUTER HANDBOOK 1967

digital

SMALL COMPUTER HANDBOOK

DIGITAL EQUIPMENT CORPORATION

THE

digital

SMALL COMPUTER HANDBOOK
1967-68 EDITION

PART I: SMALL COMPUTER PRIMER

PART II: APPLICATION PROGRAM EXAMPLES

PART III: PDP-8/I USERS HANDBOOK

PART IV: PRODUCT CATALOG

FOREWORD

The computer revolution is here in the sciences and engineering. No discipline will remain the same. Computers open up too many new ways of knowing and doing. Computers offer too much power for control and analysis.

Small general purpose computers have become an important part of this revolution. They give the scientist and engineer a way to have computer power under their direct control; they are personal approachable tools.

Research scientists use small computers to automatically control experiments and to collect and condense data. The small computer allows the researcher to use emerging data to creatively vary experiment sequences and parameters, and of course, to perform mathematical operations. Where massive computation is required, small computers are easily connected to the facilities of large scale computer systems.

Small computers are also used by the engineer as the control element in large data handling or process control systems and instruments. In these applications, small computers not only have cost advantages as compared with specially designed, fixed-purpose, sub-systems, they have almost unlimited flexibility and they can be readily expanded if the system has to grow.

The computer described in this handbook, the PDP-8/I, is a new and important member of this computer revolution. It is the newest in a family of compatible machines designed with the same word length, instructions and programs. Its predecessor, the PDP-8, the most popular computer ever made for the scientific community, is presently in use in over 1000 installations. The little brother to the PDP-8, the PDP-8/S, is active in over 500 installations. Both machines were designed to offer potential computer users the best price/performance ratio in the industry. The new PDP-8/I is designed to carry on this tradition.

There is an investment to be made by the scientist and engineer. If they are to get the most out of their machine, they have to learn how to use it. The study of the computer fundamentals outlined on the following pages is a good start. The payoff for this learning investment is high. Dividends come in terms of an entire career; the computer discipline is added to the discipline of science or engineering.

Digital's PDP-8 computer, shown here undergoing final production testing, is one of the most popular on-line computers for physics and biomedical analysis and process control.

FLIP CHIP assembly line combines automated manufacturing steps with computer controlled checkout for lower cost, more reliable circuits.

TABLE OF CONTENTS

PART I: SMALL COMPUTER PRIMER

INTRODUCTION

Robert Benchley once wrote that although he realized that a modern bridge was indeed a very large and complex structure, he nevertheless felt quite sure he could build one himself, if only he could think of the very first thing that had to be done. Once over this initial hurdle, he thought, the remaining steps would fall readily into place one after the other until the bridge was complete. It was just his inability to discover that first step that prevented him from becoming a master bridge builder!

You, perhaps, have been trying to discover the first step to be taken in building a knowledge of just what a high-speed digital computer system is, how it operates, and in what ways it might be useful to you in your work. You have heard that the computer can be an extraordinarily powerful tool, that it can somehow remember an enormous number of facts, make decisions automatically, and solve arithmetic problems at incredible speeds, doing in a matter of seconds an amount of work it would take you days or weeks to accomplish by hand. Perhaps you have a large, difficult, or complex experimental problem that is already beyond your scope without the aid of a computer. Or, you are trying to find a way to capture and analyze an elusive signal, the slight variation of the salinity of the ocean, the response of the human brain to a flash of light, or a faint and fuzzy image on a photographic plate. It has become important to you to know how to make use of the power and versatility that have characterized the computer's astonishingly rapid development in recent years.

That the modern computer is vastly more complex than Benchley's bridge should not discourage you, for it is by no means necessary to learn all there is to know about the computer before it can be put to work. Indeed, its basic operating principles are really very simple, few in number, and can be learned easily. Of course, the greater the extent and depth of your knowledge of the computer and its application, the greater will be the computer's usefulness to you as a tool. Starting with basic principles as they are applied in simple situations, you will be able to extend your knowledge gradually to include more and more complex ones. If only you could discover that first step. . .

It turns out that the first step is surprisingly simple. It involves no more than a consideration of the kinds of on-off or up-down patterns you might find represented by a row of simple switches. For the digital computer, with all its power and versatility, is basically little more than an automatic device for manipulating just such patterns at high speed according to simple fixed rules. The computer converts signals from its environment into switch patterns; it changes one set of patterns into another in arithmetically useful ways; it stores patterns away for further use, interprets them, combines them, translates them into characters on a printed page, or into pictures, or sounds, or mechanical movements. Master the motion of switch patterns, and you have made an important start in understanding how the digit computer works. Not that you must understand how the computer works in detail to use it to solve your problem. Not at all! Your *main concern* is going to be rather the careful formulation of your problem in symbolic terms using pencil and paper. Oh, occasionally you may want to flip a switch or two to modify or control the behavior of the computer, and there may be times when interpreting switch patterns as they are displayed in rows of indicator lights on a console will be helpful. But by and large you will be content simply to "feed" the computer your problem and its data in symbolic or electrical form, and have the results returned to you with as little fuss as possible on some kind of printing, plotting, or display device. You won't care how much pattern manipulation the

computer carries out in the process. The fact remains, however, that switch patterns and their manipulation underlie and define everything that the computer is capable of doing, and it is therefore a good idea to know something about them.

PATTERNS IN SWITCHES

Consider a row of three switches as shown in figure 1. Think of each switch as being in either an UP or DOWN position. For the present, suspend your interest in which position corresponds to on and which to off and describe the pattern of switch settings by means of the letters U and D strung together in the right way. Thus, you can write down the pattern represented in figure 1 as U D U.

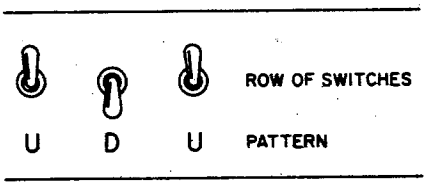


Figure 1 A Pattern of Three Switches

Now, you might ask some simple questions about these patterns. For example, how many different patterns can be represented with three such switches? With four switches? With no switches?

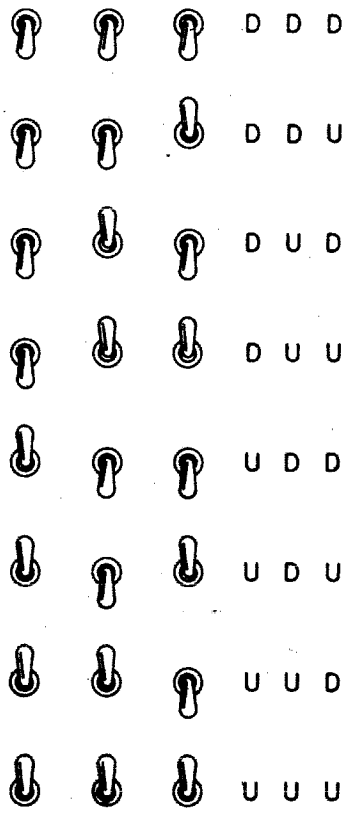
It is clear that with three switches you can represent twice as many as you can with two switches, and with two switches, twice as many as with one switch. One switch provides for the two patterns U and D; two switches, therefore, provide for $2 \times 2 = 4$ patterns; and three switches, for $2 \times 2 \times 2 = 8$ patterns. Adding a fourth switch doubles the number of possible patterns again, giving $2 \times 2 \times 2 \times 2 = 16$ patterns, and so on. Clearly, the rule is simply that the number of patterns which can be represented by n switches is 2^n.

Is there a systematic way of writing down all of the 2^n patterns possible with n switches? Yes, there are many systematic methods, the most important of which can be summarized by means of the following step-by-step procedure stated in the all too familiar style of an income tax return:

LINE	STEP
1	PUT ALL n SWITCHES DOWN
2	RECORD THE PATTERN
3	IF ALL n SWITCHES ARE UP, THEN STOP, IF NOT, GO ON TO LINE 4
4	NOTE THE SETTING OF THE RIGHTMOST SWITCH
5	IF THIS SWITCH IS DOWN, PUT IT UP AND GO BACK TO LINE 2. IF THIS SWITCH IS UP, PUT IT DOWN AND GO ON TO LINE 6
6	NOTE THE SETTING OF THE NEXT SWITCH TO THE LEFT AND GO BACK TO LINE 5

Figure 2 A Procedure for Systematically Producing All 2^n Patterns Represented by n Switches

Try this step-by-step procedure. Imagine a row of three switches and mentally carry out the specified actions. You will find that the sequence of settings and the corresponding patterns will turn out to be:



Note that if step 3 is modified slightly to read, "If all n switches are UP, go back to line 1, if not, go on to line 4," the set of patterns will be repeated endlessly. Each pattern then specifies its successor, with pattern DDD being the successor to pattern UUU. You might, in fact, think of the whole procedure as one for finding successors according to a simple fixed rule. Start at line 3 with any pattern whatever, and the procedure will give you the successor to that pattern.

FLOW DIAGRAMS

No doubt you have had at one time or another, some difficulty in following the line-by-line tabular instructions on your income tax form. You will be pleased to know, therefore, that there is another way to represent systematic procedures, a way which emphasizes the flow from step to step. Such a representation is called a flow chart or flow diagram. You are going to find this flow diagram technique extremely useful when you are ready to formulate your problem in symbolic terms for the computer, and you will be making frequent use of it. Let's recast the table of figure 2 in flow diagram form so that you can see just what a flow diagram looks like:

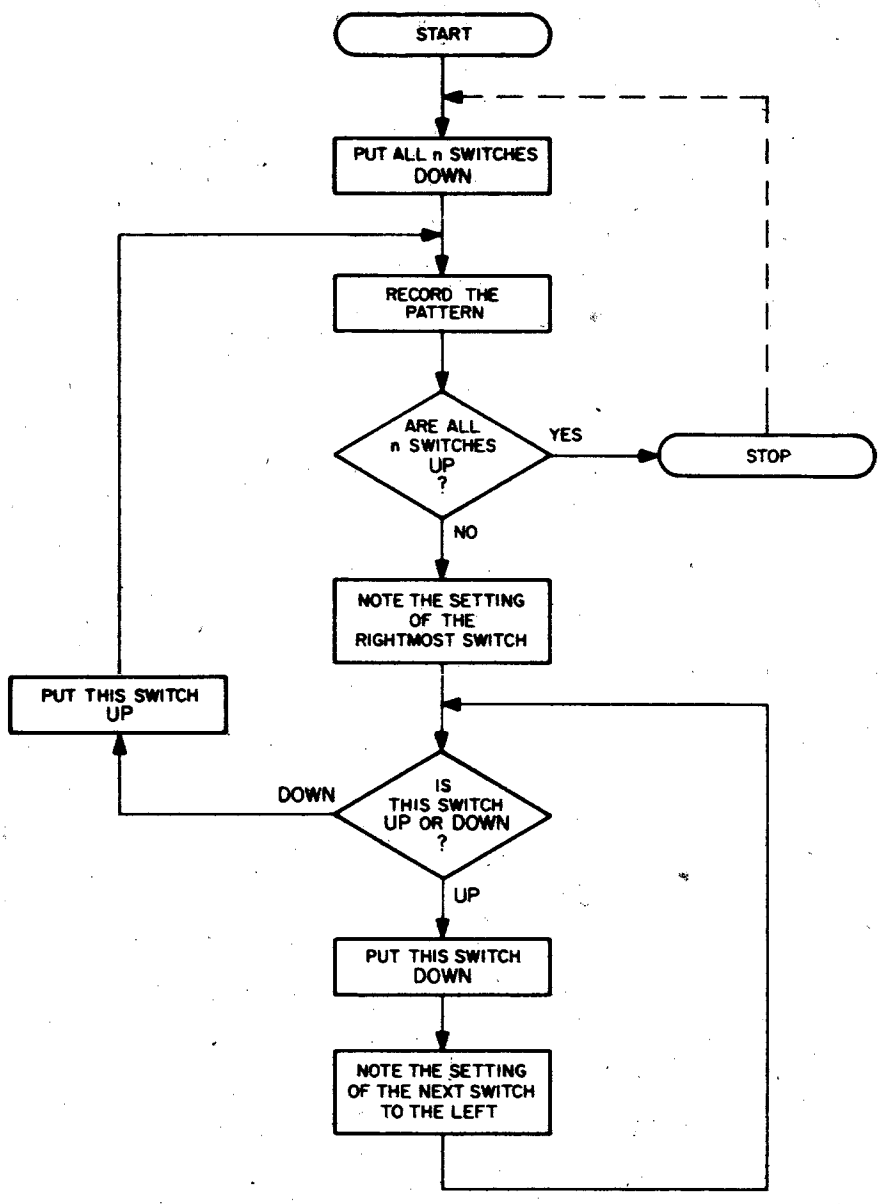


Figure 3 Flow Diagram of the Procedure for Systematically Producing All 2^n Patterns Represented by n Switches

The dotted line indicates the modification of the procedure which, if the STOP is omitted, provides for continuous pattern sequencing.

Although not as compact as the tabular form of figure 2, the flow diagram is considerably more graphic and is therefore easier to comprehend, especially in the case of procedures involving many alternative steps. The shapes of the boxes are not of primary importance, of course; the ones shown, however, are convenient and more or less standard.

BINARY COUNTING

Counting is one of the most basic operations of arithmetic. The procedure of figure 2 or figure 3 is one which generates patterns by a process of counting in a two-valued system. If you substitute the digit "0" for "D" and "1" for "U," the resulting patterns take the form of binary numbers. Unlike a decimal number, which uses the ten digits 0 through 9, a binary number uses only the two bits 0 and 1. Figure 4 shows the first four binary numbers and their decimal equivalents generated by our counting procedure:

Count in Decimal	Count in Binary
0	00
1	01
2	10
3	11

Figure 4 The First Four Binary Numbers and Their Decimal Equivalents

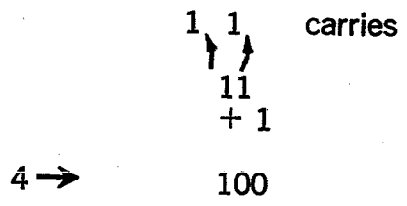
To keep the notation straight, it will be a good idea to use the subscripts 2 and 10 in any statements of equivalence that you might want to write: $00_2 = 0_{10}$; $01_2 = 1_{10}$; $10_2 = 2_{10}$; $11_2 = 3_{10}$.

The generating procedure we have chosen is one in which binary patterns are "counted out" in a way similar to the purely symbolic one you learned in grade school for decimal numbers. You learned to say "ZERO plus ONE equals ONE; ONE plus ONE equals TWO; etc.; NINE plus ONE equals ZERO with ONE to carry." In binary terms you would say "ZERO plus ONE equals ONE; ONE plus ONE equals ZERO with ONE to carry." Much simpler. The above sequence of four binary numbers was generated in just that way by our procedure. The counting went like this:

```
0 →      00
         00
       + 1
       ----
1 →      01

           1   carry
         01
       + 1
       + 1
       ----
2 →      10
         10
       + 1
       ----
3 →      11
```

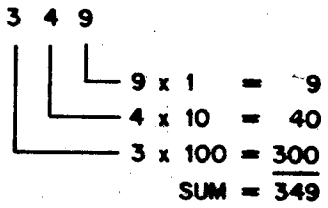
You can see that if there were more than two switches, the next number generated would be



and so on, the sequence continuing with 101, 110, 111, 1000, etc.
Symbolic procedures of the sort we have been using are known as algorithms (after the ninth century Arabian arithmetician al-Kuwarizmi, to whom the credit is due for this method of calculating by means of symbol manipulation, a vast improvement over the method of counting pebbles). Thus you would speak of the procedure of figure 3 as the binary counting algorithm, or of the familiar grade school decimal counting algorithm, and so forth. All of the fixed rules followed by a digital computer as it goes about solving a problem are, in fact, algorithms of one kind or another.

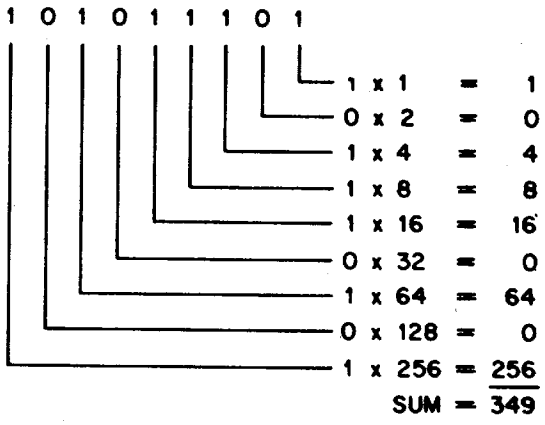
What is the meaning of a binary number, anyway? What is the quantity of pebbles represented by a given binary number? The answer is that each bit within the string of bits stands for a different portion of the whole quantity, depending upon its position in the string, just as in decimal notation.

Recall that in the decimal system, a number such as 349, for example, has the meaning "nine ones + four tens + three hundreds." The values one, ten, hundred, and so forth are the consecutive powers of ten; that is, 10^0, 10^1, 10^2, etc.



e binary system, a number such as 10110 has the meaning (again read-
)m the right) "zero ones + one two + one four + zero eights + one six-
or twenty-two. The values one, two, four, eight, sixteen, and so forth are
nsecutive powers of two; that is, 2^0, 2^1, 2^2, 2^3, 2^4, etc. The binary digits
1 are called bits (for binary digits), and just as we say that 349 is a
decimal number, we say that 10110 is a 5-bit binary number. Repre-
349 pebbles requires a 9-bit binary number:

$1\,0\,1\,0\,1\,1\,1\,0\,1_2 = 349_{10}$



BINARY ADDITION

You may be wondering how addition of two binary numbers is accomplished directly in binary. Very simple, really. The addition algorithm is just an extension of the counting algorithm. The bit strings of the two numbers, the ADDEND and the AUGEND, are lined up one over the other, and for each column, the rule "ZERO plus ONE equals ONE; ONE plus ONE equals ZERO with ONE to carry" is applied, starting from the rightmost, or least-significant bit column. You might try the following example for 12-bit numbers yourself.

Carries		1	11	11	
ADDEND	000	010	101	101	(173_{10})
AUGEND	011	010	001	011	(+ 1675_{10})
SUM	011	100	111	000	(= 1848_{10})

In the fourth column from the right, you will notice that three ONEs had to be added together. Of course, you simply add two of them together first, getting "ZERO with ONE to carry," and then add the third ONE to this, getting finally "ONE with ONE to carry." If the sum is also supposed to be a 12-bit number, any carry produced in the leftmost column, the end-carry, can simply be discarded.

But discarding the end-carry means, doesn't it, that the sum is incorrect? Yes, it does in a way, but it also makes possible a simple representation of negative numbers within certain limits. For if the sum turns out to be zero, i.e., 000 000 000 000, either the ADDEND and AUGEND are both zero, or, more to the point, the ADDEND and AUGEND can be thought of as having values which are equal in magnitude but opposite in sign since these are the only kinds of values which add up to zero. For example, in the addition

end carry

(1)	111	111	111	11	carry
	111	111	111	111	ADDEND
	000	000	000	001	AUGEND
	000	000	000	000	SUM (rightmost 12 bits)

you recognize the ADDEND as having the value "1" and it must be, therefore, that the AUGEND has the value "−1."

We say that the numbers 111 111 111 111 and 000 000 000 001 are 2's complements of one another; addition in which the end carry is discarded is called 2's complement addition.

We can arrange the set of 12-bit numbers in a way which shows all of the 2's complement pairs, and assign corresponding decimal values and their signs as in figure 5.

PAIRING	BINARY (TWO'S COMPLEMENT)	SIGNED DECIMAL
	011 111 111 111	+2047
	⋮	⋮
	000 000 000 100	+4
	000 000 000 011	+3
	000 000 000 010	+2
	000 000 000 001	+1
	000 000 000 000	0
	111 111 111 111	−1
	111 111 111 110	−2
	111 111 111 101	−3
	111 111 111 100	−4
	⋮	⋮
	100 000 000 001	−2047

Figure 5 Two's Complement 12-Bit Binary Numbers and Their Signed Decimal Number Equivalents

The leftmost bit, you will notice, is ZERO for all the positive numbers and ONE for all the negative numbers. It is called, therefore, the *sign bit* of the number. Actually there is one more "negative" number not shown in the table. It is $100\ 000\ 000\ 000_2 = 2048_{10}$ — a bonus or a nuisance depending on your point of view.

You may have perceived that the way to negate either a positive or negative number is to change all its bits from ONE to ZERO and vice versa (a process called complementing) and then add 1 to the result.
The whole business is getting to be a bit tedious, isn't it? Bear in mind, though, that the computer is going to take care of practically all of these binary details for you automatically, and that you can work with the decimal number system directly if that is what you wish to do. Conversion between the two systems can be achieved by means of suitable algorithms, and at this point perhaps, that's all you want to know about that.

WHY BINARY?

You can see that binary numbers are much less compact than decimal numbers. They are, however, quite the right sort of numbers to represent in two-state devices as two-position switches. The reason that the binary rather than the more compact decimal system is used in electronic digital computers is primarily that two-state devices have been found to be technically and economically sounder than ten-state devices in computer construction.

Two-state devices of various kinds are used extensively. The electronic version of the two-position switch is picturesquely known as a flip-flop. The computer is able to put a flip-flop into either of its two states by means of electrical signals, just as you are able to put a switch into either of *its* two states by hand. Computer memory units are composed of a large number of tiny, magnetizable doughnuts misnamed cores, wired together in geometrical arrays and arranged so that each core can be magnetized in one of two ways. Binary numbers can also be stored in the form of two-state magnetized spots on tapes, discs, or drums coated with magnetic material, or in the form of holes punched in paper tape or cards (the two states being "hole" and "no hole"). A set of n two-state, i.e., binary, devices used in the representation of an n-bit number is called an n-bit register. Thus, a row of twelve two-position switches is called a 12-bit switch register, and a set of twelve flip-flops, a 12-bit flip-flop register. A 12-bit register can represent or "hold" 2^{12}, or 4096, different binary numbers.

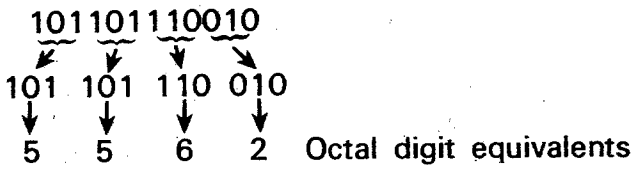
OCTAL REPRESENTATION

Even though you may be talking about or writing down a 12-bit binary number very rarely, it is still quite a nuisance to have to deal with binary digit strings like 101101110010 or 001100101101. You will be relieved to learn that there is a simple shorthand that can be used instead. It involves yet another system, the octal number system, in which numbers are based on powers of eight, and the digits 0 through 7 are used. But do not despair, for it is the notational rather than the arithmetic aspects of the octal system that will concern you. To use it as a shorthand notation for a 12-bit binary number, for example, you simply divide the twelve bits into four groups of three bits and assign in place of each group its octal equivalent taken from the 3-bit table of figure 6, which you can surely memorize:

OCTAL	BINARY
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

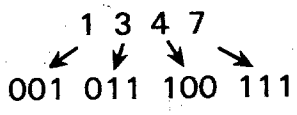
Figure 6 Octal-Binary Equivalents

For example, if you find it necessary to deal with the number 101101110010, you can subdue it as indicated below:



$$101101110010_2 = 5562_8$$

It's easy to see that the process can be reversed. Given an octal number such as 1347_8, you use figure 5 to find the 3-bit binary equivalents of each octal digit and string these together.



$$1347_8 = 001011100111_2$$

Handy. Almost as compact as decimal, furthermore.

STORAGE AND RETRIEVAL OF BINARY PATTERNS

By now you have the idea that a register can be thought of as "holding" a number. A given 12-bit register can hold only one 12-bit binary number at a time, but any of the 4096 possible 12-bit numbers can be accommodated. A number can be copied or transferred electronically into any flip-flop register large enough to accommodate it, and a computer, in fact, makes many such transfers in the course of a calculation. To accomplish a number transfer, the state of each flip-flop or switch within the source register is represented as an electrical signal which is then transmitted to the destination register where it sets a corresponding flip-flop into a matching state. If all n bits are copied simultaneously, the transfer is said to be a parallel one; if the bits are copied one at a time, a serial one.

One of the most important functions of a computer is the storage and retrieval of information—facts, dates, measurements, names, messages, instructions, data—all representable as binary patterns of one kind or another. Before the computer can solve your problem, you must supply it with all the information relevant to the problem, the procedures for solution as well as data. To do its job the computer must have rapid access to many, many numbers, far too many to hold entirely in flip-flop registers, which are relatively large and elaborate devices. Instead, a special storage unit, the core memory, is used. In a core memory, the two-state elements are the magnetic cores mentioned earlier, rather than flip-flops. These cores are arranged to function as sets of identical n-bit registers, into and out of which parallel n-bit transfers can be made one register at a time.

The registers within a core memory have identifying numbers associated with them, much the way houses on a street have addresses. These numbers, in fact, are called addresses, and you can speak of "storing a number at a cer-

tain address" in the memory. Addresses are also referred to as "patterns" or "words."

This process is so fundamental that you will want to see a specific example, perhaps one in which the address and pattern are copied from switch registers even though in a typical computer situation other kinds of number-input devices will be used. A simple 64-word 12-bit memory system and a 6-bit address register are shown in figure 7.

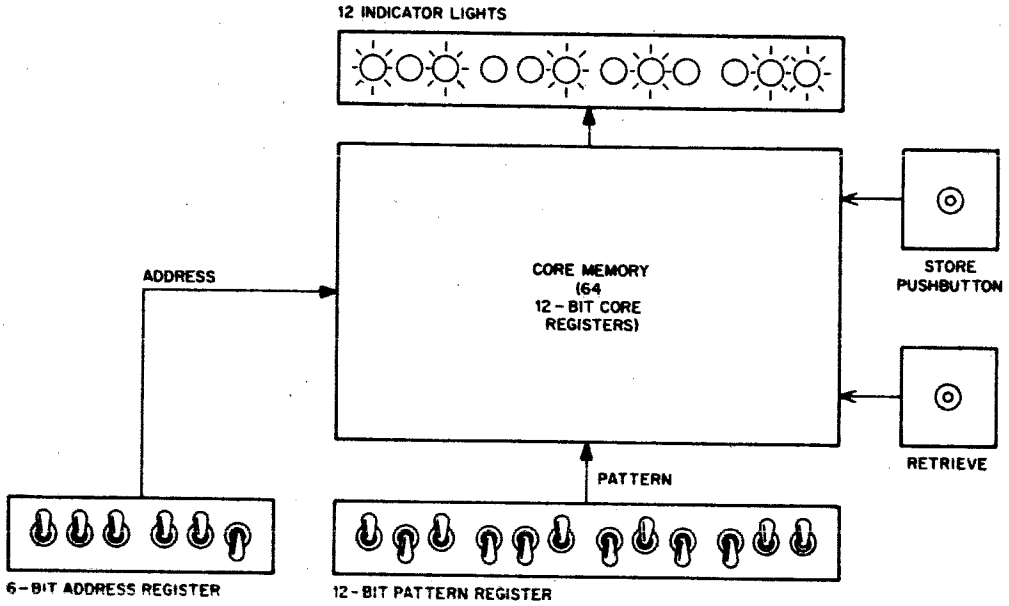


Figure 7 A Simple 64-Word 12-Bit Memory System

You will notice that because there are 64 addresses, the binary address register must have 6 switches: $2^6 = 64_{10}$. To store patterns or "words" in this simple memory, you set the switches of the address register to represent the binary form of the address designating the core register you want to use, set the switches of the pattern register to hold the pattern you want to store, and push the STORE push button. You can repeat this process with other addresses and patterns, if you wish, until all 64 registers are "full." The registers may be used in any order whatever, sequentially, if you wish, or completely at random. The core memory system is said to provide random access to its registers.

To retrieve a pattern you have stored, you again set the address switches to the address of the register holding the pattern and then push the RETRIEVE push button. The pattern held in the designated core register is copied into a special 12-bit flip-flop register, which is connected to a set of twelve lights for you to look at. Lights which are on correspond to ONEs in the binary pattern. Simple? Logically yes, but electronically, no. The process of tuning electrical signals into magnetic states and vice versa is one requiring many electronic devices and circuits.

So, in a memory system like the one above you can store a set of quite arbitrarily chosen patterns. In figure 7, the switches are set to store the number 5123_8 in the register whose address is 76_8. You can summarize the state of affairs at any point by listing the patterns and their address locations in octal

notation (using the table of figure 6 which you have, of course, memorized). You might have something like this:

LOCATION (OCTAL)	CONTENT (OCTAL)
00	4321
01	0000
02	7777
03	5123
04	0510
05	7401
06	2111
07	3043
10	6571
11	1234
⋮	⋮
76	5123
77	1234

Figure 8 The Contents of a 64-Word Memory

Register 07_8, for example, holds the number 3043_8 and the "next" register, 10_8, holds the number 6571_8. Of course, if you are actually setting switches or looking at lights, you can readily convert these octal numbers back to binary if you wish:

$07_8 = 000\ \ 111_2$ $\qquad$ $3043_8 = 011\ \ 000\ \ 100\ \ 011_2$

$10_8 = 001\ \ 000_2$ $\qquad$ $6571_8 = 110\ \ 101\ \ 111\ \ 001_2$

By the way, notice the disconcerting skip from the number 7 to the number 10 in an octal counting sequence. This will probably always bother you, but nothing can be done about it. The same thing happens at 17, which is followed by 20, and 27, 37, 47, 57, and 67. The number following 77 is 100, and so it goes. Just skip any numbers which have an 8 or 9 in them, and you can count in octal if you really want to.

ORGANIZATION OF THE COMPUTER

"This is all very well," you can be heard to say, "but what does it have to do with 'capturing and analyzing an elusive signal,' for example?" A fair question, it must be admitted. The step from storing and retrieving binary numbers using switches set by hand to solving a complex problem is seemingly great, but be assured that you are on the right track! For it turns out that a complete digital computer can be built around just such a simple storage and retrieval scheme as the one outlined above. There will have to be more core registers and correspondingly larger addresses; some kind of unit capable of carrying out various arithmetic algorithms will have to be provided; for convenience, it will probably include at least a keyboard/printing device such as a teletypewriter.

The structure and operating principles of such a computer might be represented in terms of a simple diagram:

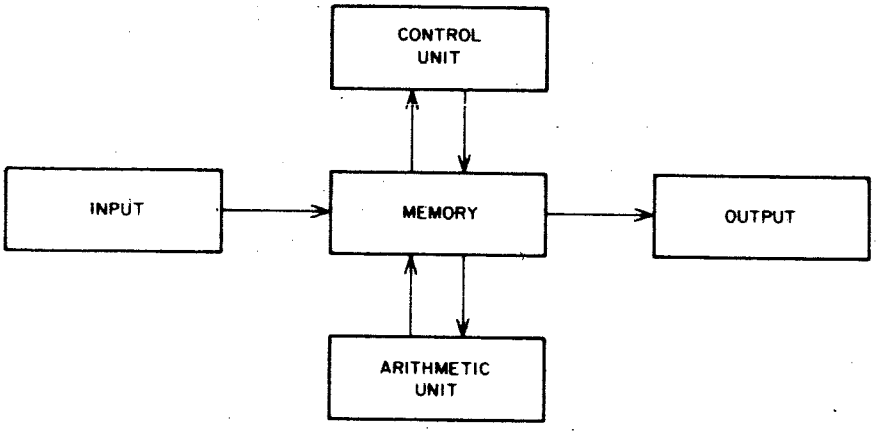


Figure 9 A Simple Model of the Computer

The input, memory, and output elements are similar in principle to the pattern switches, memory, and indicator lights of figure 7. Input and output, of course, can be and frequently are handled by a wide variety of devices much more sophisticated than switches and lights, such as display scopes, magnetic or paper tape units, analog/digital converters, and teletypewriters. No matter how complex such a device might be, however, it always transmits information into or out of a digital computer in binary patterns which could be set in switches or readin lights. (How slow that would be!)

Storing and retrieving binary patterns in a core memory certainly has little appeal unless you can also manipulate the patterns in some useful way, perhaps to do arithmetic calculations with binary data which is in the memory. The arithmetic unit not only accepts numbers previously stored in the memory, but is capable of performing various algorithmic operations on these numbers (i.e., adding two numbers together), and returning them eventually to the memory or to an output device. Since it must be able to manipulate numbers, as well as "hold" them, the arithmetic unit usually consists of specially designed flip-flop registers which are more powerful (and more expensive) than core registers.

If any part of a computer can be thought of as its "brain," it is the control unit. This unit coordinates all the parts of the computer so that events happen in a logical sequence and at the right time. To show all the pathways control signals might take to the other parts of even the simple computer model shown in figure 9 would quite obscure the drawing! This is the unit which "makes things happen."

This is neither as powerful nor as mysterious as it may sound, for the control unit only does exactly what you tell it to do. Numerically encoded instructions, which you store in the memory, can be sent to the control unit to direct it to carry out certain basic algorithmic steps. The control unit is designed to "decode" each number sent to it and to initiate a chain of events designated by that number. For example, the instruction code number 7001_8 might initiate a counting algorithm step in an arithmetic unit register. When one chain of events has been completed, the control unit is ready to receive another number from the memory and to initiate the chain of events designated by that number, and so on.

Relatively few algorithms or instructions which the control unit can interpret are actually built into the computer, but they include ones which make it pos-

sible, when combined in the proper sequence, to synthesize any algorithm whatever that could have been built in! This remarkable circumstance gives the computer enormous versatility. We speak of this kind of computer, one that stores its own instructions and provides for the general synthesis of algorithms, as a general purpose computer.

The set of instructions which the computer is directed to carry out in a specified sequence is called a program. It is the program that states the procedure the computer is to follow in solving the problem at hand. The program is thus a large problem soluation algorithm composed of many simpler algorithms, namely, the basic ones provided in the repertory of the computer. Your task in using the computer to solve a given problem is primarily one of writing an effective program based ultimately on the simple operations the computer "knows" how to do.

In principle, then, the steps you must take to use the computer to solve your problem are the following:

1) Program: The problem must be analyzed and an algorithm for its solution constructed in the form of a program of instructions which are in the repertory of the computer.
2) Input: These instructions must be encoded in binary form and stored in the memory together with any data and parameters required by the program.
3) Operation: The computer must be started at the first instruction; all the steps of the program are then automatically carried out, the computer alternately "fetching" instructions from its memory and "executing" them.
4) Output: The binary results must be retrieved from the memory.

You would find these four steps quite tedious if you had to carry them out in detail. Fortunately, in practice, there are many simplifying variations of these four steps, designed to make your task easier. It will be possible, for example, to express your problem not in the relatively simple basic instruction "language" of the computer, but rather in a more powerful language better suited to your needs, which will be translated into the basic language and encoded in the necessary binary form automatically. You will be able to incorporate into your work some of the many programs already written by others, taking advantage of an ever-increasing base of useful procedures. Devices will be available to capture that elusive signal in binary numerical form and store it away for you. Printers, plotters, and recorders of various kinds will simplify the retrieval and display of results in decimal or graphic form.

A SPECIFIC EXAMPLE

Someone once suggested that the ideal digital computer would be powerful enough to solve any problem in one second or less, would cost no more than ten dollars, and would be so compact that you could simply paint it onto any handy surface. Such a machine is not yet available. Instead, focus your attention briefly on the organization of a specific available digital computer as a preface to learning just what kinds of instructions can be used in writing programs.

Figure 10 shows a diagram of the PDP-8, a simple, high-speed, general purpose digital computer which operates on 12-bit binary numbers in parallel fashion.

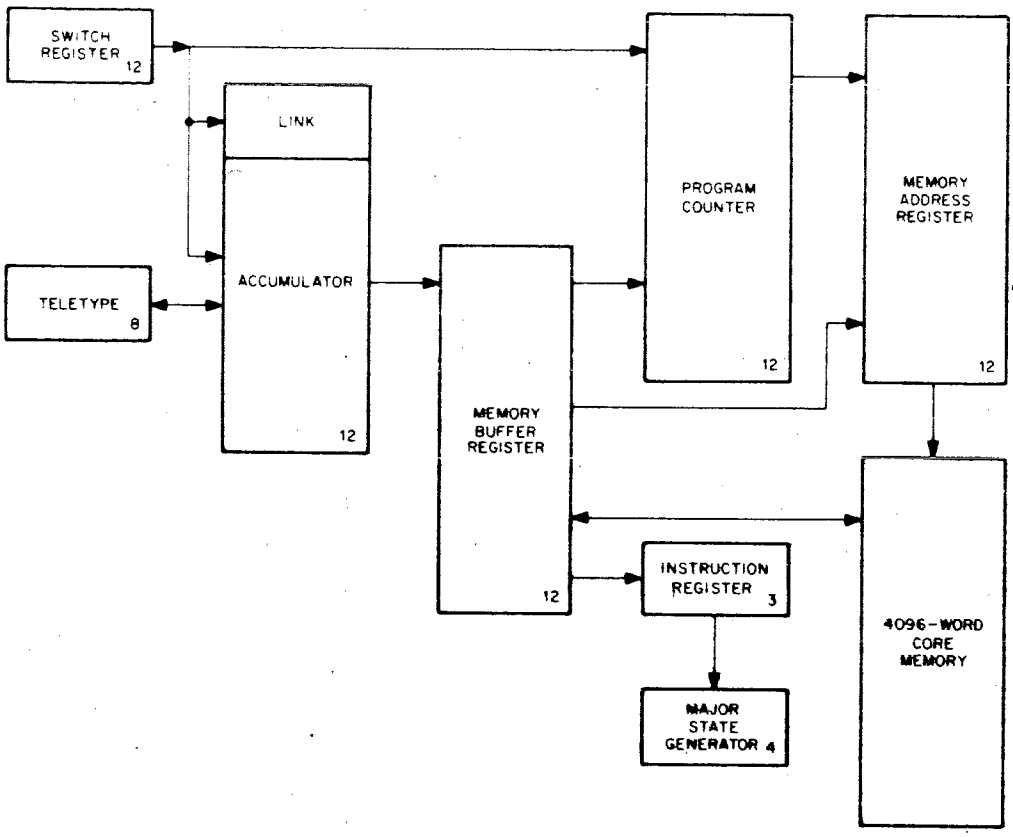


Figure 10 A Block Diagram of the PDP-8

The diagram clearly introduces several new terms with which you will become familiar, but each component can be directly associated with the simple model of figure 9. The switch register and the teletypewriter for example are typical input output devices.

The PDP-8 memory contains 4096 12-bit core registers and therefore requires 12-bit addresses. A 12-bit flip-flop register, the memory address register, is provided to hold an address during a memory reference period, or memory cycle as it is called. Another 12-bit flip-flop register, the memory buffer register, holds all words which move into and out of the memory. These two special registers and the memory make up the memory unit.

The arithmetic unit has two elements in the PDP-8. The accumulator is a 12-bit flip-flop register whose main use is in arithmetic and other operations on numbers held in the memory; it is surrounded by electronic circuits which implement the binary algorithms for these operations. Its name comes from the fact that it accumulates partial results during the execution of the program. Notice that the accumulator also communicates with a Teletype machine—about which more later—and has attached to it a 1-bit flip-flop register, the link, which is used primarily in calculations in which twelve bits are not enough to represent the numbers involved. The accumulator can also be set from the switch register.

The instruction register and major state generator, together with the program counter, can be identified as part of the control unit. The 3-bit instruction

register holds a code number specifying the main characteristics of the instruction being executed (other details are specified by other bits of the instruction being executed (other details are specified by other bits of the instruction as they appear in the memory buffer register), and the major state generator keeps track of the fetch and execution phases of operation.

The program counter is used by the control unit to keep track within the program of the addresses, i.e., the locations in memory, of the instructions to be executed. Ordinarily these instructions are stored at numerically consecutive locations, and the process of keeping track involves no more "counting along" the number in the program counter; that is, replacing it with its binary successor using built-in electronic circuits which implement the by now very familiar binary counting algorithm of figure 3. The program counter can be set initially to the address held in the 12-bit switch register so that operation can start properly with the first instruction in the program. This setting procedure can also be used in the simple kind of storing and retrieval scheme discussed earlier.

Not shown on either of the computer diagrams is a very important component called the operator console which makes it possible for you to communicate with the computer directly, should you wish to do so. A simplified picture is shown in figure 11:

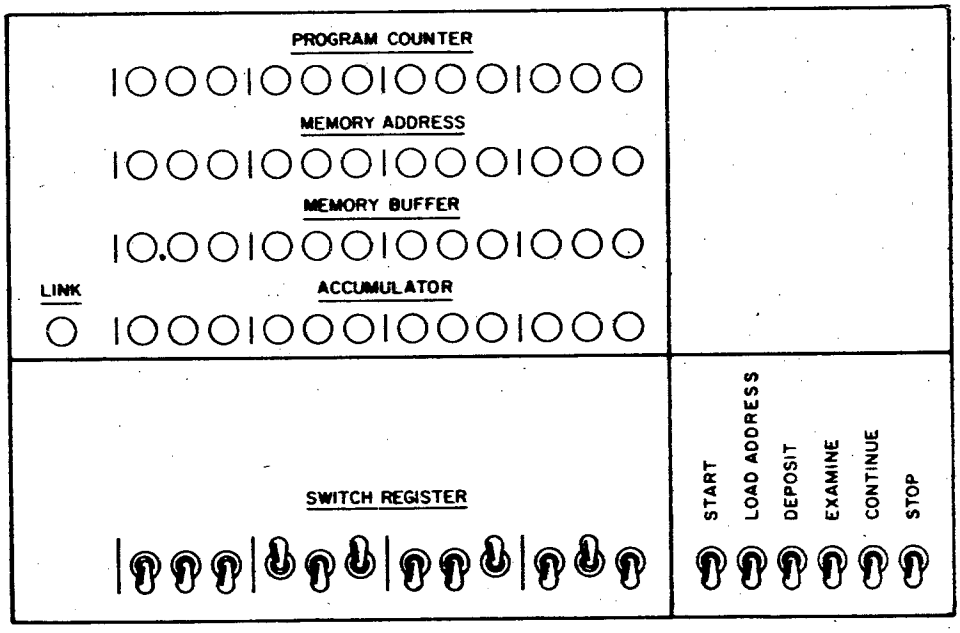


Figure 11 A Simplified Picture of the PDP-8 Operator Console

On the console are manual keys which act as push buttons for storing, retrieving, starting, etc. There are indicator lights for each flip-flop to provide a binary form of visual output, and which you can read when the computer is stopped. The single switch register, in conjunction with the keys, has many applications.

Why not start by seeing how you might use this console to store binary patterns in memory? The switch register supplies both the address and the

pattern itself, unlike the earlier example of figure 7 in which two switch registers were shown. In the PDP-8, the address is first set into the switch register and the key labeled LOAD ADDRESS is momentarily lifted. This action causes the address set in the switch register to be copied into the program counter.

Next, the pattern is set into the switch register and the key labeled DEPOSIT is lifted momentarily. This causes the pattern to be copied into the core register whose address has been put in the program counter. The number in the program counter is increased by 1 in preparation for another DEPOSIT action intended to copy the next pattern set in the switch register into the core register "following" the first one used, and the computer stops.

Direct retrieval is accomplished by setting the address in a similar fashion. The EXAMINE key corresponds to the RETRIEVAL push button of figure 7. The pattern found at the address set in the switch register will be displayed in the accumulator lights. Again, the address held in the program counter is increased by one to facilitate consecutive addressing.

PROGRAMMING THE PDP-8

Now that you have an idea of the structure of the computer, how it represents and handles binary information, and how you can communicate with it, it is time to look more specifically at some of the instructions which are in the repertory of the PDP-8 and to see how they can be arranged into useful programs.

Let us begin with some simple sets of instructions. Suppose, for example, that you have put the following numbers into the memory:

LOCATION (OCTAL)	CONTENT (OCTAL)
⋮	⋮
0040	7200
0041	7001
0042	7001
0043	7001
0044	7402
⋮	⋮

Suppose, further, that you then set the number 0040_8 in the switch register and press the *START* key. Nothing seems to happen, although the indicator lights may appear to blink. The computer is clearly not running, and the accumulator indicator lights show the pattern.

000 000 000 011

that is, the number 0003. The program counter lights show the pattern

000 000 100 101

that is, the number 0045_8. Other lights are lit in other patterns. What does it all mean?

You have just "run" a small program on the computer. The steps of the program were automatically carried out, the computer alternately "fetching" instructions from its memory and "executing" them. The numbers you have stored in registers 4 through 44 are the code numbers of instructions for generating the number 3 in the accumulator. Pressing the START key with 40 in the switch register set the program counter to 40 and started the computer in its calculating mode. The computer "looked up" the contents of register 40, found the number 7200, and sent it off to its control unit for interpretation as the first instruction in the program. The computer then executed sequentially the five instructions in registers 40, 41, 42, 43, and 44.

To understand just which instructions in the repertory might generate the number 3 in the accumulator, let us rewrite the program using mnemonically useful abbreviations. The mnemonic abbreviations used throughout the text have been generated for explanation only. There are no standard mnemonics used by the computer industry.

	Location	Contents (octal)	Mnemonic	Comment
	.	.	.	
	.	.	.	
Start →	.	.	.	
	0040	7200	CLA	Clear AC.
	0041	7001	IAC	
	0042	7001	IAC	Increment AC three times.
	0043	7001	IAC	
	0044	7402	HLT	Halt the computer.
	.	.	.	
	.	.	.	
	.	.	.	

Program Example 1 Generating the Number 3 in the Accumulator

In the PDP-8, 7200 is the code number of the instruction "clear the accumulator," i.e., the instruction to set the twelve accumulator, or AC, flip-flops to the 0 state. These clearing actions were carried out, the number in the program counter was increased by ONE to the value 41, and the computer was then ready to look up, i.e., fetch, its next instruction which it proceeded to do. In register 41 it found the number 7001, the PDP-8 code number of the instruction to "increment accumulator," i.e., to increase the number in the accumulator by ONE. This instruction was then executed, and the step of incrementing the program counter contents was also carried out as before. And so it went. Two more increment accumulator instructions were fetched and executed, leaving the number 3 in the accumulator. The instruction which was then found in register 44, with the PDP-8 code 7402, simply directed the computer to halt, with the number in the program counter set to 0045 in anticipation of another step.

The program might have been stored or relocated in a completely different set of consecutive core registers without in the least changing its effect on the accumulator. You have only to start it at the location of the first instruction, wherever that might be.

The instructions CLA, IAC, and HLT are examples of a whole group of simple built-in instructions called operate instructions, so called because almost all of them "operate on," or refer to the contents of the accumulator. Members of this group are recognized by the control unit because they always have a "7" as the leftmost (most significant) octal digit. By the way, it is always the most significant octal digit that is sent to the 3-bit instruction register for decoding and interpretation.

What about instructions of other kinds, that is, ones with code values other than 7? A very interesting and powerful instruction is the one having the code value 5. It is called the "jump" instruction, and has the effect of changing the program counter. In other words, the computer does not have to execute instructions as they appear sequentially in consecutive memory registers. The jump instruction can be used to direct the computer to another memory register, out of sequence, for the next instruction.

The rightmost three octal digits, i.e., nine bits, of the jump instruction tell the computer the address of the next instruction. For example, if the code number encountered in, say, register 123, during the instruction fetching phase is 5034_8 the "5" will direct the computer to replace the contents of the program counter with the number 0034. The next instruction will, therefore, come not from register 124 but rather from register 34, and the program will go on from that point. In effect, the program "jumps" from register 123 to register 34. We write this instruction mnemonically as JMP 34, and can describe it generally as "JMP Y."* Look at the following example:

	Location (octal)	Contents (octal)	Mnemonic	Comment
	.	.	.	
	.	.	.	
Start →	.	.	.	
	0040	7200	CLA	Clear AC.
	0041	7001	IAC	Increment AC.
	0042	5041	JMP 41	Go back to location 41.
	.	.	.	
	.	.	.	
	.	.	.	

Program Example 2 Endless Binary Counting in the Accumulator

The program endlessly repeats the entire counting sequence of 4096 numbers, replacing the number in the accumulator with its successor each time "around the loop." The contents of the program counter pass through the values 40, 41, 42, 41, 42, 41 . . . endlessly, or rather, until the *STOP* key is pressed or someone trips over the power cable.
You will agree that simply counting out the binary numbers isn't very interesting, but the four instructions which these examples introduce should give you some notion of what it means to "program a computer." Perhaps the instructions don't look quite the way you expected!

The repertory of any general purpose computer, of course, reflects the organization of the particular computer for which it was designed, and although the instructions may seem odd at first, they generally perform very simple operations which are quickly mastered by the programmer.

---

*For the present let Y stand for any number less than 200_8, i.e., any number less than 128_{10}. These are smaller numbers than you need to address the entire memory, the restriction to small numbers will be explained later.

Perhaps the most important single thing to be aware of, as you put instructions together to form a program, is that the computer is really very simple-minded. Remember the CLA instruction in the first example? If the computer had not first been directed to clear the accumulator, the LAC instructions in locations 41, 42, and 43, which "increase the number in the accumulator by ONE," would have done just that; i.e., they would effectively have added 3 to whatever number was in the accumulator at the time. The computer has no way of knowing that that wasn't quite your intention, and, of course, at another time that might have been exactly your intention. This necessary attention to detail may seem picky at first, but you will quickly learn the combinations of instructions which insure the smooth operation of your program.

The instruction repertory is then the "language" of the computer. The programming examples which follow will introduce you to some of the PDP-8 instructions. The repertory includes instructions, such as JMP and HLT, which simply direct the flow of events for the control unit. There are instructions which send information between the accumulator and input or output devices such as the Teletype, (KSF, KCC, KRS, KRB), and ones which send information between the accumulator and the memory (called "memory reference instructions").

Since all the "work" is done in the arithmetic unit, there are a number of instructions in the operate group which make it easy to use efficiently. These include instructions which complement the accumulator contents, rotate it right or left, set it to the number held in the switch register, or cause the computer to skip one instruction when the accumulator is in a certain state.

SYMBOLIC MACHINE LANGUAGE

Before proceeding with more specific programming examples, it will be helpful to introduce a few more mnemonic aids which you will be able to substitute for much of the octal notation.

Program example 2, for instance, can be relocated in another set of core registers, but notice, please, that in this case the program must be changed slightly before it will run correctly: the JMP Y must be recoded to match the new location. Examples:

```
→ 65    CLA          → 123   CLA
  66    IAC            124   IAC
  67    JMP 66         125   JMP 124
```

This can become quite a nuisance, especially with large programs in which there will be many such references to memory addresses. If, however, you assign a name, say, "CAT," to the memory register to which the JMP instruction must return, you can write

```
→ CLA
CAT, IAC
     JMP CAT
```

without, for now, concerning yourself with the actual memory location of the IAC instruction. Note, in the above example, that CAT is followed by a comma when it is used as an identifier for an instruction. When CAT is used to symbolize the address portion of an instruction, no comma is required.

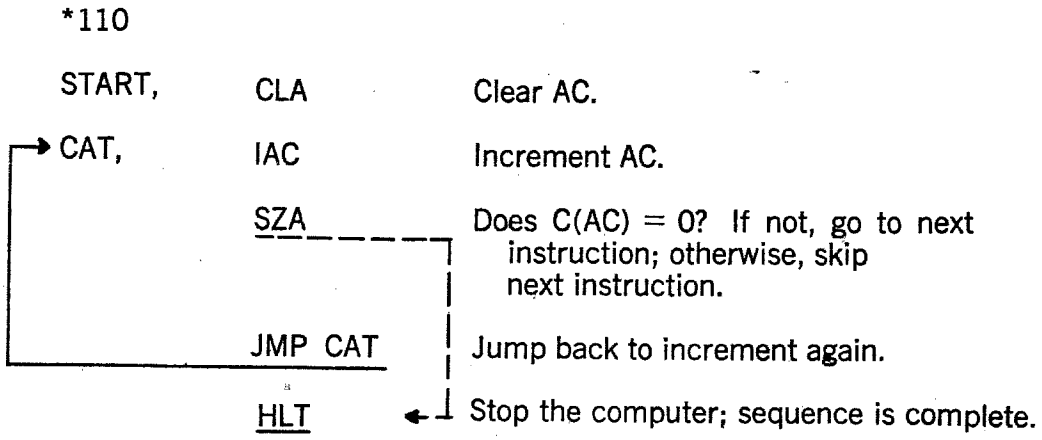
These symbolic references to instruction locations are made possible because of a special program, called an assembler. When you are ready to run your program, the assembler will first translate this symbolic language you have used* into the binary numbers which the computer will use. Clearly a very useful program!

The PDP-8 assembler is one of many such symbolic translators and we will use its notation in the examples which follow. Another translator, of which you have perhaps heard, is called FORTRAN. It can translate more complex combinations of symbols than an assembler, and may be the appropriate language for writing some of your programs.

PROGRAMMING EXAMPLES

Let us turn our attention first to the extremely important skip instructions, ones that give the computer its ability to make decisions automatically, to change the course of its calculations, to discriminate between one kind of result and another. In the PDP-8 each of these instructions takes the form of a conditional "skipping over" of the instruction stored in the immediately following register in memory. This skipping action depends on the detection of certain situations or states; for example, upon the detection of the number 0000 in the accumulator. (The control unit handles skipping by simply incrementing the contents of the program counter one extra time if the designated situation is found to exist.)

For example, if we include the instruction SZA, "skip if accumulator is zero," in program example 2, we can make the computer break out of its otherwise endless loop after it has generated one complete sequence of the 2^{12} binary numbers. SZA has the effect of skipping the instruction following it if the number in the accumulator is 0000. The program would look like example 3 if written in the symbolic language of the assembler. "*110" is called an origin; it "tells" the assembler that you want the program to occupy consecutive memory locations starting at register 110. "C(AC)" is an abbreviation of "contents" of the accumulator".



Program Example 3 Halting after Generation of One Complete Binary Counting Sequence

*Note the arrows and lines, though. They are just your notes to yourself.

The program stops itself after counting out 2^{12} numbers. As you can readily see, it might instead have been made to go on to yet another calculation by replacing the HLT with a JMP to the start of a new calculation. To complete the picture, you might construct the flow diagram for this simple program. It would look like this:

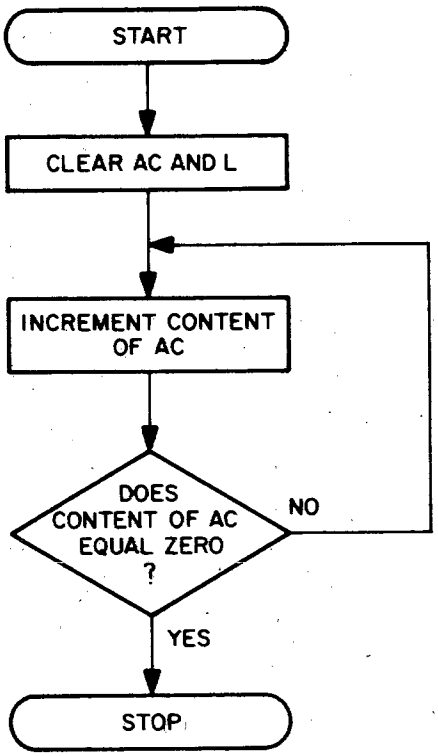


Figure 12 Flow Diagram of Example 3

The PDP-8 includes a very useful and powerful instruction which combines the operation of incrementing and skipping. It is called a "memory reference instruction", and operates on a number held in the memory, using counting and detection circuits on the memory buffer register instead of the accumulator. It is the "increment and skip if zero" instruction, abbreviated ISZ Y. Using this instruction, the equivalent of the last program might be written:

```
        ISZ     COUNT   Increment contents of COUNT. If
                        the result is 0, skip the next instruction.
        JMP     .-1     Return to increment AC again.
        HLT             Result is 0, so halt.
COUNT,  (0)             The count itself.
```

Program Example 4 - Counting in a Memory Register

The notation ".-1" in the JMP instruction is used to go back one location from the location of the JMP instruction, wherever that may be. (In other words, the period is a symbol for "this address".) Note that in this example the accumulator is not used at all; it will remain unchanged throughout the operation of the program.

ADDRESS MODIFICATION

Suppose that you want to have the computer clear a set of consecutive core registers, or what is referred to as a table of registers in memory. You will make use of the instruction, "deposit and clear accumulator," or DCA Y (with the same restriction on Y as before), which has the effect of copying the number in the accumulator into register Y and then clearing the accumulator. Each number in the table of registers, say, 100_8 through 177_8, is to be set to the value 0 in preparation for some further calculation or to be ready to accept data from an input device. The sequence of instructions you are trying to achieve is:

Start →	CLA	Clear AC.
	DCA 100	Deposit C(AC) in register 100.
	DCA 101	Deposit C(AC) in register 101.
	DCA 102	Deposit C(AC) in register 102.
	.	.
	.	.
	.	.
	DCA 176	Deposit C(AC) in register 176.
	DCA 177	Deposit C(AC) in register 177.
	HLT	Stop the computer.

There is, however, a much more compact way of programming this example, which is based on the fact that the code number of an instruction can be modified arithmetically. The code number for the instruction DCA 100, 3100, can be modified in place to become the code number for the instruction DCA 101, 3101, and that in turn, to 3102, etc. A jump back to the register holding this successively incremented DCA instruction can be included so that the DCA instruction will be repeatedly executed, each time depositing 0000 in one more register of the table. The process can be stopped by including an ISZ instruction which increments the contents of yet another register holding a number designed to become 0000 just as, or rather, just after, the last register of the table has been cleared by execution of the instruction DCA 177. The process might be diagrammed:

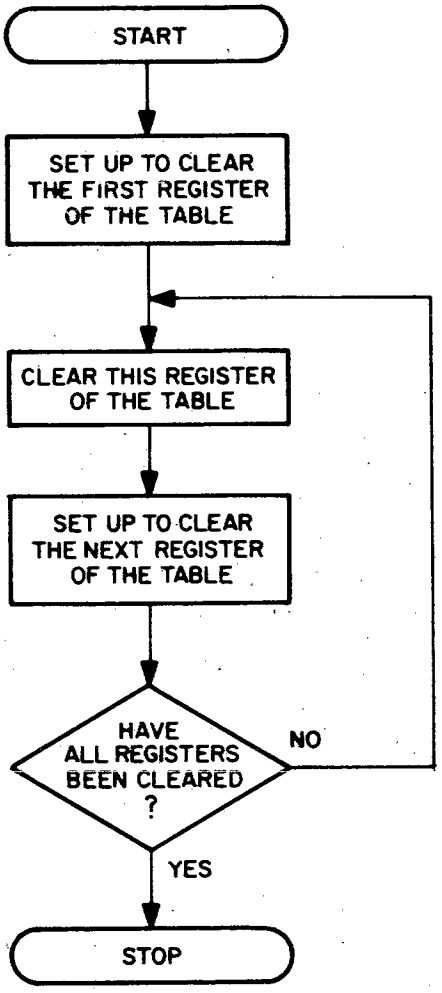


Figure 13. Flow Chart of the Procedure for Clearing a Table of Registers

The program itself might be the following:

```
CLEAR,    CLA          Clear accumulator.
          DCA   100    Deposit in next table register.
          ISZ   .-1    Increment code number of DCA Y
                         instruction itself.
          ISZ   CNTR   Increment CNTR contents. Have
                         all registers been cleared?
          JMP   .-3    No; go back to clear next register.
          HLT          Yes; stop the computer.
CNTR,     -100₈
```

Program Example 5 Clearing a Table of Registers

The initial value in CNTR. -100_8, is, of course, directly related to the number of registers to be cleared. The first pass through the program will clear register 100, advance the DCA 100 code number to 3101, the code for DCA 101, and increment the count in CNTR to -77. You can see that the ISZ instruction will never find a result of 0 after incrementing. It will therefore always direct the computer to go on to the next instruction. But the ISZ CNTR will be changing the count in CNTR in such a way that each pass brings it one step closer to 0, so we'll have:

Before the	DCA instruction is	CNTR holds	Register cleared will be
1st pass	DCA 100	-100	100
2nd pass	DCA 101	-77	101
3rd pass	DCA 102	-76	102
.	.	.	.
.	.	.	.
.	.	.	.
Final pass	DCA 177	-1	177

On the final pass, register 177 is cleared, and then the DCA 177 is advanced to DCA 200 and the -1 in CNTR is incremented to 0, the value being "watched for" by the ISZ CNTR instruction, which then causes a skip to the HLT instruction to stop the computer.

This trick of continuing toward 0 by counting with an ISZ instruction is enormously useful. To go through a part of any program exactly n times, you supply an initial count value of -n. Notice, however, that once you have executed the program in example 5, you cannot get it to work again until the initial value, -100, has been restored and the DCA instruction reset to the code number for the instruction DCA 100, namely 3100.

SETTING UP INITIAL VALUES

The way to handle this problem of setting up initial values is to store the numbers -100 and 3100 in two extra registers, ones that won't be altered by operation of the program, and to copy each of them in turn into the accumulator and from there into registers CNTR and CLEAR + 1 respectively by means of a programmed "preface" to the table-clearing process. We'll use the instruction TAD Y, "2's complement add," execution of which adds a copy of the contents of Y to the contents of the accumulator, leaving the result in the accumulator. (Of course we'll be careful to clear the accumulator first so that the effect of executing the TAD instruction will be only that of copying.) The preface to program example 5 might then look like this

```
BEGIN,  CLA              Copy initial count value into AC.
        TAD  MCNT        Deposit in CNTR.
        DCA  CNTR
        TAD  K           Copy code number for DCA 100 into
        DCA  CLEAR + 1   AC and then into CLEAR + 1.
        JMP  CLEAR       Initialization complete; go to CLEAR program.
MCNT,   -100
   K,   3100
```

Program Example 5a Initialization Prefaces to Program Example 5

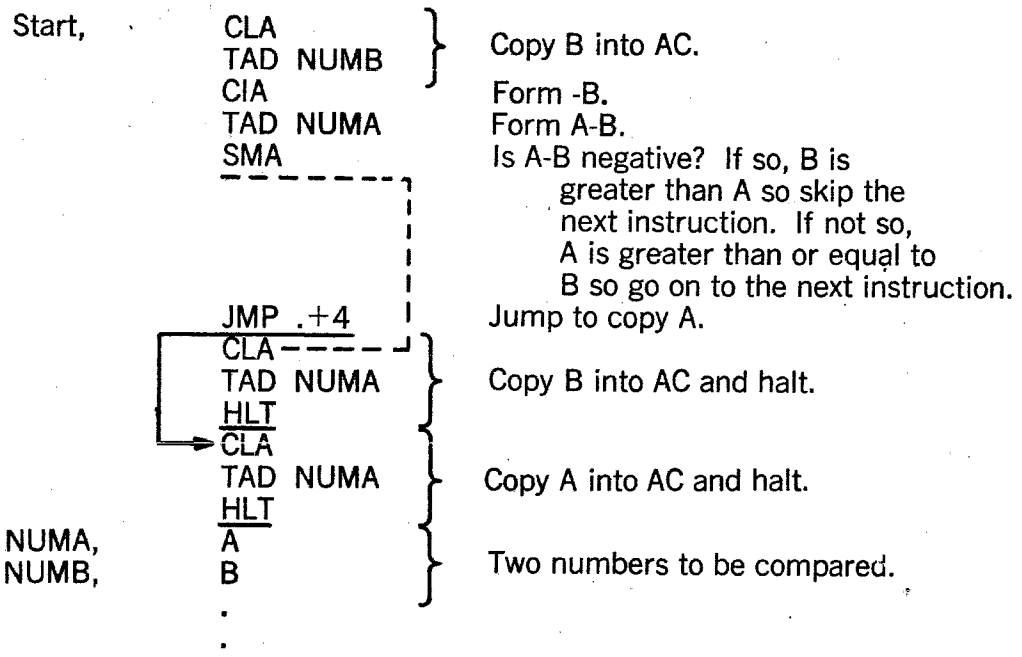
The complete program now starts at BEGIN rather than at CLEAR, and may be repeated as often as you like. Notice that execution of the DCA CNTR instruction clears the accumulator so that execution of the TAD K will, like the TAD MCNT instruction, merely copy. The general process of prefacing a program by setting to their proper initial values any numbers which are to be modified during execution of the program is extremely important and helpful. Best to carry out these set ups before rather than after execution of the modifying program because the intended execution, for some reason, may not go to completion (somebody really ought to put a cover over that power cable!).

SUBTRACTION

Now, of course, the TAD instruction can be used for more interesting purposes than merely copying. After all, it is an arithmetic instruction, and in fact all arithmetic procedures can make use of it: subtraction, by adding negated numbers; multiplication, by repeated additions; division, by repeated subtractions; and the generation of square roots, sines, polynomials, etc., by combining addition, subtraction, multiplication, and division steps.

Look at the subtraction, for example. To find the difference A-B, you can first copy B into the accumulator, then negate it (using the instruction CIA, "complement and increment accumulator") to get -B, then add A. Or, if A is already in the accumulator from a previous calculation, you can negate to get -A, then add B to get B-A, and then negate the result to get: -(B-A) = -A-B.

Here is an example of an interesting and very useful application of subtraction combined with a test of the sign of the result using the instruction SMA, "skip on minus accumulator." It is a program which finds the larger of two numbers, and leaves the answer in the accumulator.



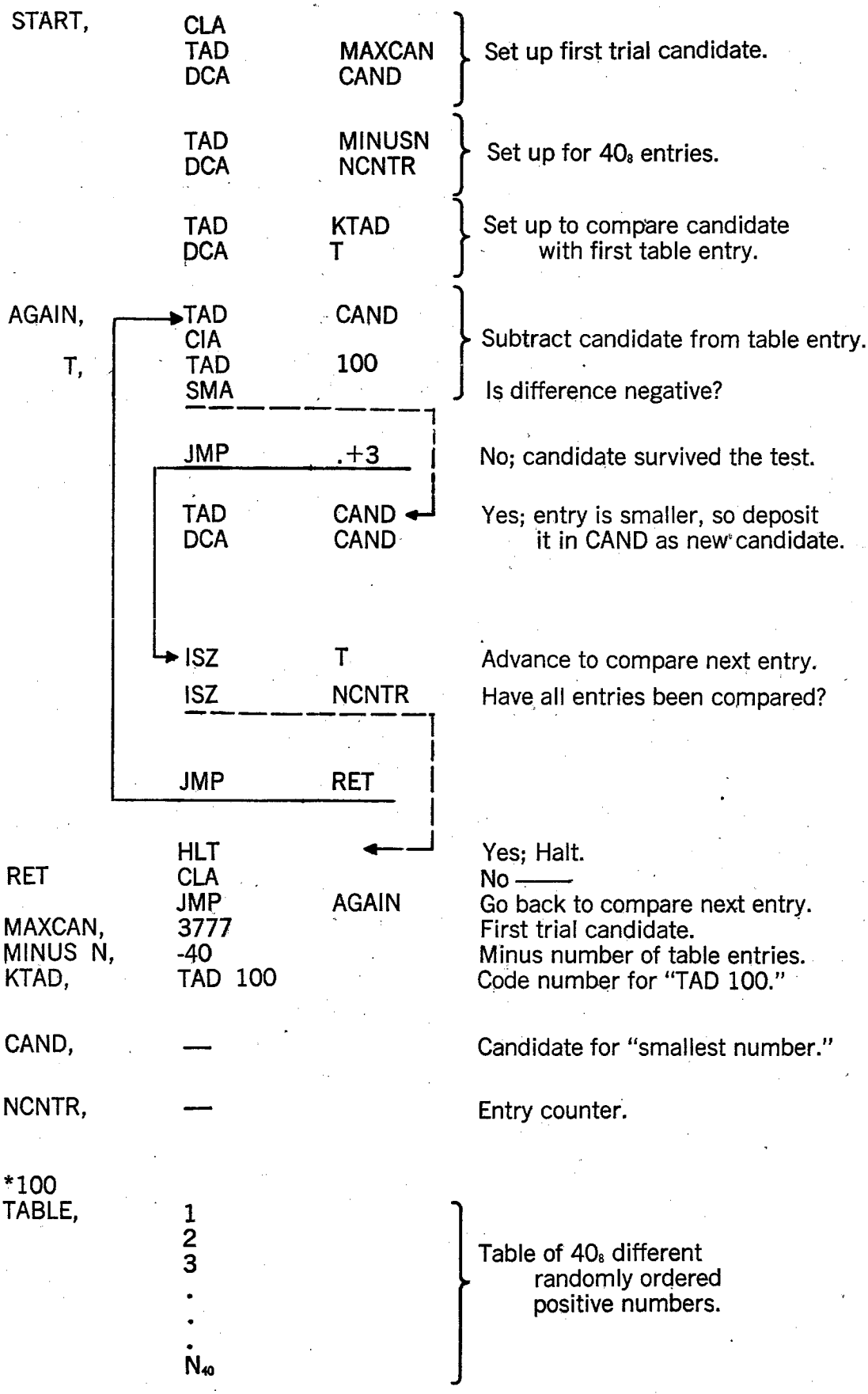
Program Example 6 Finding the Larger of Two Numbers

Number comparison is basic to many processes. In example 6 only the question of which of two numbers is larger is answered, but you can see that it is just as easy to find the smaller of two numbers instead. If the instruction SZA, "skip if AC = 0," is used, the question answered would be whether or not A-B = 0, that is, A = B, enabling you to have the computer look for specific numerical values.

By using two comparisons you can have the computer find out whether a number x is in a specified range, being smaller than some numerical upper bound M, and greater than some numerical lower bound N, perhaps having the computer discard any x that is not in this range.

By combining the process of number comparison with the process of scanning numbers stored in a table of registers, you can have the computer find either the largest or the smallest of a whole list of numbers. There are many ways to do this. One way to find the smallest number is to start by picking a candidate for the smallest number which is at least as large as the largest number in the list. Each number in the list is then compared with this candidate, and whenever a number in the list is found to be smaller, it becomes the candidate and displaces the old one. This "survival of the smallest" process clearly leads to the identification of the correct member of the list.

The following program example will help to clarify things for you. It finds the smallest number in the list of $40_8(32_{10})$ randomly ordered numbers stored in registers 100-137. All of the numbers are positive, and the largest any can be is therefore 3777_8, i.e. $+2047_{10}$.



Program Example 7 Finding the Smallest of a Set of Numbers

After setting up proper initial values, the process begins with a comparison of the trial candidate 3777 with the first table entry by forming their numerical difference and checking its sign. A negative difference indicates that the table entry is smaller than the candidate; a positive difference, that the table entry is not smaller than the candidate.

If a negative difference is found, a copy of the table entry replaces the candidate. Notice that execution of the TAD CAND following the SMA simply recovers the table entry by adding the candidate back into the difference.
If a positive difference is found, the candidate has evidently survived the comparison test, that is, is still at least as small as the smallest number found so far, and no replacement is made.

In either case, the table entry address is advanced, and the entry counter is incremented. If there are more entries to compare, the comparison process is repeated. If all entries have been compared, the process is complete and the program halts leaving a copy of the smallest number in CAND.

Yes, this all seems to be a rather elaborate way of doing what you could do simply yourself by scanning a list of numbers (such as you might find, for example, on a supermarket shopping receipt) to pick out the smallest one. But the computer's repertory of basic operations is very limited. On the other hand, the computer is tireless, extremely fast, and quite accurate, rewarding your patient attention to the initial bother of writing the program by performing your task on demand at any time thereafter. Furthermore, you will be able to build on your efforts and on those of others, incorporating the programs which carry out simple tasks into larger and more powerful programs for more difficult tasks.

INDIRECT ADDRESSING

By now of course, you are familiar with some of the memory reference instructions such as TAD Y, JMP Y, DCA Y, and ISZ Y and with the fact that they refer directly to some register Y during their execution. If, for example, you write ISZ 46, you know that the contents of the register whose address is 46 will be incremented. This is called direct addressing, and while it seems a natural way to refer to a memory register, you will recall that the 9 bits of an instruction word available for an address cannot address the entire 4096 registers of the PDP-8 memory. Not directly, anyway. You can, however, address any register indirectly by putting the full 12-bit address you want an instruction to use into a register of its own.

For example, if you want to store the contents of the accumulator at location 3765, you can write

```
1703765
170 3765
      .
      .
      .
→     DCA  1  170
      .
      .
      .
3765 -
```

Accumlator contents stored here. The new symbol, 1, for indirect address, directs the computer to look in register 170 for the actual address to be used. The accumulator contents will then be stored in register

3765. The contents of register 170 will not be changed.

Of course, you can write this with symbolic names, as

```
STORE,      LATER
                .
                .
                .
            → DCA   1   STORE
                .
                .
                .
LATER,          —
```

Not only does this ability to address indirectly make it possible to refer to all memory registers, but also it simplifies the process of initialization and address modification as the following example illustrates. Example 8 shows the use of indirect addressing in a program to add a constant to each entry in a table of N entries:

```
BEGIN,      CLA                 } Set up for N entries.
            TAD  MINUSN
            DCA  NCNTR
            TAD  TABLOC         } Set up initial entry location.
            DCA  ENTLOC

A,          → TAD  1  ENTLOC    } Add constant K to table entry and
              TAD  CONST            replace in table.
B,            DCA  1  ENTLOC    }
              ISZ  ENTLOC       Advance to next entry.
              ISZ  NCNTR        Have all entries been handled?
              JMP  A            No; go back for next entry.
              HLT               Yes; stop.
              ———

MINUSN,       -N                Minus the number of entries.
TABLOC,       TABLE             Location of first entry in table.
CONST,          K               Constant to be added to each entry.
ENTLOC,       —                 The location of the entry.
NCNTR,        —                 Entry counter.
TABLE,        X1                }
              X2                }
                .               } Table of N entries.
                .               }
                .               }
              Xn                }
```

Program Example 8 Adding a Constant to a Set of Numbers Using Indirect Addressing

Notice that the two instructions at A and B both make indirect reference to the same entry through ENTLOC, yet only the single number, ENTLOC, has to be set up initially and incremented as the program proceeds.

SUBROUTINES

It has been mentioned that you will frequently be able to make use of programs already written to help perform the task which you want the computer to do. This is often accomplished by incorporating other programs as subprograms, or subroutines, in your own program. Or, if there is a particular sequence of instructions which your program must execute frequently, you might want to treat those instructions as a subroutine for convenience and economy of memory registers. Think of subordinates as program building blocks.

A subroutine is a self-contained sequence of instructions which carry out a single task. A good example might be the subtraction of a number in the accumulator from a constant, a task which may be of importance in some larger program and execution of which may be required at many different points in that program. The sequence is of course:

```
        .
        .
        .
        CIA
        TAD   CONST
        CIA
        .
        .
        .
CONST,  K
```

Instead of incorporating the required three instructions at every point in the program requiring this subtraction, the sequence is put into the form of a PDP-8 subroutine called, let us say, SUBCON:

```
SUBCON,
        0
        CIA
        TAD   CONST
        CIA
        JMP   1   SUBCON
CONST,  K
```

At each point in the program requiring the subtraction, the instruction JMS SUBCON is written. JMSY, "jump to subroutine Y," puts the location number following the location number of itself into register Y, and then jumps to register Y + 1. In the above case, if JMS SUBCON were executed at location P, the number P + 1 would be put into register SUBCON and the program would jump to SUBCON + 1, executing the required sequence. After the sequence has been completed, the JMP 1 SUBCON jumps indirectly to SUBCON, finding the number P + 1 therein, and returning control to the program at location P + 1.

Many subroutines have been written for the PDP-8 and may be incorporated in your program saving you a great deal of work. They constitute a library of useful procedures and calculations from which appropriate routines may be bor-

rowed, including routines for square roots, multiplication, division, trigonometric function evaluation, and various high precision calculations; decimal/binary conversion; memory content printout; Teletype readin; and so forth.
These program building blocks greatly enhance the usefulness of the computer and simplify your programming task. In the next section we will see some subroutines which can be used for the Teletype.

DEALING WITH THE PRINTED CHARACTERS

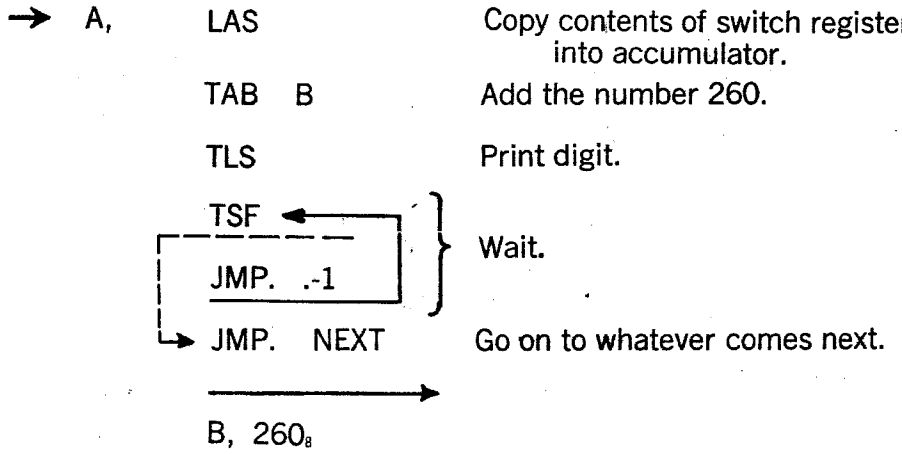
The Teletype is a device for both input and output of information. On your side of the device, the information appears as ordinary type-print like that found on a typewriter. On the computer's side, the information appears as 8-bit binary numbers. Within the Teletype are mechanisms for translating one form into the other.
For example, if you type the character "A" on the Teletype keyboard, it is translated into the code number 301_8. This code number can then be copied into the accumulator by the instruction designed KRB, "read keyboard." Similarly, if the code number 253_8 is held in the accumulator and the instruction designated TLS, "load teleprinter" is executed, the character "+" will be printed. Each character has a unique 8-bit code number, and 8-bit code numbers for such things as carriage-return, space, rub-out, and so forth are also provided. Teletypes, though well matched to a typist in speed, are very, very slow tortoises to the computer's hare. The computer must wait or do something else while the Teletype is catching up at the rate of ten characters per second. When the Teletype is ready, it simply raises a "flag" which is being watched for by the computer, and the computer then proceeds to transfer the code number and lower the flag (so that there will be no confusion with respect to the next character).
A sequence of instructions to read a character from the keyboard would then look like this:

```
.
.
KCC          Lower flag to prevent initial confusion.
KSF          Is flag up?
JMP .-1      No; look again.        } Wait.
KRB          Yes; read character into accumulator and lower flag.
.
.
etc.
```

and a sequence to print a character whose code number is in the accumulator might look like this:

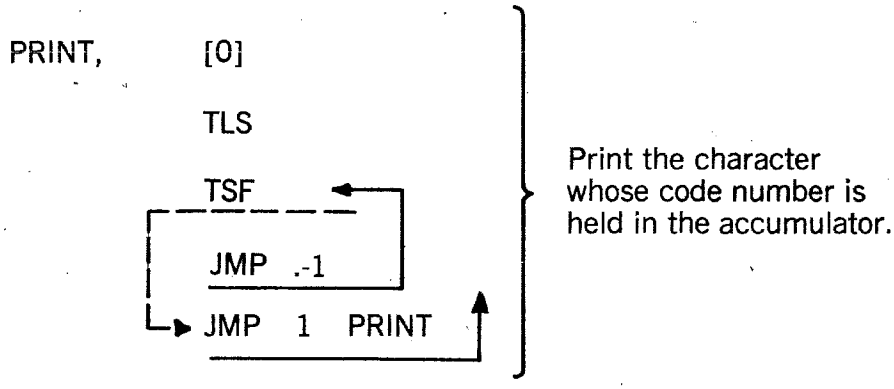
```
.
.
.
TLS          Lower flag, transfer character, and print.
TSF          Is flag up?
JMP .-1      No; look again.        } Wait.
             Yes; continue with program.
.
etc.
```

KSF and TSF are flag-watching instructions of the skip variety.
Now you can see how numerical forms might be handled. A simple example follows in which a binary number in the switch register is read into the accumulator by the instruction designated LAS; "load accumulator with switch register," and then a digit representing its decimal value is printed (we'll restrict the binary number to decimal values less than ten for simplicity):



Program Example 9 Printing the Decimal Digit Equivalent of the Binary Number in the Switch Register

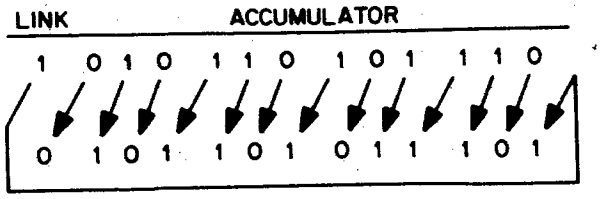
The Teletype code numbers for the digit characters 0, 1, ...9, happen to be The Teletype code numbers for the digit characters 0, 1, . . . 9, happen to be 260, 261, . . . 271 (octal) so that simply adding the number 260 to the binary number in the switch register generates the proper code number. For example, if the switches are set to 000 000 000 011, the addition of the number 260 produces the number 263, the code number for the character "3." The waiting loop doesn't make much sense in this example unless the continuation designated NEXT somehow returns to execute the program again (you would have trouble setting switches at the rate of ten numbers per second to keep up, though!). However, it would certainly be necessary in a general character print subroutine such as the following:



You might use such a subroutine in a program to print the full four octal digits of the number in the switch register. In such a program, you would also want another subroutine to aid in isolating each octal digit. The AND Y instruction will be useful. Execution of AND Y sets to 0 each bit of the number in the accumulator for which the corresponding bit of the number in register Y is 0.

The number in register Y can be thought of as a mask or template through which certain bits of the number in the accumulator can be "erased."

The 12-bit number can be decomposed into the four groups of three bits by shifting it three places to the left each time it is used. In the PDP-8, shifting is carried out in the accumulator and link. The instruction RAL, "rotate accumulator and link left one place," causes each bit to be copied one position to the left, the leftmost bit being copied into the link just as the bit in the link is being copied into the rightmost position of the accumulator, in the manner of a digital Mad Tea Party; for example:



After an initial shift of one place to take account of the inclusion of the link in the "ring" formed by the shift-left pathways, the following subroutine can be used repeatedly to isolate each octal digit in turn, form the digit code number, and print the corresponding digit character using PRINT as a sub-subroutine.

```
NXTDIG,   0
          CLA
          TAD    TEMP
          RAL
          RAL
          RAL
          DCA    TEMP
          TAD    TEMP
          AND    MASK
          TAD    DCODE
          JMS    PRINT
          JMP I  NXTDIG

TEMP.

MASK.  0007

DCODE,  260
```

Shift number in TEMP three places left around the ring, leaving result in TEMP and in the Accumulator.

Form code number and go off to the PRINT subroutine.

Return.

Program Example 10 Octal Digit Isolation and Print Subroutine

Now that the subroutines PRINT and NXTDIG have been provided, they can be used as components of the program to translate a 12-bit binary number in the switch register (or, by an obvious extension, any other 12-bit number that can be copied into the accumulator) into its printed octal equivalent:

```
OCPRINT,  LAS              } Read switch register,
          RAL              } make corrective shift,
          DCA  TEMP        } and deposit in TEMP.

          JMS  NXTDIG      }
          JMS  NXTDIG      } Print the four octal digits.
          JMS  NXTDIG      }
          JMS  NXTDIG      }

          CLA              }
          TAD  CARRTN      } Return the carriage.
          JMS  PRINT       }

          CLA              }
          TAD  LN  FEED    } Feed paper up one line and halt.
          JMS  PRINT       }
          HLT              }

CARRTN,   215              Carriage return code number.
LNFEED,   212              Line feed code number.
```

Program Example 11 Printing the Octal Equivalent of the 12-Bit Number in the Switch Register

After printing the four octal digits, the program uses the PRINT subroutine directly to return the carriage and to feed the paper up one line.
You can readily see that similar programs can be written to read in and combine octal digits as they are typed on the keyboard. Or, in translating from symbolic language to binary number, to read in the characters "C," "L," "A," and by a process of matching their code numbers to ones stored in a table, decide that this string of 3 characters, CLA, should be assigned the value 7200, the code for "clear accumulator." Or, after reading in the string of characters "2," "+," "3," "c," a program might respond by printing out the digit "5." You can think of many other examples based on these simple notions yourself. The possibilities are quite limitless.

PART II: APPLICATION PROGRAM EXAMPLES

INTRODUCTION

Now that you are acquainted with the fundamentals of computer logic and programming, you are probably anxious to see the machine in action. After all, for most of you, the main question is "How will the computer assist me in solving problems?"

To answer your question, we have selected several representative scientific and engineering computer program examples, specifically in the fields of oceanography, physics, biomedicine, and process control. In the first example, we describe a simple program language, similar to algebra, and its value to oceanographers. In the second example, we analyze pulses generated by a gamma-ray detector. The third example is a program for generating and displaying a time-interval histogram designed for the biomedical scientist. Computer-directed process control techniques are discussed in the fourth example.

REAL TIME TECHNIQUES FOR OCEANOGRAPHIC APPLICATIONS

INTERFACING MARINE SENSORS

Marine sensing instruments are often considered unique devices that cannot be connected directly to a computer. However, most instruments can be connected directly to the computer with little additional equipment, figure 1. Let us examine the basic types of interfaces that would be necessary to connect the computer to oceanographic sensors.

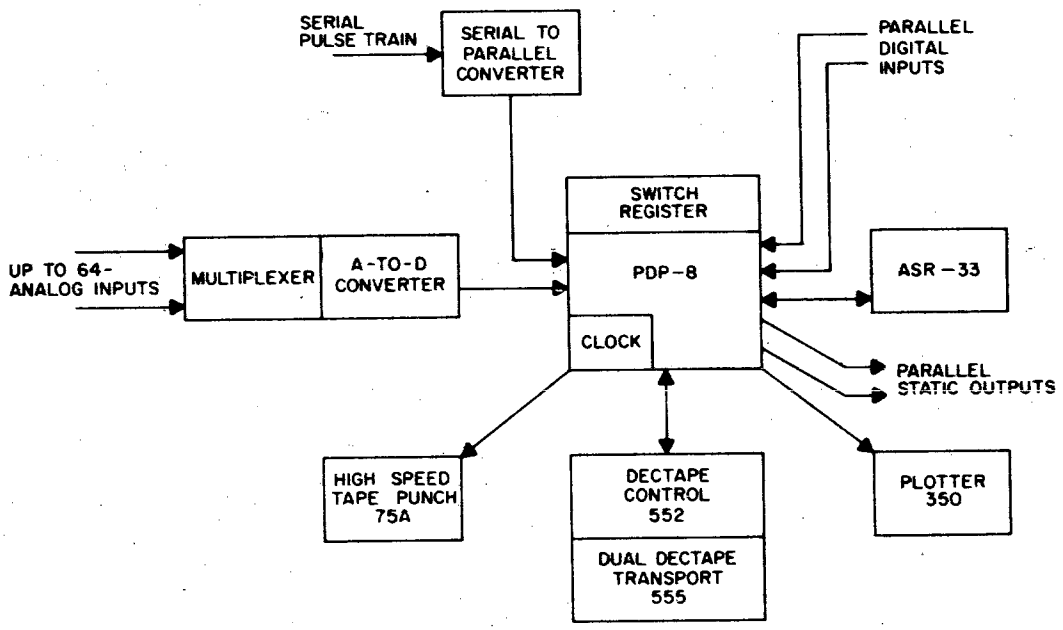


Figure 1. Typical Data Acquisition System

INTERFACING TRANSDUCERS

Three basic types of interfaces are used with the PDP-8 computer to receive data from environment sensors. These interfaces allow data to enter the computer under control of a simple real-time symbolic compiler that gives you the following flexibility when you sample the environment:

1. A variety of preprogrammed interface devices that easily connect the computer to instrumentation.
2. Simple symbolic assignment of identifying names to physical input variables such as pressure, temperature, etc.
3. Absolute control in sampling the experimental environment with respect to time.
4. Computer compatibility with respect to rapid response sensors using up to 176 separate sensing devices.
5. Capability to make logical decisions concerning the acquisition of data while sampling the environment.
6. Storage of data for future computations as well as immediate output for checking quality while obtaining data.

Transducers are sometimes considered unique devices that do not lend themselves to being connected directly to a computer; however, by the use of basic standard interface types, most instruments can be connected directly to a computer with little additional equipment. Let us examine the basic types of interfaces that would be necessary to interconnect a computer to various transducers.

INTERFACE TYPES

Digital-Parallel Input Signal Buffer

This buffer permits the direct parallel insertion of a digital number into the computer, using a method by which the number immediately becomes a computer word. Examples of devices feeding data in by this method are shaft-position encoders, switch registers, and other allied devices. Figure 2 shows the general method for digital-parallel input.

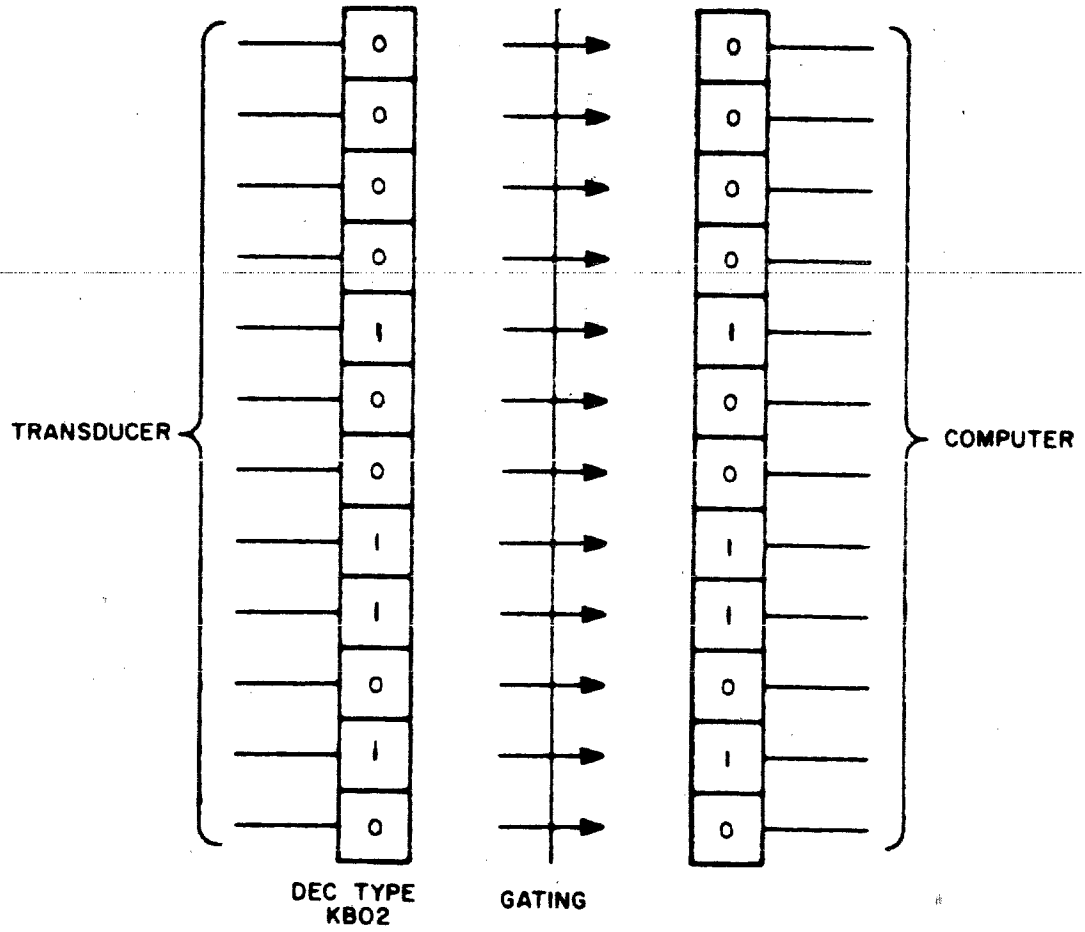


Figure 2. Digital-Parallel Signal Buffer

This method can accommodate signals or pulses from a minimum range of 0 to —10mv up to a maximum of 20 to —15v. The system can include one buffer for each of several dozen variables.

Serial-Parallel Input Signal Buffer

This method would be used, for example, in telemetry applications where the

transmitter is remotely located and able to transmit a number of input words one bit at a time along a single conductor cable or by radio (see figure 3).

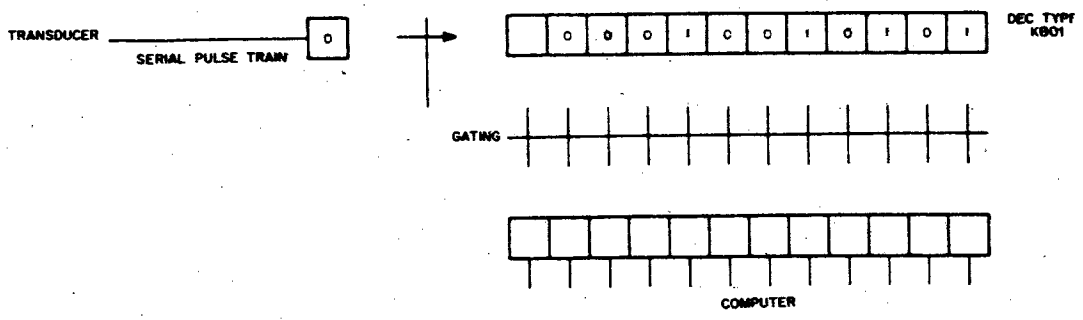


Figure 3. Serial-Parallel Signal Buffer

This buffer can convert a serial pulse train to a 12-bit word in 6 μsec, or one bit every 500 nsec. It accommodates ranges of input levels from a minimum of 0 to —10mv, up to a maximum of 20 to —15v. The flexibility provided in this buffer allows you to format the data before it is finally assembled as computer words; that is, you can insert octal constants in the specific serial word of your choice. It provides the programmer with the following additional instructions in the computer:

1. Skip if data flag = 0.
2. Skip if start flag = 0.
3. Clear data flag and start flag.
4. Read data into the accumulator.

Multiplexed Analog-to-Digital Conversion Input

This method is particularly advantageous in data acquisition when many devices such as thermistors, pressure sensors, and strain gages are working together. One example of where this method would prove advantageous is a thermistor chain in which each thermistor, pressure sensor, or conductivity sensor could be individually sampled by the computer. Figure 4 indicates this relationship.

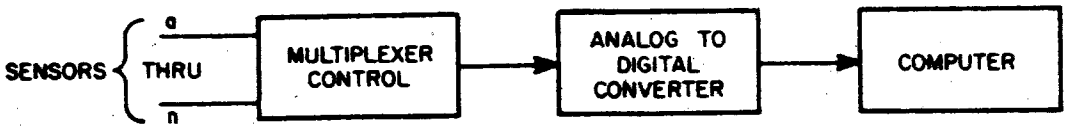


Figure 4. Multiplexed Analog-to-Digital Conversion

Switching from one input to the next in the multiplexer is accomplished in 2 μsec, and up to 64 separate inputs can be sampled directly by the computer. The analog-to-digital converter uses the range of 0 to —10v.

PROGRAMMING

The task of writing a special-purpose program for each installation could be a formidable problem involving a great deal of time and money. In order to alleviate this problem, Digital has written DATAK, an algebraic compiler that allows you to format your data acquisition problem in a simple language similar to algebra. By using the DATAK language, you are given a great deal of flexibility

concerning the interface hardware available. This program offers you flexibility in choosing the frequency and conditions under which you sample the experimental environment, as well as makes possible the addition or change of sensors without the major task of reprogramming in machine coding. This program is available for the PDP-8, a compact 12-bit computer with a 1.5 μsec cycle time.

This program allows you to analyze a sample using the following inputs.

Up to 96 independent data variables using digital-parallel input.
Up to 64 independent data variables using a multiplexed A-to-D converter.
Up to 25 independent data variables via serial buffer input.

Each independent variable can be sampled at a rate of up to 100 times per second.

System Inputs

Data can come to the computer from the digital-parallel input buffer, the serial input buffer, and the multiplexed analog-digital converter. Each of these devices is assigned a symbolic name that tells the computer which device is transmitting information.

The symbols are:

DGIN: Digital-parallel signal buffer; input can be converted from Gray code to binary.

BUFR: Serial buffer input.

ADCV: Multiplexed analog-digital converter.

System Outputs

Data output can be distributed to a number of specifically named devices to allow immediate presentation as well as permanent storage. The following output symbols and their associated devices are available:

TYPE On-line teleprinter. Variables can be typed in decimal or octal. Decimal is specified by ↑ immediately following the variable name.

PNCH High-speed paper tape punch. Variables can be punched in decimal or octal. Decimal is specified by ↑ immediately following the variable name.

DCTP Digital's compact DECtape. Variables are recorded magnetically on DECtape in binary with identifying words.

PLOT X-Y plotter. The plotter pen is moved to a new position each time an output is specified.

DIG 1 Up to four digital outputs are available through parallel buffers to other devices such as relays, buffers, sense lines, and range
DIG 4 switching devices.

Variables

Input variables to the computer are assigned alphanumeric symbols by the investigator. They can be one to four characters long, and must begin with a letter. Some examples of these would be:

T123, SURF, DEEP, AIR, X, Y, TEMP, H20, H202

In addition, variables that are inputted through the multiplexer have specified channels; that is, T123(1) would be input through channel 1 of the multiplexer.

Time

Four types of time can be used by the computer: basic, program, variable, and reference.

BASIC TIME

Basic time represents the basic interval of 0.01 sec in which a clock interrupts the program.

PROGRAM TIME

This time represents the basic rate at which the investigator desires to interrogate the sensors. It is some multiple of the basic time and is under program control. Its symbology is simply expressed as follows:

QUNT:50 The investigator has specified that the program time will be $50 \times .01 = 0.5$ sec; that is, each sensor will be sampled every 0.5 sec. If he desires the fastest rate possible, he may express the following:

QUNT:1 In which case each variable will be sampled every 0.01 sec.

VARIABLE TIME

In order to allow more flexibility in timing, digital inputs can be sampled at a slower rate than the program time specifies. For example, the expression:

DGIN:L1 (4, L2 (4

specifies that the digital input variables L1 and L2 should be sampled once in four cycles of the program time or every $4 \times 0.5 = 2$ sec.

REFERENCE TIME

It is often desirable to know the reference time in order to associate data with time. Within the program is a 3-word variable, CLOK, which counts the number of seconds, minutes, and hours that have elapsed since start-up time; it can be used as an output variable to reference data with time.

Arithmetic Operation

ADDITION AND SUBTRACTION

Variables can have constants added to or subtracted from them as they are sampled, or the variables can be added to or subtracted from each other.

All arithmetic operations are done in 2's complement arithmetic, with the operands being considered signed, fixed-point numbers. The following examples

mean that the variable T2 will have constants or variables added or subtracted before output:

T2 + 137	Add a constant to the variable.
T2 — 3	Subtract a constant.
T2 + T3	Add a second variable.

ARITHMETIC COMPARISON

Variables can be compared against constants, compared against other variables, or compared against themselves with respect to sample time. The basic comparison instructions are:

IFEQ X, Y	if X is equal to Y
IFLS X, Y	if X is less than Y
IFGR X, Y	if X is greater than Y

An example of the comparison of a variable against a constant would be:

IFEQ X, 1000;

meaning that if X is equal to 1000_8, execute the operation following the semicolon; otherwise, go to the next line.

Every time a variable is recorded and outputted, its value is preserved and is given the name of the original variable. Thus, X and @ X represent the current value of the variable X, and its value when last used as output, respectively.

The following example of the comparison of a variable and its predecessor:

IFLS X, @ X;

means that if X is less than it was when last recorded and output, execute the operation following the semicolon; otherwise, go to the next line.

Gray Binary Conversion

Gray binary code can be converted to simple binary under program control if the input method is digital (DGIN). This provides you with a rapid means of conversion in order to intercompare the usual shaft-encoded Gray binary numbers, if the shaft encoder does not convert from Gray Code to simple binary prior to buffering.

The conversion is accomplished by inserting ↑ immediately before the time multiple.

DGIN L1↑4

This means that the Gray binary variable L1 is sampled every fourth time through the program and is converted to a simple binary number before comparison or storing.

Format Statements

Format statements are numbered from 1-15 (decimal). They contain the names of variables and their output forms (octal or decimal for the Teletype or punch). Format numbers appear along with a device name in every output

statement. Thus, the statement:

FORM: 1,X,Y↑,Z

indicates that the variables X, Y, and Z are to be outputted to the Teletype, with X typed in octal, Y typed in decimal, and Z typed in decimal.

GOTO Statement

Program control can be unconditionally transferred through the use of the GOTO statement.

PROGRAM EXAMPLES

A Simple Programming Problem

An *in situ* pressure, temperature, and salinity sensing instrument is lowered into the ocean. Data is transmitted along a single conductor cable and is brought into the computer using a serial buffer input.

We want to sample the ocean in the following manner:

1. From the surface to 100 meters, record at each meter the pressure, temperature, and salinity.
2. From 100 meters to 1000 meters, record the pressure, temperature, and salinity whenever the absolute change of temperature is greater than 0.05°C or the absolute change of salinity is greater than 0.02‰. Also record the pressure, temperature, and salinity every 100 meters from 100 meters to 1000 meters.

Let us assume that the oceanographic sensors have the following precision; that is, unity is equal to the following:

1 unit of pressure = 1 meter
1 unit of temperature = 0.01°C
1 unit of salinity = 0.01‰

A program to accomplish this sampling is written as follows:

```
BUFR :      PRES, TEMP, COND
FORM :      1, PRES, TEMP, COND

[    :      OUTP (1, DCTP)
1    :      IFGR PRES, 144; GOTO 2
     :      IFGR (PRES –@PRES), 1; OUTP (1, DCTP)
     :      GOTO 1
2    :      IFLS (PRES –@PRES), 144; IFLS (TEMP –@TEMP), 5;'
            IFLS (COND –@COND), 1; GOTO 2
     :      OUTP (1, DCTP); GOTO 2
     ]
            END
```

This program says in effect:

BUFR: PRES, TEMP, COND

Three variables named PRES, TEMP, and COND are to be sampled using the serial buffer.

FORM: 1, PRES, TEMP COND

Three variables named, PRES, TEMP, and COND are to be outputted together.

```
:OUTP (1, DCTP)
```

This says output is to be recorded on magnetic tape.

```
1:IFGR PRES, 144; GOTO 2
```

This states that if the absolute change of pressure is greater than $100(144_8)$, the control of the sampling will be transferred to statement number 2; otherwise, it will go to the next line.

```
:IFGR(PRES –@PRES), 1; OUTP (1, DCTP)
```

This line states that if the absolute change of the pressure between two successive readings is greater than 1, output onto magnetic tape according to format 1; that is, OUTP (1, DCTP) which means store data on DECtape using format 1; otherwise, go to the next line.

```
GOTO 1
```

This says to go to statement number 1 and test the environment again.

```
2:  IFLS (PRES –@PRES), 144; IFLS (TEMP –@TEMP), 5;'
    IFLS (COND –@COND), 1; GOTO 2
```

This states that if the absolute change of pressure is less than 100 meters or the absolute change of temperature is less than .05°C, or if the absolute change in salinity is less than .02‰ then go to statement number 2 which begins the tests over again. Otherwise go to the next line.

```
:OUTP (1, DCTP); GOTO 2
```

This says to output data onto magnetic tape and transfer control to statement number 2. The sampling and testing procedure begins again.

```
END
```

This last instruction is self explanatory.

As shown in the above description, the computer has been programmed to make logical decisions specified by the investigator in sampling the marine environment. It also has been used as a means of storing data. In the above instance, data has been stored on magnetic tape and can be used in other programs to determine variables such as Sigma T, anomaly of specific volume, and sound velocity. Table 1 shows a portion of the calculated output from stored data on magnetic tape transport number 1 that can be run immediately after the sample program.

TABLE 1 REDUCED DATA

Depth	Input Source? T Observed Values Temp.	Salin.	Sigma-T	Delta-A	Sound-Vel
0000	8.35	34.17	+26.590	+145.53	+1483.4
0001	8.28	34.19	+26.616	+143.08	+1483.2
0002	8.20	34.21	+36.644	+140.45	+1482.9
0003	8.13	34.24	+26.678	+137.20	+1482.7
0004	8.05	34.26	+26.706	+134.59	+1482.4
0005	7.98	34.28	+26.732	+132.19	+1482.2
0006	7.91	34.31	+26.766	+128.98	+1482.0
0007	7.83	34.33	+26.793	+126.38	+1481.7
0008	7.76	34.35	+26.819	+123.94	+1481.4
0009	7.69	34.37	+26.845	+121.45	+1481.2
0010	7.61	34.39	+26.873	+118.89	+1480.9

A More Sophisticated Program

For a better demonstration of the flexibility of this programming technique, consider the following program.

An investigator desires to use a thermistor, pressure, and conductivity chain towed from an oceanographic vessel. He will sample at the same time a telemetering buoy that transmits data from these current meters. In addition, he desires to obtain Loran lines of position and sample the ship's speed and ship's heading. These can be summarized as follows:

1. Log the time on magnetic tape every 50 meters of distance traveled. Ship's speed is 10 knots; it will cover 50 meters in approximately 9.70 sec.
2. Sample each thermistor, conductivity, and pressure sensor in the chain every half second. This defines the program time as QUNT: 62.
3. Sample the Loran, ship's speed, and ship's heading every 2 sec, thus L1(4, L2(4, SP(4, HEAD(4.
4. Conditional Output — Since the near-surface values of temperature and conductivity will fluctuate the most, it might be most desirable to set thresholds so that relatively large changes of temperature and salinity will be stored. However, deeper values will not change as significantly, so small incremental changes have more meaning and thus should be outputted and stored. Arbitrary values have been chosen and are shown in table 2.

Determination of Octal Constants to be Used in Testing — In order to test the variable it must be determined to what its unit value corresponds. This is found by dividing the range of the thermistor, pressure transducer, or other device by the precision of measurement; thus if the range of the thermistor is 20°C and the precision of measurement is 1 part in 2000, then each unit equals 0.01°C.

5. General Output Requirement
   Every variable should be recorded on magnetic tape if the specified conditions are met.

   Plot TO versus SO if it is recorded.

   Type current meter data in decimal if it is recorded.

   Punch TO, PO, and SO if they are recorded.

By outlining the problem, the investigator will have thresholds established for recording changes in the variables listed in tables 3, 4, and 5. Figure 5 shows the equipment needed to do the work.

TABLE 2 SUMMARY

Variable	Range	Precision	UNITY Corresponds to
Thermistor	20°C	1:2000	≅ .01°C
Pressure	100 meters	1:2000	≅ .05 meters
Conductivity	20‰	1:2000	≅ .01‰
Vane	360°	1:120	≅ 3°
Compass	360°	1:120	≅ 3°
Rotor			≅ 1 centimeter per second

TABLE 3 MULTIPLEXER A-D CONVERTER ASSIGNMENT
(Data Originating in Thermistor Chain)

Depth in Meters	Thermistor Variable Name	Multiplexer Channel #	Record if Absolute Change of	Pressure Variable Name	Multiplexer Channel #	Record if Absolute Change of	Conductivity Variable Name	Multiplexer Channel #	Record if Absolute Change of
0	T0	(0)	.1°C	P0	(6)	.5 meter	S0	(14)	.05‰
10	T1	(1)	.07°C	P1	(7)	.5 meter	S1	(15)	.05‰
20	T2	(2)	.05°C	P2	(10)	.5 meter	S2	(16)	.04‰
30	T3	(3)	.03°C	P3	(11)	.3 meter	S3	(17)	.03‰
40	T4	(4)	.02°C	P4	(12)	.2 meter	S4	(20)	.02‰
50	T5	(5)	.01°C	P5	(13)	.1 meter	S5	(21)	.01‰

TABLE 4 SERIAL DATA INPUT BUFFER ASSIGNMENT
(Data Originating in Moored Current Meter String)

	Vane Variable Name	Compass Variable Name	Record if	Rotor Variable Name	Record if
Current Meter 1	V1	C1	V1 + C1 = 9°	R1	R1 > 10
Current Meter 2	V2	C2	V2 + C2 = 6°	R2	R2 > 10
Current Meter 3	V3	C3	V3 + C3 = 3°	R3	R3 > 10

TABLE 5 DIGITAL INPUT BUFFER ASSIGNMENT
(Data Originating in Ship's Instrumentation)

	Variable Name	Time Multiple	Gray Code?
Loran Line	L1	(4	
Loran Line	L2	(4	
Ship's Speed	SP	(4	
Ship's Head	HEAD	↑4	yes

The program listing to sample these variables and record those which exceed the established thresholds is given below.

```
ADCV :    T0(0), T1(1), T2(2), T3(3), T4(4), T5(5), P0(6),'
          P1(7), P2(10), P3(11), P4(12), P5(13), S0(14), S1(15),'
          S2(16), S3(17), S4(20), S5(21)
BUFR :    V1, C1, R1, V2, C2, R2, V3, C3, R3
DGIN :    L1(4, L2(4, SP(4, HEAD ↑4
QUNT :    62
FORM:     1, T0, T1, T2, T3, T4, T5, P0, P1, P2, P3,'
          P4, P5, S0, S1, S2, S3, S4, S5
FORM:     2, T0, S0
FORM:     3, V1, C1, R1, V2, C2, R2, V3,'
          C3, R3
FORM:     4, T0, P0, S0
FORM:     5, L1, L2, SP, HEAD, CLOK
          <5, DCTP, 22>
[    :    OUTP(1, DCTP); OUTP(2, PLDT); OUTP(3, TYPE); OUTP(4, PNCH)
3    :    IFEQ (V1 + C2), 3; GOTO 1
          IFEQ (V2 + C2), 2; GOTO 1
          IFEQ (V3 + C3), 1; GOTO 1
          IFLS R1, 12; IFLS R2, 12; IFLS R3, 12; GOTO 2
```

```
1       :       OUTP (3, TYPE)
2       :       IFLS (T0–@T0), 12; IFLS (P0–@P0), 12; IFLS (S0–@S0), 5;'
                IFLS (T1–@T1), 7; IFLS (P1–@P1), 12; IFLS (S1–@S1), 5;'
                IFLS (T2–@T2), 5; IFLS (P2–@P2), 2; IFLS (S2–@S2), 4;'
                IFLS (T3–@T3), 3; IFLS (P3–@P3), 6; IFLS(S3–@S3), 3;'
                IFLS (T4–@T4), 2; IFLS (P4–@P4), 4; IFLS (S4–@S4), 2;'
                IFLS (T5–@T5), 2; IFLS (P5–@P5), 2; IFLS (S5–@S5), 1;'
                GOTO 2
        :       OUTP(1, DCTP); OUTP(2, PLDT); OUTP(4, PNCH); GOTO 3
]
                END
```

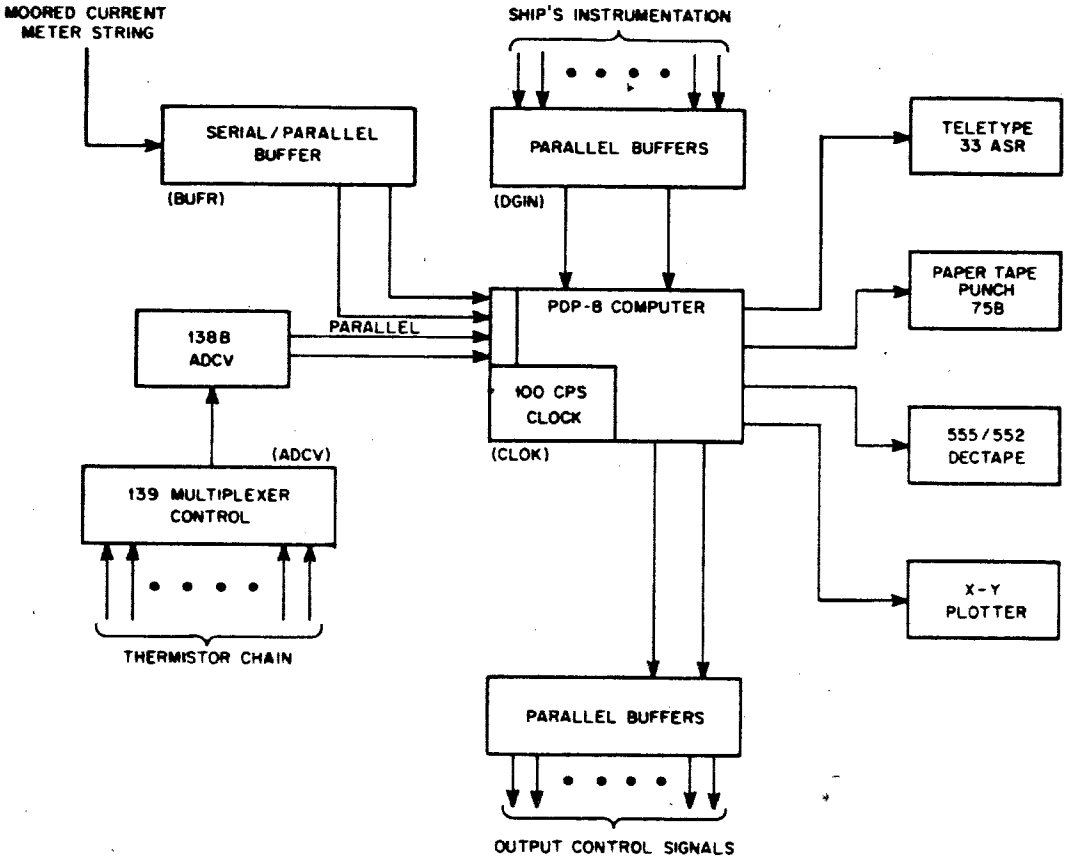


Figure 5. Equipment Configuration for Problem 2

A BASIC PROGRAM FOR PULSE HEIGHT ANALYSIS

The problem of pulse height analysis is basic to the physicist. With this example, we will demonstrate the computer's ability to rapidly sense and analyze large numbers of events in real time. In this example, the pulses to be analyzed are generated by a charged-particle or gamma-ray detector that produces a stream of pulses proportional to the energies of the particles intercepted by the detector. If we write a computer program to sort and count the resulting pulses according to height, and then display the result, we will obtain the radioactive spectrum of the source, as shown in figure 6.

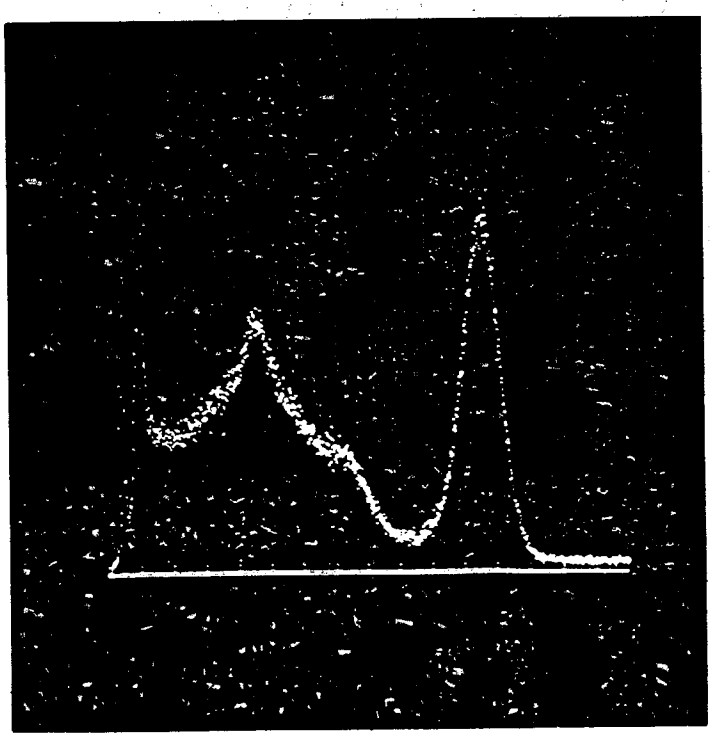


Figure 6. An Oscilloscope Photograph that Shows the Number of Nuclear Particles as a Function of Energy.

The first step in setting up the experiment is to construct a basic algorithm for the required computer program (figure 7). The computer continuously monitors the detector, waiting for output pulses. When a particle is detected, the program notes the amplitude of the resulting output pulse. The particle is then counted by incrementing a location in memory used to record the number of detected particles within that particular energy level. When a statistically significant number of particles are counted, the results are displayed on the oscilloscope.

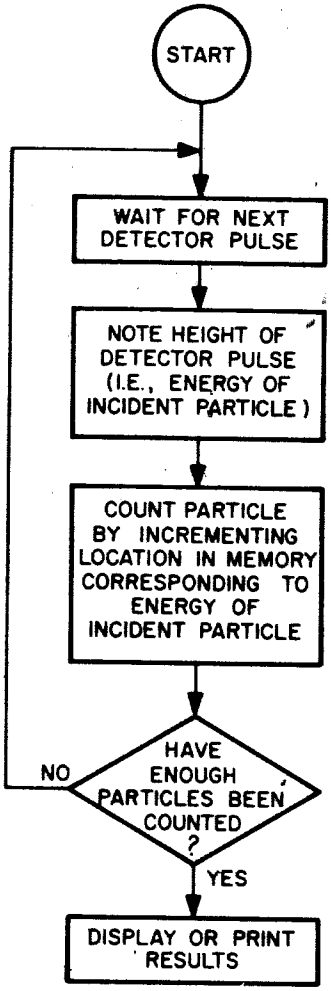


Figure 7. Basic Algorithm for Pulse Height Analysis Experiment.

The algorithm shown in figure 7 becomes somewhat unsatisfactory when we consider that we are requiring computers capable of some 30,000 to 300,000 instructions a minute to wait for a detector that outputs pulses at irregular intervals. Could not this wasted time be put to better use? Say, to produce a continuous, dynamic display of the energy spectrum. The answer is yes — we need only connect the analog-to-digital converter to the computer interrupt line. Then, whenever a particle is detected, the computer program will be interrupted. The interruption is a signal to the program that the analog-to-digital converter should be read and the appropriate memory register incremented. At all other times, the program generates a continuous display of the radioactive source's energy spectrum.

The computer responds to the interrupting signal as follows:

1. the computer concludes the instruction being executed;
2. the location of the instruction that would normally be executed next by the main program (that is, the contents of the program counter) is stored in memory location 0;
3. the computer takes its next instruction from memory location 1.

Thus, the first instruction of any program responding to an interrupt signal must be placed in location 1. The interrupt-servicing program should also save the contents of the accumulator and any other active register that is to be used by the interrupt program before issuing any instructions that alter the contents of these registers. When the interrupt-servicing program is complete, the original contents of the accumulator must be restored, and an indirect jump through location 0 (JMP I 0) must be made to return control to the main routine at the point where the interruption occurred.

Figure 8 illustrates the equipment needed for the experiment. The particle detector generates a voltage proportional to the energy of the incident particle; this is converted to a binary number by the analog-to-digital converters. Conversely, two digital-to-analog converters are used to drive the X and Y deflection amplifiers of the display oscilloscope.

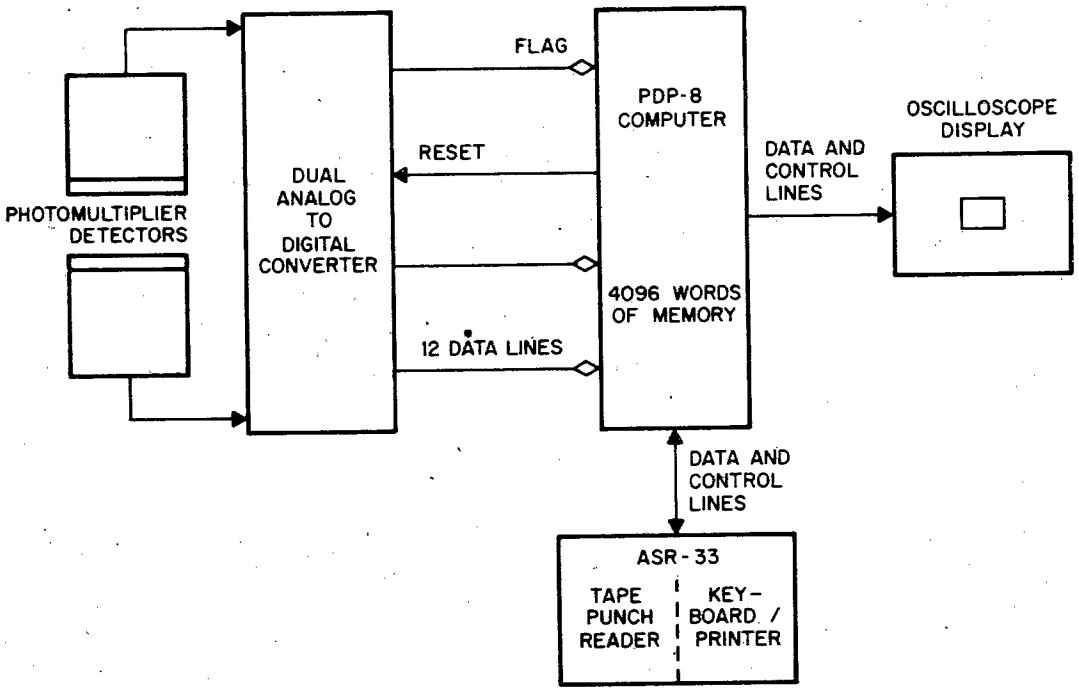


Figure 8. Block Diagram of Pulse Height Analysis Experiment.

CONSTRUCTING THE DETAILED ALGORITHM

The pulse height analysis program, then, will consist of two separate and distinct routines: the display routine to provide the dynamic display, and the interrupt routine, which periodically interrupts the display routine to store new data. The flow charts for these routines are shown in figures 9 and 10, respectively.

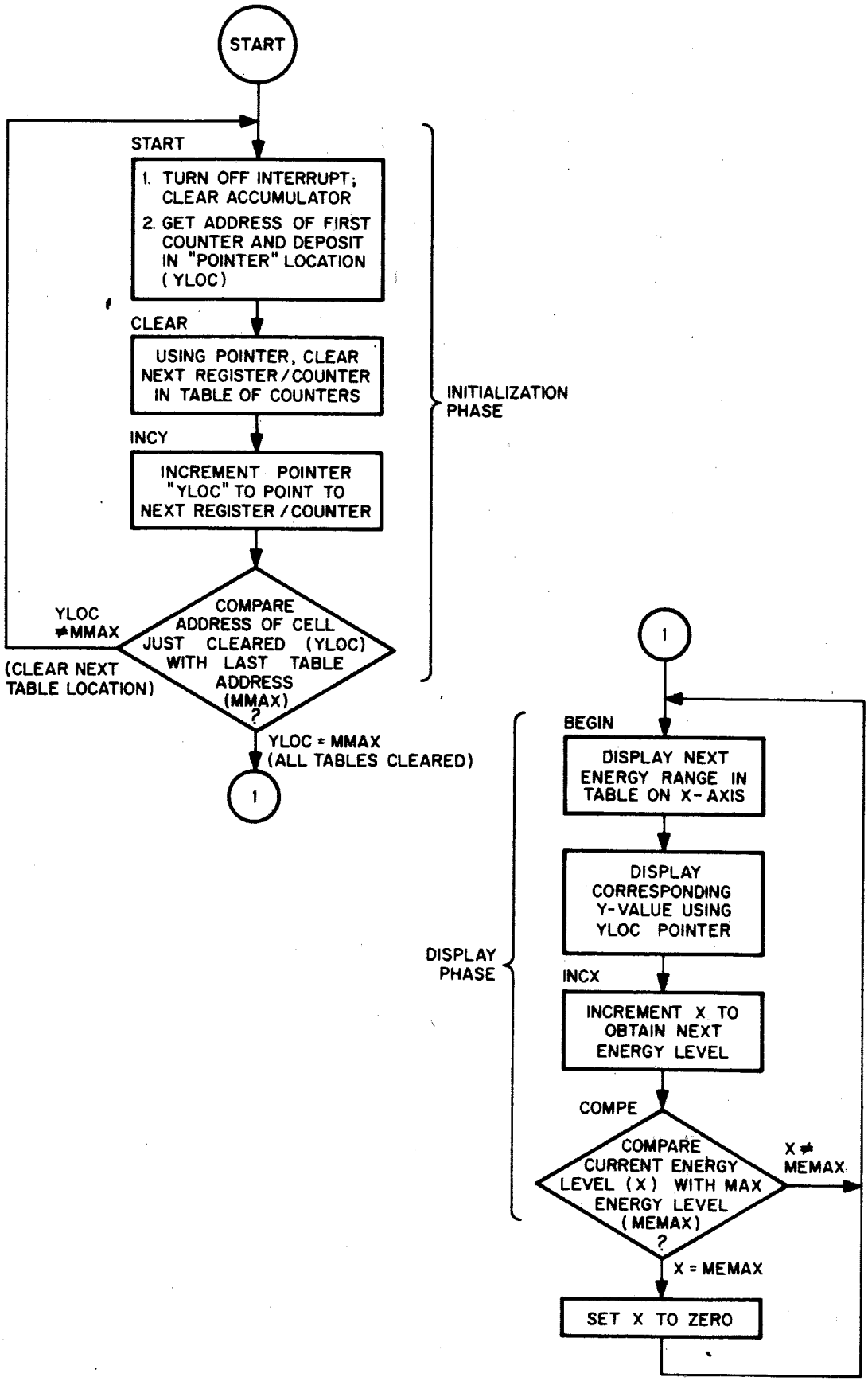


Figure 9. Display Routine.

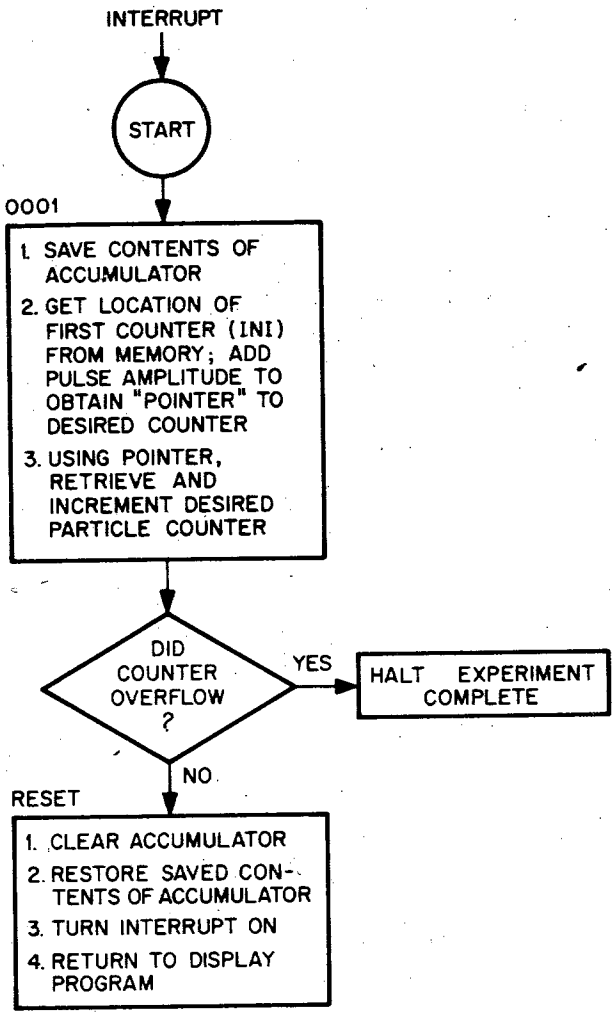


Figure 10. Interrupt Routine.

Note that the display program consists of two phases: the value initiation phase, and the display phase. In the initial phase, the table of memory locations used by the program to count the incident particles is cleared. In the display phase, the number of particles are displayed on the scope as a function of energy.

As a first step in clearing the table of particle-counting registers, we load the accumulator with the address of the first counter within the table and store this address in symbolic location YLOC. YLOC can then be used as a "pointer" to enable clearing of the first counter, as follows:

```
CLA              /CLEAR AC
TAD INI          /GET ADDRESS OF FIRST COUNTER
DCA YLOC         /STORE FOR USE AS POINTER
CLA              /CLEAR AC
DCA I YLOC       /CLEAR COUNTER
```

Similarly, by incrementing YLOC, we can clear the next counter in the table, and so on until all the counters are cleared.

The routine for displaying counter contents is equally simple: for each discrete energy level (X-axis position) the contents of the corresponding counter are displayed on the Y-axis, proceeding from the lowest selected energy level to the highest in a continuous, interruptible loop.

The task of the interrupt routine (figure 10) is to count each particle by incrementing the appropriate memory register. The simplest way to accomplish this is to let the pulse amplitude itself specify the address of the appropriate memory register. Thus, the address of the first counter (symbolic location INI) is added to the binary value of the pulse amplitude, and the sum is used as a pointer to locate and increment the desired register. The interrupt routine compares the resultant counter contents with an arbitrary maximum (full-scale) value — in this case, 1023.

The coding of the complete program — interrupt routine and main routine — is shown in table 6. Labels appended to boxes on the flow charts correspond to symbolic location names within the program to simplify the task of keying the flow charts to the program.

TABLE 6. COMPLETE PULSE HEIGHT ANALYSIS PROGRAM

```
 PULSE HEIGHT ANALYSIS PROGRAM
/RECORDING NUMBER OF PARTICLES (P) AS A FUNCTION OF ENERGY (E).
/PLOTTING NUMBER OF PARTICLES (Y AXIS) VS. ENERGY (X AXIS).
/DATA INPUT WRITING.
0000                        /RESERVED FOR INTERRUPT.
0001        DCA STORE       /SAVE ACCUMULATOR.
            ADRR            /READ ENERGY FROM ADC INTO AC.
            TAD INI         /OBTAIN P LOCATION BY ADDING.
                            /ADDRESS OF FIRST COUNTER.
            DCA TEMP        /STORE P LOCATION FOR INDIRECT
                             ADDRESSING.
            ISZ I TEMP
            JMP RESET
            HLT
/RESET AND RETURN
RESET       CLA             /CLEAR AND
            TAD STORE       /RESTORE AC.
            ION             /TURN ON INTERRUPT.
            JMP I 0         /RETURN TO DISPLAY.
/CLEAR ROUTINE
START,      IOF             /TURN OFF INTERRUPT.
            CLA             /CLEAR AC.
            TAD INI         /GET ADDRESS OF FIRST COUNTER.
            DCA YLOC        /DEPOSIT FOR INDIRECT ADDRESSING.
/CLEAR REGISTERS
CLEAR       CLA             /CLEAR AC.
            DCA I YLOC      /CLEAR COUNTER
/ADVANCE ADDRESS
INCY        TAD YLOC        /READ ADDRESS.
            IAC             /INCREMENT ADDRESS.
            DCA YLOC        /DEPOSIT ADDRESS FOR NEXT LOOP.
/CHECK FOR COMPLETION AND LOOP OR ADVANCE
ENDAD       TAD YLOC        /READ ADDRESS.
```

```
          TAD MMAX      /SUBTRACT MAXIMUM VALUE OF Y
                         LOCATION.
          SZA           /IS IT ZERO?
          JMP CLEAR     /IF NO, CLEAR NEXT COUNTER.
          ION           /IF YES, TURN ON INTERRUPT.
/DISPLAY ROUTINE
BEGIN,    CLA           /CLEAR AC.
          TAD X         /LOAD AC WITH ENERGY RANGE.
          DXL           /DISPLAY ON X AXIS
          TAD INI       /COMPUTE Y LOCATION FOR INDIRECT
                         ADDRESSING.
          DCA YLOC      /STORE FOR INDIRECT ADDRESSING.
          TAD I YLOC    /READ NUMBER OF PARTICLES.
          DYS           /DISPLAY ON Y AXIS AND INTENSIFY.
/ADVANCE ADDRESS
INCX      CLA           /CLEAR AC.
          TAD X         /READ ENERGY.
          IAC           /INCREMENT ENERGY.
          DCA X         /DEPOSIT NEXT ENERGY.
/CHECK FOR FULL SCALE
COMPE     TAD X         /READ NEW ENERGY.
          TAD MEMAX     /SUBTRACT E MAXIMUM.
          SZA           /IS IT ZERO?
          JMP BEGIN     /IF NO DISPLAY NEXT POINT.
          DCA X         /IF YES, SET ENERGY = 0
          JMP BEGIN     /JUMP TO DISPLAY FIRST POINT
/LIST OF CONSTANTS
STORE,      0           /STORAGE REGISTER FOR AC DURING
                        /INPUT ROUTINE.
INI         1000        /LOCATION OF INITIAL DATA REGISTER.
TEMP,          0        /TEMP. STORAGE FOR ADDRESS OF P.
MFS,        -1023       /MINUS FULL SCALE.
YLOC,          0        /TEMP. STORAGE FOR ADDRESS OF Y.
MMAX,       -2024       /MINUS (MAXIMUM Y LOCATION +1)
X,             0        /X
MEMAY,      -1024       /MINUS (E MAXIMUM +1)
```

A PROGRAM TO GENERATE AND DISPLAY A TIME-INTERVAL HISTOGRAM

The time-interval histogram generated and displayed by a computer has become an important biomedical tool in the analysis of neuroelectric and cardiovascular data. The post-stimulus time histogram, for example, is an effective means of revealing the response patterns elicited by controlled experimental stimuli. In brief, it provides an estimate of the relative firing rates of a single unit in successive intervals of time following the presentation of a stimulus.

Another example of the value of the time-interval histogram is in the study of heart action. From the standpoint of the computer, cardiovascular research has much in common with neurophysiological research. The basic data is often a time-voltage function produced by the system under study. Perturbations in the rhythmic activity of the heart may be detectable, and quantifiable, in distributions of interbeat intervals derived from the EKG.

Digital has pioneered in the development of computer applications for biomedical research. The most popular Digital computer for these applications is the LINC-8, specifically designed for the research laboratory. The LINC-8 combines, in a single, self-contained system, all the features of the standard PDP-8 computer, plus the unusual man-machine communication capability of the LINC (Laboratory Instrument Computer). The LINC section includes an oscilloscope display, analog-to-digital converter, mass storage through LINC-tape, relay register, and an instruction repertoire that greatly enhances man-machine communication. The basic LINC computer was developed by the Massachusetts Institute of Technology, supported by grants from the National Institute of Health and the National Aeronautics and Space Administration.

The LINC-8 operates in one of two modes. In one mode, it operates as a standard PDP-8 computer. In the other mode, it operates as a LINC, having certain special input-output and speed characteristics, but otherwise functionally identical to the original LINC. In the following discussion, you will be introduced to some of the basic concepts of programming for the LINC.

LINC MODE PROGRAMMING

As you learned from the previous example of pulse height analysis, the sampling and display of on-line data requires a substantial number of instructions when a conventional computer such as the PDP-8 is used. The LINC-8 greatly simplifies such programming tasks. For example, the single instruction

```
SAM n
```

will cause analog channel n to be sampled, the voltage to be converted to a binary number, and deposited in the accumulator. Again, the single instruction

```
DIS n
```

will cause a spot to be displayed and intensified on the scope. The horizontal

coordinate of the spot will be obtained from the nine rightmost bits of the word at location n, and the vertical coordinate of the spot will be obtained from the nine rightmost bits of the word in the accumulator. If an "i" bit is included in the instruction thusly,

DIS i n

the contents of location n will be incremented by one ("indexed") before the spot is brightened. Of course, the indexing will also increase the horizontal coordinate by one. Which suggests a convenient way to get a horizontal trace across the scope.

The following brief program will display a continuous horizontal line through the middle of the scope (display coordinate = 0) via display channel 0. Memory addresses and instruction codes are shown both in symbolic code and as absolute octal values.

Memory Address	Memory Contents		Comments
	Symbolic	Octal	
5	0	0000	Horizontal Coordinate location
.	.	.	
.	.	.	
.	.	.	
START → 20	CLR	0011	Clear accumulator (Set Vert. Coord. = 0)
21	DIS i 5	0165	Increment horizontal coordinate, and display spot
22	JMP 20	6020	Repeat indefinitely to obtain trace.

Now let's try something a trifle more ambitious and practical — say, a dynamic display of a voltage waveform hooked up to one of the LINC input channels. The following program continuously displays the input from channel 12 until the operator strikes a key.

Memory Address	Memory Content		Comments
	Symbolic	Octal	
1	0	0000	Horizontal Coordinate location
.	.	.	
.	.	.	
.	.	.	
START → 20	SAM 12	0112	Sample line 12 (i.e., Vert. Coord. to accumulator)
21	DIS i 1	0161	Increment Horiz. Coord. and display spot
22	KST	0415	Skip next instruction if key has been struck
23	JMP 20	6020	Key not struck; continue sampling and displaying
24	HLT	0000	Key struck; halt

Thus, the LINC-8 is in very close touch with the "outside world" through its very communicative instructions, which sample input channels, display data, and respond readily to operator commands, through console switches and other controls. LINC programming is somewhat more complex than other 12-bit computers since LINC addressing and control schemes are so much more flexible and extensive. However, the versatility and power of the LINC language should prove to be an ample reward for the additional investment of learning time.

Before describing the time-interval histogram program, it will be necessary to discuss in detail certain LINC instructions. In particular, we will define the formation of the "effective address," and the function of the SET instruction.

The "effective address" of a memory reference instruction is defined as the actual address referred to by the instruction after all address modification and indirect addressing operations have been completed. (Recall that these operations were discussed earlier in the primer.) The LINC uses the i bit in conjunction with location $1\text{-}17_8$ of memory to specify effective addresses for so-called "index class" memory reference instructions. A typical index class memory reference instruction, "load the accumulator" (LDA) has the general form

LDA i B

where i = 1 or 0, and B equals value between 0 and 17. If either i or B is to be set to zero, it is simply omitted from the symbolic code. If B is to be non-zero, the value must be speciified. According to the values assumed by i and B, LINC

assigns the effective address of an index class memory reference instruction as follows:

i (Bit 7)	B (Bits 8-11)	Location of Effective Address
0	0	The contents of bits 1-11 of the register immediately following the instruction.
1	0	The address of the register immediately following the instruction. (The operand itself appears in this register.)
0	$1\text{-}17_8$	The contents of bits 1-11 of register B
1	$1\text{-}17_8$	The contents, incremented by 1, of bits 1-11 of register B

The SET instruction as the form

SET i a

C

and always occupies two consecutive memory locations. As LINC fetches sequential memory instructions, it always skips the second location. If the i bit of the SET instruction equals zero, LINC replaces the contents of the register at memory location "a" with the contents of the register at memory location C. If the i bit of the SET instruction equals one, LINC replaces the contents of the register at memory location "a" with the value C.

We will build our time interval histogram program out of a series of subroutines, each of these subroutines does a specific function (accept data, display a point in a histogram, clear storage areas to start a new histogram, etc.). It is important therefore to understand how these subroutines can be called into play and how they can return control to the correct place in the main program. This is accomplished through the JMP instruction. Whenever JMP instruction is executed, a JMP P + 1 (where P is contents of the program counter) is inserted into location 0. Thus, when we execute JMP 2C, a JMP (P + 1) or JMP 3H is inserted into location ϕ, to exit from the initialization subroutine, then a JMP ϕ is executed and control reverts to location 0. Since location 0 contains a JMP 3H instruction, the program then goes to symbolic location 3H in the main program. Note that when the initialization subroutine is entered from the display subroutine (after results of all bins have been displayed), the JMP 0 in this case reverts to location 4H + 1 since the last JMP executed was from 4H. In a similar manner, all JMP 0 instructions in this sample program provide a return to the main program.

THE TIME-INTERVAL HISTOGRAM PROGRAM

This program shows you how to use the LINC to accumulate a frequency distribution of time between events (the occurrence of a pulse) and to continuously display the distribution as an interval histogram. When an acceptable sample is detected, LINC records the event by incrementing a location in memory, called a "bin", that records the number of samples within the corresponding "class interval" or frequency range. Five hundred and twelve bins are used to accumulate the occurrence of up to 512 separate time intervals. Suppose that we have a histogram with four horizontal graduates or bins of 1-millisecond intervals as shown in figure 11.

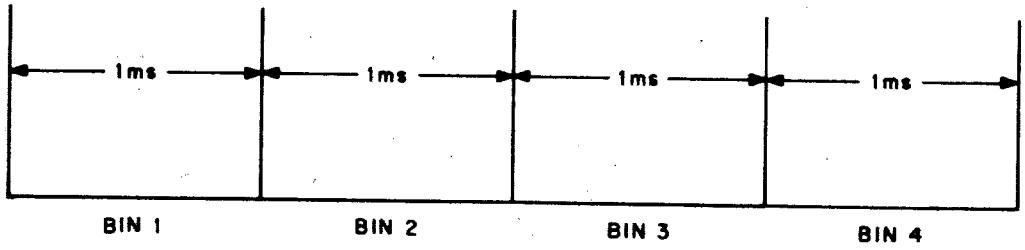


Figure 11. Four 1-Millisecond Interval Bins

When recording the time interval of pulses (events), for all pulse intervals that occur between 0 and 1 milliseconds, a one will be added to bin 1; for intervals of 1 to 2 milliseconds, a one will be added to bin 2, etc. If a pulse train (figure 12) were monitored over a period of time, a histogram of the pulse train would be as shown in figure 13.

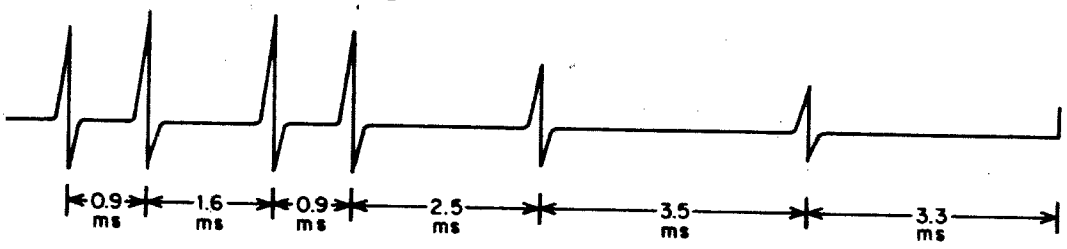


Figure 12. Pulse Train

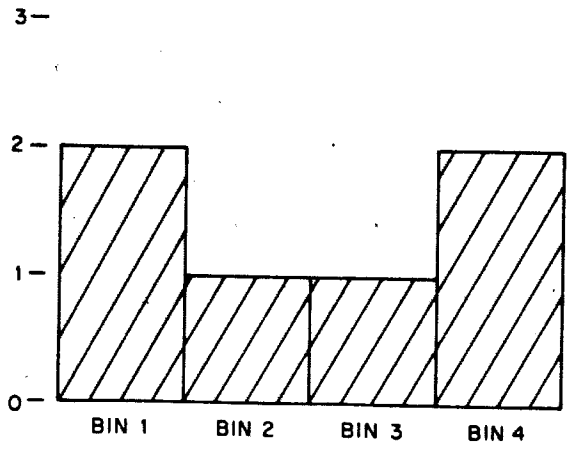


Figure 13. Pulse Train Histogram

Since there are two 0.9-millisecond pulse intervals that fall within the 0 to 1 ms interval, a vertical bar length of two is recorded for bin 1. Similarly, vertical bar lengths of 1, 2, and 1 are recorded for bins 2 through 4, respectively. The foregoing histogram provides a foundation for further perusal of the histogram generating program developed for the LINC-8 programming example. We will retain the bin widths of 1 ms for this example.

We will also use external clock circuitry (external to the LINC-8 equipment, but connected to the LINC-8 as an input/output device) to detect the event and record the time interval. The clock is a digital counter with input detection circuits. An input event stops the digital counter and sets a flip-flop to "flag" the occurrence of an event. The program tests for the occurrence of the event by using a "Skip on External Level" instruction to sense the flag. The program reads the digital counter into the accumulator and resets the flip-flop and the digital counter to 0. The digital counter then starts counting for the next event.

Now let us examine the data collection subroutine which will classify the events into bins of 1 ms intervals. Our histogram will contain 512 bins, hence a classification range of 0-511 ms. (Overflows stop the counter so that **all** intervals greater than 511 ms would increment bin 512 by 1 to indicate the overflow condition.) We will refer to both the flow diagram (figure 14) and subroutine listing for the explanation of the data collection subroutine.

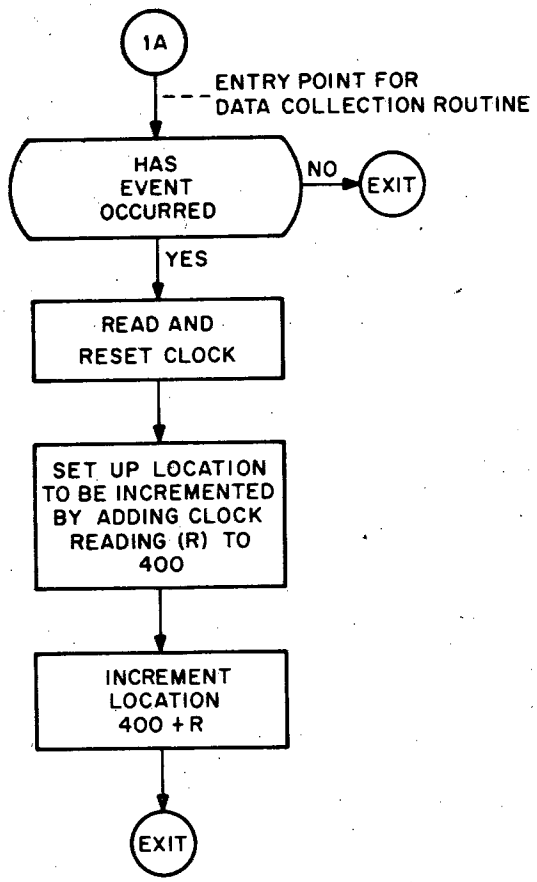
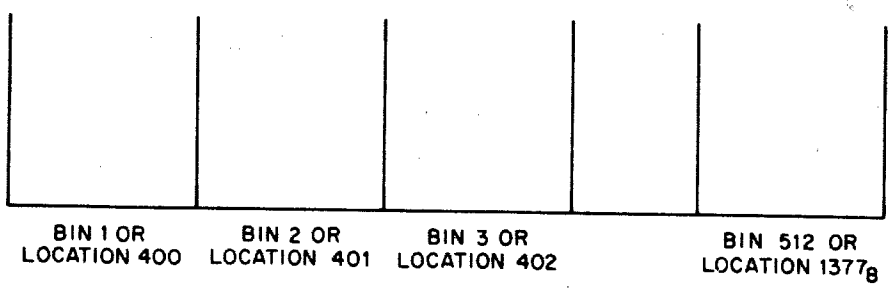


Figure 14. Data Collection Subroutine

DATA COLLECTION SUBROUTINE

```
#1A   SXL   0      / IS THERE AN EVENT
      JMP   0      / NO; EXIT
      OPR   0      / YES; READ AND RESET CLOCK
      ADD   1C     / SET UP ADDRESS FOR INCREMENTING
      STC   1B     / TEMPORARILY STORE ADDRESS OF BIN TO BE
                     INCREMENTED
      LDA   i      / ONE TO ACCUMULATOR
      1
      ADM          / INCREMENT APPROPRIATE BIN
#1B   0            / ADDRESS OF BIN TO BE INCREMENTED STORED HERE
      JMP   0      / EXIT
```

The external clock is connected to the LINC-8 computer via the I/O bus and the clock flip-flop is connected to LINC external line ϕ. Thus, the first instruction in the data collection subroutine, (SXL ϕ) senses line ϕ and skips the next instruction if an event occurred; if the event did not occur, the next instruction JMP ϕ is executed causing control to exit from this subroutine. Assuming an event occurred, we go to the OPR ϕ* instruction which reads the external clock and resets the clock so that it starts recording time for the next interval. After the execution of OPR ϕ the clock reading is held in the LINC accumulator. The next instruction, ADD 1C, adds 400 to the accumulator which has the clock reading and then stores the results into symbolic location 1B. Let us digress a moment to see the significance of this.



---

* The OPR instruction is executed by the PDP-8 section of the LINC-8 computer; when the LINC detects the OPR instruction, it alerts the PDP-8 computer via computer interrupt. The PDP-8 executes the instruction through a PDP-8 subroutine that reads the clock, resets the clock and flag, and transfers the clock reading to the LINC accumulator. To the LINC programmer, it appears that the LINC executed the instruction.

If we let location 400 be the address of bin 1, and 401 bin 2, etc., then by adding 400 to the clock reading and storing the results into symbolic location #1B, location #1B contains the address of the appropriate bin.

Going back to the subroutine, the next instructions add one to the addressed bin and control exists. Hence, if the time-interval read from the clock was 2.1 ms, then location #1B would address 402 or bin 3, and bin 3 is incremented by one. Note that each entry into the data collection subroutine processes only one event (if it occurs). If we connect the subroutines so that we continually loop through the data collection subroutine, we will accumulate data counts for those bins corresponding to the number of pulse interval detections. For example, over a period of time, 5 events of 3.5 ms were detected, then bin 4 (location 405), which holds the 3-4 ms intervals, contains a 5.

Suppose we let the data collection subroutine collect data over a period of time and then in some manner terminate the subroutine. We, in effect, have our histogram stored in locations 400-1377_8. Now, we want to display our histogram. The display subroutine will accomplish this.

Let us first examine the display initialization subroutine (Figure 15) which sets up the display subroutine. The SET i 1 and 400 instruction combination sets index register 1 to a value of 400. Similarly, index register 2 is set to 0 so that the histogram display starts at the left side of the CRT at a horizontal coordinate of 000. The LDA 1 instruction loads the accumulator with the content of memory location 400 which contains the count of bin 1 (the height of the bar to be displayed). To this height we add —377, so that the vertical display will start at the bottom of the CRT. The height is then stored in symbolic location 2B.

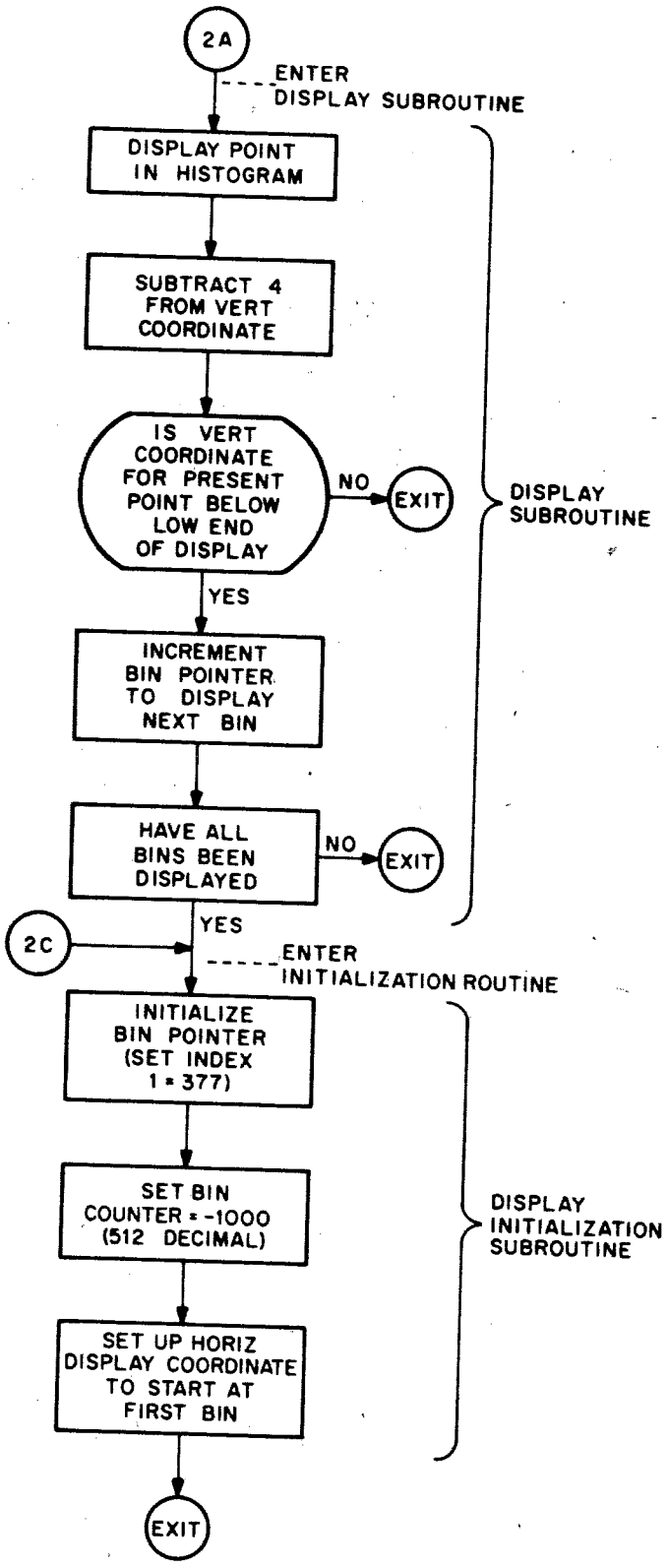


Figure 15. Display and Display Initialization Subroutines

POINT DISPLAY OF HISTOGRAM SUBROUTINE

```
2A      LDA          /VERTICAL COUNT TO ACCUMULATOR
        2B
        DIS 3        /GENERATE DISPLAY
        ADA i        /SUBTRACT 4 FROM ACCUMULATOR
        —4
        STA          /STORE VERTICAL COORDINATE FOR NEXT DISPLAY
        2B
        ADA i        /ACCUMULATOR WILL BE NEGATIVE IF NEXT DIS-
        377           PLAY IS BELOW BOTTOM EDGE OF SCOPE
        APO i        /SKIP IF ACCUMULATOR NEGATIVE POINT IS BE-
                      LOW BOTTOM EDGE OF SCOPE
        JMP 0        /NO; EXIT
        LDA i 1      /LOAD COUNT OF VERTICAL COORDINATE FOR
                      NEXT BIN
        STA i        /AND STORE IN 2B
2B      0000
        CLR          /CLEAR ACCUMULATOR
        XSK i 3      /MOVE HORIZONTAL COORDINATE
        XSK i 2      /HAVE ALL BINS BEEN DISPLAYED?
        JMP 0        /NO; EXIT
Display Initialization Subroutine
2C      SET i 1      /SET INITIAL TABLE ADDRESS
1C      400          /DATA BEGINS IN 400
        SET i 2      /SET COUNTER FOR 512 BINS
        —1000
        LDA 1        /LOAD COUNT FOR FIRST BIN
        ADA i        /BIN COUNTS DISPLAYED FROM BOTTOM OF SCOPE
        —377
        STC 2B       /LOAD VERTICAL COORDINATE
        SET i 3      /SET HORIZONTAL COORDINATE
          0
        JMP 0        /EXIT
```

The display subroutine has been set up by the display initialization subroutine so that it starts with bin 1 or location 400. Upon entry into the display subroutine at symbolic location #2A, we load the accumulator with the vertical coordinate of the first bin. The DIS 3 instruction is issued to generate a display; the DIS 3 instruction takes the vertical coordinate from the accumulator and horizontal coordinate from index register 3 which was initially set to equal 0, thus an intensified spot appears at a CRT horizontal coordinate of 000 with the vertical coordinate as specified by the content of bin 1.

Next, we subtract 4 from the vertical coordinate. This causes the display subroutine to skip 4 vertical CRT coordinate positions between points in the vertical bar for each entry into the subroutine. This reduces the overall display program time required to display a complete histogram. This is not detrimental to the display since the bar appears to be continuous.

It should be noted that since the maximum vertical coordinate is +377 and the vertical coordinates of the histogram are obtained by incrementing a memory location, if the data collection subroutine obtains a count in excess of 777_8 for any bin, the display will show a full length line independent of how much in excess the count in that bin was.

We next test to see if the complete bar of the histogram has been displayed. This is accomplished by adding 377 to the vertical coordinates. If the vertical coordinate is in the interval —377 to 377, a negative number results from the addition (i.e., if the vertical coordinate is below the bottom line, the result of the addition is negative). Therefore the APO i senses for a positive accumulator; if positive, we exit from the subroutine and re-enter later to complete the present vertical bar. If negative, we have completed the vertical bar and must go to the next bin to obtain the vertical coordinate of that bin. This is accomplished by the LDA i 1 which indexes (since i= 1) location 1 to a value of 401 (the second bin) and then loads the accumulator with the content of 401, the vertical coordinate of the second bin. This coordinate is stored in location #2B which is used by display subroutine to obtain the vertical coordinate. We must now increment the horizontal display coordinate; this is accomplished by the XSK i 3 which increments location 3 (the horizontal coordinate). To test for the end of the histogram, the routine executes an XSK i 2 which adds 1 to location 2, skips the next instruction if bits 2 thru 11 of location 2 equal 1777_8. If all bins have not been displayed, we will not skip; therefore, we exit. If all bins have been displayed location 2 (bits 2 thru 11) contains 1777_8 and we skip the next instruction and enter the display initialization routine to initialize the display routine so that we can redisplay the histogram.

We have not discussed the core initialization subroutine (Figure 16); This subroutine serves to clear all bins in corememory so that a new histogram may be accumulated and stored. This is accomplished by setting index register 1 (memory location 1) to 377 and index register 2 to —1000 (the octal number of bins in the histogram), the accumulator is cleared. The STA i 1 instruction increments the content of index register 1 (the first increment is to 400, the start of our histogram), and then stores a zero into the addressed location. We loop through this routine until index register 2 contains 1777; we then return to the main program.

We next test to see if the complete bar of the histogram has been displayed. This is accomplished by adding 377 to the vertical coordinates. If the vertical coordinate is in the interval —377 to 377, a negative number results from the addition (i.e., if the vertical coordinate is below the bottom line, the result of the addition is negative). Therefore the APO i senses for a positive accumulator; if positive, we exit from the subroutine and re-enter later to complete the present vertical bar. If negative, we have completed the vertical bar and must go to the next bin to obtain the vertical coordinate of that bin. This is accomplished by the LDA i 1 which indexes (since i = 1) location 1 to a value of 401 (the second bin) and then loads the accumulator with the content of 401, the vertical coordinate of the second bin. This coordinate is stored in location #2B which is used by display subroutine to obtain the vertical coordinate. We must now increment the horizontal display coordinate; this is accomplished by the XSK i 3 which increments location 3 (the horizontal coordinate). To test for the end of the histogram the routine executes an XSK i 2 which adds 1 to location 2 skips the next instruction if bits 2 thru 11 of location 2 equal 1777_8. If all bins have not been displayed, we will not skip; therefore, we exit. If all bins have been displayed, location 2 (bits 2 thru 11) contains 1777_8 and we skip the next instruction and enter the display initialization routine to initialize the display routine so that we can redisplay the histogram.

We have not discussed the core initialization subroutine (Figure 16); This subroutine serves to clear all bins in core memory so that a new histogram may be accumulated and stored. This is accomplished by setting index register 1

(memory location 1) to 377 and index register 2 to —1000. (The octal number of bins in the histogram). The accumulator is cleared. The STA i 1 instruction increments the content of index register 1 (the first increment is to 400, the start of our histogram and then stores a zero into the addressed location. We loop through this routine until index register 2 contains 1777; we then return to the main program.

We have now seen how to accumulate data for the histogram, display the histogram, and initialize core storage for the data collection and histogram display. Now we will combine all these subroutines with a main program which calls for these routines under manual control of the sense switches. (The sense switches are front panel controls which the program can sense via the SNS instruction to change the programming sequence). From Figure 17, we see that after starting, we enter subroutines that initialize the data collection and display subroutines. Assuming sense switch 1 is in down position, we loop through the "accumulate data" and "display" subroutines. This loop permits us to accumulate data for this histogram, with an apparently concurrent display of the generated histogram. When enough data has been collected, we can terminate the data collection by setting sense switch ϕ to the up position. We can continue displaying the histogram by leaving sense switch 1 in the down position. When we want to generate a new histogram, we set both sense switches 0 and 1 to the up position so that we enter 1H to clear out the old histogram from core memory and to initialize the display subroutine. We then set sense switch 1 down and ϕ down so that we loop through the "accumulate data" subroutine.

CORE INITIALIZATION
SUBROUTINE

```
1Z      SET i 1         /INITIALIZE POINTER TO FIRST LOCATION OF
        377              HISTOGRAM
        SET i 2         /SET NUMBER OF BINS
        —1000           /1000₈ = 512₁₀
        CLR             /CLEAR ACCUMULATOR
        STA i 1         /CLEAR NEXT BIN
        XSK i 2         /ADD 1 TO LOCATION 2; SKIP WHEN CONTENTS OF
                         LOCATION 2 >= 1777₈ (BITS 2 THROUGH 11)
        JMP p-2         /JUMP BACKWARD TWO LOCATIONS (CONTINUE
                         CLEARING)
        JMP 2H.         /RETURN TO MAIN PROGRAM TO SETUP DISPLAY
                         COUNTERS AND POINTERS
```

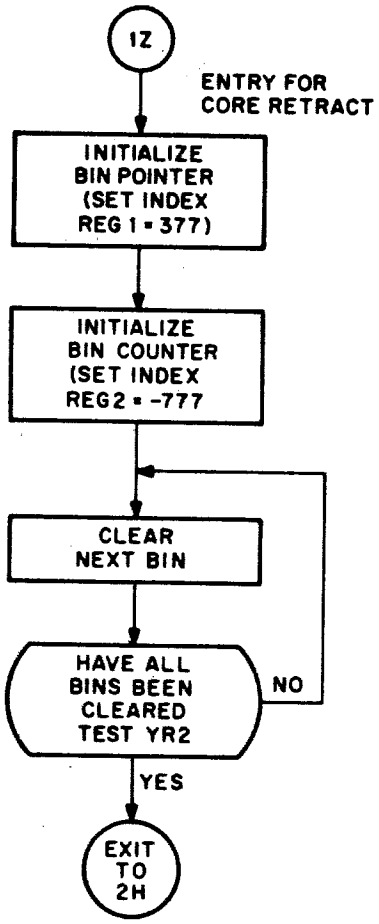


Figure 16. Core Initialization Subroutine

MAIN PROGRAM — TIME INTERVAL HISTOGRAM

```
1H    JMP   1Z        /CLEAR TABLES
2H    JMP   2C        /SET UP COUNTERS AND POINTERS
3H    JMP   1A        /GET DATA
4H    JMP   2A        /DISPLAY ONE DATA POINT
      SNS   0         /SKIP IF SENSE SWITCH 0 IS UP
      JMP   3H        /GET NEXT POINT
      SNS   1         /SKIP IF SENSE SWITCH 1 IS UP
      JMP   4H        /STATIC DISPLAY SELECTED
      JMP   1H        /0 AND 1 UP; RESTART
```

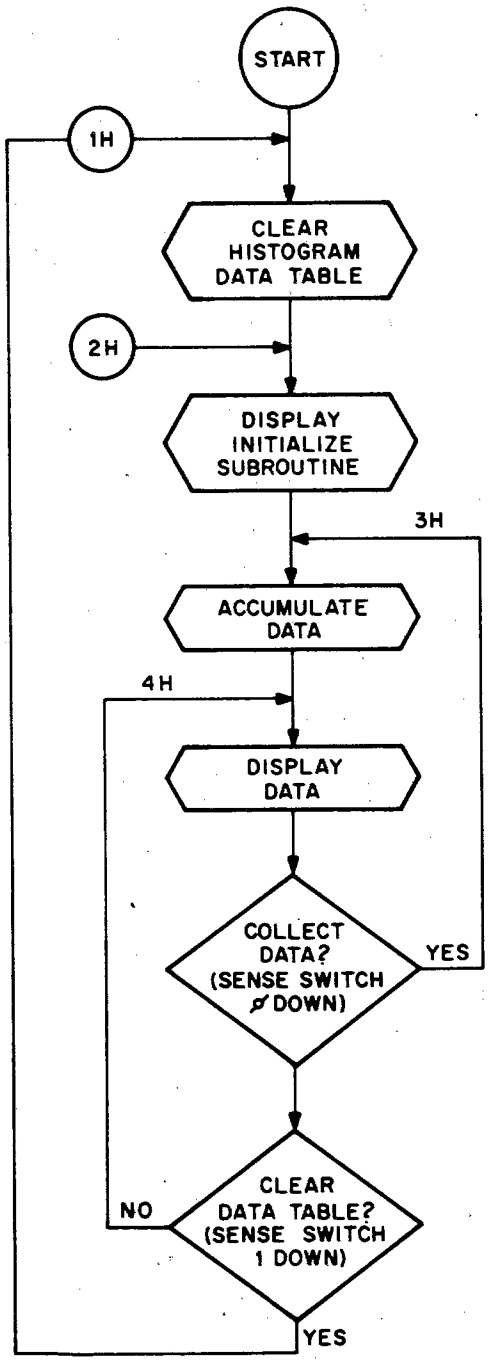


Figure 17. Main Program

COMPUTER-DIRECTED PROCESS CONTROL TECHNIQUES

Among the most rapidly developing applications of digital computers are those requiring automatic control of machines and processes. In industrial process control, for example, digital computers are displacing conventional analog controllers and other components of conventional control loops with ever-increasing frequency.

Before we explore the techniques of computer-directed process control, let's examine the characteristics of the conventional process control loop. Figure 18 shows such a system for controlling a single process-variable. The measured process-variable signal (pressure, temperature, flow, etc.) is compared to the value entered by the operator to determine the magnitude and direction of the error. The error signal then serves as input to a small, special-purpose analog device or controller. The controller calculates the valve position that will reduce the steady-state error to zero.

The equations solved by the controller will usually be one of the following forms:

$$\text{Valve position} = K_0 + K_1 e \quad \text{(proportional control P)}$$

$$\text{Valve position} = K_0 + K_1 e + K_2 \int_0^t e\,dt \quad \text{(proportional integral control PI)}$$

$$\text{Valve position} = K_0 + K_1 e + K_2 \int_0^t e\,dt + K_3 (de/dt) \quad \text{(proportional derivative control PID)}$$

Where K_0 = initial valve position

K_1 = proportional-term adjustment

K_2 = integral term adjustment

K_3 = derivative term adjustment

e = error (i.e., measured variable — set point)

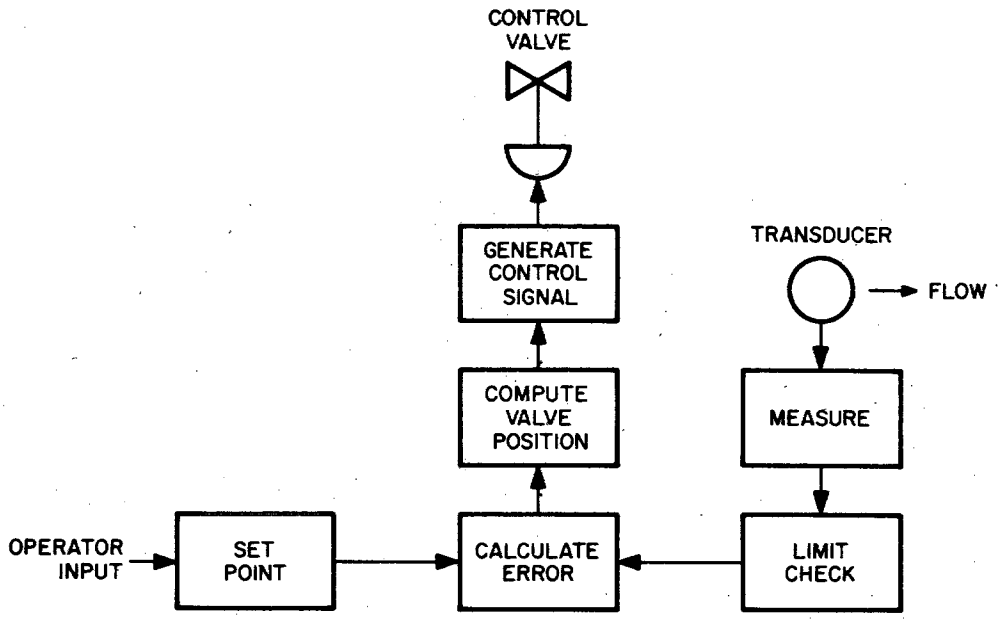


Figure 18. Single Process-Variable System

The adjustment terms (K_0, K_1, K_2, and K_3) permit adaptation of the control loop to a broad range of processes having different control characteristics. By varying the adjustment terms, the controller can be "tuned" to the process characteristics. Since each controller solves only a single loop equation, large conventional process control systems require a correspondingly large number of controllers. The control engineer must choose the equation, and thus the controller, which best fits the process. Once an analog controller has been installed, an equipment change is required if process characteristics change significantly.

Now let us examine a process control system directed by a digital computer such as the PDP-8 (figure 19). In this system, the control loop variables are applied to a single multiplexer, permitting the computer to monitor all variables within the system on a time-shared basis. The control loop algorithms are solved by computer subroutines instead of analog controllers. The subroutines compute the required error signals, which are converted to analog voltage swings and multiplexed to the appropriate valve-positioning device.

You can readily see some of the advantages of digital computer control over the conventional analog controller: to change the control algorithm, you have only to change parameters within the computer program; a costly and time-consuming redesign of the equipment is not required. To increase the number of control loops, you need add only a transducer and actuator or control valve for each additional loop. Addition of such features as limit checking of valve positioning to prevent damage can easily be effected, since the program can be written to clamp the error signal to specified limits before sending it the valve positioning device. Changes of tuning adjustments, alarm limits, and set points can easily be accomplished with conventional computer input devices such as typewriters. In most cases, however, these inputs will be channeled through a specially designed operator console.

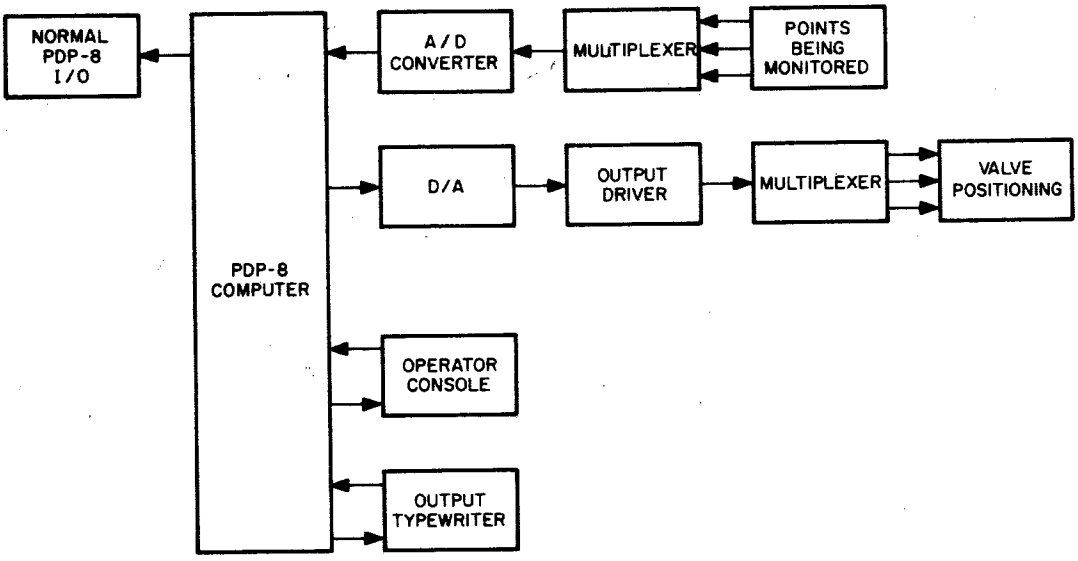


Figure 19. Computer-Directed System

Although digital computers have been in use in process control for over a decade, their inclusion as the computational element in the control loop (direct digital control) is a comparatively recent phenomenon. The early computers in process applications operated at a speed of about one thousand operations per second. Computers of this speed are presently guiding the operations of chemical plants and refineries, starting up power plants, and replacing conventional controls in case of failures. These machines made no attempt to compete with standard controls; the amount of computer time required would have produced too severe a drain on the computational capacity of the system. These systems justified their existence in process control by performing functions of which standard analog controls were incapable.

Today, computers such as the PDP-8 operate at speeds which are more than sufficient to provide direct digital control for every loop in a typical process control system. Moreover, as the speed has increased the price of computers has decreased, so that today a redundant two-computer PDP-8 system costs about the same as an earlier single-computer system.

A PROCESS CONTROL SYSTEM USING DIRECT DIGITAL CONTROL

To achieve direct digital control of all loops within a process system such as the one shown in figure 19, we require an "executive" program which will sequentially scan or monitor the field points (measured variables). Let's examine, to begin with, the over-simplified executive control program of figure 20. Here, a program called the level executor supervises the allocation of processing time to the various programs and subroutines that scan the measured variables, effect the required control loop computations, service operator requests, and output timely messages to assist him in his control decisions. Transfer of control between major system programs is accomplished as follows. A program sets a request for entry into another program, then transfers control to the level executor. The level executor either grants the transfer

request immediately, or places it in a "queue" for later servicing, depending upon the relative priorities of the program in progress and the requested program.

Typically, a real-time clock is used to initiate the scanning of the measured variables, or "field points." The interrupt handler program services the real-time clock interrupt by requesting entry into the scan initiation and multiplexing (SCAN) level program. SCAN directs the hardware multiplexer to start multiplexing the field points.

When the multiplexer has obtained the field point reading, it generates an interrupt to reenter SCAN. When reentered, SCAN linearizes the field point reading. If the point is outside specified limits, SCAN requests entry into the message compiler and output program to output the alarm message to the control process operator. SCAN then requests entry into the algorithm processor (ALGO), and initiates multiplexing of the next field point. Using the previously acquired point reading, ALGO computes the error signal and requests entry into the Output Driver program to provide the output to the processing system.

The operator console handler enables the operator to change parameters for control computations and field points, such as set points and alarm limits.

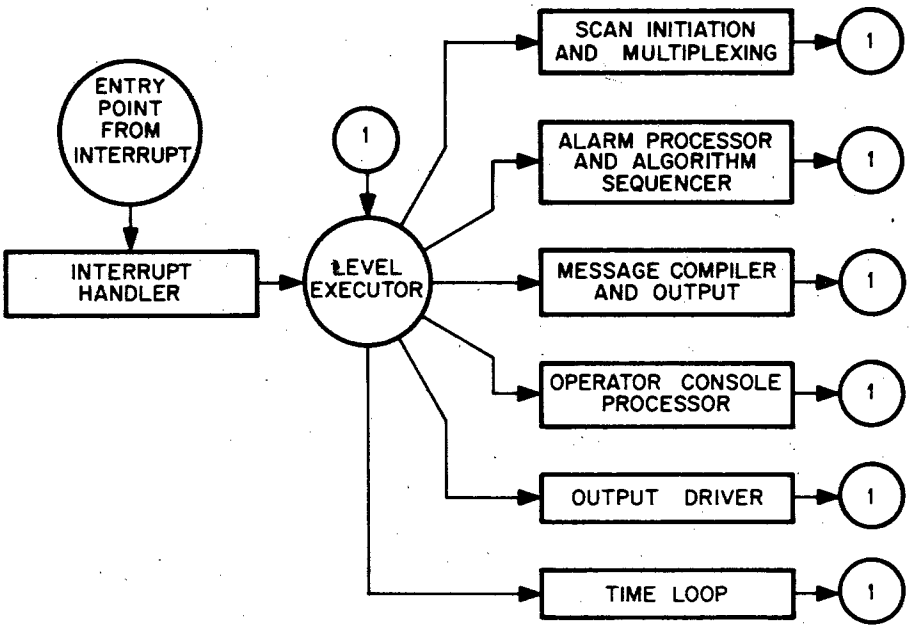


Figure 20. Process Control Software System

It is useful to divide the several system programs into four priority levels, as follows. To the first (highest) priority level, we assign the programs that perform the principal functions of the conventional analog controller; i.e., field point scanning and error signal output. To the second priority level, we assign such functions as error signal computation. The lower two priority levels perform such functions as message output and servicing operator requests. Processing time for lower level programs is normally allocated during "wait" periods, as, for example, when the program is waiting for the multiplexer to return the measured variable.

TIMING THE POINT-SCANNING PROCESS

One of the principal differences between direct digital control and analog control is that the analog controller continuously monitors the process-variable, while in direct digital control systems, the process-variable is monitored periodically. Thus, the control engineer is required to determine how frequently the control loop should be monitored. When this parameter has been established, it is used to set the real-time clock, which periodically interrupts the computer and initiates a timing program.

The timing program initiates the scanning of all field points through a sequencing program (SEQ). The timing program also schedules execution of the other time-based level programs, as shown in figure 21. Note that since the timing program is initiated periodically, it is a simple matter to implement a time-of-day clock. The time-of-day clock may be used for logging and data collecting operations. For example, if an alarm condition occurs, the alarm message generated on the output typewriter can log the time of occurrence by referencing the time-of-day clock.

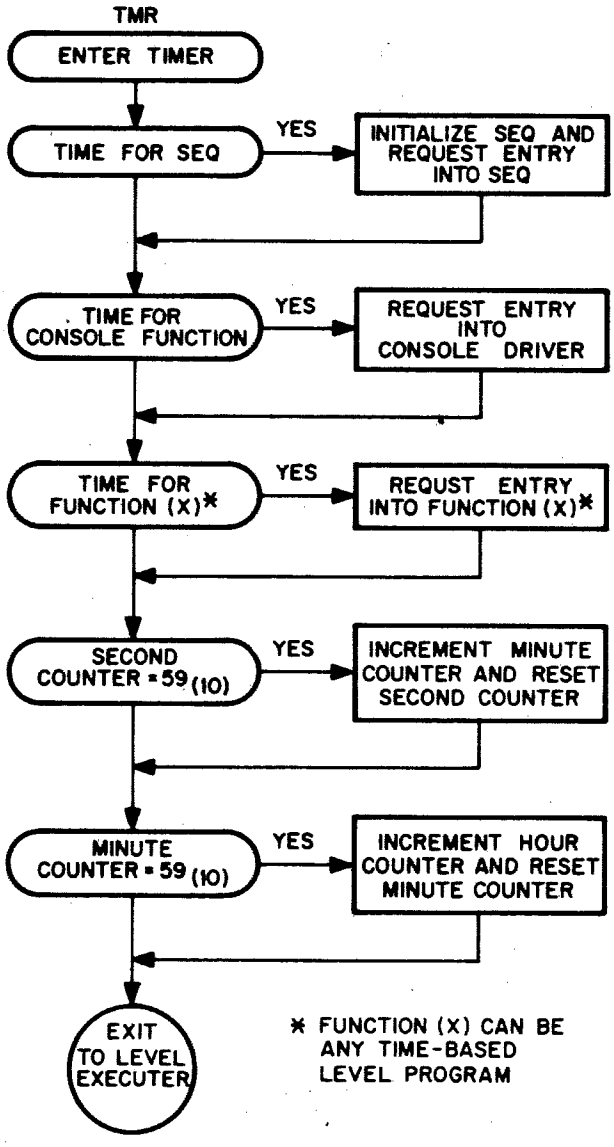


Figure 21. The Timing Program

Sequencing and Multiplexing the Point-Scanning Process

The point-scanning process (figure 22) is initiated by a sequencing program (SEQ). SEQ maintains a list or table of all points to be scanned. The table describes such parameters as its point number or multiplexer address. To sequentially select field points for multiplexing, a pointer cell is used to indicate the present field point being processed. The pointer is initialized to point to the first word in the list of field points. The table is completely scanned each period of the real-time clock. SEQ obtains the multiplexer address of the point being processed, inserts the address into a location or "slot" within the Multiplexer program (MPXR), then increments the pointer so that on the next entry into SEQ, the next pointer will be processed. SEQ then requests entry into MPXR to obtain the measurement.

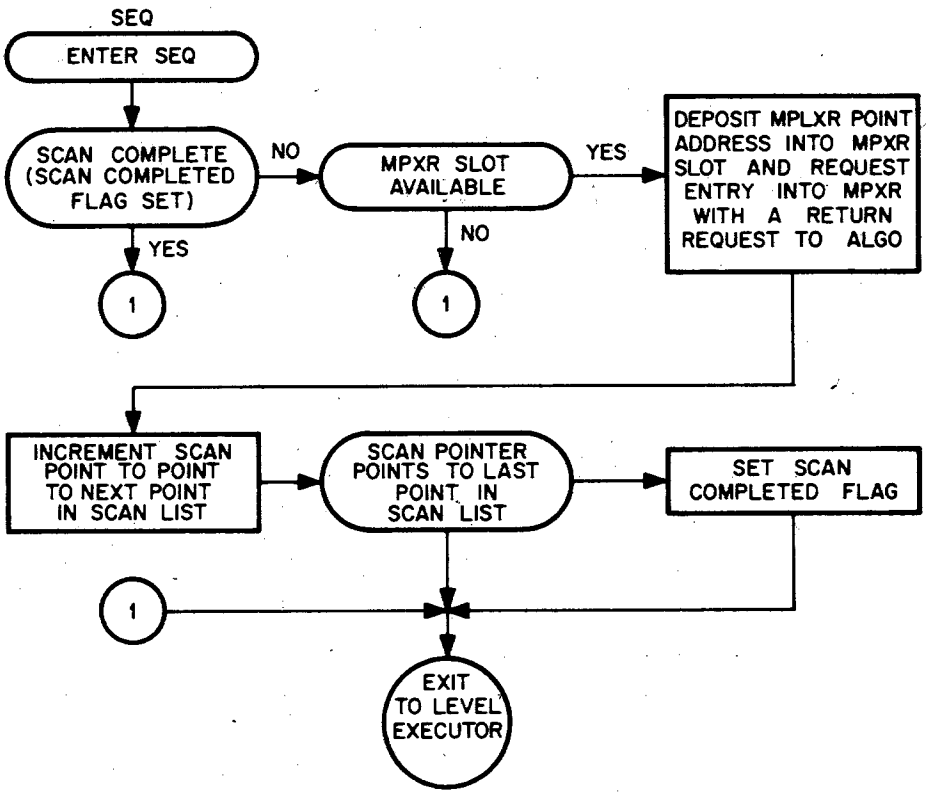


Figure 22. Point-Scanning Process

The multiplexer program sets up the hardware multiplexer to obtain the measurement, then transfers control to the level executor. When the multiplexer hardware has obtained the measurement, an interrupt is generated, and the measurement is set into MPXR. MPXR then recalls the program that requested the measurement. However, when SEQ is the requesting program, control is transferred to the Algorithm Processor.

PROCESSING THE ALGORITHM

The algorithm processor (figure 23) solves the basic proportional, rate, and reset control equations shown earlier in this discussion.

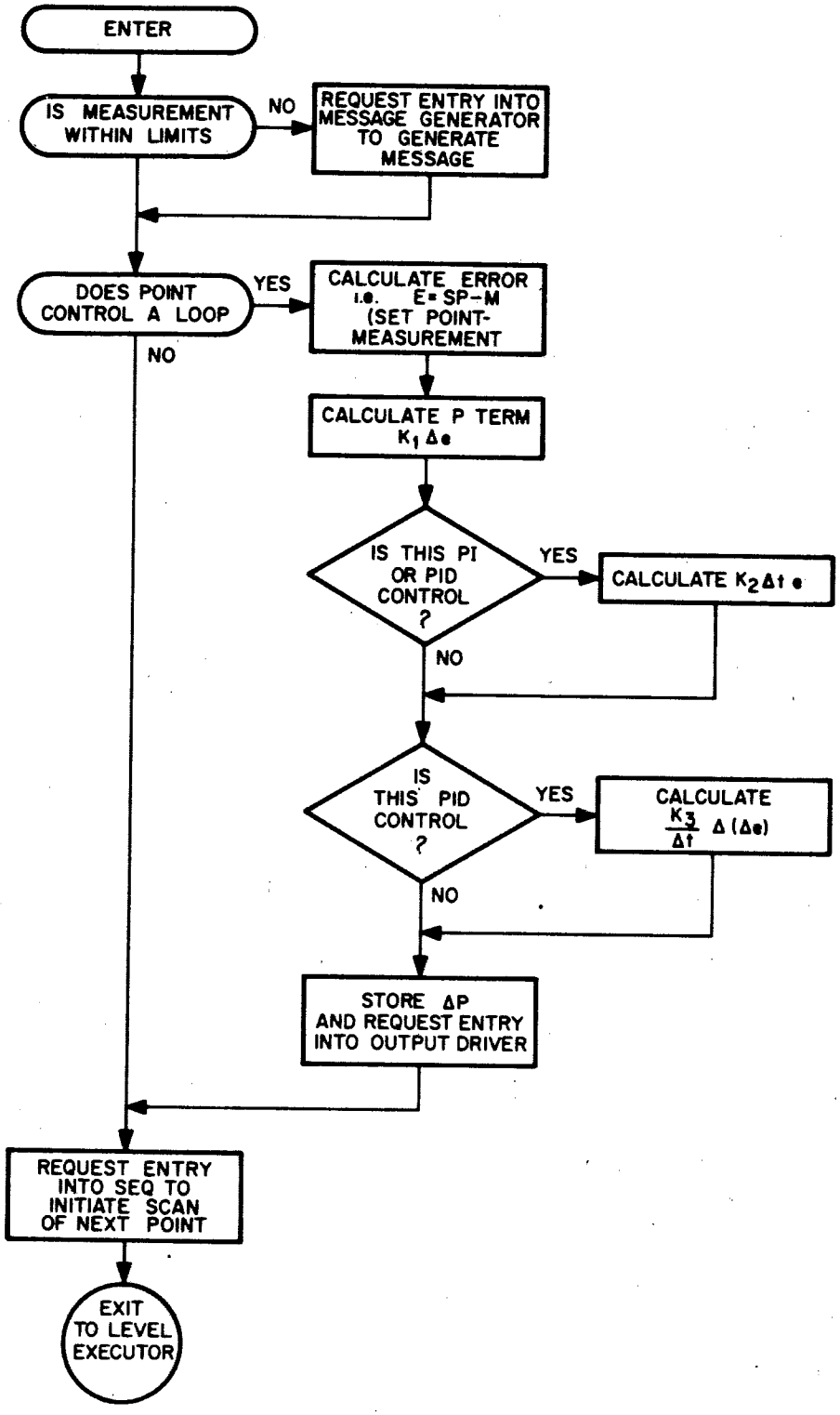


Figure 23. Algorithm Processor

Moreover, feed-forward and combination systems can be used more widely, since the only elements which must be added to the system are transducers and mathematical expressions. In brief, computer control of processes enables you to engineer the control system, rather than requiring you to select an alternative solution from among the available analog devices.

However, we cannot solve the equations for the proportional, proportional integral, and proportional derivative (P, PI, and PID) algorithms in as straightforward a manner as the analog controller, since the computer can only add or subtract. Since the terms of the P and PI equations also appear in the PID algorithm, we will show the implementation of the PID algorithm for direct digital control. The PID algorithm is defined as:

$$P = K_1 e + K_2 \int^{e} dt + K_3 \frac{de}{dt} \tag{1}$$

where P represents the steady state value position and e the steady state error, which is the difference between set point and measurement.

Since we monitor the field points at discrete intervals of time (Δ_t), we can represent the PIP equation as follows:

$$\frac{\Delta p}{\Delta t} = K_1 \frac{\Delta e}{\Delta t} + K_2 e + K_3 \frac{\Delta \left(\frac{\Delta e}{\Delta t}\right)}{\Delta t} \tag{2}$$

Where Δt is the sampling interval and Δ_e is the difference between e signals from present sampling and the previous sampling period.

Equation (2) can be reduced to:

$$\Delta p = K_1 \Delta e + K_2 \Delta t\, e + \frac{K_3}{\Delta t} \Delta(\Delta e) \tag{3}$$

where Δ (Δe) is the second order change in error signal, i.e.

$$\Delta(\Delta e) = (\Delta e)_n - (\Delta e)_{n-1}$$

Equation (3) represents PID equation in a form that can be computed by the program. Note that the $K_2\ \Delta t$, and $\frac{K_3}{\Delta t}$ terms can be considered constants since we know the sampling interval Δt. From equation (3) the P and PI equations are as follows:

$$\text{P algorithm} = \Delta P = K_1 \Delta e \tag{4}$$

$$\text{PI algorithm} = \Delta P = K_1 \Delta e + K_2 \Delta t e \tag{5}$$

Now we have the P, PI, and PID equations in forms that are readily resolved by the computer. Furthermore, since the terms are common for all three, we can use one routine (figure 27) to calculate the algorithms for all control loops in the system. The constants for the control loops are stored in each

control loops respective table entries.

As shown in figure 27, the P algorithm is calculated first since its term appears in all three algorithms; hence, if the loop requires PI control, the $K_1\Delta e$ term is computed an added to the $K_2\Delta te$ term; if the control loop requires P control, the $K_2\Delta te$ and $\frac{K_3}{\Delta t}\Delta^{(\Delta e)}$ calculations are by passed.

PART III: PDP-8/I USERS HANDBOOK

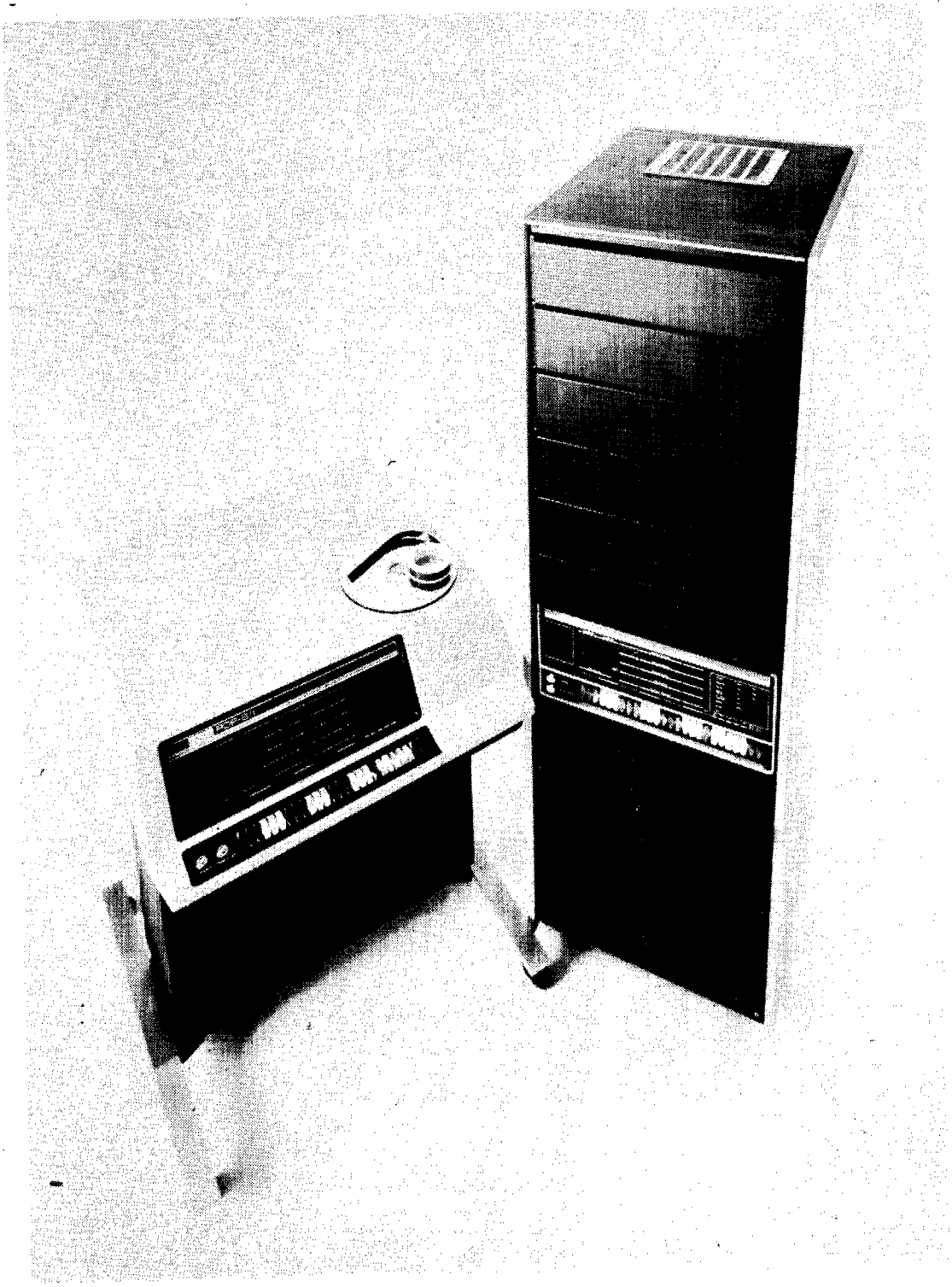


Figure 1. Typical PDP-8/I in Table Top and Rack Mounted Configurations

CHAPTER 1
SYSTEM INTRODUCTION

The Digital Equipment Corporation Programmed Data Processor-8/I (PDP-8/I) is a small-scale general-purpose computer. TTL monolithic integrated circuit modules are used throughout thereby providing efficient packaging, high reliability and noise immunity. The PDP-8/I is a one-address, fixed word length, parallel computer using 12 bit, two's complement arithmetic. Cycle time of the 4096-word random-address magnetic-core memory is 1.5 microseconds. Standard features of the system include indirect addressing and facilities for instruction skipping and program interruption as functions of input-output device conditions.

The 1.5-microsecond cycle time of the machine provides a computation rate of 333,333 additions per second. Addition is performed in 3.0 microseconds (with one number in the accumulator) and subtraction is performed in 6.0 microseconds (with the subtrahend in the accumulator). Multiplication is performed in approximately 315 microseconds by a subroutine that operates on two signed 12-bit numbers to produce a 24-bit product, leaving the 12 most significant bits in the accumulator. Division of two signed 12-bit numbers is performed in approximately 444 microseconds by a subroutine that produces a 12-bit quotient in the accumulator and a 12-bit remainder in core memory. Similar multiplication and division operations are performed by means of the optional extended arithmetic element in approximately 6.0 and 6.5 microseconds, respectively.

Flexible, high-capacity, input-output capabilities of the computer allow it to operate a variety of peripheral equipment. In addition to standard Teletype and perforated tape equipment, the system is capable of operating in conjunction with a number of optional devices such as high-speed perforated tape readers and punches, card equipment, a line printer, analog-to-digital converters, cathode-ray-tube displays, magnetic drum systems, magnetic-tape equipment, and a 32,000 word random access disc file. Equipment of a special design is easily adapted for connection into the PDP-8/I system. The computer does not have to be modified when peripheral devices are added.

The PDP-8/I is completely self-contained, requiring no special power sources or environmental conditions. A single source of 115-volt, 60-cycle, single-phase power is required to operate the machine. Internal power supplies produce all of the operating voltages required. Modules utilizing TTL monolithic integrated circuits and built-in provisions for marginal checking insure reliable operation in ambient temperatures between 32 and 130 degrees Fahrenheit.

COMPUTER ORGANIZATION

The PDP-8/I system is organized into a processor, core memory, and input/output equipment and facilities. All arithmetic, logic, and system control operations of the standard PDP-8/I are performed by the processor. Permanent (longer than one instruction time) local information storage and retrieval operations are performed by the core memory. The memory is continuously cycling, automatically performing a read and write operation during each computer cycle. Input and output address and data buffering

for the core memory is performed by registers of the processor, and operation of the memory is under control of timing signals produced by the processor. Due to the close relationship of operations performed by the processor and the core memory, these two elements are described together in this chapter of this handbook.

Interface circuits for the processor allow bussed connections to a variety of peripheral equipment. Each input/output device is responsible for detecting its own select code and for providing any necessary input or output gating. Individually programmed data transfers between the processor and the peripheral equipment take place through the processor accumulator. Data transfers can be initiated by peripheral equipment, rather than by the program, by means of the data break facilities. Standard features of the PDP-8/I also allow peripheral equipment to perform certain control functions such as instruction skipping, and a transfer of program control initiated by a program interrupt.

Standard peripheral equipment provided with each PDP-8/I system consists of a Teletype Model 33 Automatic Send Receive set and a Teletype control. The Teletype unit is a standard machine operating from serial 11-unit-code characters at a rate of ten characters per second. The Teletype provides a means of supplying data to the computer from perforated tape or by means of a keyboard, and supplies data as an output from the computer in the form of perforated tape or typed copy. The Teletype control serves as a serial-to-parallel converter for Teletype inputs to the computer and serves as a parallel-to-serial converter for computer output signals to the Teletype unit.

The Teletype and all optional input/output equipment are discussed in Chapter 7 of this handbook.

SYMBOLS

The following special symbols are used throughout this handbook to explain the function of equipment and instructions:

Symbol	Explanation
A => B	The content of register A is transferred into register B
0 => A	Register A is cleared to contain all binary zeros
Aj	Any given bit in A
A5	The content of bit 5 of register A
A5(1)	Bit 5 of register A contains a 1
A6 — 11	The content of bits 6 through 11 of register A
A6 — 11 => B0 — 5	The content of bits 6 through 11 of register A is transferred into bits 0 through 5 of register B
Y	The content of any core memory location
V	Inclusive OR
⊻	Exclusive OR
∧	AND
$\overline{A}$	Ones complement of the content of A

MEMORY AND PROCESSOR FUNCTIONS

Major Registers

To store, retrieve, control, and modify information and to perform the required logical, arithmetic, and data processing operations, the core memory and the processor employ the logic complement shown in Figure 2 and described in the following paragraphs.

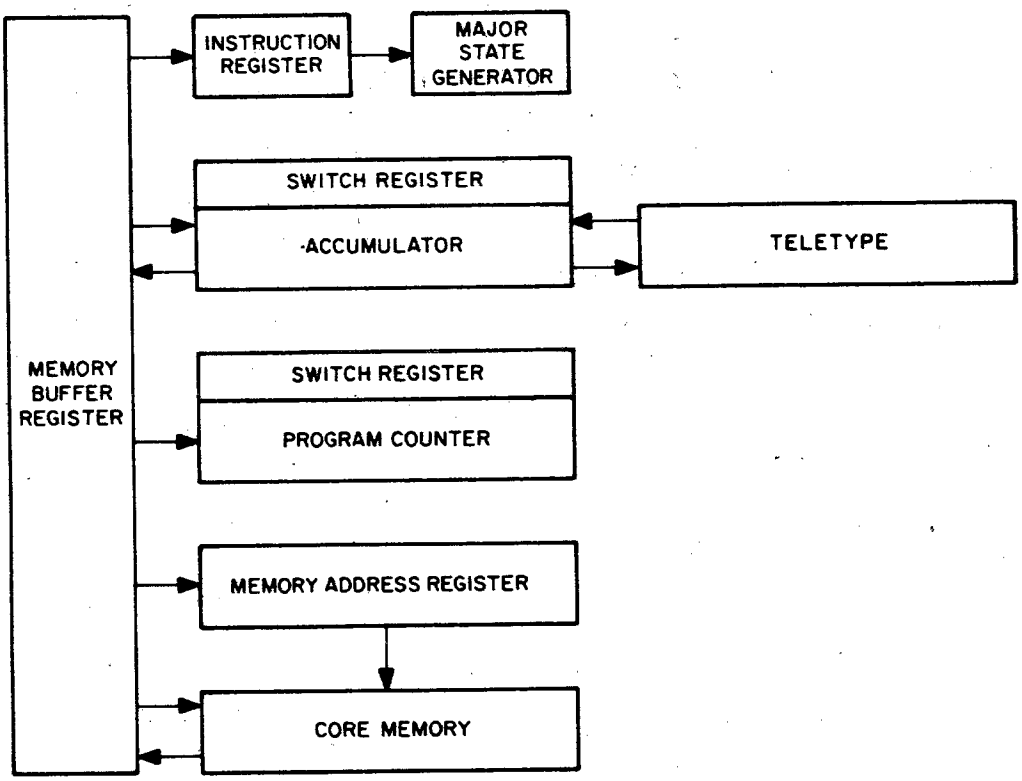


Figure 2 Simplified Block Diagram

ACCUMULATOR (AC)

Arithmetic and logic operations are performed in this 12-bit register. Under program control the AC can be cleared or complemented, its content can be rotated right or left with the link. The content of the memory buffer register can be added to the content of the AC and the result left in the AC. The content of both of these registers may be combined by the logical operation AND, the result remaining in the AC. The memory buffer register and the AC also have gates which allow them to be used together as the shift register and buffer register of a successive approximation analog-to-digital converter. The inclusive OR may be performed between the AC and the switch register on the operator console and the result left in the AC.

The accumulator also serves as an input-output register. All programmed information transfers between core memory and an external device pass through the accumulator.

LINK (L)

This one-bit register is used to extend the arithmetic facilities of the accumulator. It is used as the carry register for two's complement arithmetic. Overflow into the L from the AC can be checked by the program to greatly simplify and speed up single and multiple precision arithmetic routines. Under program control the link can be cleared and complemented, and it can be rotated as part of the accumulator.

PROGRAM COUNTER (PC)

The program sequence, that is the order in which instructions are performed, is determined by the PC. This 12-bit register contains the address of the core memory location from which the next instruction will be taken. Information enters the PC from the core memory, via the memory buffer register, and from the switch register on the operator console. Information in the PC is transferred into the memory address register to determine the core memory address from which each instruction is taken. Incrementation of the content of the PC establishes the successive core memory locations of the program and provides skipping of an instruction based upon a programmed test of information or conditions.

MEMORY ADDRESS REGISTER (MA)

The address in core memory which is currently selected for reading or writing is contained in this 12-bit register. Therefore, all 4096 words of core memory can be addressed directly by this register. Data can be set into it from the memory buffer register, from the program counter, or from an I/O device using the data break facilities.

SWITCH REGISTER (SR)

Information can be manually set into the switch register for transfer into the PC as an address by means of the LOAD ADDRESS key, or into the AC as data to be stored in core memory by means of the DEPOSIT key.

CORE MEMORY

The core memory provides storage for instructions to be performed and information to be processed or distributed. This random address magnetic core memory holds 4096 12-bit words in the standard PDP-8/I. Optional equipment extends the storage capacity in fields of 4096 words or expands the word length to 13 bits to provide parity checking. Memory location 0_8 is used to store the content of the PC following a program interrupt, and location 1_8 is used to store the first instruction to be executed following a program interrupt. (When a program interrupt occurs, the content of the PC is stored in location 0_8, and program control is transferred to location 1 automatically.) Locations 10_8 through 17_8 are used for auto-indexing. All other locations can be used to store instructions or data.

The core memory contains numerous circuits such as read-write switches, address decoders, inhibit drivers, and sense amplifiers. These circuits perform the electrical conversions necessary to transfer information into or out of the core array and perform no arithmetic or logic operations upon the data. Since their operation is not discernible by the programmer or operator of the PDP-8/I, these circuits are not described here in detail.

MEMORY BUFFER REGISTER (MB)

All information transfers between the processor registers and the core memory are temporarily held in the MB. Information can be transferred into the MB from the accumulator or memory address register. The MB can

be cleared, incremented by one or two, or shifted right. Information can be set into the MB from an external device during a data break or from core memory, via the sense amplifiers. Information is read from a memory location in 0.75 microsecond and rewritten in the same location in another 0.75 microsecond of one 1.5 microsecond memory cycle.

INSTRUCTION REGISTER (IR)

This 3-bit register contains the operation code of the instruction currently being performed by the machine. The three most significant bits of the current instruction are loaded into the IR from the memory buffer register during a Fetch cycle. The content of the IR is decoded to produce the eight basic instructions, and affect the cycles and states entered at each step in the program.

MAJOR STATE GENERATOR

One or more major states are entered serially to execute programmed instructions or to effect a data break. The major state generator establishes one state for each computer timing cycle. The Fetch, Defer, and Execute states are entered to determine and execute instructions. Entry into these states is produced as a function of the current instruction and the current state. The Word Count, Current Address, and Break states are entered during a data break. The Break state or all three of these states are entered based upon request signals received from peripheral I/O equipment.

Fetch

During this state an instruction is read into the MB from core memory at the address specified by the content of the PC. The instruction is restored in core memory and retained in the MB. The operation code of the instruction is transferred into the IR to cause enactment, and the content of the PC is incremented by one.

If a multiple-cycle instruction is fetched, the following major state will be either Defer or Execute. If a one-cycle instruction is fetched, the operations specified are performed during the last part of the Fetch cycle and the next state will be another Fetch.

Defer

When a 1 is present in bit 3 of a memory reference instruction, the Defer state is entered to obtain the full 12-bit address of the operand from the address in the current page or page 0 specified by bits 4 through 11 of the instruction. The process of address deferring is called indirect addressing because access to the operand is addressed indirectly, or deferred, to another memory location.

Execute

This state is entered for all memory reference instructions except jump. During an AND, two's complement add, or increment and skip if zero instruction, the content of the core memory location specified by the address portion of the instruction is read into the MB and the operation specified by bits 0 through 2 of the instruction is performed. During a deposit and clear accumulator instruction the content of the AC is transferred into the MB and is stored in core memory at the address specified in the instruction. During a jump to subroutine instruction this state occurs to write the content of the PC into the core memory address designated by the instruction and to transfer this address into the PC to change program control.

Word Count

This state is entered when an external device supplies signals requesting a data break and specifying that the break should be a 3-cycle break. When this state occurs, a transfer word count in a core memory location designated by the device is read into the MB, is incremented by 1, and is rewritten in the same location. If the word count overflows, indicating that the desired number of data break transfers will be enacted at completion of the current break, the computer transmits a signal to the device. The Current Address state immediately follows the Word Count state.

Current Address

As the second cycle of a 3-cycle data break, this cycle establishes the address for the transfer that takes place in the following cycle (Break state). Normally the location following the word count is read from core memory into the MB, is incremented by 1 to establish sequential addresses for the transfers, and is transferred into the MA to determine the address selected for the next cycle. When the word count operation is not used, the device supplies an inhibit signal to the computer so that the word read during this cycle is not incremented. Transfers occur at sequential addresses due to incrementing during the Word Count state. During this sequence the word in the MB is rewritten at the same location and the MB is cleared at the end of the cycle. The Break state immediately follows the Current Address state.

Break

This state is entered to enact a data transfer between computer core memory and an external device, either as the only state of a 1-cycle data break or as the final state of a 3-cycle data break. When a break request signal arrives and the cycle select signal specifies a 1-cycle break, the computer enters the Break state at the completion of the current instruction. Information transfers occur between the external device and a device-specified core memory location, through the MB. When this transfer is complete, the program sequence resumes from the point of the break. The data break does not affect the content of the AC, L, and PC.

OUTPUT BUS DRIVERS

Output signals from the computer processor are power amplified by output bus driver modules of the standard PDP-8/I; allowing these signals to drive a heavy circuit load.

FUNCTIONAL SUMMARY

Operation of the computer is accomplished on a limited scale by keys on the operator console. Operation in this manner is limited to address and data storage by means of the switch register, core memory data examination, the normal start/stop/continue control, and the single step or single instruction operation that allows a program to be monitored visually as a maintenance operation. Most of these manually initiated operations are performed by executing an instruction in the same manner as by automatic programming, except that the gating is performed by special pulses rather than by the normal clock pulses. In automatic operation, instructions stored in core memory are loaded into the memory buffer register and executed during one or more computer cycles. Each instruction determines the major control states that must be entered for its execution. Each control state lasts for one 1.5-microsecond computer cycle and is divided into distinct time states which can be used to perform sequential logical operations. Performance of any function of the computer is controlled by gating of a specific instruction during a specific major control state and a specific time state.

TIMING AND CONTROL ELEMENTS

The circuit elements that determine the timing and control of the operation of the major registers of the PDP-8/I are added to Figure 2 to form Figure 3.

Figure 3 shows the timing and control elements described in the succeeding paragraphs and indicates their relationship to the major registers. These elements can be grouped categorically into timing generators, register controls, and program controls.

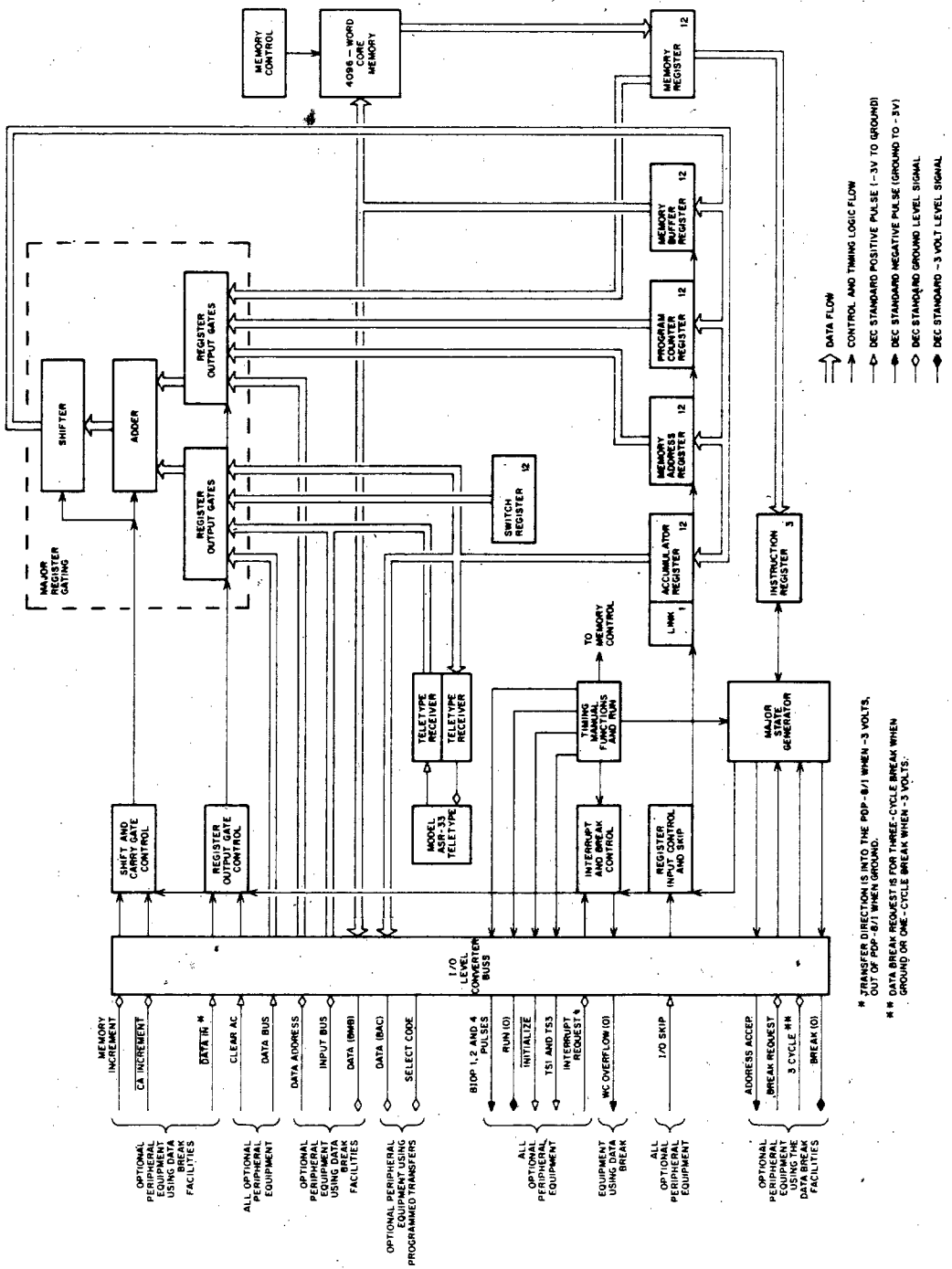


Figure 3. PDP-8/I Timing and Control Element Block Diagram

TIMING GENERATORS

Timing pulses used to determine the computer cycle time and used to initiate sequential time-synchronized gating operations are produced by the timing signal generator. Timing pulses used during operations resulting from the use of the keys and switches on the operator console are produced by the special pulse generator. Pulses that reset registers and control circuits during power turn on and turn off operations are produced by the power clear pulse generator. Several of these pulses are available to peripheral devices using programmed or data break information transfers.

Register Controls

The AC, MA, MB and PC each have gated inputs and gated outputs. The gated input bus of each register is tied to a common register bus that is the output of the major register gating circuit. The data on the common register bus originates from the various outputs of each register and can be modified by the ADDER or SHIFTER in the major register gating circuit. When the contents of a register are to be transferred to another register, its contents are gated by the register output gate control onto the common register bus and strobed into the appropriate register by the register input control. Data can therefore be transferred between registers directly by disabling the ADDER and SHIFTER or can be modified during transfer to provide SHIFT, CARRY and SKIP operations. Operations such as incrementing a register are accomplished simply by gating the output of the register onto the register bus, enabling the ADDER, and strobing the results back into the same register.

PROGRAM CONTROLS

Circuits are also included in the PDP-8/I that produce the IOP pulses which initiate operations involved in input-output transfers, determine the advance of the computer program, and allow peripheral equipment to cause a program interrupt of the main computer program to transfer program control to a subroutine which performs some service for the I/O device.

Interface

The input/output portion of the PDP-8/I is extremely flexible and interfaces readily with special equipment, especially in real time data processing and control environments.

The PDP-8/I utilizes positive logic within the computer but inverts the input/output "bus" system into negative logic. This makes the PDP-8/I compatible with existing peripheral equipment offered by Digital and other manufacturers. Two options expand the flexibility of the PDP-8/I for interfacing to peripheral equipment, particularly those designed with positive logic. An internal option modifies the standard negative (—3v) "bus" system to a positive (+3v) "bus" system. An external option splits the standard "bus" into both a positive (+3v) and a negative (—3v) "bus" system. (Information on this option may be obtained from your nearest DIGITAL EQUIPMENT CORP. sales office.)

The PDP-8/I utilizes a "bus" I/O system rather than the more conventional "radial" system. The "bus" system allows a single set of data and control lines to communicate with all I/O devices.The bus simply goes from one device to the next. No additional connections to the computer are required. A "radial" system requires that a different set of signals be transmitted to each device; and thus the computer must be modified when new devices are added. The PDP-8/I need not be modified when adding new peripheral devices.

Data transfers may also be made directly with core memory at a high speed using the data break facility. This is a completely separate I/O system from the one described previously. It is standard equipment in every PDP-8/I and is ordinarily used with fast I/O devices such as magnetic drums or tapes. Transfers through the data break facility are interlaced with the program in progress. They are initiated by a request from the peripheral device and not by programmed instruction. Thus, the device may transfer a word with memory whenever it is ready and does not have to wait for the program to issue an instruction. Computation may proceed on an interlaced basis with these transfers.

Interface signal characteristics are indicated in Chapter 11.

CHAPTER 2
STANDARD PDP-8/I OPERATION

Controls and Indicators

Manual control of the PDP-8/I is exercised by means of keys and switches on the operator console. Visual indications of the machine status and the content of major registers and control flip-flops is also given on this console. Indicator lamps light to denote the presence of a binary 1 in specific register bits and in control flip-flops. The function of these controls and indicators is listed in Table 1, and their location is shown in Figure 4. The functions of all controls and indicators of the Model 33 ASR Teletype unit are described in Table 2, as they apply to operation of the computer. The Teletype console is shown in Figure 5.

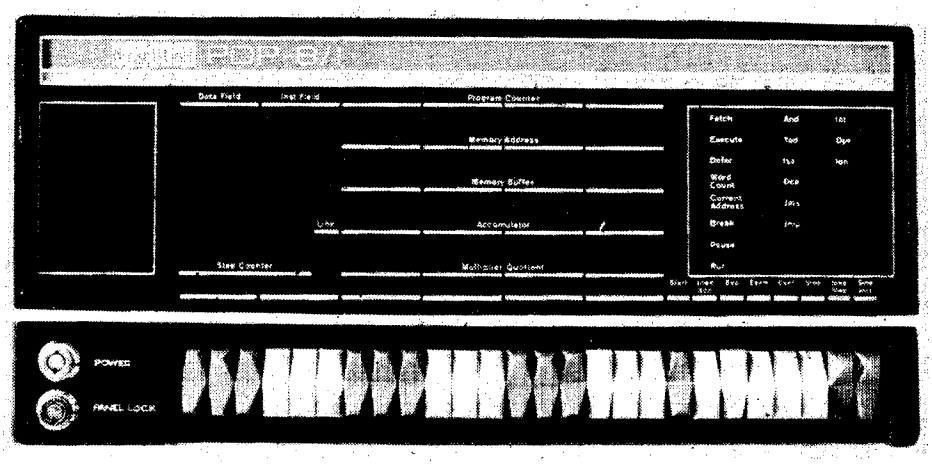


Figure 4. PDP-8/I Operator Console

TABLE 1. OPERATOR CONSOLE CONTROLS AND INDICATORS

Control or Indicator	Function
PANEL LOCK switch	With this key-operated switch turned clockwise, all keys and switches except the SWITCH REGISTER switches on the operator console are disabled. In this condition the program can not be disturbed by inadvertent key operation. The program can, however, monitor the content of the SR by execution of the OSR instruction. With this switch turned counterclockwise, all operator console keys and switches function normally.
POWER switch	In the counterclockwise position this key-operated switch removes primary power from the computer, and in the clockwise position it applies power.

TABLE 1. OPERATOR CONSOLE CONTROLS AND INDICATORS (continued)

Control or Indicator	Function
START key	Starts the computer program by turning off the program interrupt circuits; clearing the AC, L, MB, and IR; setting the Fetch state, transferring the content of the PC into the MA; and setting the RUN flip-flop. Therefore, the word stored at the address currently held by the PC is taken as the first instruction.
LOAD ADDRESS key	Pressing this key sets the content of the SR into the PC, sets the content of the INST FIELD switches into the IF, and sets the content of the DATA FIELD switches into the DF.
DEPOSIT key	Lifting this key sets the content of the SR into the MB and core memory at the address specified by the current content of the PC. The content of the PC is then incremented by one, to allow storing of information in sequential memory addresses by repeated operation of the DEPOSIT key.
EXAMINE key	Pressing this key sets the content of core memory at the address specified by the content of the PC into the MB. The content of the PC is then incremented by one to allow examination of the content of sequential core memory addresses by repeated operation of the EXAMINE key.
CONTINUE key	Pressing this key sets the RUN flip-flop to continue the program in the state and instruction designated by the lighted console indicators, at the address currently specified by the PC.
STOP key	Causes the RUN flip-flop to be cleared at the end of the cycle in progress at the time the key is pressed.
SINGLE STEP switch	The switch is off in the down position. In the up position the switch causes the RUN flip-flop to be cleared to disable the timing circuits at the end of one cycle of operation. Thereafter, repeated operation of the CONTINUE key steps the program one cycle at a time so that the content of registers can be observed in each state.

TABLE 1. OPERATOR CONSOLE CONTROLS AND INDICATORS (continued)

Control or Indicator	Function
SINGLE INSTRUCTION switch	The switch is off in the down position. In the up position the switch causes the RUN flip-flop to be cleared at the end of the next instruction execution. When the computer is started by means of the START or CONTINUE key, this switch causes the RUN flip-flop to be cleared at the end of the last cycle of the current instruction. Therefore, repeated operation of the CONTINUE key steps the program one instruction at a time.
SWITCH REGISTER switches	Provide a means of manually setting a 12-bit word into the machine Switches in the up position; corresponds to binary ones, down to zeros. The content of this register is loaded into the PC by the LOAD ADDRESS key or into the MB and core memory by the DEPOSIT key. The content of the SR can be set into the AC under program control by means of the OSR instruction.
DATA FIELD indicators and switches*	The indicators denote the content of the data field register (DF) and the switches serve as an extension of the SR to load the DF by means of the LOAD ADDRESS key. The DF determines the core memory field of data storage and retrieval.
INST FIELD indicators and switches*	The indicators denote the content of the instruction field register (IF) and the switches serve as an extension of the SR to load the IF by means of the LOAD ADDRESS key. The IF determines the core memory field from which instructions are to be taken.
PROGRAM COUNTER indicators	Indicate the content of the PC. When the machine is stopped the content of the PC indicates the core memory address of the first instruction to be executed when the START or CONTINUE key is operated. When the machine is running the content of the PC indicates the core memory address of the next instruction.
MEMORY ADDRESS indicators	Indicate the content of the MA. Usually the content of the MA denotes the core memory address of the word currently or previously read or written. After operation of either the DEPOSIT or EXAMINE key, the content of the MA indicates the core memory address at which information was just written or read.

*Activated only on systems containing the Type MC8/I Memory Extension Control option.

TABLE 1. OPERATOR CONSOLE CONTROLS AND INDICATORS (continued)

Control or Indicator	Function
MEMORY BUFFER indicators	Indicate the content of the MB. Usually the content of the MB designates the word just read or written at the core memory address held in the MA.
ACCUMULATOR indicators	Indicates the content of the AC.
LINK indicator	Indicates the content of the L.
MULTIPLIER QUOTIENT indicators*	Indicate the content of the multiplier quotient (MQ). The MQ holds the multiplier at the beginning of a multiplication and holds the least significant half of the product at the conclusion. It holds the least significant half of the dividend at the start of a division and at the end holds the quotient.
STEP COUNTER indicators*	Indicate the contents of the step counter (SC). The step counter holds the complement of the contents of bits 7-11 of the memory location following the instruction.
Instruction indicators (AND, TAD, ISZ, DCA, JMS, JMP, IOT, OPR)	Indicate the decoded output of the IR as the instruction currently in progress.
FETCH, EXECUTE DEFER, WORD COUNT, CURRENT ADDRESS, BREAK indicators	Indicate the primary control state of the machine and that the current memory cycle is a Fetch, Execute, Defer or Break cycle, respectively. Word Count and Current Address indicate the first and second cycles of a Break cycle, respectively.
ION indicator	Indicates the 1 status of the INT. ENABLE flip-flop. When lit, the program in progress can be interrupted by receipt of a Program Interrupt Request signal from an I/O device.
PAUSE indicator	Indicates the 1 status of the PAUSE flip-flop when lit. An IOT instruction sets the PAUSE flip-flop at TP1 time to initiate operation of the IOP generator and to inhibit advance of the normal timing generator. When IOP generator operation is completed (approximately 3.8 microseconds later), a TP4 pulse is generated and the PAUSE flip-flop is cleared to enable advance of the timing generator in synchronism with the basic computer clock.

*Activated only on systems containing the KE-8/I Extended Arithmetic Element option.

TABLE 1. OPERATOR CONSOLE CONTROLS AND INDICATORS (continued)

Control or Indicator	Function
RUN indicator	Indicates the 1 status of the RUN flip-flop. When lit, the internal timing circuits are enabled and the machine performs instructions.

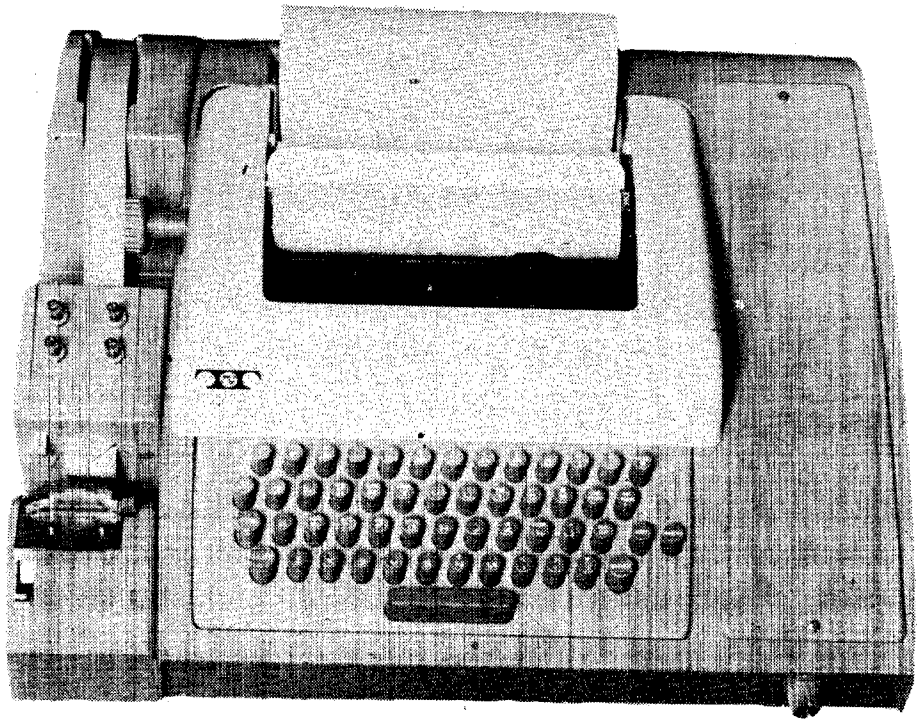


Figure 5. Teletype Model 33 ASR Console

TABLE 2. TELETYPE CONTROLS AND INDICATORS

Control or Indicator	Function
REL. pushbutton	Disengages the tape in the punch to allow tape removal or tape loading.
B. SP. pushbutton	Backspaces the tape in the punch by one space, allowing manual correction or rub out of the character just punched.
OFF and ON pushbuttons	Control use of the tape punch with operation of the Teletype keyboard/printer.

Control or Indicator	Function
START/STOP/FREE switch	Controls use of the tape reader with operation of the Teletype. In the lower FREE position the reader is disengaged and can be loaded or unloaded. In the center STOP position the reader mechanism is engaged but de-energized. In the upper START position the reader is engaged and operated under program control.
Keyboard	Provides a means of printing on paper in use as a typewriter and punching tape when the punch ON pushbutton is pressed, and provides a means of supplying input data to the computer when the LINE/OFF/LOCAL switch is in the LINE position.
LINE/OFF/LOCAL switch	Controls application of primary power in the Teletype and controls data connection to the processor. In the LINE position the Teletype is energized and connected as an I/O device of the computer. In the OFF position the Teletype is de-energized. In the LOCAL position the Teletype is energized for off-line operation, and signal connections to the processor are broken. Both line and local use of the Teletype require that the computer be energized through the POWER switch.

OPERATING PROCEDURES

Many means are available for loading and unloading PDP-8/I information. The means used are, of course, dependent upon the form of the information, time limitations, and the peripheral equipment connected to the computer. The following procedures are basic to any use of the PDP-8/I, and although they may be used infrequently as the programming and use of the computer become more sophisticated, they are valuable in preparing the initial programs and learning the function of machine input and output transfers.

Manual Data Storage and Modification

Programs and data can be stored or modified manually by means of the facilities on the operator console. Chief use of manual data storage is made to load the readin mode leader program into the computer core memory. The readin mode (RIM) loader is a program used to automatically load programs into PDP-8/I from perforated tape in RIM format. This program and the RIM tape format are described in Appendix 5 and in Digital Program Library descriptions. The RIM program listed in the Appendix can be used as an exercise in manual data storage. To store data manually in the PDP-8/I core memory:

1. Turn the PANEL LOCK switch counterclockwise and turn the POWER switch clockwise.

2. Set the bit switches of the SWITCH REGISTER (SR) to correspond with the address bits of the first word to be stored. Press the LOAD ADDRESS key and observe that the address set by the SR is held in the PC, as designated by lighted PROGRAM COUNTER indicators corresponding to switches in the 1 (up) position and unlighted indicators corresponding to switches in the 0 (down) position.

3. Set the SR to correspond with the data or instruction word to be stored at the address just set into the PC. Lift the DEPOSIT key and observe that the MB, and hence the core memory, hold the word set by the SR.
Also, observe that the PC has been incremented by one so that additional data can be stored at sequential addresses by repeated SR setting and DEPOSIT key operation.

To check the content of an address in core memory, set the address into the PC as in step 2, then press the EXAMINE key. The content of the address is then designated by the MEMORY BUFFER indicators. The content of the PC is incremented by one with operation of the EXAMINE key, so the content of sequential addresses can be examined by repeated operation after the original (or starting) address is loaded. The content of any address can be modified by repeating both steps 2 and 3.

Loading Data Under Program Control

Information can be stored or modified in the computer automatically only by enacting programs previously stored in core memory. For example, having the RIM loader stored in core memory allows RIM format tapes to be loaded as follows:

1. Turn the PANEL LOCK switch counterclockwise and turn the POWER switch clockwise.

2. Set the Teletype LINE/OFF/LOCAL switch to the LINE position.

3. Load the tape in the Teletype reader by setting the START/STOP/FREE switch to the FREE position, releasing the cover guard by means of the latch at the right, loading the tape so that the sprocket wheel teeth engage the feed holes in the tape, closing the cover guard, and setting the switch to the STOP position. Tape is loaded in the back of the reader so that it moves toward the front as it is read. Proper positioning of the tape in the reader finds three bit positions being sensed to the left of the sprocket wheel and five bit positions being sensed to the right of the sprocket wheel.

4. Load the starting address of the RIM loader program (not the address of the program to be loaded) into the PC by means of the SR and the LOAD ADDRESS key.

5. Press the computer START key and set the 3-position Teletype reader switch to the START position. The tape will be read automatically.

Automatic storing of the binary loader (BIN) program is performed by means of the RIM loader program as previously described. With the BIN loader

stored in core memory, program tapes assembled in the program assembly language (PAL III) binary format can be stored as described in the previous procedure except that the starting address of the BIN loader (usually 7777) is used in step 4. When storing a program in this manner, the computer stops and the AC should contain all zeros if the program is stored properly. If the computer stops with a number other than zero in the AC, a checksum error has been detected. When the program has been stored, it can be initiated by loading the program starting address (usually designated on the leader of the tape) into the PC by means of the SR and LOAD ADDRESS key, then pressing the START key.

Off-Line Teletype Operation

The Teletype can be used separately from the PDP-8/I for typing, punching tape, or duplicating tapes. To use the Teletype in this manner:

1. Assure that the computer PANEL LOCK switch is turned counterclockwise and turn the POWER switch clockwise.

2. Set the Teletype LINE/OFF/LOCAL switch to the LOCAL position.

3. If the punch is to be used, load it by raising the cover, manually feeding the tape from the top of the roll into the guide at the back of the punch, advancing the tape through the punch by manually turning the friction wheel, and then closing the cover. Energize the punch by pressing the ON pushbutton, and produce about two feet of leader. The leader-trailer can be code 200 or 377. To produce the code 200 leader, simultaneously press and hold the CTRL and SHIFT keys with the left hand; press and hold the REPT key; press and release the @ key. When the required amount of leader has been punched release all keys. To produce the 377 code, simultaneously press and hold both the REPT and RUB OUT keys until a sufficient amount of leader has been punched.

If an incorrect key is struck while punching a tape, the tape can be corrected as follows: if the error is noticed after typing and punching N characters, press the punch B. SP. (backspace) pushbutton N + 1 times and strike the keyboard RUB OUT key N + 1 times. Then continue typing and punching with the character which was in error.

To duplicate and obtain a listing of an existing tape: Perform the procedure under the current heading. Then load the tape to be duplicated as described in step 2 of the procedure under Loading Data Under Program Control. Initiate tape duplication by setting the reader START/STOP/FREE switch in the START position. The punch and teleprinter stop when the tape being duplicated is completely read.

Corrections to insert or delete information on a perforated tape can be made by duplicating the correct portion of the tape, and manually punching additional information or inhibiting punching of information to be deleted. This is accomplished by duplicating the tape and carefully observing the information being typed as the tape is read. In this manner the reader START/STOP/FREE switch can be set to the STOP position just before the point of the correction is typed. Information to be inserted can then be punched manually by means of the keyboard. Information can be deleted by pressing the punch OFF push-

button and operating the reader until the portion of the tape to be deleted has been typed. It may be necessary to backspace and rub out one or two characters on the new tape if the reader is not stopped precisely on time. The number of characters to be rubbed out can be determined exactly by the typed copy. Be sure to count spaces when counting typed characters. Continue duplicating the tape in the normal manner after making the corrections.

New, duplicated, or corrected perforated tapes should be verified by reading them off line and carefully proofreading the typed copy.

Program Control

If the program is stopped at the end of an instruction by raising the SINGLE STEP key, then the LOAD ADDRESS, EXAMINE, and DEPOSIT keys may be used without changing the AC. The program may then be resumed by resetting the PC using LOAD ADDRESS and by pressing CONTINUE.

CHAPTER 3

MEMORY AND PROCESSOR BASIC PROGRAMMING*

MEMORY ADDRESSING

The following terms are used in memory address programming:

Term	Definition
Page	A block of 128 core memory locations (200_8 addresses). The page containing the instruction being executed; as determined by bits 0 through 4 of the program counter.
Current Page	The page containing the instruction being executed; as determined by bits 0 through 4 of the program counter.
Page Address	An 8-bit number contained in bits 4 through 11 of an instruction which designates one of 256 core memory locations. Bit 4 of a page address indicates that the location is in the current page when a 1, or indicates it is in page 0 when a 0. Bits 5 through 11 designate one of the 128 locations in the page determined by bit 4.
Absolute Address	A 12-bit number used to address any location in core memory.
Effective Address	The address of the operand. When the address of the operand is in the current page or in page 0, the effective address is a page address. Otherwise, the effective address is an absolute address stored in the current page or page 0 and obtained by indirect addressing.

Organization of the standard core memory or any 4096-word field of extended memory is summarized as follows:

Total locations (decimal)	4096
Total addresses (octal)	7777
Number of pages (decimal)	32
Page designations (octal)	0-37
Number of locations per page (decimal)	128
Addresses within a page (octal)	0-177

*See Appendix I for Program Abstracts.

Four methods of obtaining the effective address are used as specified by combinations of bits 3 and 4.

Bit 3	Bit 4	Effective Address
0	0	The operand is in page 0 at the address specified by bits 5 through 11.
0	1	The operand is in the current page at the address specified by bits 5 through 11.
1	0	The absolute address of the operand is taken from the content of the location in page 0 designated by bits 5 through 11.
1	1	The absolute address of the operand is taken from the content of the location in the current page designated by bits 5 through 11.

The following example indicates the use of bits 3 and 4 to address any location in core memory. Suppose it is desired to add the content of locations A, B, C, and D to the content of the accumulator by means of a routine stored in page 2. The instructions in this example indicate the operation code, the content of bit 4, the content of bit 3, and a 7-bit address. This routine would take the following form:

Page 0 Location	Page 0 Content	Page 1 Location	Page 1 Content	Page 2 Location	Page 2 Content	Remarks
				R	TAD 00 A	DIRECT TO DATA IN PAGE 0
				S	TAD 01 B	DIRECT TO DATA IN SAME PAGE
				T	TAD 10 M	INDIRECT TO ADDRESS SPECIFIED IN PAGE 0
				U	TAD 11 N	INDIRECT TO ADDRESS SPECIFIED IN SAME PAGE
					. . . .	
A	xxxx	C	xxxx	B	xxxx	
M	C	D	xxxx	N	D	

Routines using 128 instructions, or less, can be written in one page using direct addresses for looping and using indirect addresses for data stored in other pages. When planning the location of instructions and data in core memory, remember that the following locations are reserved for special purposes:

Address	Purpose
0_8	Stores the contents of the program counter following a program interrupt.
1_8	Stores the first instruction to be executed following a program interrupt.
10_8 through 17_8	Auto-indexing.

Indirect Addressing

When indirect addressing is specified, the address part (bits 5-11) of a memory reference instruction is interpreted as the address of a location containing not the operand, but containing the address of the operand. Consider the instruction TAD A. Normally, A is interpreted as the address of the location containing the quantity to be added to the content of the AC. Thus, if location 100 contains the number 5432, the instruction TAD 100 causes the quantity 5432 to be added to the content of the AC. Now suppose that location 5432 contains the number 6543. The instruction TAD I 100 (where I signifies indirect addressing) causes the computer to take the number 5432, which is in location 100, as the effective address of the instruction and the number in location 5432 as the operand. Hence, this instruction results in the quantity 6543 being added to the content of the AC.

Auto-Indexing

When a location between 10_8 and 17_8 in page 0 of any core memory field is addressed indirectly (by an instruction in which bit 3 is a 1) the content of that location is read, incremented by one, rewritten in the same location, and then taken as the effective address of the instruction. This feature is called auto-indexing. If location 12_8 contains the number 5432 and the instruction DCA I Z 12 is given, the number 5433 is stored in location 12, and the content of the accumulator is deposited in core memory location 5433.

STORING AND LOADING

Data is stored in any core memory location by use of the DCA Y instruction. This instruction clears the AC to simplify loading of the next datum. If the data deposited is required in the AC for the next program operation, the DCA must be followed by a TAD Y for the same address.

All loading of core memory information into the AC is accomplished by means of the TAD Y instruction, preceded by an instruction that clears the AC such as CLA or DCA.

Storing and loading of information in sequential core memory locations can make excellent use of an auto-index register to specify the core memory address.

PROGRAM CONTROL

Transfer of program control to any core memory location uses the JMP or JMS instructions. The JMP I (indirect address, 1 in bit 3) is used to transfer program control to any location in core memory which is not in the current page or page 0.

The JMS Y is used to enter a subroutine which starts at location Y + 1 in the current page or page 0. The content of the PC + 1 is stored in the specified

address Y, and address Y + 1 is transferred into the PC. To exit a subroutine the last instruction is a JMP I Y, which returns program control to the location stored in Y.

INDEXING OPERATIONS

External events can be counted by the program and the number can be stored in core memory. The core memory location used to store the event count can be initialized (cleared) by a CLA command followed by a DCA instruction. Each time the event occurs, the event count can be advanced by a sequence of commands such as CLA, TAD, IAC, and DCA.

The ISZ instruction is used to count repetitive program operations or external events without disturbing the content of the accumulator. Counting a specified number of operations is performed by storing a two's complement negative number equal to the number of iterations to be counted. Each time the operation is performed, the ISZ instruction is used to increment the content of this stored number and check the result. When the stored number becomes zero, the specified number of operations have occurred and the program skips out of the loop and back to the main sequence.

This instruction is also used for other routines in which the content of a memory location is incremented without disturbing the content of the accumulator, such as storing information from an I/O device in sequential memory locations or using core memory locations to count I/O device events.

LOGIC OPERATIONS

The PDP-8/I instruction list includes the logic instruction AND Y. From this instruction short routines can be written to perform the inclusive OR and exclusive OR operations.

Logical AND

The logic AND operation between the content of the accumulator and the content of a core memory location Y is performed directly by means of the AND Y instruction. The result remains in the AC, the original content of the AC is lost, and the content of Y is unaffected.

Inclusive OR

Assuming value A is in the AC and value B is stored in a known core memory address, the following sequence performs the inclusive OR. The sequence is stated as a utility subroutine called IOR.

```
/CALLING SEQUENCE                    JMS IOR
/                                    (ADDRESS OF B)
/                                    (RETURN)
/ENTER WITH ARGUMENT IN AC; EXIT WITH LOGICAL RESULT IN AC

          IOR,    0
                  DCA TEM1
                  TAD I IOR
                  DCA TEM2
                  TAD TEM1
                  CMA
                  AND I TEM2
```

```
                    TAD TEM1
                    ISZ IOR
                    JMP I IOR
TEM1,               0
TEM2,               0
```

Exclusive OR

The exclusive OR operation for two numbers, A and B, can be performed by a subroutine called by the mnemonic code XOR. In the following general purpose XOR subroutine, the value A is assumed to be in the AC, and the address of the value B is assumed to be stored in a known core memory location.

```
/CALLING SEQUENCE                JMS XOR
/                                (ADDRESS OF B)
/                                (RETURN)
/ENTER WITH ARGUMENT IN AC; EXIT WITH LOGICAL RESULT IN AC
                XOR,             0
                                 DCA TEM1
                                 TAD I XOR
                                 DCA TEM2
                                 TAD TEM1
                                 AND I TEM2
                                 CMA IAC
                                 CLL RAL
                                 TAD TEM1
                                 TAD I TEM2
                                 ISZ XOR
                                 JMP I XOR
                TEM1,            0
                TEM2,            0
```

An XOR subroutine can be written using fewer core memory locations by making use of the IOR subroutine; however, such a subroutine takes more time to execute. A faster XOR subroutine can be written by storing the value B in the second instruction of the calling sequence instead of the address of B; however, the resulting subroutine is not as utilitarian as the routine given here.

ARITHMETIC OPERATIONS

One arithmetic instruction is included in the PDP-8/I order code, the two's complement add: TAD Y. Using this instruction, routines can easily be written to perform addition, subtraction, multiplication, and division in two's complement arithmetic.

Two's Complement Arithmetic

In two's complement arithmetic,addition, subtraction, multiplication, and division of binary numbers is performed in accordance with the common rules of binary arithmetic. In PDP-8/I, as in other machines utilizing complementation techniques, negative numbers are represented as the complement of positive numbers, and subtraction is achieved by complement addition. Representation of negative values in one's complement arithmetic is slightly different from that in two's complement arithmetic.

The one's complement of a number is the complement of the absolute positive value; that is, all ones are replaced by zeros and all zeros are replaced by ones. The two's complement of a number is equal to the one's complement of the positive value plus one.

In one's complement arithmetic a carry from the sign bit (most significant bit) is added to the least significant bit in an end-around carry. In two's complement arithmetic a carry from the sign bit complements the link (a carry would set the link to 1 if it were properly cleared before the operation), and there is no end-around carry.

PROGRAMMING SYSTEM

The programming system for the PDP-8/I includes: the Symbolic Assemblers, FORTRAN System Compiler, Symbolic Tape Editor, Floating Point Package, DISC/DECtape Keyboard monitor, mathematical function subroutines, and utility and maintenance programs. All operate with the basic computer. The programming system was designed to simplify and accelerate the process of learning to program. At the same time, experienced programmers will find that it incorporates many advanced features. The system is intended to make immediately available to each user the full, general-purpose data processing capability of the computer and to serve as the operating nucleus for a growing library of programs and routines to be made available to all installations. New techniques, routines, and programs are constantly being developed, field-tested, and documented in the Digital Program Library for incorporation in users' systems.

Assemblers

The use of an assembly program has become standard practice in programming digital computers. This process allows the programmer to code his instructions in a symbolic language, one he can work with more conveniently than the 12-bit binary numbers which actually operate the computer. The assembly program then translates the symbolic language program into its machine code equivalent. The advantages are significant: the symbolic language is more meaningful and convenient to a programmer than a numeric code; instructions or data can be referred to by symbolic names without concern for, or even knowledge of, their actual addresses in core memory; decimal and alphabetical data can be expressed in a form more convenient than binary numbers; programs can be altered without extensive changes; and debugging is considerably simplified.

Two Assemblers Are Available:

1. PAL III is a basic assembler allowing symbolic references, symbolic origins, and expressions. The output is in a form suitable for input to the binary loader. High or low-speed paper tape input is accepted.
2. MACRO-8 is an advanced assembler which has the same basic features of PAL III and in addition, MACRO capability, literals, off-page references, and high/low-speed paper tape input and output.

DISC/DECtape Keyboard Monitor

A Keyboard Monitor is available to users with a DISC or DECtape System which allows the user to save core images on the DISC or DECtape System device and restore these core images to memory. Programs modified to work under the Monitor include: FORTRAN, EDITOR, DDT LOADER, and an Assembler. In addition, the user may save his own core images and restore them and use the remainder of the available device storage for temporary storage of source or binary data.

FORTRAN Compiler (4K)

The FORTRAN (for FORmula TRANslation) compiler lets the user express the problem he is trying to solve in a mixture of English words and mathematical statements that is close to the language of mathematics and is also intelligible to the computer. In addition to reducing the time needed for program preparation, the compiler enables users with little or no knowledge of the computer's organization and operating language to write effective programs for it. The FORTRAN Compiler contains the instructions the computer requires to perform the clerical work of translating the FORTRAN version of the problem statement into an object program in machine language. It also produces diagnostic messages. After compilation, the object program, the operating system and the data it will work with, are loaded into the computer for solution of the problem.

The FORTRAN language consists of four general types of statements: arithmetic, logic, control, and input/output. FORTRAN functions include addition, subtraction, multiplication, division, sine, cosine, arctangent, square root, natural logarithm, and exponential.

FORTRAN Compiler (8K)

The 8K FORTRAN compiler is an extension of the 4K FORTRAN compiler which features the following additions:

1. U.S.A. Standard FORTRAN Syntax
2. Subroutines
3. Two levels of subscripting
4. Function subprograms
5. I/O supervisor
6. Relocatable link loadable output
7. Common
8. I, E, F, H, A, X, format specification
9. Arithmetic and trigonometric library

This compiler will utilize all of available core from 8K to 32K and will correctly load programs over field boundary.

Symbolic On-Line Debugging Program

On-line debugging with DDT-8 gives the user dynamic printed program status information. It gives him close control over program execution, preventing errors ("bugs") from destroying other portions of his program. He can monitor the execution of single instructions or subsections, change instructions or data in any format, and output a corrected program at the end of the debugging session.

Using the standard Teletype keyboard/reader and teleprinter/punch, the user can communicate conveniently with the PDP-8/I in the symbols of his source language. He can control the execution of any portion of his object program by inserting breaks, or traps, in it. When the computer reaches a break, it transfers control of the object program to DDT. The user can then examine and modify the content of individual core memory registers to correct and improve his object program.

Symbolic Tape Editor

The Symbolic Tape Editor program is used to edit, correct, and update symbolic program tapes using the PDP-8/I, the teletype unit and/or the high-speed reader. With the editor in core memory, the user reads in portions of his symbolic tape, removes, changes, or adds instructions or operands, and gets back a complete new symbolic tape with errors removed. He can work through the program instruction by instruction, spot check it, or concentrate on new sections. A character string search is available. The user can move one or more lines of text from one place to another.

Floating Point Package

The Floating Point Package permits the PDP-8/I to perform arithmetic operations that many other computers can perform only after the addition of costly optional hardware. Floating point operations automatically align the binary points of operands, retaining the maximum precision available by discarding leading zeros. In addition to increasing accuracy, floating point operations relieve the programmer of scaling problems common in fixed point operations. This is of particular advantage to the inexperienced programmer.

Mathematical Function Routines

The programming system also includes a set of mathematical function routines to perform the following operations in both single and double precision: addition, subtraction, multiplication, division, square root, sine, cosine, arctangent, natural logarithm, and exponential.

Utility and Maintenance Programs

PDP-8/I utility programs provide printouts or punchouts of core memory content in octal, decimal, or binary form, as specified by the user. Subroutines are provided for octal or decimal data transfer and binary-to-decimal, decimal-to-binary, and Teletype tape conversion.

A complete set of standard diagnostic programs is provided to simplify and expedite system maintenance. Program descriptions and manuals permit the user to effectively test the operation of the computer for proper core memory functioning and proper execution of instructions. In addition, diagnostic programs to check the performance of standard and optional peripheral devices are provided with the devices.

CHAPTER 4

MEMORY AND PROCESSOR INSTRUCTIONS

Instruction words are of two types: memory reference and augmented. Memory reference instructions store or retrieve data from core memory, while augmented instructions do not. All instructions utilize bits 0 through 2 to specify the operation code. Operation codes of 0_8 through 5_8 specify memory reference instructions, and codes of 6_8 and 7_8 specify augmented instructions. Memory reference instruction execution times are multiples of the 1.5-microsecond memory cycle. Indirect addressing increases the execution time of a memory reference instruction by 1.5 microseconds. The augmented instructions, input-output transfer and operate, are performed in 4.25 and 1.5 microseconds, respectively. (All computer times are ±20%.)

MEMORY REFERENCE INSTRUCTIONS

Since the PDP-8/I system contains a 4096-word core memory, 12 bits are required to address all locations. To simplify addressing, the core memory is divided into blocks, or pages, of 128 words (200_8 addresses). Pages are numbered 0_8 through 37_8, each field of 4096-words of core memory uses 32 pages. The seven address bits (bits 5 through 11) of a memory reference instruction can address any location in the page on which the current instruction is located by placing a 1 in bit 4 of the instruction. By placing a 0 in bit 4 of the instruction, any location in page 0 can be addressed directly from any page of core memory. All other core memory locations can be addressed indirectly by placing a 1 in bit 3 and placing a 7-bit effective address in bits 5 through 11 of the instruction to specify the location in the current page or page 0 which contains the full 12-bit absolute address of the operand.

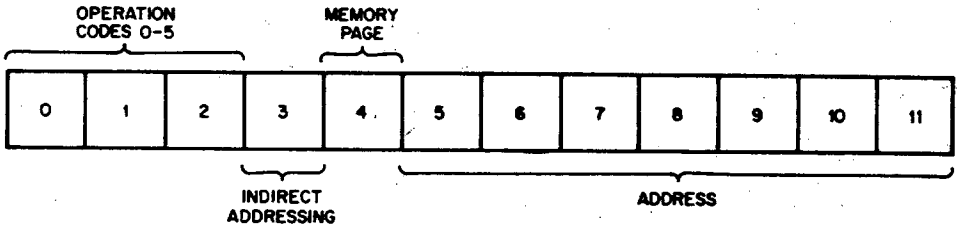


Figure 6. Memory Reference Instruction Bit Assignments

Word format of memory reference instructions is shown in Figure 4 and the instructions perform as follows:

Logical AND (AND Y)

Octal Code: 0
Indicators: AND, FETCH, EXECUTE
Execution Time: 3.0 microseconds with direct addressing, 4.5 microseconds with indirect addressing.
Operation: The AND operation is performed between the content of memory location Y and the content of the AC. The result is left in the AC, the original content of the AC is lost, and the content of Y is restored. Corresponding bits of the AC and Y are operated upon independently. This instruction, often called extract or mask, can be considered as a bit-by-bit multiplication. Example:

Original ACj	Yj	Final ACj
0	0	0
0	1	0
1	0	0
1	1	1

Symbol: ACj $\wedge$ Yj $=>$ ACj

Two's Complement Add (TAD Y)

Octal Code: 1
Indicators: TAD, FETCH, EXECUTE
Execution Time: 3.0 microseconds with direct addressing, 4.5 microseconds with indirect addressing.
Operation: The content of memory location Y is added to the content of the AC in two's complement arithmetic. The result of this addition is held in the AC, the original content of the AC is lost, and the content of Y is restored. If there is a carry from AC0, the link is complemented. This feature is useful in multiple precision arithmetic.
Symbol: AC0 $-$ 11 + Y0 $-$ 11 $=>$ AC0 $-$ 11

Increment and Skip If Zero (ISZ Y)

Octal Code: 2
Indicators: ISZ, FETCH, EXECUTE
Execution Time: 3.0 microseconds with direct addressing, 4.5 microseconds with indirect addressing.
Operation: The content of memory location Y is incremented by one in two's complement arithmetic. If the resultant content of Y equals zero, the content of the PC is incremented by one and the next instruction is skipped. If the resultant content of Y does not equal zero, the program proceeds to the next instruction. The incremented content of Y is restored to memory. The content of the AC is not affected by this instruction.
Symbol: Y + 1 $=>$ Y
If resultant Y0 $-$ 11 = 0, then PC + 1 $=>$ PC

Deposit and Clear AC (DCA Y)

Octal Code: 3
Indicators: DCA, FETCH, EXECUTE
Execution Time: 3.0 microseconds with direct addressing, 4.5 microseconds with indirect addressing.
Operation: The content of the AC is deposited in core memory at address Y and the AC is cleared. The previous content of memory location Y is lost.
Symbol: AC $=>$ Y
then 0 $=>$ AC

Jump to Subroutine (JMS Y)

Octal Code: 4
Indicators: JMS, FETCH, EXECUTE
Execution Time: 3.0 microseconds with direct addressing, 4.5 microseconds with indirect addressing.
Operation: The content of the PC is deposited in core memory location Y and the next instruction is taken from core memory location Y + 1. The content of the AC is not affected by this instruction.
Symbol: PC + 1 $=>$ Y
Y + 1 $=>$ PC

Jump to Y (JMP Y)

Octal Code: 5
Indicators: JMP, FETCH
Execution Time: 1.5 microseconds with direct addressing, 3.0 microseconds with indirect addressing.
Operation: Address Y is set into the PC so that the next instruction is taken from core memory address Y. The original content of the PC is lost. The content of the AC is not affected by this instruction.
Symbol: Y = > PC

AUGMENTED INSTRUCTIONS

There are two augmented instructions which do not reference core memory. They are the input-output transfer, which has an operation code of 6, and the operate which has an operation code of 7. Bits 3 through 11 within these instructions function as an extension of the operation code and can be microprogrammed to perform several operations within one instruction. Augmented instructions are one-cycle (Fetch) instructions that initiate various operations as a function of bit microprogramming.

Input/Output Transfer Instruction

Microinstructions of the input-output transfer (IOT) initiate operation of peripheral equipment and effect information transfers between the processor and an I/O device. Specifically, upon recognition of the operation code 6 as an IOT instruction, the computer enters a 4.25 μsec expanded computer FETCH cycle by setting the PAUSE flip-flop and enabling the IOP generator to produce IOP 1, IOP 2 and IOP 4 pulses as a function of the three least significant bits of the instruction (bits 9 thru 11). These pulses occur at 1 microsecond intervals designated as event times 3, 2 and 1 as follows:

Instruction Bit	IOP Pulse	IOT Pulse	Event Time
11	IOP 1	IOT 1	1
10	IOP 2	IOT 2	2
9	IOP 4	IOT 4	3

The IOP pulses are gated in the device selector of the program-selected equipment to produce IOT pulses that enact a data transfer or initiate a control operation. Selection of an equipment is accomplished by bits 3 through 8 of the IOT instruction. These bits form a 6-bit code that enables the device selector in a given device.

The format of the IOT instruction is shown in Figure 7. Operations performed by IOT microinstructions are explained in Chapter 7.

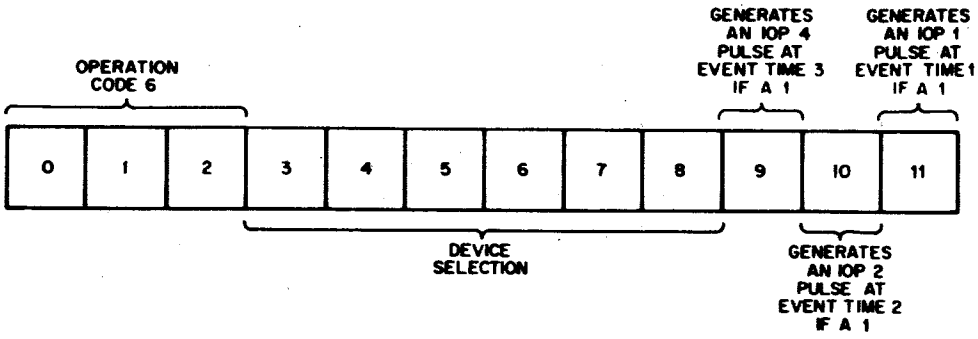


Figure 7. IOT Instruction Bit Assignments

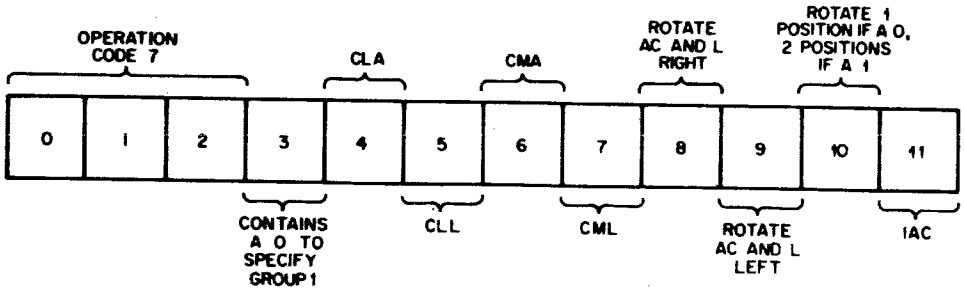
Operate Instruction

With operate instructions, the programmer can consider logical sequences occurring during one computer FETCH cycle. These sequences provide a logical method of forming microinstructions.

The operate instruction consists of two groups of microinstructions. Group 1 (OPR 1) is principally for clear, complement, rotate, and increment operations and is designated by the presence of a 0 in bit 3. Group 2 (OPR 2) is used principally in checking the content of the accumulator and link and continuing to, or skipping, the next instruction based on the check. A 1 in bit 3 designates an OPR 2 microinstruction.

GROUP 1

The Group 1 operate microinstruction format is shown in Figure 8. and the microinstructions are explained in the succeeding paragraphs. Any logical combination of bits within this group can be combined into one microinstruction. For example, it is possible to assign ones to bits 5, 6, and 11; although it is not logical to assign ones to bits 8 and 9 simultaneously since they specify conflicting operations. (The most frequently used combinations are listed in Appendix 2.)



LOGICAL SEQUENCE:

1 — CLA, CLL
2 — CMA, CML
3 — IAC
4 — RAR, RAL, RTR, RTL,

Figure 8. Group 1 Operate Instruction Bit Assignments

No Operation (NOP)

Octal Code: 7000
Sequence: None
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: This command causes a 1-cycle delay in the program and then the next sequential instruction is initiated. This command is used to add execution time to a program, such as to synchronize subroutine or loop timing with peripheral equipment timing.
Symbol: None

Increment Accumulator (IAC)

Octal Code: 7001
Sequence: 3
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is incremented by one in two's complement arithmetic.
Symbol: AC + 1 => AC

Rotate Accumulator Left (RAL)

Octal Code: 7004
Sequence: 4
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is rotated one binary position to the left with the content of the link. The content of bits AC1 — 11 are shifted to the next greater significant bit, the content of ACO is shifted into the L, and the content of the L is shifted into AC11.
Symbol: ACj => ACj — 1
ACO => L
L => AC11

Rotate Two Left (RTL)

Octal Code: 7006
Sequence: 4
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is rotated two binary positions to the left with the content of the link. This instruction is logically equal to two successive RAL operations.
Symbol: ACj => ACj — 2
AC1 => L
ACO => AC11
L => AC10

Rotate Accumulator Right (RAR)

Octal Code: 7010
Sequence: 4
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is rotated one binary position to the right with the content of the link. The content of bits ACO — 10 are shifted to the next less significant bit, the content of AC11 is shifted into the L, and the content of the L is shifted into ACO.
Symbol: ACj => ACj + 1
AC11 => L
L => ACO

Rotate Two Right (RTR)

Octal Code: 7012
Sequence: 4
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is rotated two binary positions to the right with the content of the link. This instruction is logically equal to two successive RAR operations.
Symbol: ACj $=>$ Acj + 2
AC10 = L
AC11 = ACO
L $=>$ AC1

Complement Link (CML)

Octal Code: 7020
Sequence: 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the L is complemented.
Symbol: $\overline{L} => L$

Complement Accumulator (CMA)

Octal Code: 7040
Sequence: 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is set to the one's complement of the current content of the AC. The content of each bit of the AC is complemented individually.
Symbol: $\overline{ACj} => ACj$

Complement and Increment Accumulator (CIA)

Octal Code: 7041
Sequence: 2, 3
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the AC is converted from a binary value to its equivalent two's complement number. This conversion is accomplished by combining the CMA and IAC commands, thus the content of the AC is complemented during sequence 2 and is incremented by one during sequence 3.
Symbol: $\overline{ACj} => Acj$,
then AC + 1 $=>$ AC

Clear Link (CLL)

Octal Code: 7100
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the L is cleared to contain a 0.
Symbol: 0 $=>$ L

Set Link (STL)

Octal Code: 7120
Sequence: 1, 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The L is set to contain a binary 1. This instruction is logically equal to combining the CLL and CML commands.
Symbol: 1 = > L.

Clear Accumulator (CLA)

Octal Code: 7200
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of each bit of the AC is cleared to contain a binary 0.
Symbol: 0 = > AC

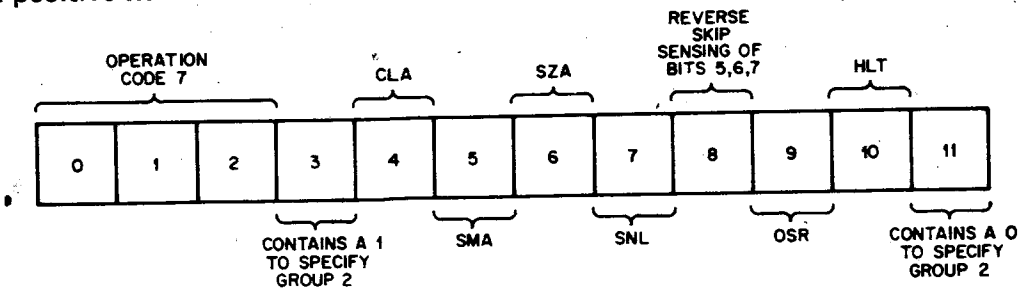
Set Accumulators (STA)

Octal Code: 7240
Sequence: 1, 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: Each bit of the AC is set to contain a binary 1. This operation is logically equal to combining the CLA and CMA commands.
Symbol: 1 = > ACj

GROUP 2

The Group 2 operate microinstruction format is shown in Figure 9. and the primary microinstructions are explained in the following paragraphs. Any logical combination of bits within this group can be composed into one microinstruction. (The instructions constructed by most logical command combinations are listed in Appendix 2.)

If skips are combined in a single instruction the inclusive OR of the conditions determines the skip when bit 8 is a 0; and the AND of the inverse of the conditions determines the skip when bit 8 is a 1. For example, if ones are designated in bits 6 and 7 (SZA and SNL), the next instruction is skipped if either the content of the AC = 0, or the content of L = 1. If ones are contained in bits 5, 7, and 8, the next instruction is skipped if the AC contains a positive number and the L contains a 0.



Logical Sequence:

1 (Bit 8 is a zero) — Either SMA or SZA or SNL
1 (Bit 8 is a one) — Both SPA and SNA and SZL
2 — CLA
3 — OSR, HLT

Figure 9. Group 2 Operate Instruction Bit Assignments

Halt (HLT)

Octal Code: 7402
Sequence: 3
Indicators: OPR, not RUN
Execution Time: 1.5 microseconds
Operation: Clears the RUN flip-flop at Sequence 3, so that the program stops at the conclusion of the current machine cycle. This command can be combined with others in the OPR 2 group that are executed during either sequence 1, or 2, and so are performed before the program stops.
Symbol: 0 = > RUN

OR with Switch Register (OSR)

Octal Code: 7404
Sequence: 3
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The inclusive OR operation is performed between the content of the AC and the content of the SR. The result is left in the AC, the original content of the AC is lost, and the content of the SR is unaffected by this command. When combined with the CLA command, the OSR performs a transfer of the content of the SR into the AC.
Symbol: ACj V Srj = > ACj

Skip, Unconditional (SKP)

Octal Code: 7410
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: PC + 1 = > PC

Skip on Non-Zero Link (SNL)

Octal Code: 7420
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the L is sampled, and if it contains a 1 the content of the PC is incremented by one so that the next sequential instruction is skipped. If the L contains a 0, no operation occurs and the next sequential instruction is initiated.
Symbol: If L = 1, then PC + 1 = > PC

Skip on Zero Link (SZL)

Octal Code: 7430
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the L is sampled, and if it contains a 0 the content of the PC is incremented by one so that the next sequential instruction is skipped. If the L contains a 1, no operation occurs and the next sequential instruction is initiated.
Symbol: If L = 0, then PC + 1 = > PC

Skip on Zero Accumulator (SZA)

Octal Code: 7440
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of each bit of the AC is sampled, and if any bit contains a 0 the content of the PC is incremented by one so that the next sequential instruction is skipped. If all bits of the AC contain a 0, no operation occurs and the next sequential instruction is initiated.
Symbol: If ACO $-$ 11 $=$ 0, then PC $+$ 1 $=>$ PC

Skip on Non-Zero Accumulator (SNA)

Octal Code: 7450
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of each bit of the AC is sampled, and if any bit contains a 1 the content of the PC is incremented by one so that the next sequential instruction is skipped. If all bits of the AC contain a 0, no operation occurs and the next sequential instruction is initiated.
Symbol: If ACO $-$ 11 $\neq$ 0, then PC $+$ 1 $=>$ PC

Skip on Minus Accumulator (SMA)

Octal Code: 7500
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the most significant bit of the AC is sampled, and if it contains a 1, indicating the AC contains a negative two's complement number, the content of the PC is incremented by one so that the next sequential instruction is skipped. If the AC contains a positive number no operation occurs and program control advances to the next sequential instruction.
Symbol: If ACO $=$ 1, then PC $+$ 1 $=>$ PC

Skip on Positive Accumulator (SPA)

Octal Code: 7510
Sequence: 1
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the most significant bit of the AC is sampled, and if it contains a 0, indicating a positive (or zero) two's complement number, the content of the PC is incremented by one so that the next sequential instruction is skipped. If the AC contains a negative number, no operation occurs and program control advances to the next sequential instruction.
Symbol: If ACO $=$ 0, then PC $+$ 1 $=>$ PC

Clear Accumulator (CLA)

Octal Code: 7600
Sequence: 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: Each bit of the AC is cleared to contain a binary 0.
Symbol: 0 $=>$ AC

PROGRAM INTERRUPT

The program interrupt feature allows certain external conditions to interrupt the computer program. It is used to speed the information processing of input-output devices or to allow certain alarms to halt the program in progress and initiate another routine. When a program interrupt request is made the computer completes execution of the instruction in progress before acknowledging the request and entering the interrupt mode. A program interrupt is similar to a JMS to location 0; that is, the content of the program counter is stored in location 0, and the program resumes operation in location 1. The interrupt program commencing in location 1 is responsible for identifying the signal causing the interruption, for removing the interrupt condition, and for returning to the original program. Exit from the interrupt program, back to the original program, can be accomplished by a JMP I Z 0 instruction.

Instructions

The two instructions associated with the program interrupt synchronization element are IOT microinstructions that do not use the IOP generator. These instructions are:

Interrupt Turn On (ION)

Octal Code: 6001
Event Time: Not applicable
Indicators: IOT, FETCH, ION
Execution Time: 1.5 microseconds
Operation: This command enables the computer to respond to a program interrupt request. If the interrupt is disabled when this instruction is given, the computer executes the next instruction, then enables the interrupt. The additional instruction allows exit from the interrupt subroutine before allowing another interrupt to occur. This instruction has no effect upon the condition of the interrupt circuits if it is given when the interrupt is enabled.
Symbol: 1 = > INT. ENABLE

Interrupt Turn Off (IOF)

Octal Code: 6002
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: This command disables the program interrupt synchronization element to prevent interruption of the current program.
Symbol: 0 = > INT. ENABLE, INT. DELAY

Programming

When an interrupt request is acknowledged, the interrupt is automatically disabled by the program interrupt synchronization circuits (not by instructions). The next instruction is taken from core memory location 1. Usually, the instruction stored in location 1 is a JMP, which transfers program control to a subroutine which services the interrupt. At some time during this subroutine, an ION instruction must be given. The ION can be given at the end of the subroutine to allow other interrupts to be serviced after program control is transferred back to the original program. In this application, the ION instruc-

tion immediately precedes the last instruction in the routine. A delay of one instruction (regardless of the execution time of the following instruction) is inherent in the ION instruction to allow transfer of program control back to the original program before enabling the interrupt. Usually exit from the subroutine is accomplished by a JMP I Z 0 instruction.

The ION command can be given during the subroutine as soon as it has determined the I/O device causing the interrupt. This latter method allows the subroutine which is handling a low priority interrupt to be interrupted, possibly by a high priority device. Programming of an interrupt subroutine which checks for priority and allows itself to be interrupted, must make provisions to relocate the content of the program counter stored in location 0; so that if interrupted, the content of the PC during the subroutine is stored in location 0, and the content of the PC during the original program is not lost.

CHAPTER 5

DATA BREAK

Peripheral equipment connected to the data break facility can cause a temporary suspension in the program in progress to transfer information with the computer core memory, via the MB. One I/O device can be connected directly to the data break facility or up to seven devices can be connected to it through the Type DM01 Data Multiplexer. This cycle stealing mode of operation provides a high-speed transfer of individual words or blocks of information at core memory addresses specified by the I/O device. Since program execution is not involved in these transfers, the program counter, accumulator, and instruction register are not disturbed or involved in these transfers. The program is merely suspended at the conclusion of an instruction execution, and the data break is entered to perform the transfer, then the Fetch state is entered to continue the main program.

Data breaks are of two basic types: single-cycle and three-cycle. In a single-cycle data break, registers in the device (or device interface) specify the core memory address of each transfer and count the number of transfers to determine the end of data blocks. (See discussion of Data Break Transfers in Chapter 10.) In the three-cycle data break two computer core memory locations perform these functions, simplifying the device interface by omitting two hardware registers.

The computer receives the following signals from the device during a data break:

Signal	−3 Volts	0 Volts
Break Request	No break request	Break request
Cycle Select	One-cycle break	Three-Cycle break
Transfer Direction	Data into PDP-8/I	Data out of PDP-8/I
Increment CA Inhibit	CA incremented	CA not incremented
Increment MB (pulse)	MB not incremented	MB incremented
Address (12 bits)	Binary 0	Binary 1
Data (12 bits)	Binary 0	Binary 1

The computer sends the following signals to the device during a data break:

Signal	Characteristics
Data (12 bits)	−3 volts = binary 0, 0 volts = binary 1
Address Accepted	.35 — .45 microsecond negative pulse beginning at TP4 of a break cycle.
WC Overflow	−3 volts level change occuring at TP2 time of the WC state and lasting for one machine cycle.
Buffered Break	−3 volts when in Break state.

(The above input/output signals can be changed to positive "bus" logic as described in Chapter 1.)

To initiate a data break an I/O device must supply four signals simultaneously to the data break facility. These signals are the Break Request signal, which sets the BRK SYNC flip-flop in the major state generator to control entry into the data break states (Word Count for a three-cycle data break or Break for a single-cycle data break); a Transfer Direction signal, supplied to the MB control element to allow data to be strobed into the MB from the peripheral

equipment and to inhibit reading from core memory; a Cycle Select Signal which controls gating in the major state generator to determine if the one-cycle or three-cycle data break is to be selected; and a core memory address of the transfer which is supplied to the input of the MA. When the break request is made, the data break replaces entry into the Fetch state of an instruction. Therefore the data break is entered at the conclusion of the Execute state of most memory reference instructions and at the conclusion of a Fetch state of augmented instructions. Having established the data break, each machine cycle is a Word Count, Current Address, or Break cycle until all data transfers have taken place, as indicated by removal of the Break Request signal by the peripheral equipment.

More exactly, the Break Request signal enables the BRK SYNC flip-flop. At TP1 time, the BRK SYNC flip-flop is set if the Break Request signal has been received, and is cleared otherwise.

At TP4 time of each machine cycle, the major state generator is set to establish the state for the cycle. At this time, the status of the BRK SYNC flip-flop is sampled and if in the 1 state, the Word Count or Break state is set into the major state generator and a data break commences.

Therefore, to initiate a data break, the Break Request must be at ground potential for at least 500 nanoseconds preceeding TP1 of the cycle preceeding the data break cycle. A Break Request signal should be supplied to the computer when the address, data, transfer direction, and cycle select signals are supplied to the computer.

When a data break occurs, the address designated by the device is loaded into the MA at TP4 time of the last cycle of the current instruction, and the major state generator is set to the World Count state if the Cycle Select signal is at ground, or is set to the Break state if this signal is at −3 volts. The program is delayed for the duration of the data break, commencing in the following cycle. A break request is granted only after completion of the current instruction as specified by the following conditions:

1. At the end of the Fetch cycle of an OPR or IOT instruction, or a directly addressed JMP instruction.
2. At the end of the Defer cycle of an indirectly addressed JMP instruction.
3. At the end of the Execute cycle of a JMS, DCA, ISZ, TAD, or AND instruction.

At the beginning of the Word Count cycle of a three-cycle data break or the Break cycle of a one-cycle data break, the address supplied to the input of the MA is strobed into the MA and the computer supplies an Address Accepted signal to the device. Entry into the Break cycle is indicated to the peripheral equipment by a Buffered Break signal and by an Address Accepted signal that can be used to enable gates in the device to perform tasks associated with the transfers. The Address Accepted signal is the most convenient control to be used by I/O equipment to disable the Break Request signal, since this signal must be removed at TP4 time to prevent continuance of the data break into the next cycle. If the Transfer Direction signal establishes the direction as out of the computer, the content of the core memory register at the address specified is transferred into the MB and is immediately available for strobing by the peripheral equipment. If the Transfer Direction signal specifies a data direction into the PDP-8/I, reading from core memory is inhibited and data is transferred into the MB from peripheral equipment.

The status of the BRK SYNC flip-flop is sensed at the beginning of a Break cycle to determine if an additional Break cycle is required. If a Break Request signal has been received since TP4, the Break state is maintained in the major state generator; if the Break Request signal has not been received by this time; the Fetch state is set into the major state generator to continue the program. The Break Request signal should be removed by the end of the Address Accepted signal if additional Break cycles are not required.

SINGLE-CYCLE DATA BREAK

One-cycle breaks transfer a data word into the computer core memory from the device, transfer a data word into a device from the core memory, or increment the content of a device-specified core memory location. In each of these types of data break one computer cycle is stolen from the program by each transfer; Break cycles occur singly (interleaved with the program steps) or continuously (as in a block transfer), depending upon the timing of the Break Request signal at rates of up to 660 kh.

During the memory strobe portion of the Break cycle, the content of the addressed cell is read into the MB if the transfer direction is out of the computer (into the I/O device). If the transfer direction is into the computer, generation of the Memory Strobe pulse is inhibited so that the MB (cleared during the previous cycle) remains cleared. Information is transferred from the output data register of the I/O device into the MB and is written into core memory during TS3 and TS4 times of the Break cycle. In an outward transfer, the write operation restores the original content of the address cell to memory.

If there is a further break request, another Break cycle is initiated. If there is no break request, the content of the PC is transferred into the MA, the IR is cleared, and the major state generator is set to Fetch. The program then executes the next instruction.

The increment MB facility is useful for counting iterations or events by means of a data break, so that the PC and AC are not disturbed. Within one Break cycle of 1.5 microseconds, a word is fetched from a device-specified core memory location, is incremented by one, and is restored to the same memory location. The Increment MB signal input must be supplied to the computer only during a Break cycle in which the direction of transfer is out of the PDP-8/I. These restrictions can be met by a simple AND-gate in the device; an Increment MB signal is generated only when an event occurs, the Buffered Break signal from the computer is present, and the Transfer Direction signal supplied to the computer is at ground potential.

THREE-CYCLE DATA BREAK

The three-cycle data break provides an economical method of controlling the transfer of data between the computer core memory and fast peripheral devices. Transfer rates in excess of 220 kh are possible using this feature of the PDP-8/I.

The three-cycle data break differs from the one-cycle break in that a ground-level Cycle Select signal is supplied so that when the data break conditions are fulfilled the program is suspended and the Word Count state is entered. The Word Count state is entered to increment the fixed core memory location containing the word count. The device requesting the break supplies this address as in the one-cycle break, except that this is a fixed address supplied by wired ground and —3v signals rather than from a register.

Following the Word Count state a Current Address state occurs in which the location following the Word Count address (bit 11 = 1 after + 1 = > MA) is read, incremented by one, restored to memory, and loaded into the MA to be used as the transfer address. Then the normal Break state is entered to effect the transfer between the device and the computer memory cell specified by the MA.

Word Count State

When this state is entered, the contents of the core memory address specified by the external device plus 1 is loaded into the MB at TP2 time. The word count, established previously by instructions, is the 2's completment negative number equal to the required number of transfers. If the word becomes 0 when incremented, the computer generates a WC overflow signal and supplies it to the device. During TS3 and TS4 times, the incremented word count is rewritten in memory, the contents of the MA is incremented by 1 to establish the next location as the address for the following memory cycle, and the major state generator is set to the Current Address state.

Current Address State

Operations during the second cycle of the three-cycle data break depend upon the condition of the Increment CA Inhibit (+1 → CA Inhibit) signal supplied to the computer from the I/O device. At TP2 time, the MB is loaded with either the contents of the memory cell following the word count (Current Address register) or the incremented contents of the current address register (i.e. if CA Inhibit is at ground, the contents are loaded; if CA Inhibit is at —3 volts, the incremented contents are loaded). The Current Address register may be incremented to advance the address of the transfer to the next sequential location. During TS3 and TS4 times, the contents of the MB is rewritten into core memory, the address word in the MB is transferred into the MA to designate the address to be used in the succeeding memory cycle, and the major state generator is set to Break state.

Break State

The actual transfer of data between the external device and the core memory, through the MB, occurs during the Break state as during a single-cycle data break, except that the address is determined by the current content of the MA rather than directly by the device.

CHAPTER 6

OPTIONAL MEMORY AND PROCESSOR EQUIPMENT AND INSTRUCTIONS

MEMORY EXTENSION CONTROL AND MEMORY MODULE (MC8/I) MEMORY MODULE (MM8/I)

(The logic for the Memory Extension Control Type MC8/I is located in the PDP-8/I central processor.)

Extension of the storage capacity of the standard 4096-word core memory is accomplished by adding fields of 4096-word core memories. Field select control and extended address control for up to 32,768 words is provided by the MEMORY EXTENSION CONTROL (MC8/I) that also adds 4096-word core memory internal to the machine for a total of 8K of memory. Each MEMORY MODULE (MM8/I) adds either 4096-word core memory or 8192 word core memory external to the machine. Up to six fields of 4096-word core memory can be added external to the machine, providing a maximum storage of 32,768 words (internal and external). Direct address of 32,768 words requires 15 bits ($2^{15} = 32,768$). However, since programs and data need not be directly addressed for execution of each instruction, a field can be program-selected, and all 12-bit addresses are then assumed to be within the current memory field. Program interrupt of a program in any field automatically specifies field 0, address 0 for storage of the program count. The memory extension control consists of several 3-bit flip-flop registers that extend addresses to 15 bits to establish or select a field.

Addition of a memory extension control to a standard PDP-8/I requires a simple modification of the operator console to activate indicators and switches associated with the instruction field register and the data field register of the control. These switches function in the same manner as the switch register, to load information into associated registers when the LOAD ADDRESS key is pressed.

The functional circuit elements which comprise the memory extension control perform as follows:

Instruction Field Register (IF): The IF is a 3-bit register that serves as an extension of the PC. The content of the IF determines the field from which all instructions are taken and the field from which operands are taken in directly-addressed AND, TAD, ISZ, or DCA instructions. Operating the LOAD ADDRESS key JAM transfers the contents of the INSTRUCTION FIELD switch register on the operator console into the IF register. During a JMP or JMS instruction the IF is set by a transfer of information contained in the instruction buffer register. When a program interrupt occurs, the content of the IF is automatically stored in bits 0 through 2 of the save field register for restoration to the IF from the instruction buffer register at the conclusion of the program interrupt subroutine.

Data Field Register (DF): This 3-bit register determines the memory field from which operands are taken in indirectly-addressed AND, TAD, ISZ, or DCA instructions. Operating the LOAD ADDRESS key JAM transfers the contents of the DATA FIELD switch register on the operator console into the DF register. The DF is set by a transfer of information from bits 6 through 8 of the MB during a CDF microinstruction to establish a microprogrammed data field. When a program interrupt occurs, the content of the DF is automatically stored in the save field register. The DF is set by a transfer of information from bits 3 through 5 of the save field register by the RMF microinstruction to restore the data field at the conclusion of the program interrupt subroutine.

Instruction Buffer Register (IB): The IB serves as a 3-bit input buffer for the instruction field register. All field number transfers into the instruction field register are made through the instruction buffer, except transfers from the operator console switches. The IB is set by operation of the LOAD ADDRESS key in the same manner as the instruction field register. A CIF microinstruction loads the IB with the programmed field number contained in MB 6-8. An RMF microinstruction transfers the content of bits 0 through 2 of the save field register into the IB to restore the instruction field to the conditions that existed prior to a program interrupt.

Save Field Register (SF): When a program interrupt occurs, this 6-bit register is loaded from the instruction field and data field registers. The RMF microinstruction can be given immediately prior to the exit from the program interrupt subroutine to restore the instruction field and data field by transferring the content of the SF into the instruction buffer and the data field register. The SF is cleared during the cycle in which the program count is stored at address 0000 of the JMS instruction forced by a program interrupt request, then the instruction field and data field are strobed into the SF.

Break Field Register (BF): This 3-bit register receives three ADDRESS EXTEND signals from any I/O device using the data break facility. When the B SET signal arrives from the processor, this register is loaded with the bit combination of the three inputs.

Extended Address Signal Generator When the PDP-8/I core memory capacity is extended, the standard memory is designated as field 0. This circuit produces the EXTEND ADDRESS FIELD 0 signal when data field 0 is selected, or instruction field 0 is selected. This circuit will produce the other seven possible EXTEND ADDRESS FIELD signals determined by the bit combination applied to its input.

Accumulator Transfer Gating: This gating allows the contents of the save field register, instruction field register, or the data field register to be strobed into the accumulator. Transfer of information in this manner is accomplished by circuits which sample the content of registers and supply positive pulses to the AC upon receipt of IOT command pulses. During an RIB microinstruction, bits 6 through 11 of the AC are set by the content of the save field register. During an RIF microinstruction, bits 6 through 8 of the AC are set by the content of the instruction field register. During an RDF microinstruction, bits 6 through 8 of the AC are set by the content of the data field register.

Device Selector: Bits 3 through 5 of the IOT instruction are decoded to produce the IOT command pulses for the memory extension control. Bits 6 through 8 of the instruction are not used for device selection since they specify a field number in some commands. Therefore, the select code for this device selector is designated as 2X.

The Memory Control and Memory Module option Type MM 8/I adds memory control and read/write switches for each 8K of additional memory. Two 4K memory modules or one 4K memory module may be specified. Each 4K memory consists of a core array, address selection circuits and inhibit selection circuits which are identical with those housed with the PDP-8/I.

Instructions

The instructions for the Type MC 8/I option do not use the IOP generator and extend the IOT instruction list to include the following:

Change to Data Field N (CDF)

Octal Code: 62N1
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: The data field register is loaded with the program-selected field number (N = 0 to 7). All subsequent memory requests for operands are automatically switched to that data field until the data field number is changed by a new CDF command, or during a program interrupt.
Symbol: MB6 − 8 = > DF

Change Instruction Field (CIF)

Octal Code: 62N2
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: The instruction buffer register is loaded with the program-selected field number (N = 0 to 7). The next JMP or JMS instruction causes the new field to be entered.
Symbol: MB6 − 8 = > IB

Read Data Field (RDF)

Octal Code: 6214
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the data field register is transferred into bits 6, 7, 8 of the AC. All other bits of the AC are unaffected.

Symbol: DF = > AC6 − 8

Read Instruction Field (RIF)

Octal Code: 6224
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the instruction field register is transferred into bits 6, 7, 8 of the AC. All other bits of the AC are unaffected.
Symbol: IF = > AC6 − 8

Read Interrupt Buffer (RIB)

Octal Code: 6234
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: The instruction field and data field held in the save field register during a program interrupt are transferred into bits 6 through 8, and 9 through 11 of the AC respectively.
Symbol: SF 0 − 2 = > AC6 − 8
SF 3 − 5 = > AC9 − 11

Restore Memory Field (RMF)

Octal Code: 6244
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: This command is used upon exit from the program interrupt subroutine in another field. The data and instruction fields that were interrupted by the subroutine are restored by transferring the content of the save field register into the instruction buffer and data field registers.
Symbol:

SF 0 − 2 => IB
SF 3 − 5 => DF

Programming

Instructions and data are accessed from the currently assigned instruction and data fields, where instructions and data may be stored in the same or different memory fields. When indirect memory references are executed, the operand address refers first to the instruction field to obtain an effective address, which in turn, refers to a location in the currently assigned data field. All instructions and operands are obtained from the field designated by the content of the instruction field register, except for indirectly-addressed operands which are specified by the content of the data field register. In other words, the DF is effective only in the Execute cycle that is directly preceded by the Defer cycle of a memory reference instructions, as follows:

Indirect (Bit 3)	Page or Z Bit (Bit 0)	Field In IF	Field In DF	Effective Address
0	0	m	n	The operand is in page 0 of field m at the page address specified by bits 5 through 11.
0	1	m	n	The operand is in the current page of field m at the page address specified by bits 5 through 11.
1	0	m	n	The absolute address of the operand in field n is taken from the content of the location in page 0 of field m designated by bits 5 through 11.
1	1	m	n	The absolute address of the operand in field n is taken from the content of the location in the current page of field m designated by bits 5 through 11.

Each field of extended memory contains eight autoindex registers in addresses 10 through 17. For example, assume that a program in field 2 is running (IF = 2) and using operands in field 1 (DF = 1) when the instruction TAD I 10 is fetched. The Defer cycle is entered (bit 3 = 1) and the content of location 10 in field 2 is read, incremented, and rewritten. If address 10 in field 2 originally contained 4321, it now contains 4322. In the Execute cycle the operand is fetched from location 4322 of field 1.

Program control is transferred between memory fields by the CIF commands. The instruction does not change the instruction field directly, since this would make it impossible to execute the next sequential instruction. The CIF instruction sets the new instruction field into the IB for automatic transfer into the IF when either a JMP or JMS instruction is executed. The DF is unaffected by the JMP and JMS instructions. The 12-bit program counter is set in the normal manner and, since the IF is an extension on the most significant end of the PC, program sequence resumes in the new memory field following a JMP or JMS. Entry into a program interrupt is inhibited after the CIF instruction until a JMP or JMS is executed.

To call a subroutine that is out of the current field, the data field register is set to indicate the field of the calling JMS, which establishes the location of the operands as well as the identity of the return field. The instruction field is set to the field of the starting address of the subroutine. The following sequence returns program control to the main program from a subroutine that is out of the current field.

```
/PROGRAM OPERATIONS IN MEMORY FIELD 2
/INSTRUCTION FIELD = 2; DATA FIELD = 2
/CALL A SUBROUTINE IN MEMORY FIELD 1
/INDICATE CALLING FIELD LOCATION BY THE CONTENT OF THE DATA FIELD

            CIF         10          /CHANGE TO INSTRUCTION
                                    /FIELD 1 = 6212
            JMS         1 SUBRP     /SUBRP = ENTRY ADDRESS
            CDF         20          /RESTORE DATA FIELD
SUBRP,      SUBR                    /POINTER
/CALLED SUBROUTINE
            0                       /SUBR = PC + 1 AT CALLING POINT
            RDF                     /READ DATA FIELD INTO AC
            TAD RETURN              /CONTENT OF THE AC = 6202 + DATA
                                    /FIELD BITS
            DCA EXIT                /STORE INSTRUCTION SUBROUTINE
              .
              .
              .
              .
EXIT,       0                       /A CIF INSTRUCTION
            JMP I SUBR              /RETURN
RETURN,     CIF
```

When a program interrupt occurs, the current instruction and data field numbers are automatically stored in the 6-bit save field register, then the IF and DF are cleared. The 12-bit program count is stored in location 0000 of field 0 and program control advances to location 0001 of field 0. At the end of the program interrupt subroutine the RMF instruction restores the IF and DF from the content of the SF. The following instruction sequence at the end of the program interrupt subroutine continues the interrupted program after the interrupt has been processed:

```
.                    /RESTORE MQ IF REQUIRED
.
.
.
.                    /RESTORE L IF REQUIRED
.
.
.
CLA
TAD  AC              /RESTORE AC
RMF                  /LOAD IB FROM SF
ION                  /TURN ON INTERRUPT SYSTEM
JMP  I 0             /RESTORE PC WITH CONTENT OF
                     /LOCATION 0 AND LOAD IF FROM IB
```

A device using the computer data break facility supplies a 12-bit address to the MA and a 3-bit address extension to the Memory Extension Control Type MC 8/I. The address extension is received by a break field decoder which selects the memory field used for the data break.

MEMORY PARITY (MP-8/I) (The logic for this option is housed within the PDP-8/I central processor.)

Data transmission checking of each word written in and read from core memory is provided by this option. The option replaces the 12-bit core memory with a 13-bit system (driving, inhibiting, sensing circuits as well as a core array constructed of 13 planes) and includes a parity generator and a parity checking circuit. The parity generator produces the 13th bit for each 12-bit data word written in core memory so that the entire word contains an odd number of binary ones. The parity checking circuit monitors each word read from core memory to assure that the odd parity is maintained. If a word read contains an even number of ones a transmission error is indicated by setting a parity error flag. This flag is connected to the program interrupt synchronization element of the computer to initiate a program interrupt subroutine. This routine sequentially checks all equipment error flags to determine the option causing the interrupt and initiates an appropriate service and returns to the main program; or provides a suitable error printout and halts programmed operations. Upon determining that a memory parity error has occurred the program interrupt subroutine can repeat the main program step that caused the error to check the reliability of the error condition, can perform a simple write/read/check routine at the error address, or can determine the status of the machine when the error was detected and re-establish or print out these conditions and halt.

Instructions

Two instructions are associated with the Type MP 8/I option. They are:

Skip on No Memory Parity Error (SMP)
Octal Code: 6101
Event Time: 1
Indicator: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds

Operation: The memory parity error flag is sensed and if it contains a 0 (signifying no error has been detected) the PC is incremented so that the next successive instruction is skipped.
Symbol: If Memory Parity Error Flag = 0, then PC + 1 = > PC

Clear Memory Parity Error Flag (CMP)
Octal Code: 6104
Event Time: 3
Indicator: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The memory parity error flag is cleared.
Symbol: 0 = > Memory Parity Error Flag

Programming

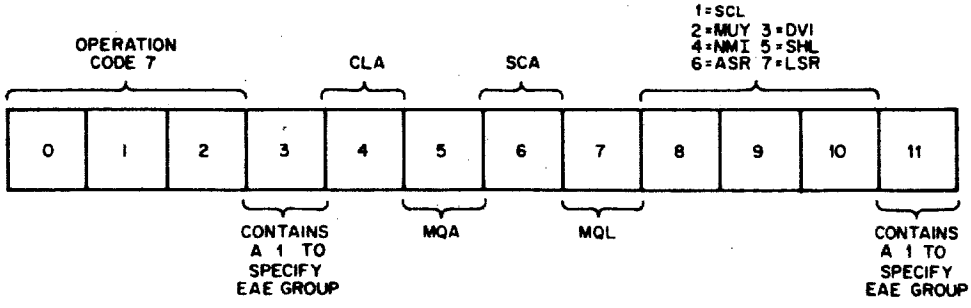
Both instructions for this option are used in the program interrupt subroutine and in diagnostic maintenance programs. The SMP command is used as a programmed check for memory parity error. In the program interrupt subroutine this command can be followed by a jump to a portion of the routine that services the memory parity option as described previously. The CMP command is used to initialize the memory parity option in preparation for normal programmed operation of the computer.

EXTENDED ARITHMETIC ELEMENT (KE-8/I) (The logic for this option is located within the PDP-8/I central processor.)

This option consists of circuits that perform parallel arithmetic operations on positive binary numbers. A 12-bit multiplier quotient register (MQ), a 5-stage step counter (SC), and various shifting and control logic constitute the option. The AC and MB are used in conjunction with these logic elements to perform arithmetic operations. With the addition of this option to a PDP-8/I system, indicators on the operator console for the content of each bit of the MQ are activated and a class of instructions is added to the Group 2 Operate instruction list.

Instructions

The extended arithmetic element (EAE) microinstructions are specified by an operate instruction (operation code 7) in which bits 3 and 11 contain binary ones. Being augmented instructions, the EAE commands are microprogrammed and can be combined with each other to perform non-conflicting logical operations. Format and bit assignments of the EAE commands are indicated in Figure 10.



Logical Sequence:

1 — CLA
2 — MQA, MQL, SCA
3 (Bits 8 thru 10 = 1) — SCL
3 (Bits 8 thru 10 = 2) — MUY
3 (Bits 8 thru 10 = 3) — DVI
3 (Bits 8 thru 10 = 4) — NMI
3 (Bits 8 thru 10 = 5) — SHL
3 (Bits 8 thru 10 = 6) — ASR
3 (Bits 8 thru 10 = 7) — LSR

Figure 10. EAE Microinstruction Bit Assignments

Multiply (MUY)

Octal Code: 7405
Sequence: 3
Indicators: OPR, FETCH, PAUSE
Execution Time: 4.8 to 7.2 microseconds
Operation: The number held in the MQ is multiplied by the number held in core memory location PC + 1 (or the next successive core memory location after the MUY command). At the conclusion of this command the link contains a 0, the most significant 12 bits of the product are contained in the AC and the least significant 12 bits of the product are contained in the MQ.
Symbol:

$$Y \times MQ => AC, MQ$$
$$0 => L$$

Divide (DVI)

Octal Code: 7407
Sequence: 3
Indicators: OPR, FETCH, PAUSE
Execution Time: 5.2 to 7.8 microseconds
Operation: The 24-bit dividend held in the AC (most significant 12 bits) and the MQ (least significant 12 bits) is divided by the divisor held in core memory location PC + 1 (or the next successive core memory location following the DVI command). At the conclusion of this command the quotient is held in the MQ, the remainder is in the AC, and the L contains a 0. If the L contains a 1, divide overflow occurred so the operation was concluded after the first cycle of the division.

Symbol: $AC, MQ \div Y => MQ$

Normalize (NMI)

Octal Code: 7411
Sequence: 3
Indicators: OPR, FETCH, PAUSE
Execution Time: 1.5 microseconds + 0.25 microsecond for each shift
Operation: This instruction is used as part of the conversion of a binary number to a fraction and an exponent for use in floating-point arithmetic. The combined content of the AC and the MQ is shifted left by this one command until the content of AC0 is not equal to the content of AC1, or until 6000 0000 is contained in the combined AC and MQ, to form the fraction. Zeros are shifted into vacated MQ11 positions for each shift. At the

conclusion of this operation, the step counter contains a number equal to the number of shifts performed, which can be loaded into the AC by an SCA command to form the exponent. The content of L is lost. Both positive and negative two's complement numbers can be normalized.
Symbol:

ACj => ACj — 1
AC0 => L
MQ0 => AC11
MQj => MQj — 1
0 => MQ11 until AC0 ≠ AC1 or until AC MQ = 6000 0000

Shift Arithmetic Left (SHL)

Octal Code: 7413
Sequence: 3
Indicators: OPR, FETCH, PAUSE
Execution Time: 3.0 microseconds + 0.25 microsecond for each shift
Operation: This instruction is used for scaling by shifting the combined content of the AC and MQ to the left one position more than the number of positions indicated by the content of core memory at address PC + 1 (or the next successive core memory location following the SHL command). During the shifting, zeros are shifted into vacated MQ11 positions. The L, AC, and MQ are treated as one long register during this operation. Bits shifted out of AC0 enter the L, and bits shifted out of the L are lost.
Symbol:

Shift Y + 1 positions as follows:
ACj => ACj — 1
AC0 => L
MQ0 => AC11
MQj => MQj — 1
0 => MQ11

Arithmetic Shift Right (ASR)

Octal Code: 7415
Sequence: 3
Indicators: OPR, FETCH, PAUSE
Execution Time: 3.0 microseconds + 0.25 microsecond for each shift.
Operation: This instruction is used for scaling and treats the AC and MQ as one long register. The combined content of the AC and the MQ is shifted right one position more than the number contained in memory location PC + 1 (or the next successive core memory location following the ASR command). The sign bit, contained in AC0, enters vacated positions, the sign bit is preserved in the link, informaton shifted out of MQ11 is lost, and the L is set to correspond to the sign bit during this operation.
Symbol:

Shift Y + 1 positions as follows:
AC0 => L
AC0 => AC0
ACj => ACj + 1
AC11 => MQ0
MQj => MQj + 1

Logical Shift Right (LSR)

Octal Code: 7417
Sequence: 3
Indicators: OPR, FETCH, PAUSE
Execution Time: 3.0 microseconds + 0.25 microsecond for each shift.
Operation: This instruction is used for scaling and treats the AC and MQ as one long register. The combined content of the AC and MQ is shifted right one position more than the number contained in memory location PC + 1 (or the next successive core memory location following the LSR command). This command is similar to the ASR command except that zeros enter vacated positions instead of the sign bit entering these locations. Information shifted out of MQ11 is lost and the L is cleared during this operaton.

Shift Y + 1 positions as follows:
0 => L
0 => ACO
ACj => ACj + 1
AC11 => MQ0
MQj => MQj + 1

Load Multiplier Quotient (MQL)

Octal Code: 7421
Sequence: 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: This command clears the MQ, loads the content of the AC into the MQ, then clears the AC. This operation is essential to initializing any multiply or divide routine and can be combined with a MUY or DVI command to perform the operation just prior to executing a multiplication or a division using a 12-bit dividend.
Symbol:

0 => MQ
AC => MQ
0 => AC

Step Counter Load from Memory (SCL)

Octal Code: 7403
Sequence: 3
Indicators: OPR, FETCH, EXECUTE
Execution Time: 3.0 microseconds
Operation: Loads complement of bits 7 thru 11 of the word in memory following the instruction into the step counter.
Symbol:

$\overline{MB_{7-11}}$ => SC
PC + 2 => PC

Step Counter Load into Accumulator (SCA)

Octal Code: 7441
Sequence: 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the step counter is transferred into the AC. This command is used following an NMI command to establish the exponent of a normalized number to be used in floating point arithmetic. The AC should be cleared prior to issuing this command or the CLA command can be combined with the SCA to clear the AC then effect the transfer.
Symbol: SC V AC => AC

Multiplier Quotient Load into Accumulator (MQA)

Octal Code: 7501
Sequence: 2
Indicators: OPR, FETCH
Execution Time: 1.5 microseconds
Operation: The content of the MQ is transferred into the AC. This command is given to load the 12 least significant bits of the product into the AC following a multiplication or to load the quotient into the AC following a division. The AC should be cleared prior to issuing this command or the CLA command can be combined with the MQA to clear the AC then effect the transfer.
Symbol: MQ V AC = > AC

Clear Accumulator (CLA)

Octal Code: 7601
Sequence: 1
Execution Time: 1.5 microseconds
Indicators: OPR, FETCH
Operation: The AC is cleared, allowing this command to be combined with the other EAE commands that load the AC (such as SCA and MQA).
Symbol: 0 = > AC

Programming

MULTIPLICATION

Multiplication is performed as follows:

1. Load the AC with the multiplier using the TAD instruction.
2. Transfer the content of the AC into the MQ using the MQL command.
3. Give the MUY command.

Note that steps 2 and 3 can be combined into one instruction.
The content of the MQ is then multiplied by the content of the next successive core memory address (PC + 1). At the conclusion of the multiplication the most signicant 12 bits of the product are held in the AC and the least significant bits are held in the MQ. This operation takes a maximum of 7.2 microseconds, at the end of this time the next instruction is executed.

The following multiplication program examples indicate the operation of the KE 8/I option in closed subroutines (routines which are incorporated into larger routines and are not written in a form which allows them to be called as a normal mathematical subroutine).

Multiplication of 12-Bit Unsigned Numbers

Enter with a 12-bit multiplicand in AC and a 12-bit multiplier in core memory. Exit with high order half of product in a core memory location labelled HIGH, and with low order half of product in the AC. Program time is from 9.3 to 11.7 microseconds.

```
MQL MUY     /LOAD MQ WITH MULTIPLICAND, INITIATE
            /MULTIPLICATION
MLTPLR      /MULTIPLIER
DCA HIGH    /STORE HIGH ORDER PRODUCT
MQA         /LOAD AC WITH LOW ORDER PRODUCT
```

Multiplication of 12-Bit Signed Numbers, 24-Bit Signed Product

Enter with a 12-bit multiplicand in AC and a 12-bit multiplier in core memory. Exit with signed 24-bit product in core memory locations designated HIGH and LOW. Program time is from 36.3 to 52.5 microseconds.

```
        CLL
        SPA                     /MULTIPLICAND POSITIVE?
        CMA CML IAC             /NO. FORM TWO'S COMPLEMENT
        MQL                     /LOAD MULTIPLICAND INTO MQ
        TAD     MLTPLR
        SPA                     /MULTIPLIER POSITIVE?
        CMA CML IAC             /NO. FORM TWO'S COMPLEMENT
        DCA     MLTPLR
        RAL
        DCA     SIGN            /SAVE LINK AS SIGN INDICATOR
        MUY                     /MULTIPLY
MLTPLR, 0                       /MULTIPLIER
        DCA     HIGH
        TAD     SIGN
        RAR                     /LOAD LINK WITH SIGN INDICATOR
        MQA
        SNL                     /IS PRODUCT NEGATIVE?
        JMP     LAST            /NO
        CLL CMA IAC             /YES
        DCA     LOW
        TAD     HIGH
        CMA
        SZL
        IAC
        DCA     HIGH
        SKP
LAST,   DCA     LOW
```

DIVISION

Division is performed as follows:

1. Load the 12 least significant bits of the dividend into the AC using the TAD instruction, then transfer the content of the AC into the MQ using the MQL command.
2. Load the 12 most significant bit of the dividend into the AC.
3. Give the DVI command.

The 24-bit dividend contained in the AC and MQ is then divided by the 12-bit divisor contained in the next successive core memory address (PC + 1). This operation takes a maximum of 7.8 microseconds and is concluded with a 12-bit quotient held in the MQ, the 12-bit remainder in the AC, and the link holding a 0 if divided overflow did not occur. To prevent divide overflow, the divisor in the core memory must be greater than the 12-bits of the dividend held in the AC. When divide overflow occurs, the link is set and the division

is concluded after only one cycle. Therefore the instruction following the divisor in core memory should be an SZL microinstruction to test for overflow. The instruction following the SZL can be a jump to a subroutine that services the overflow. This subroutine can cause the program to type out an error indication, rescale the divisor or the dividend, or perform other mathematical corrections and repeat the divide routine.

The following division program examples indicate the operation of the Type KE 8/I option in closed subroutines.

Division of 12-Bit Unsigned Numbers

Enter with a 12-bit unsigned dividend in the AC and a 12-bit unsigned divisor in core memory. Exit with remainder in core memory location labeleld REMAIN and with the quotient in the AC. Program time is a maximum of 15.3 microseconds.

```
CLL
MQL  DVI          /LOAD MQ, INITIATE DIVISION
DIVSOR            /DIVISOR
SZL               /OVERFLOW?
JMP               /YES, EXIT
DCA REMAIN
MQL               /LOAD AC WITH QUOTIENT
```

Division of a 12-Bit Signed Numbers

Enter with a 12-bit signed dividend in the AC and a 12-bit signed divisor in core memory. Exit with unsigned remainder in core memory location REMAIN and a 12-bit signed quotient in the AC. Program time is a maximum of 36.3 microseconds.

```
CLL
SPA               /DIVIDEND POSITIVE?
CMA CML IAC       /NO
MQL
TAD .+11
SPA               /DIVISOR POSITIVE?
CMA CML IAC       /NO
DCA .+6
SNL               /QUOTIENT NEGATIVE?
CMA               /NO
CLL
DCA SIGN          /SET SIGN INDICATOR
DVI
DIVSOR            /DIVISOR
SZL               /OVERFLOW
JMP               /EXIT ON OVERFLOW
MQL
ISZ SIGN
CMA IAC
```

POWER FAILURE (KP-8/I) (The logic for this option is housed within the PDP-8/I central processor.)

This prewired option protects an operating program in the event of failure of the source of computer primary power. If a power failure occurs, this option causes a program interrupt and enables continued operation for 1 millisecond, allowing the interrupt routine to detect the power low condition as initiator of the interrupt, and to store the content of active regisers (AC, L, MQ,etc.) and the program count in known core memory locations. When power is restored, the power low flag clears and a routine beginning in address 0000 starts automatically. This routine restores the content of the active registers and program counter to the conditions that existed when the interrupt occurred, then continues the interrupted program.

The KP8/I option consists of three logic circuits:

A power interrupt circuit monitors the status signal of the computer power supply, and sets a power low flag when power is interrupted (due to a power failure or due to the operation of the POWER lock on the operator console). This flag causes a program interrupt when an interruption in computer power is detected.

A restart circuit assures that when a power interrupt occurs the logic circuits of the computer continue operation for 1 millisecond to allow a program subroutine to store the content of the active registers; maintains the inoperative condition of the computer during periods of power fluctuation; and clears the power low flag and restarts the program when power conditions are suitable for computer operation. A manual RESTART switch enables or disables the automatic restart operation. With this switch in the ON (down) position, the option clears the program counter immediately and produces a signal to simulate operation of the START key on the operator console 200 milliseconds after power conditions are satisfactory. The PC is clear so that operation restarts by executing the instruction in address 0000. This instruction is a JMP to the starting address of the subroutine which restores the content of the active registers and the program counter to the conditions that existed prior to the power low interrupt. The 200-millisecond delay assures that slow mechanical devices, such as Teletype equipment, have come to a complete stop before the program is resumed. Simulation of the manual START function causes the processor to generate a Power Clear pulse to clear internal controls and I/O device registers. With the RESTART switch in the OFF (up) position, the power low flag is cleared but the program must be started manually, possibly after resetting peripheral equipment or by starting the interrupted program from the beginning.

A skip circuit provides programmed sensing of the condition of the power low flag by adding the following instruction to the computer repertoire:

Skip on Power Low (SPL)
Octal Code: 6102
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds

Operation: The content of the power low flag is sampled, and if it contains a 1 (indicating a power failure has been detected) the content of the PC is incremented by one so the next sequential instruction is skipped.

Symbol: If Power Low flag = 1, then PC +1 = > PC

Since the time that operation of the computer can be extended after a power failure is limited to 1 millisecond, the condition of the power low flag should be the first status check made by the program interrupt subroutine. The beginning of the program interrupt subroutine, containing the SPL microinstruction and the power fail program sequence can be executed in 26 microseconds on a basic PDP-8/I with an extended arithmetic element. The power fail program sequence stores the content of the active register and program count in designated core memory location, then relocates the calling instruction of the power restore subroutine to address 0000, as follows:

```
Address   Instruction        Remarks
0000          —          /STORAGE FOR PC AFTER PROGRAM INTERRUPT
0001      JMP FLAGS      /INSTRUCTION EXECUTED AFTER PROGRAM
                         /INTERRUPT
FLAGS,    SPL            /SKIP IF POWER LOW FLAG = 1
          JMP OTHER      /INTERRUPT NOT CAUSED BY POWER LOW,
                         /CHECK OTHER FLAGS
          DCA AC         /INTERRUPT WAS CAUSED BY POWER LOW,
                         /SAVE AC
          RAR            /GET LINK
          DCA LINK       /SAVE LINK
          MQA            /GET MQ
          DCA MQ         /SAVE MQ
          TAD 0000       /GET PC
          DCA PC         /SAVE PC
          TAD RESTRT     /GET RESTART LOCATION
          DCA 0000       /DEPOSIT RESTART LOCATION IN 0000
          HLT
RESTRT    JMP ABCD       /ABCD IS LOCATION OF RESTART ROUTINE
```

Automatic program restart begins by executing the instruction stored in address 0000 by the power fail routine. The power restore subroutine restores the content of the active registers, enables the program interrupt facility, and continues the interrupted program from the point at which it was interrupted, as follows:

```
Address   Instruction        Remarks
0000      JMP ABCD
ABCD,     TAD MQ         /GET MQ
          MQL            /RESTORE MQ
          TAD LINK       /GET LINK
          CLL RAL        /RESTORE LINK
          TAD AC         /RESTORE AC
          ION            /TURN ON INTERRUPT
          JMP I PO       /RETURN TO INTERRUPTED PROGRAM
```

CHAPTER 7
INPUT/OUTPUT EQUIPMENT INSTRUCTIONS

TELETYPE AND CONTROL
Teletype Model 33 ASR

(The control circuitry for this device is located in the PDP-8/I central processor.)

The standard Teletype Model 33 ASR (automatic send-receive) can be used to type in or print out information at a rate of up to ten characters per second, or to read in or punch out perforated paper tape at a ten characters per second rate. Signals transferred between the 33 ASR and the control logic are standard serial, 11 unit code Teletype signals. The signals consist of marks and spaces which correspond to idle and bias current in the Teletype, and to zeros and ones in the control and computer. The start mark and subsequent eight character bits are one unit of time duration and are followed by the stop mark which is two units.

The 8-bit code used by the Model 33 ASR Teletype unit is the American Standard Code for Information Interchange (ASCII) modified. To convert the ASCII code to Teletype code add 200 octal (ASCII + 200_8 = Teletype). This code is read in the reverse of the normal octal form used in the PDP-8/I since bits are numbered from right to left, from 1 through 8, with bit 1 having the least significance. Therefore perforated tape is read:

8 7 6 5 4 S 3 2 1

6 5 4: Most Significant Octal Bit

3 2 1: Least Significant Octal Bit

The Model 33 ASR set can generate all assigned codes except 340 through 374 and 376. Generally codes 207, 212, 215, 240 through 337, and 377 are sufficient for Teletype operation. The Model 33 ASR set can detect all characters, but does not interpret all of the codes that it can generate as commands. The standard number of characters printed per line is 72. The sequence for proceeding to the next line is a carriage return followed by a line feed (as opposed to a line feed followed by a carriage return). Appendix 3 lists the character code for the Teletype. Punched tape format is as follows:

	Tape Channel			
	87	654	S	321
Binary Code (Punch = 1)	10	110		100
Octal Code	2	6		4

Teletype Control

Serial information read or written by the Teletype unit is assembled or disassembled by the control for parallel transfer to the accumulator of the processor. The control also provides the program flags which cause a program interrupt or an instruction skip based upon the availability of the Teletype and the processor as a function of the program.

In all programmed operation, the Teletype unit and control are considered as a Teletype in (TTI) as a source of input intelligence from the keyboard or the perforated-tape reader and is considered a Teletype out (TTO) for computer output information to be printed and/or punched on tape. Therefore; two device selectors are used; the select code of 03 initiates operations associated with the keyboard/reader, and the device selector, assigned the select code of 04, performs operations associated with the teleprinter/punch. Parallel input and output functions are performed by corresponding IOT pulses produced by the two device selectors. Pulses produced by IOP1 pulse trigger skip gates; pulses produced by the IOP2 pulse clear the control flags and/or the accumulator; and pulses produced by the IOP4 pulse initiate data transfers to or from the control.

Keyboard/Reader

The keyboard and tape reader control contains an 8-bit buffer (TTI) which assembles and holds the code for the last character struck on the keyboard or read from the tape. Teletype characters from the keyboard/reader are received serially by the 8-bit shift register TTI. The code of a teletype character is loaded into the TTI so that spaces correspond with binary zeros and holes (marks) correspond to binary ones. Upon program command the content of the TTI is transferred in parallel to the accumulator.

When a Teletype character starts to enter the TTI the control de-energizes a relay in the Teletype unit to release the tape feed latch. When released, the latch mechanism stops tape motion only when a complete character has been sensed, and before sensing of the next character is started.

A keyboard flag is set to a binary one, and causes a program interrupt when an 8-bit computer character has been assembled in the TTI from a Teletype character. The program must sense the condition of this flag with a KSF microinstruction, and if the flag is set, issue a KRB microinstruction which clears the AC, clears the keyboard flag, transfers the content of the TTI into the AC, and enables advance of the tape feed mechanism.

Instructions for use in supplying data to the computer from the Teletype are:

Skip on Keyboard Flag (KSF)

Octal Code: 6031
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The keyboard flag is sensed, and if it contains a binary 1 the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Keyboard Flag = 1, then PC + 1 = > PC

Clear Keyboard Flag (KCC)

Octal Code: 6032
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds

Operation: The AC is cleared in preparation for another microinstruction to transfer a character from the TTI into the AC. The keyboard flag is also cleared, this allows the hardware to begin assembling the next input charac-

ter in the TTI. If there is tape in the reader and the reader is on, the character over the read head will be loaded into the TTI and the tape advanced one frame. If there is no tape or the reader is turned off (STOP or FREE) the character struck on the keyboard will be assembled into the TTI. In either case, when the character is completely assembled in the TTI the hardware causes the keyboard flag to be set to a binary 1.

Symbol: 0 = > AC
0 = > Keyboard flag allowing the hardware to cause:
Keyboard/Tape Character = > TTI
1 > Keyboard flag when done

Read Keyboard Buffer Static (KRS)

Octal Code: 6034
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the TTI is transferred into bits 4 through 11 of the AC. This is a static command in that neither the AC nor the keyboard flag is cleared.
Symbol: TTI V AC 4-11 = > AC 4-11

Read Keyboard Buffer Dynamic (KRB)

Octal Code: 6036
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: This microinstruction combines the functions of the KCC and KRS. The AC and keyboard flag are both cleared and the content of the TTI is transferred into bits 4-11 of the AC. Clearing the keyboard flag allows the hardware to begin assembling the next input character into the TTI (as discussed with the KCC). When the character is completely assembled in the TTI, the hardware causes the flag to be set indicating it again has a character ready for transfer.

Symbol: 0 = > AC C(TTI) V C(AC 4-11) = > AC 4-11
0 = > Keyboard Flag allowing the hardware to cause:
Keyboard/Tape Character = > TTI
1 = > Keyboard flag when done.

The following are examples of possible sequences of instruction to read a character into the AC from the teletype:

```
LOOK,   KSF          /SKIP IF FLAG = 1
        JMP LOOK     /JMP BACK & TEST FLAG AGAIN
        KRB          /TRANSFER TTI CONTENTS INTO AC
```

This sequence waits for the TTI to set its flag, indicating that it has a character ready to be transferred. It then skips to the KRB command which causes the character to be read into the AC from the TTI.

By making this sequence of instructions a subroutine of a larger program, it can be accessed each time an input character is desired.

```
READ,   0               /STORE DC HERE FOR RETURN ADDRESS
        KSF             /SKIP IF FLAG = 1
        JMP.-1          /TEST FLAG AGAIN
        KRB             /READ CHAR INTO AC
        JMP I READ      /EXIT TO MAIN PROGRAM
        .
        .
        .
```

The above sequence will operate properly on a PDP-8/I since all flags are cleared upon pressing START, however, the flags are not cleared on the PDP-5 when START is pressed, hence the reader flag should be cleared by a KCC as part of the initialization done at the beginning of any program. Failure to clear this flag could cause an extraneous character to be input (whatever happened to be in the TTI buffer would be interpreted as the first input character).

```
        KCC             /CLEAR TTI FLAG
        .
        .
        .
READ,   0
        KSF             /SKIP IF FLAG = 1
        JMP.-1          /TEST FLAG AGAIN
        KRB             /READ CHARACTER INTO AC
        JMP I READ      /EXIT
        .
        .
        .
```

Teleprinter/Punch

On program command a character is sent in parallel from the accumulator (AC) to the TTO shift register for transmission to the teleprinter/punch unit. The control generates the start space, then shifts the eight character bits serially into the printer selector magnets of the teletype unit, and then generates the stop marks. This transfer of information from the TTO into the teleprinter/punch unit is accomplished at the normal teletype rate and requires 100 milliseconds for completion. The flag in the teleprinter control is again set to a 1 when the last of the character code has been sent to the teleprinter/punch, indicating that the TTO is ready to receive a new character from the AC. The flag is connected to both the program interrupt synchronization element and the instruction skip element. Unless using the interrupt, the program must check the flag and, upon detecting the ready or set (binary 1) condition of the flag by means of the TSF microinstruction, the program must issue a TLS microinstruction which clears the flag and sends a new character from the AC to the TTO to be shifted out to the teleprinter/punch. The process of sending a character to the TTO from the AC is a great deal shorter than that of shifting the character out to the teleprinter/punch, therefore, the program must account for the time differential by waiting for flag to be set (1) before issuing a TLS.

Instructions for use in outputting data to the teletype are as follows:

Skip on Teleprinter Flag (TSF)

Octal Code: 6041
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The teleprinter flag is sensed, and if it contains a binary 1 the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Teleprinter Flag = 1, then PC + 1 = > PC

Clear Teleprinter Flag (TCF)

Octal Code: 6042
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The teleprinter flag is cleared to 0.
Symbol: 0 = > Teleprinter Flag

Load Teleprinter and Print (TPC)

Octal Code: 6044
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of bits 4-11 of the AC are sent to the TTO, then the hardware starts shifting the character out to the printer/punch unit. This microinstruction does not clear the teleprinter flag.

Symbol: C(AC 4-11) = > TTO causing:
C(TTO) = > printed and (if punch on) punched

Load Teleprinter Sequence (TLS)

Octal Code: 6046
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds

Operation: This microinstruction combines the functions of the TCF and the TPC. The teleprinter flag is cleared (set to 0) then the contents of bits 4-11 of the AC are sent to the TTO, where the hardware shifts the character out to the printer/punch unit. When the printer/punch has finished outputting the character and is ready for another character, the hardware has again raised the teleprinter flag (set it to a 1) to indicate this free condition. The whole operation, from the time at which the TLS has cleared the flag and sent out the character until the time at which the hardware finishes with the character and sets the flag to a 1 again, requires 100 milliseconds with the time required for the character to travel from the TTO to the paper being considerably greater than that required for it to be sent from the computer to the TTO.

Symbol: 0 = > Teleprinter flag
C(AC 4-11) = > TTO causing:
C(TTO) = > Printed and (if punch on) punched
1 = > Teleprinter flag when done

The following are examples of possible ways to use these instructions to output a character to the teletype. The last is recommended:

```
          .
          .
          .
          CLA
          TAD X          /PUT CHARACTER CODE INTO AC FROM
                         /LOCATION X
          TLS            /LOAD TTO FROM AC & PRINT/PUNCH
FREE,     TSF            /TEST FLAG TO SEE IF DONE PRINTING,
                         /SKIP IF = 1
          JMP FREE       /TEST FLAG AGAIN
          CLA            /CLEAR CHARACTER CODE FROM AC
          .
          .
          .              continue program
```

This sequence sends one character code to the TTO and waits for it to finish printing/punching before continuing program. It does not require that the flag to be set, in order to output the character. By making this sequence of instructions a subroutine of a larger program, it can be accessed (by a JMS) each time a character is to be output. Assume that the subroutine is entered with the character code in the AC:

```
TYPE,     0
          TLS            /LOAD TTO FROM AC AND PRINT/PUNCH
          TSF            /TEST FLAG, SKIP IF = 1
          JMP.-1         /JMP BACK & TEST FLAG AGAIN
          CLA            /CLEAR CHARACTER FROM AC
          JMP I TYPE     /EXIT TO MAIN PROGRAM
          .
          .
          .
```

By rearranging this subroutine the present time spent waiting for the character to be output and the flag to be set to 1 (100 milliseconds) can be used to continue the calculations, etc., of the main program thus making more efficient use of time.

```
TYPE,     0
          TSF            /TEST FLAG TO SEE IF PRINTER FREE,
                         /SKIP IF YES OR . . .
          JMP.-1         /WAIT TIL IT IS BY TESTING AGAIN AND
                         /AGAIN
          TLS            /OUTPUT CHARACTER
          CLA
          JMP I TYPE     /EXIT TO CONTINUE PROGRAM
          .
          .
          .
```

This subroutine tests the flag first and waits only if a previous character is still being output. It clears the AC and exits immediately after sending the character to the TTO and is continuing to run the user's program instead of waiting while the teletype (a much slower device) is off typing/punching the last character. The PDP-8/I clears all flags which are on the clear flag bus (this includes teletype flags) when key START is depressed. This means that the user program must account for setting the teleprinter flag initially and after each TCF (if any) or else the program will hang up in the wait loop of the print routine. The only way to set the flag to a 1 is through issuing a microinstruction which leaves the flag set when alone. This instruction should appear among the first few executed and must appear before any attempt to output a character.

The following example initializes the flag with a TLS as the first instruction of the program and makes optimum use of the time that would be spent waiting for the teletype to finish.

```
BEGIN,    TLS             /INITIALIZE TELEPRINTER FLAG
          .
          .
          .
TYPE,     0
          TSF             /SKIP IF FLAG = 1 or . . .
          JMP.-1          /WAIT UNTIL IT IS LOAD TTO &
          TLS             /TYPE CHARACTER
          CLA
          JMP I TYPE      /EXIT & CONTINUE PROGRAM WHILE
          .               /TELETYPE IS FINISHING CHARACTER
          .
          .
```

TELETYPE OPTION (TYPE PT08)

The Teletype facility of the basic computer can be expanded to accommodate several Model 33 or Model 35 Automatic Send Receive or Keyboard Send Receive units with the PT08 option. A PT08 option allows a Teletype to be interfaced to the PDP-8/I. Each Teletype line added contains logic elements that are functionally identical to those of the basic Teletype control. Therefore, instructions and programming for each PT08 are similar to those described previously for the basic Teletype unit. The following device select codes have been assigned for 5 PT08 options.

Line Unit	Select Codes
1	40 and 41
2	42 and 43
3	44 and 45
4	46 and 47
5	11 and 12

Instruction mnemonics for Teletype equipment in the PT08 system are not recognized by the program assembler (PAL III) and must be defined by the programmer. Mnemonic codes can be defined by the mnemonic code of the comparable basic Teletype microinstruction, suffixed with "PT" and the line number. For example, the following instructions can be defined for line 3:

Mnemonic	Octal	Operation
TSFPT3	6441	Skip if teleprinter 3 flag is a 1.
TCFPT3	6442	Clear teleprinter 3 flag.
TPCPT3	6444	Load teleprinter 3 buffer (TT03) from the content of AC4-11 and print and/or punch the character.
TLSPT3	6446	Load TT03 from the content of AC4-11, clear teleprinter 3 flag, and print and/or punch the character
KSFPT3	6451	Skip if keyboard 3 flag is a 1.
KCCPT3	6452	Clear AC and clear keyboard 3 flag.
KRSPT3	6454	Read keyboard 3 buffer (TTI3) static. The content of TTI3 is loaded into AC4-11 by an OR transfer.
KRBPT3	6456	Clear the AC, clear keyboard 3 flag, and read the content of TTI3 into AC4-11.

HIGH-SPEED PERFORATED TAPE READER AND CONTROL (TYPE PR8/I)

(The control circuitry for this device is located in the PDP-8/I central processor.)

This device senses 8-hole perforated paper or Mylar tape photoelectrically at 300 characters per second. The reader control requests reader movement, transfers data from the reader into the reader buffer (RB), and signals the computer when incoming data is present. Reader tape movement is started by a reader control request to simultaneously release the brake and engage the clutch. The 8-bit reader buffer sets the reader flag to 1 when it has been filled from the reader and transfers data into bits 4 through 11 of the accumulator under program control. The reader flag is connected to the computer program interrupt and instruction skip facilities, and is cleared by IOT pulses. Tape format is as described for the Teletype unit. Computer instructions for the reader are:

Skip on Reader Flag (RSF)

Octal Code: 6011
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The reader flag is sensed, and if it contains a binary 1 the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Reader Flag = 1, then PC + 1 = > PC

Read Reader Buffer (RRB)

Octal Code: 6012
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the reader buffer is transferred into bits 4 through 11 of the AC and the reader flag is cleared. This command does not clear the AC.
Symbol: RB V AC 4-11 = > AC 4-11
0 = > Reader Flag

Reader Fetch Character (RFC)

Octal Code: 6014
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The reader flag and the reader buffer are both cleared, one character is loaded into the reader buffer from tape, and the reader flag is set when this operation is completed.
Symbol: 0 = > Reader Flag, RB
Tape Data = > RB
1 = > Reader Flag when done

A program sequence loop to read a character from perforated tape can be written as follows:

```
        RFC         /FETCH CHARACTER FROM TAPE
LOOK,   RSF         /SKIP WHEN RB FULL
        JMP LOOK
        CLA
        RRB         /LOAD AC FROM RB
```

HIGH-SPEED TAPE PUNCH AND CONTROL (TYPE PP8/I)

(The control circuitry for this device is located in the PDP-8/I central processor.)

This option consists of a Royal McBee paper tape punch that perforates 8-hole tape at a rate of 50 characters per second. Information to be punched on a line of tape is loaded in an 8-bit punch buffer (PB) from AC bits 4 through 11. The punch flag becomes a 1 at the completion of punching action, signaling that new information may be transferred into the punch buffer, and punching initiated. The punch flag is as described for the Teletype unit. The punch instructions are:

Skip on Punch Flag (PSF)

Octal Code: 6021
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The punch flag is sensed, and if it contains a binary 1 the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Punch Flag = 1, then PC + 1 = > PC

Clear Punch Flag (PCF)

Octal Code: 6022
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Both the punch flag and the punch buffer are cleared in preparation for receiving a new character from the computer.
Symbol: 0 = > Punch Flag, PB

Load Punch Buffer and Punch Character (PPC)

Octal Code: 6024
Event Time. 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: An 8-bit character is transferred from bits 4 through 11 of the AC into the punch buffer and then this character is punched. This command does not clear the punch flag or the punch buffer.
Symbol: AC4-11 V PB = > PB

Load Punch Buffer Sequence (PLS)

Octal Code: 6026
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The punch flag and punch buffer are both cleared, the content of bits 4 through 11 of the AC is transferred into the punch buffer, the character in the PB is punched in tape, and the punch flag is set when the operation is completed.
Symbol: 0 = > Punch Flag, PB
AC4-11 = > PB
1 = > Punch Flag when done

A program sequence loop to punch a character when the punch buffer is "free" can be written as follows:

```
FREE,    PSF          /SKIP WHEN FREE
         JMP FREE
         PLS          /LOAD PB FROM AC AND PUNCH
                      /CHARACTER
```

DIGITAL-TO-ANALOG CONVERTER (TYPE AA01A)

The general-purpose Digital-to-Analog Converter Type AA01A converts 12-bit binary computer output numbers to analog voltages. The basic option consists of three channels, each containing a 12-bit digital buffer register and a digital-to-analog converter (DAC). Digital input to all three registers is provided, in common, by one 12-bit input channel which receives bussed output connections from the accumulator. Appropriate precision voltage reference supplies are provided for the converters.

One IOT microinstruction simultaneously selects a channel and transfers a digital number into the selected register. Each converter operates continuously on the content of the associated register to provide an analog output voltage.

Type AA01A options can be specified in a wide range of basic configurations; e.g., with from one to three channels, with or without output operational amplifiers, and with internally or externally supplied reference voltages. Configurations with double buffer registers in each channel are also available.

Each single-buffered channel of the equipment is operated by a single IOT command. Select codes of 55, 56, and 57 are assigned to the AA01A, making it possible to operate nine single-buffered channels or various configurations of double-buffered channels. A typical instruction for the AA01A is:

Load Digital-to-Analog Converter 1 (DAL1)

Octal Code: 6551
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the accumulator is loaded into the digital buffer register of channel 1.
Symbol: AC = > DAC1

The analog output voltage of a standard converter is from ground to −9.9976 volts (other voltages are available in equipment containing output operational amplifiers). All binary input numbers are assumed to be 12 bits in length with negative numbers represented in 2's complement notation. An input of 4000_8 yields an output of ground potential; an input of 0000_8 yields an output of −5 volts; and an input of 1777_8 yields an output of −10 volts minus the analog value of the least significant digital bit. Output accuracy is ±0.0125% of full scale and resolution is 0.025% of full scale value. Response time, measured directly at the converter output, is 3 microseconds for a full-scale step change to 1 least significant bit accuracy. Maximum buffer register loading rate is 2 megahertz.

DISPLAY EQUIPMENT

Cathode-ray tube display equipment available for use with the PDP-8/I includes the Oscilloscope Display Type VC8/I and the Precision Display Type 30N. The Light Pen Type 370 operates with either of these devices.

OSCILLOSCOPE DISPLAY CONTROL (TYPE VC8/I)

(The control circuitry for this device is located in the PDP-8/I central processor.)

The oscilloscope available for use with the Type VC8/I control is a Textronic Oscilloscope Model RM503 with 10 bits per axis.

Type VC8/I is a two axis digital-to-analog converter and an intensifying circuit, which provides the Deflection and Intensify signals needed to plot data on an oscilloscope. Coordinate data is loaded into an X buffer (XB) or a Y buffer (YB) from bits 2 through 11 of the accumulator. The binary data in these buffers is converted to a −10 to 0 volt Analog Deflection signal. The 30-volt Intensify signal is connected to the grid of the oscilloscope CRT. The duration of this signal, and hence the intensity of the point displayed, is determined by a 2-bit brightness register (BR). The content of the BR controls timing circuits that establish nominal durations of 1-, 2-, or 4-microsecond for the Intensify signal. The BR is loaded from a number contained in the appropriate IOT instruction. Application of power to the computer or pressing of the START key resets the BR to the maximum brightness. Points can be plotted at approximately a 30-kilohertz rate. The instructions for this display are:

Clear X Coordinate Buffer (DCX)

Octal Code: 6051
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The X coordinate buffer is cleared in preparation for receiving new X-axis display data.
Symbol: 0 = > XB

Clear and Load X Coordinate Buffer (DXL)

Octal Code: 6053
Event Time: 1, 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The X coordinate buffer is cleared, then loaded with new X-axis data from bits 2 through 11 of the AC.
Symbol: 0 = > XB
AC2-11 = > XB

Clear Y Coordinate Buffer (DCY)

Octal Code: 6061
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The Y coordinate buffer is cleared in preparation for receiving new Y-axis display data.
Symbol: 0 = > YB

Clear and Load Y Coordinate Buffer (DYL)

Octal Code: 6063
Event Time: 1, 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The Y cordinate buffer is cleared then loaded with new Y-axis data from bits 2 through 11 of the AC.
Symbol: 0 = > YB
AC 2-11 = > YB

Intensify (DIX)

Octal Code: 6054
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Intensify the point defined by the content of the X and Y coordinate buffers. This command can be combined with the DXL command.
Symbol: None

Intensify (DIY)

Octal Code: 6064
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Intensify the point defined by the content of the X and Y coordinate buffers. This command is identical to the DIX command except that it can be combined with the DYL command.
Symbol: None

X Coordinate Sequence (DXS)

Octal Code: 6057
Event Time: 1, 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: This command executes the combined functions performed by the DXL and DIX commands. The X coordinate buffer is cleared then loaded from the content of AC2 through AC11, then the point defined by the content of the X and Y buffers is intensified.
Symbol: 0 = > XB
AC 2-11 = > XB
then intensify

Y Coordinate Sequence (DYS)

Octal Code: 6067
Event Time: 1, 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: This command executes the combined functions performed by the DYL and DIY commands. The Y coordinate buffer is cleared, then loaded from the content of bits AC2 through 11, then the point defined by the content of the X and Y coordinate buffers is intensified.
Symbol: 0 = > YB
AC 2-11 = > YB
then intensify

Set Brightness Control (DSB)

Octal Code: 607X
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The brightness register (BR) is loaded from the content of bits 10 and 11 of the instruction. When the instruction is 6075 the minimum brightness (0.4 microsecond) is set, when 6076 the medium brightness (0.8 microsecond) is set, and when 6077 the maximum brightness (3.0 microseconds) is set. 6074 Instruction sets zero brightness.
Symbol: MB10-11 = > BR

The following program sequence to display a point assumes that the coordinate data is stored in known addresses X and Y.

```
X,
Y,
BEG,  CLA
      TAD X    /LOAD AC WITH X
      DXL      /CLEAR AND LOAD XB
      CLA
      TAD Y    /LOAD AC WITH Y
      DYS      /CLEAR AND LOAD YB, DISPLAY POINT
```

Precision CRT Display Type 30N

Type 30N functions are similar to those of the Type VC8/I Oscilloscope Display in plotting points on a self-contained 16-inch cathode ray tube. A 3-bit brightness register is contained in Type 30N to control the duration of the Intensify signal supplied to the CRT. The content of this register specifies the brightness of the point being displayed according to the following scale:

BR Content	Intensity
3	brightest
2	
1	
0	average
7	
6	
5	
4	dimmest

The BR register is loaded by jam transfer (transfer ones and zeros so that clearing is not required) from the AC by the instruction:

Load Brightness Register (DLB)
Octal Code: 6074
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The brightness register (BR) is loaded by a jam transfer of information contained in bits 9 through 11 of the AC.
Symbol: AC 9-11 = > BR

All other instructions and the instruction sequence are similar to those used in the Type VC8/I.

Light Pen Type 370

The light pen is a photosensitive device which detects the presence of information displayed on a CRT. If the light pen is held against the face of the CRT at a point displayed, the display flag will be set to a 1. The light pen display flag is connected into the computer instruction skip facility. The commands are:

Skip on Display Flag (DSF)
Octal Code: 6071
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the display flag is sensed, and if it contains a 1 the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Display Flag = 1, then PC + 1 = > PC

Clear the Display Flag (DCF)
Octal Code: 6072
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The display flag is cleared in preparation for sensing another point on the CRT.
Symbol: 0 = > Display Flag

INCREMENTAL PLOTTER AND CONTROL (TYPE VP8/I)

(The control circuitry for this device is located in the PDP-8/I central processor.)

Four models of California Computer Products Digital Incremental Recorder can be operated from a Digital Type VP8/I Incremental Plotter Control. Characteristics of the four recorders are:

CCP Model	Step Size (inches)	Speed (steps/minute)	Paper Width (inches)
563	0.01 or 0.005	12,000	31
565	0.01 or 0.005	18,000	12

The principles of operation are the same for each of the four models of Digital Incremental Recorders. Bidirectional rotary step motors are employed for both the X and Y axes. Recording is produced by movement of a pen relative to the surface of the graph paper, with each instruction causing an incremental step. X-axis deflection is produced by motion of the drum; Y-axis deflection, by motion of the pen carriage. Instructions are used to raise and lower the pen from the surface of the paper. Each incremental step can be in any one of eight directions through appropriate combinations of the X and Y axis instructions. All recording (discrete points, continuous curves, or symbols) is accomplished by the incremental stepping action of the paper drum and pen carriage. Front panel controls permit single-step or continuous-step manual operation of the drum and carriage, and manual control of the pen solenoid. The recorder and control are connected to the computer program interrupt and instruction skip facility.

Instructions for the recorder and control are:

Skip on Plotter Flag (PLSF)

Octal Code: 6501
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter flag is sensed, and if it contains a 1 the content of the PC is incremented by one so the next sequential instruction is skipped.
Symbol: If Plotter Flag = 1, then PC + 1 = > PC

Clear Plotter Flag (PLCF)

Octal Code: 6502
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter flag is cleared in preparation for issuing a plotter operation command.
Symbol: 0 = > Plotter Flag

Pen Up (PLPU)

Octal Code: 6504
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter pen is raised from the surface of the paper.
Symbol: None

Pen Right (PLPR)
Octal Code: 6511
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter pen is moved to the right in either the raised or lowered position.
Symbol: None

Drum Up (PLDU)
Octal Code: 6512
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter paper drum is moved upward. This command can be combined with the PLPR and PLDD commands.
Symbol: None

Drum Down (PLDD)
Octal Code: 6514
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter paper drum is moved downward.
Symbol: None

Pen Left (PLPL)
Octal Code: 6521
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter pen is moved to the left in either the raised or lowered position.
Symbol: None

Drum Up (PLUD)
Octal Code: 6522
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter paper drum is moved upward. This command is similar to command 6512 except that it can be combined with the PLPL or PLPD commands.
Symbol: None

Pen Down (PLPD)
Octal Code: 6524
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The plotter pen is lowered to the surface of the paper.
Symbol: None

Program sequence must assume that the pen location is known at the start of a routine since there is no means of specifying an absolute pen location in an incremental plotter. Pen location can be preset by the manual controls on the recorder. During a subroutine, the PDP-8/I can track the location of the

pen on the paper by counting the instructions that increment position of the pen and the drum.

CARD READER AND CONTROL TYPE (CR8/I)

(The control circuitry for this device is located in the PDP-8/I central processor.)

The Card Reader and Control Type CR8/I reads standard 12-row, 80-column punched cards at a maximum rate of 100 cards per minute. Cards are read by column, beginning with column 1. One select instruction starts the card moving past the read station. Once a card is in motion, all 80 columns are read. Data in a card column is sensed by mechanical star wheels which close an electrical contact when a hole (binary 1) is detected. Column information is read in one of two program selected modes: alphanumeric and binary. In the alphanumeric mode the 12 information bits in one column are automatically decoded and transferred into the least significant half of the accumulator as a 6-bit Hollerith code. Appendix 3 lists the Hollerith card codes. In the binary mode the 12 bits of a column are transferred directly into the accumulator so that the top row (12) is transferred into AC0 and the bottom row (9) is transferred into AC11. A punched hole is interpreted as a binary 1 and no hole is interpreted as a binary 0.

Three program flags indicate card reader conditions to the computer. The data ready flag rises and requests a program interrupt when a column of information is ready to be transferred into the AC. A read alphanumeric or read binary command must be issued within 1.5 milliseconds after the data ready flag rises to prevent data loss. The card done flag rises and requests a program interrupt when the card leaves the read station. A new select command must be issued within 25 milliseconds after the card done flag rises to keep the reader operating at maximum speed. Sensing of this flag can eliminate the need for counting columns, or combined with column counting can provide a check for data loss. The reader-not-ready flag can be sensed by a skip command to provide indication of card reader power off, no card in the read station, or that a reader failure has been detected. When this flag is raised the reader cannot be selected and select commands are ignored. The reader-not-ready flag is not connected to the program interrupt facility and cannot be cleared under program control. Manual intervention is required to clear the reader-not-ready flag. Instructions for the CR8/I are:

Skip on Data Ready (RCSF)

Octal Code: 6631
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the data ready flag is sensed, and if it contains a 1 (indicating that information for one card column is ready to be read) the content of the PC is incremented by one so the next sequential instruction is skipped.
Symbol: If Data Ready Flag = 1, then PC + 1 => PC

Read Alphanumeric (RCRA)

Octal Code: 6632
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The 6-bit Hollerith code for the 12 bits of a card column are transferred into bits 6 through 11 of the AC, and the data ready flag is cleared.
Symbol: AC6-11 V Hollerith Code $=>$ AC6-11
0 $=>$ Data Ready Flag

Read Binary (RCRB)

Octal Code: 6634
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The 12-bit binary code for a card column is transferred directly into the AC, and the data ready flag is cleared. Information from the card column is transferred into the AC so that card row 12 enters AC0, row 11 enters AC1, row 0 enters AC2, . . . and row 9 enters AC 11.
Symbol: AC V Binary Code $=>$ AC
0 $=>$ Data Ready Flag

Skip on Card Done Flag (RCSP)

Octal Code: 6671
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the card done flag is sensed, and if it contains a 1 (indicating that the card has passed the read station) the content of the PC is incremented to skip the next sequential instruction.
Symbol: If Card Done Flag $= 1$, then PC $+ 1 =>$ PC

Select Card Reader and Skip If Ready (RCSE)

Octal Code: 6672
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the reader-not-ready flag is sensed and if it contains a 1 (indicating that the card reader is ready for programmed operation) the PC is incremented to skip the next sequential instruction; a card is started towards the read station from the feed hopper; and the card done flag is cleared. If the reader-not-ready flag contains a 0 (indicating power is off or no card is in the read station) card selection (motion) does not occur and the skip does not occur.
Symbol: If Reader-Not-Ready Flag $= 1$, then PC $+ 1 =>$ PC
0 $=>$ Card Done Flag

Clear Card Done Flag (RCRD)

Octal Code: 6674
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The card done flag is cleared. This command allows a program to stop reading at any point in a card deck.
Symbol: 0 $=>$ Card Done Flag

A logical instruction sequence to read cards is:

```
START,    RCSE              /START CARD MOTION AND SKIP IF READY
          JMP NOT RDY       /JUMP TO SUBROUTINE THAT TYPES OUT
                            /"CARD READER MANUAL INTERVENTION
                            /REQUIRED" OR HALTS
NEXT,     RCSF              /DATA READY?
          JMP .-1           /NO, KEEP WAITING
          RCRA or RCRB      /YES, READ ONE CHARACTER OR ONE
                            /COLUMN
          DCA I STR         /STORE DATA
          RCSD              /END OF CARD?
          JMP NEXT          /NO, READ NEXT COLUMN
          JMP OUT           /YES, JUMP TO SUBROUTINE THAT CHECKS
                            /CARD COUNT OR REPEATS AT START FOR
                            /NEXT CARD
```

No validity or registration checking is performed by the CR8/I. A programmed validity check can be made by reading each card column in both the alphanumeric and the binary mode (within the 1.5 millisecond time limitation), then performing a comparison check.

Before commencing a card reading program energize the reader, load the feed hopper with cards, and manually feed the first card to the read station. The function of the manual controls and indicators are as follows (as they appear from right to left on the card reader):

Control or Indicator	Function
ON/OFF switch	Controls the application of primary power to the reader. When power is applied, the reader is ready to respond to operation of the other keys or programmed commands.
AUTO/MAN switch	Controls card reading. In the manual position this switch disables the card feed mechanism so that cards must be manually placed on the read table and registered by pressing the REG key. In the automatic position card motion from the feed hopper through the read station is under program control.
REG switch	When the AUTO/MAN switch is in the AUTO position the REG key is used to feed the first card to the read station. When the AUTO/MAN switch is in the MAN position the REG key is used to feed a card manually placed on the read table.
SKIP switch	This key is not connected on the CR8/I and has no effect on equipment operation.
CHECK READER indicator	This lamp is not connected on the CR8/I.

READY indicator	Lights when the reader is energized and cards are present in the feed hopper. The plastic card cover should always be used on top of a deck of cards to assure that the ready switch and indicator are activated.
CARD RELEASE pushbutton	When pressed, this pushbutton (adjacent to the read station) releases a card already in the read station.

AUTOMATIC LINE PRINTER AND CONTROL (TYPE 645)

The line printer can print 300 lines of 120 characters per minute. Each character is selected from a set of 64 available, by a 6-bit binary code (Appendix 3 lists the ASCII character specified for each code). Each 6-bit code is loaded separately into a core storage printing buffer (LPB) from bits 6 through 11 of the AC. The LPB is divided into two 120-character sections. To load one section of the LPB requires 120 load instructions. A print command causes the characters specified by the last-loaded section of the LPB to be printed on one line. As printing of one section of the LPB is in progress, the other section can be reloaded. After the last character in a line is printed, the section of the LPB from which characters were just printed is cleared automatically. The section of the LPB that is loaded and printed is alternated automatically within the printer and is not program specified.

The line printer can load characters into the LPB at a 10-microsecond rate, clears one section of the LPB in 3 to 6 milliseconds, and moves paper at the rate of one line every 18 milliseconds. When transfer of one code into the LPB is completed, the line printer done flag rises to indicate that the printer is ready to receive another code. When printing of the last character of a section of the LPB is completed, the line printer done flag rises and causes a program interrupt to request reloading of that section of the LPB. A line printer error flag rises and causes a program interrupt if the line printer detects an inoperative condition (printer power off, control circuits not reset, paper supply low, etc.).

A 3-bit format register (FR) in the printer is loaded from bits 9 through 11 of the AC during a print command. This register selects one of eight channels of a perforated tape in the printer to control spacing of the paper. The tape moves in synchronism with the paper until a hole is sensed in the selected channel to halt paper advance. A recommended tape has the following characteristics:

FR Code (Octal)	Paper Spacing	Tape Track
0	1 line	2
1	2 lines	3
2	3 lines	4
3	6 lines (1/4 page)	5
4	11 lines (1/2 page)	6
5	22 lines (3/4 page)	7
6	33 lines (line feed)	8
7	top of form	1

The IOT instructions which command the line printer are:

Skip on Line Printer Error (LSE)

Octal Code: 6651
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of line printer error flag is sensed, and if it contains a binary 1, indicating that an error has been detected, the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Line Printer Error Flag = 1, then PC + 1 = > PC

Clear Printer Buffer (LCB)

Octal Code: 6652
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Both sections of the line printer buffer are cleared in preparation for receiving new character information.
Symbol: 0 = > LPB

Load Printer Buffer (LLB)

Octal Code: 6654
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: A section of the printer buffer is loaded from the content of bits 6 through 11 of the AC, then the AC is cleared.
Symbol: AC6 — 11 = > LPB, then 0 = > AC

Skip on Line Printer Done Flag (LSD)

Octal Code: 6661
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the line printer done flag is sensed and if it contains a binary 1 the content of the PC is incremented by one so that the next sequential instruction is skipped.
Symbol: If Line Printer Done Flag = 1, then PC + 1 = > PC

Clear Line Printer Flags (LCF)

Octal Code: 6662
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The line printer done and error flags are cleared
Symbol: 0 = > Line Printer Done Flag
0 = > Line Printer Error Flag

Clear Format Register (LPR)

Octal Code: 6664
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The line printer format register (FR) is cleared then loaded from the content of bits 9 through 11 of the AC, and the AC is cleared. The line contained in the section of the printer buffer (LPB) loaded last is printed. Paper is advanced in accordance with the selected channel of the format tape if the content of the AC8 is a 1. If AC8 is a 0 paper advance is inhibited.

Symbol:

0 = > FR
AC9 − 11 = > FR
0 = > AC
The content of half of the LPB is printed
If AC8 = 1, then advance paper according to format tape channel FR

The following routine demonstrates the use of these commands in a sequence which prints an unspecified number of 120-character lines. This sequence assumes that the printer is not in operation, that the paper is manually positioned for the first line of print, and that one-character words are stored in sequential core memory locations beginning at 2000. The PRINT location starts the routine.

```
PRINT,  LCB          /INITIALIZE PRINTER BUFFER
        CLA
        TAD LOC      /LOAD INITIAL CHARACTER ADDRESS
        DCA 10       /STORE IN AUTO-INDEX REGISTER
LRPT,   TAD CNT      /INITIALIZE CHARACTER COUNTER
        DCA TEMP
LOOP,   LSD          /WAIT UNTIL PRINTING BUFFER READY
        JMP LOOP
        LCF          /CLEAR LINE PRINTER FLAG
        TAD I 10     /LOAD AC FROM CURRENT CHARACTER
                     /ADDRESS
        LLB          /LOAD PRINTING BUFFER
        ISZ TEMP     /TEST FOR 120 CHARACTERS LOADED
        JMP LOOP

        TAD FRM      /LOAD SPACING CONTROL AND
        LPR          /PRINT A LINE
        JMP LRPT     /JUMP TO PRINT ANOTHER LINE
LOC,    1777         /INITIAL CHARACTER ADDRESS −1
CNT,    −170         /CHARACTER COUNTER − 120 DECIMAL
TEMP,   0            /CURRENT CHARACTER ADDRESS
FRM,    10           /SPACING CONTROL AND FORMAT
```

SERIAL MAGNETIC DRUM SYSTEM (TYPE 251)

The Type 251 Serial Magnetic Drum System is a standard option that serves as an auxiliary data storage device. Information in the PDP-8/I can be stored (written) in the drum system and retrieved (read) in sectors of 128 computer words. After program initialization, sectors are transferred automatically between the computer core memory and the drum system, transfer of each word being interleaved with the running computer program under control of the computer data break facility. A word is transferred in parallel (12 bits at a time) and is read or written around the surface of the drum serially (one bit at a time). Within the drum system words consist of 12 information bits and a parity bit. Parity bits are generated internally during writing, and are read and checked during reading. Each word is transferred in about 66 microseconds; a sector transfer is completed in 8.2 milliseconds. Average access time is 8.65 milliseconds (17.3 milliseconds maximum). Track and sector format on the drum surface is such that all transfers require the same amount of time, so track and sector are specified together as an 11-bit address for 128 words.

Drum systems are available with 8, 16, 32, 64, 128, 192, or 256 tracks; each track holds 8 sectors of 128 13-bit words. The various drum system capacities are designated by a letter suffix to the system type number as follows: Type 251A, 8K words; 251B, 16K words; 251C, 32K words; 251D, 65K words; 251E, 131K words; 251F, 196K words; and 251G, 262K words.

Indicator lamps on a front panel usually display the content of the four major registers and the status of control flip-flops. The major registers are:

Drum Core Location Counter (DCL): A 15-bit register which addresses the next core memory location to or from which a word is to be transferred. As a word is transferred, DCL is incremented by one.

Drum Address Register (DAR): An 11-bit register which addresses the drum track and sector which is currently transferring data. The eight most significant bits of the DAR specify the track and the least significant three bits specify a sector on that track. At the completion of a successful sector transfer (error flag is 0) DAR is incremented by one.

Drum Final Buffer (DFB): A 12-bit register under control of the data break facility which is a buffer between the memory buffer register and the drum serial buffer. During writing, the DFB holds the next word to be written. During reading, the DFB stores the word just read from the drum until it is transferred to the PDP-8/I.

Drum Serial Buffer (DSB): A 14-bit register which contains a data word and two control bits. It is a serial-to-parallel converter during drum reading, and a parallel-to-serial converter during drum writing. Information is read from the drum into DSB serially and transferred to DFB in parallel. During drum writing, a word is transferred in parallel from DFB into DSB and written serially around the drum.

Instructions

The commands for the drum system are as follows:

Load Drum Core Location Counter and Read (DRCR)

Octal Code: 6603
Event Time: 1, 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The core memory location information in the AC is transferred into the DCL and the drum is prepared to read one sector of information for transfer to the specified core memory location.*
Symbol: AC => DCL
1 => Read Control

*The sector, track, and core memory address are suitably incremented and allow transfer of the next sequential sector without respecifying addresses. The DRCN instruction must be given within 50 microseconds after the completion flag is set to 1 during the previous sector.

Load Drum Core Location Counter and Write (DRCW)

Octal Code: 6605
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The core memory location information in the AC is transferred into the DCL and the drum is prepared to write on one sector the information beginning at the specified core memory address.*
Symbol: AC = > DCL
1 = > Write Control

Clear Drum Flags (DRCF)

Octal Code: 6611
Event Time: 1
Indicators: IOT, FETCH, PAUSE on computer and COMPLETION FLAG and ERROR FLAG on the drum system become dark.
Execution Time: 4.25 microseconds
Operation: Both completion flag and error flag are cleared
Symbol: 0 = > Completion Flag
9 = > Error Flag

Load Parity and Data Error (DREF)

Octal Code: 6612
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of both the parity error and data timing error flip-flops of the drum control is transferred into bits AC0 and AC1, respectively. The command allows the program to evaluate the cause of an error flag setting.
Symbol: Parity Error = > AC0
Data Timing Error = > AC1

Load the Track and Sector (DRTS)

Octal Code: 6615
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Track and sector information in bits 1-11 of the AC is transferred into the DAR, the completion and error flags are cleared, and a transfer (reading or writing) is begun.
Symbol: AC1-11 = > DAR
0 = > Completion Flag
0 = > Error Flag

Skip on Drum Error (DRSE)

Octal Code: 6621
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The error flag is sampled and if it contains a 0 (indicating no error has been detected) the PC is incremented to skip the next instruction.
Symbol: If Error Flag = 0, then PC + 1 = > PC

Skip on Drum Completion (DRSC)

Octal Code: 6622
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The completion flag is sampled and if it contains a 1 (indicating a sector transfer is complete) the PC is incremented to skip the next instruction.
Symbol: If completion Flag = 1, then PC+ 1 = > PC

Initiate Next Transfer (DRCN)

Octal Code: 6624
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Both the error and completion flags are cleared, then the transfer of the next sector is initiated.
Symbol: 0 = > Error Flag
0 = > Completion Flag
then start transfer

Programming

Two instructions cause the transfer of a 128-word sector. The first (DRCR or DRCW) specifies the initial core memory location of the transfer and the direction of the transfer (drum-to-core or core-to-drum). The second instruction (DRTS) specifies the track and sector address and initiates the transfer. Transfer of each word is under control of the computer data break facility and completion of a sector transfer is indicated by a completion flag that causes a program interrupt.

The eight most significant bits of a drum address select one of 256 tracks; the three least significant bits select one of eight sectors on the track. A 300-microsecond gap identifies the beginning of a track (sector 0). Even numbered sectors (0, 2, 4, and 6) are recorded consecutively following the 300-microsecond gap. A 50-microsecond gap identifies the beginning of the odd number sectors (sector 1). Odd numbered sectors (1, 3, 5, and 7) are recorded consecutively following the 50-microsecond gap. This format allows the transfer of two consecutively numbered sectors in one drum revolution, provided the continuation instruction (DRCN) is issued in the first 50 microseconds after the drum flag rises to indicate completion of a sector transfer. The program interrupt subroutine can easily determine that the drum system caused an interrupt, check the number of sectors transferred, check for drum errors by sampling the error flag, and issue the continuation instruction in 50 microseconds.

Because the selection of a track read-write head requires 200 microseconds stabilization time, a new track must be specified during the first 200 microseconds of the 300-microsecond gap for continuous transferring. If selected tracks and sectors are consecutive, uninterrupted transferring may be programmed merely by specifying continuation, since the drum system address and the core memory address are automatically incremented. However, if a data timing or parity error occurs, the track and sector number is not advanced and operations stop at the conclusion of a sector transfer. This feature allows the program to sense for error conditions and to locate the track and sector at which transmission fails.

The drum completion flag is set to 1 upon completion of a sector transfer, causing a program interrupt. The flag is cleared either by a clear flag instruction (DRCF) or automatically when one of two transfer instructions (DRTS, DRCN) is given.

The error flag, which should be checked at the completion of each transfer, indicates either of the following conditions:

(1) That a parity error has been detected after reading from drum to core.

(2) That the Break Request signal from the drum system was not answered within the required 66-microsecond period. This condition occurs either because other devices with higher priority are being serviced by the data break facility, or because an instruction requiring longer than 66 microseconds for completion is in progress when the break request is made.

    In reading from the drum, a data word is incorrect in core memory. In writing on the drum, the next word has not been received from the computer.

The following program examples indicate the operation of the drum system in single and multiple sector transfers.

SUBROUTINE TO TRANSFER (READ) ONE SECTOR

```
        CLA             /CALLING SEQUENCE
        TAD ADDR        /INITIAL CORE MEMORY ADDRESS
        JMS READ
        0               /TRACK AND SECTOR ADDRESS
        0               /RETURN
READ,   0
        DRCR            /DRCW TO WRITE
        TAD I READ      /LOAD AC WITH TRACK AND SECTOR ADDRESS
        DRTS
        DRSC            /DONE?
        JMP .-1         /NO
        DRSE            /ERRORS?
        JMP ERR         /JUMP TO ERROR CHECK ROUTINE
        ISZ READ
        JMP I READ      /RETURN
```

SUBROUTINE TO TRANSFER SUCCESSIVE (TWO) SECTORS

```
        CLA             /CALLING SEQUENCE
        TAD ADDR        /INITIAL CORE MEMORY ADDRESS
        JMS READ
        0               /TRACK AND SECTOR ADDRESS
        0               /RETURN
```

```
READ,   0
        DRCR            /DRCW TO WRITE
        TAD I READ      /LOAD AC WITH TRACK AND SECTOR ADDRESS
        DRTS
        DRSC            /DONE?
        JMP .-1         /NO
        DRSE            /ERRORS?
        JMP ERR         /JUMP TO ERROR CHECK ROUTINE
        DRCN            /CLEAR FLAGS, CONTINUE TRANSFER
                        /OF NEXT SECTOR
        DRSC
        JMP .-1
        DRSE
        JMP ERR
        ISZ READ
        JMP I READ      /RETURN
```

SERIAL MAGNETIC DRUM SYSTEM (TYPE RM08)

The Type RM08 Serial Magnetic Drum System is an option for Programmed Data Processor — 8/I (PDP-8/I) that serves as an auxiliary data storage device. Information in the PDP-8/I can be stored (written) in the Type RM08 and retrieved (read) in sectors of 16 computer words. After program initialization, sectors are transferred between the computer core memory and the drum automatically, transfer of each word being interleaved with the running computer program under control of the computer data break facilities. A word is transferred in parallel (12 bits at a time) and is read or written around the surface of the rotating drum serially (one bit at a time). Within the drum system words consist of 12 information bits and a parity bit. Parity bits are generated internally when writing and checked when reading. Each word is transferred in about 15.6 μs (3600 rpm) 19.5 μs (3000 rpm); a sector transfer is completed in 250 μs (3600 rpm); 313 μs (3000 rpm). Average access time is 10.3 milliseconds (20.5 milliseconds maximum) (3000 rpm) and 8.65 μs (17.3 ms. maximum for 3600 rpm). The Drum system has 64 tracks; each track holds 64 sectors of 16, 13-bit words. The drum is expandable to 256 tracks (262K words).

Drum Core Location Counter (DCL)

A 15-bit register which addresses the next core location or from which a word is to be transferred. As a word is transferred, DCL is incremented by one.

Drum Address Register (DAR)

A 14-bit register (8 track, 6 sector) which addresses the drum track and sector which is currently transferring data. At the completion of a successful last sector transfer (error flag is 0) DAR is incremented by one.

Drum Final Buffer (DFB)

A 12-bit register under control of the data break facility which is a buffer between the memory buffer register and the drum serial buffer. During writing, the DFB holds the next word to be written. During reading the DFB stores the word just read from the drum until it is transferred to the PDP-8.

Drum Serial Buffer (DSB)

A 14-bit register which contains a data word and two control bits. It is a serial-to-parallel converter during drum reading, and a parallel-to-serial converter during drum writing. Information is read from the drum into DSB serially and transferred to DFB in parallel. During drum writing, a word is transferred in parallel from DFB to DSB and written serially around the drum.

The Drum Field and Sector Number

An 8-bit Register that holds the drum field and the number of sectors to be transferred. The contents of this register can only be changed by an IOT. The Drum Field, a 2-bit Register specifies in which one of four Drum Fields (64K works per field) will the data be transferred. The selector number, a 6-bit register, contains the number of sectors to be transferred from 1 to 100_8.

The Sector Number Counter (7 bits)

Contains the two's complement of the Sector Number Register. It counts the number of sectors transferred and its overflow produces a flag interrupt condition.

Instructions

The commands for the drum system are as follows:

Load Drum Core Location Counter and Read (DRCR)

Octal Code: 6603
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The core memory location information in the AC is transferred into the DCL and the drum is prepared to read one sector of information for transfer to the specified core memory location.* Clears AC on completion.
Symbol: AC => DCL
1 => Read Control

Load Drum Core Location Counter and Write (DRCW)

Octal Code: 6605
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The core memory location information in the AC is transferred into the DCL and the drum is prepared to write on one sector the information beginning at the specified core memory address.* Clears the AC.
Symbol: AC => DCL
1 => Write Control

---

*The sector, track, and core memory address are suitably incremented and allow transfer of the next sequential sector without respecifying addresses up to 64 sectors.

Clear Drum Flags (DRCF)

Octal Code: 6611
Event Time: 1
Indicators: IOT, FETCH, PAUSE on computer and COMPLETION FLAG and ERROR FLAG on the drum system become dark.
Execution Time: 4.25 microseconds
Operation: Both completion flag and error flag are cleared
Symbol: 0 = > Completion Flag
9 = > Error Flag

Load Parity and Data Error and Sector Counter (DRES)

Octal Code: 6612
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of both parity error and data timing error flip-flops of the drum control are transferred into bits AC0 and AC1, respectively. The contents of the drum sector counter are transferred into bits AC6 → AC11. The command allows the program to evaluate the cause of an error flag setting and evaluate the drum address. The instruction first clears the AC, then loads.
Symbol: Parity Error = > AC0
Data Timing Error = > AC1
DSC = > AC_{6-11}

Octal Code: 6615
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Track and sector information in bits 1-11 of the AC is transferred into the DAR, the completion and error flags are cleared, and a transfer (reading or writing) is begun.
Symbol: AC1-11 = > DAR
0 = > Completion Flag
0 = > Error Flag

Skip on Drum Error (DRSE)

Octal Code: 6621
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The error flag is sampled and if it contains a 0 (indicating no error has been detected) the PC is incremented to skip the next instruction.
Symbol: If Error Flag = 0, then PC + 1 = > PC

Skip on Drum Completion (DRSC)

Octal Code: 6622
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The completion flag is sampled and if it contains a 1 (indicating a sector transfer is complete) the PC is incremented to skip the next instruction.
Symbol: If completion Flag = 1, then PC + 1 = > PC

Load Drum Field and Sector Registers (DRFS)

Octal Code: 6624
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Loads the Drum Field Register with the contents of the accumulator Bits 10 and 11 (AC_{10}, AC_{11} = > Field 0). Loads the Sector Number Register with the contents of the accumulator Bits 0-5, to specify the number of sectors to be transferred (1 to 100_8).* Loads the three most significant bits of the Drum Core Location Register (DCL_{0-2}) with the contents of the AC bits 6, 7, 8, to specify the core memory block to be used during the drum transfer. If the DRFS is not given before each DRTS instruction, the drum will transfer the field and sector number left in the Drum Field and Sector Number Register.

If memory blocks are changed (4K), then 6624 must be given before 6605 in order to break to the correct extended memory block.

Symbol: AC0-5 = > SNR
AC6-8 = > DCL_{0-2}
AC_{10-11} = > DFR

---

*100_8 = 00

PROGRAMMING

Three instructions cause the transfer of from 16 to 1024 words. The first IOT specifies the core memory bank, the drum memory field and the number of sectors to transfer. This instruction needs to be given only once, if the core memory bank (4K), drum field (64K) and number of sectors to be transferred remain the same. The second specifies the core memory location of the transfer and the direction of transfer (drum-to-core or core-to-drum). The third instruction specifies the initial sector and track number and initiates the transfer. Transfer of each word is under contol of the computer data break facility and computation continues during a sector transfer.

A transfer begins when the continuously rotating drum reaches the selected sector address, 1.2 microseconds before the beginning of the data in a selected track and sector. A 300-microsecond interval separates the end of the last sector from the beginning of the first on one track. The selection of the track read-write head requires 200 microseconds of stabilization time. The largest single data transfer is 1024 words. Since the drum automatically increments the track and sector counter during the gap time, a 1024 word transfer can begin on one track and end up on the next. The drum completion flag is set to 1 upon completion of a data transfer, causing a program interrupt. The flag is cleared either by a clear flag instruction (DRFC) or automatically when the transfer instruction (DRTS) is given.

The error flag, which should be checked at the completion of each transfer, indicates either of the following conditions:

1. That a parity error has been detected after reading from drum to core.

2. That the break request signal from the drum was not answered within the 15.6 microsecond period. This condition occurs either because other devices with higher priority are connected to the data break facility, or because an instruction requiring longer than 15.6 microseconds for completion

was in progress when the break request was made. In reading from the drum, a data word is therefore incorrect in core memory. In writing on the drum, the next word has not been received from the computer.

RANDOM ACCESS DISC FILE (TYPE DF32)

The Type DF32 Disc File is a fast, low-cost, random-access, bulk-storage device and control for the PDP-8/I. Operating through the 3-cycle data-break channel, the DF32 provides 32,768 13-bit words (12 bits plus parity) of storage, and is economically expandable to 131,072 using Expander Disc Type DS32.

Transfer rate of the DF32 is 66 μsec per word; average access time is 16.67 msec for 60-cycle power (20 msec with 50-cycle power).

Two basic assemblies comprise the DF32: the storage unit with read/write electronics, and computer interface logic. The storage unit contains a nickel-cobalt plated disc driven by a hysteresis synchronous motor. Data is recorded on a single disc surface by 16 read/write heads which are in a fixed position. A photo-reflective marker is used on the disc's outer perimeter to denote beginning and end of timing and address tracks.

Disc motor and shaft, read/write data heads, timing and address heads, and photocell assembly are mounted on a rack assembly which permits sliding the unit in and out of a standard Digital Equipment Corporation cabinet.

The disc is designed for rack mounting in a 19 inch relay rack.

INSTRUCTIONS

The commands for the disc system are as follows:

Clear Disc Memory Address Register (DCMA)

Octal Code: 6601
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Clears Memory Address Register, parity error and completion flags. This instruction clears the disc memory request flag and interrupt flags.
Symbol: 0 = > completion flag
0 = > error flag

Load Disc Memory Address Register and Read (DMAR)

Octal Code: 6603
Event Time: 1, 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of the AC are loaded into the disc memory address register and the AC is cleared. Begin to read information from the disc into the specified core location. Clears parity error and completion flags. Clears interrupt flags.
Symbol: $AC_{0-11} \rightarrow DMA_{0-11}$
0 = > completion flag
0 = > error flag

Load Disc Memory Address Register and Write (DMAW)

Octal Code: 6605
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of the AC are loaded into the disc memory address register and the AC is cleared. Begin to write information into the disc from the specified core location. Clears parity error and completion flags. Clears interrupt flags. Data break must be allowed to occur within 66 μsec after issuing this instruction.
Symbol: $AC_{0-11} \rightarrow DMA_{0-11}$
$0 = >$ completion flag
$0 = >$ error flag

Clear Disc Extended Address Register (DCEA)

Octal Code: 6611
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Clears the Disc Extended Address and memory address extension register.
Symbol: $0 = >$ Disc Extended Address Register
$0 = >$ Memory Address Extension Register

Skip on Address Confirmed Flag (DSAC)

Octal Code: 6612
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Skips next instruction if address confirmed Flag is a 1. Flag is set for 16 μsec (AC is cleared)
Symbol: If address confirmed flag = 1, then
PC + 1 = > PC

Load Disc Extended Address (DEAL)

Octal Code: 6615
Event Time: 1, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The Disc extended address and memory address extension registers are cleared and loaded with the track address data held in the AC.
Symbol: $AC_{6-8} = > EA_{3-1}$ Core Memory Extension
$AC_{1-5} = > EMA_{5-1}$ Disc Address Extension 32K, 64K, 96K, 128K
$AC_{0,\ 9-11}$ used in DEAC instruction

Note: Write lock switch status is true only when disc module contains write command. The non-existent disc condition will appear following the completion of a data transfer during read, where the address acknowledged was the last address of a disc and the next word to be addressed falls within a non-existent disc. The completion flag for this data transfer is set by the non-existent disc condition 16 microseconds following the data transfer.

Read Disc Extended Address Register (DEAC)

Octal Code: 6616
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Clear the AC then loads the contents of the disc extended address register into the AC to allow program evaluation. Skip next instruction if address confirmed flag is a 1.
Symbol: 32K, 64K, 120K: $EMA_{5-11} => AC_{1-5}$
Computer Memory $EA_{3-1} => AC_{6-8}$
Photo-cell sync mark $=> AC_0$ (Available 200 μsec)
Data Request Late flag $=> AC_9$
Non-existent or write Lock switch on $=> AC_{10}$
Parity Errors $=> AC_{11}$

Skip Old Zero Error Flag (DFSE)

Octal Code: 6621
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Skips next instruction if partial error, data request late, or write lock switch flag is a zero. Indicates no errors.
Symbol: If parity error flag = 1, then PC + 1 => PC
If Data Request late flag = 1, then PC + 1 => PC
If Write lock switch Flag = 1, then PC + 1 => PC

Skip on Data Completion Flag (DFSC)

Octal Code: 6622
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Skips next instruction if the completion flag is a 1. Indicates data transfer is complete.
Symbol: If completion flag = 1, PC + 1 => PC

Read Disc Memory Address Register (DMAC)

Octal Code: 6626
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Clears the AC then loads contents of the Disc Memory Address Register into the AC to allow program evaluation. During read, the final address will be the last one transferred.
Symbol: $DMA_{0-11} => AC_{0-11}$

SOFTWARE

DF32 Disc System, available with PDP-8/I, is a fast convenient keyboard oriented monitor which will enable the user to efficiently control the flow of programs through his PDP-8/I. This system is modular and open ended, allowing the user to build the software components required in his environment. The user may specify the system device (Disc or DECtape), the amount of core, number of discs available and the number, name and size of his resident system programs.

AUTOMATIC MAGNETIC TAPE CONTROL, TYPE TC58
Functional Description

The Type TC58 will control the operation of a maximum of eight digital magnetic tape transports, Types TU20 and TU20A. The Type TC58 interfaces to and uses the PDP-8/I 3-cycle data break facility for data transfer directly to or from system core memory and magnetic tape. The tape transports offer industry-compatible (or IBM-compatible) in both 7 and 9 channel tape transports with the following characteristics:

TRANSPORT	TAPE SPEED (ips)	DENSITIES (bpi)
TU20 (7-channel)	45 ips	200/556/800
TU20A (9-channel)	45 ips	800

Transfers are governed by the in-memory word count (WC) and current address (CA) register associated with the assigned data channel (memory locations 32_8 and 33_8). Since the CA is incremented before each data transfer, its initial contents should be set to the desired initial address minus one. The WC is also incremented before each transfer and must be set to the 2's complement of the desired number of data words to be transferred. In this way, the word transfer which causes the word count to overflow (WC becomes zero) is the last transfer to take place. The number of IOT instructions required for the TYPE TC58 is minimized by transferring all necessary control data (i.e., unit number, function, mode, direction, etc.) from the PDP-8/I accumulator (AC) to the control using IOT instructions. Similarly, all status information (i.e., status bits, error flags, etc.) can be read into the AC from the control unit by IOT instructions.

During normal data reading, the control assembles 12-bit computer words from successive frames read from the information channels of the tape. During normal data writing, the control disassembles 12-bit words and distributes the bits so they are recorded on successive frames of the information channels.

INSTRUCTIONS

The commands for the Magnetic Tape Control System are as follows:

Skip on Error Flag or Magnetic Tape Flag (MTSF)

Octal Code: 6701
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The status of the error flag (EF) and the magnetic tape flag (MTF) are sampled. If either or both are set to 1, the content of the PC is incremented by one to skip the next sequential instruction.
Symbol: If MTF or EF = 1, PC + 1 = > PC

Skip on Tape Control Ready (MTCR)

Octal Code: 6711
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: If the tape control is ready to receive a command, the PC is incremented by one to skip the next sequential instruction.
Symbol: If Tape Control Ready, PC + 1 = > PC

Skip on Tape Transport Ready (MTTR)

Octal Code: 6721
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The next sequential instruction is skipped if the tape transport is ready.
Symbol: If tape unit ready, PC + 1 = > PC

Clear Registers, Error Flag and Magnetic Tape Flag (MTAF)

Octal Code: 6712
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Clears the status and command registers, and the EF and MTF if tape control is ready. If tape control not ready, clears MTF and EF flags only.
Symbol: If tape control is ready, 0 = > MTF, 0 = > EF,
0 = > command register
If tape control not ready, 0 = > MTF, 0 = > EF

Inclusive OR Contents of Command Register

Octal Code: 6724
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Inclusively OR the contents of the command register into bits. 0-11 of the AC.
Symbol: AC $\vee$ command register = > AC

Inclusive OR Contents of Accumulator (MTCM)

Octal Code: 6714
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Inclusively OR the contents of AC bits 0-5, 9-11 into the command register; JAM transfer bits 6, 7, 8 (command function).
Symbol: AC_{0-5}, AC_{9-11} $\vee$ command register = > command register
AC_{6-8} = > command register bits 6-8

Load Command Register (MTLC)

Octal Code: 6716
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Load the contents of AC bits 0-11 into the command register.
Symbol: AC_{0-11} = > command register

Inclusive OR Contents of Status Register

Octal Code: 6704
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Inclusively OR the contents of the status register into bits 0-11 of the AC.
Symbol: Status Register $\vee$ AC = > AC_{0-11}

Read Status Register (MTRS)

Octal Code: 6706
Event Time: 2, 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Read the contents of the status register into bits 0-11 of the AC.
Symbol: Status Register $=>$ AC_{0-11}

Mag Tape "GO" (MTGO)

Octal Code: 6722
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Set "GO" bit to execute command in the command register if command is legal.
Symbol: None

Clear the AC

Octal Code: 6702
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Clear the accumulator.
Symbol: $0 => AC$

Although any number of tapes may be simultaneously rewinding, data transfer may take place to or from only one transport at any given time. In this context, data transfer includes these functions: read or write data, write EOF (end of file), read/compare, and space. When any of these functions are in process, the tape control is in the "not ready" condition. A transport is said to be "not ready" when tape is in motion, when transport power is off, or when it is off line.

Data transmission may take place in either parity mode, odd-binary or even-BCD. When reading a record in which the number of characters is not a multiple of the number of characters per word, the final characters come into memory left-justified.

Ten bits in the magnetic tape status register retain error and tape status information. Some error types are combinations, such as lateral and longitudinal parity errors (parity checks occur after both reading and writing of data), or have a combined meaning, such as illegal command, to allow for the maximum use of the available bits.

The magnetic tape status register reflects the state of the currently selected tape unit. Interrupts may occur only for the selected unit. Therefore, other units which may be rewinding, for example, will not interrupt when done.

A special feature of this control is the "Write Extended Inter-Record Gap" capability. This occurs on a write operation when Command Register bit 5 is set. The effect is to cause a 3-inch inter-record gap to be produced before the record is written. The bit is automatically cleared when the writing begins.

This is very useful for creating a 3-inch gap of blank tape over areas where tape performance is marginal.

Magnetic Tape Functions

For all functions listed below, upon completion of the data operation (after the end-of-record character passes the read head), the MTF (magnetic tape flag) is set, an interrupt occurs (if enabled), and errors are checked.

No Operation

A NO OP command defines no function in the command register. A MTGO instruction with NO OP will cause an illegal command error (set EF).

Space

There are two commands for spacing records, SPACE FORWARD and SPACE REVERSE. The number of records to be spaced (2's complement) is loaded into the WC. CA need not be set. MTF (magnetic tape flag) is set, and an interrupt occurs at WC overflow, EOF (end of file), or EOT (end of tape), whichever occurs first. When issuing a space command, both the density and parity bits must be set to the density and parity in which the records were originally written.

Load Point or Beginning of Tape (BOT) detection during a backspace terminates the function with the BOT bit set. If a SPACE REVERSE command is given when a transport is set at BOT, the command is ignored, the illegal command error and BOT bits are set, and an interrupt occurs.

Read Data

Records may be read into memory only in the forward mode. Both CA and WC must be set: CA, to the initial core address minus one; WC, to the 2's complement of the number of words to be read. Both identify and parity bits must be set.

If WC is set to less than the actual record length, only the desired number of words are transferred into memory. If WC is greater than or equal to the actual record length, the entire record is read into memory. In either case, both parity checks are performed, the MTF is set, and an interrupt occurs when the end-of-record mark passes the read head. If either lateral or longitudinal parity errors or bad tape have been detected, or an incorrect record length error occurs, (WC not equal to the number of words in the record), the appropriate status bits are set. An interrupt occurs only when the MTF is set.

To continue reading without stopping tape motion, MTAF (clear MTF) and MTGO instructions must be executed. If the MTGO command is not given before the shut down delay terminates, the transport will stop.

Write Data

Data may be written on magnetic tape in the FORWARD DIRECTION ONLY. For the WRITE DATA function, the CA and WC registers and density and parity bits must be set. WRITE DATA is controlled by the WC, such that when the WC overflows, data transfer stops, and the EOR (end of record) character and IRG (inter-record gap) are written. The MTF is set after the EOR has passed the read head. To continue writing, a MTGO command must be issued before the shut down delay terminates. If any errors occur, the EF will be set when the MTF is set.

Write EOF

The WRITE EOF command transfers a single character (17_8) record to magnetic tape and follows it with EOR character. CA and WC are ignored for WRITE EOF. The density bits must be set, and the command register parity bit should be set to even (BCD) parity. If it is set to odd parity, the control

will automatically change it to even.

When the EOF marker is written, the MTF is set and an interrupt occurs. The tape transport stops, and the EOF status bit is set, confirming the writing of EOF. If odd parity is required after a WRITE EOF, it must be specifically requested through the MTLC command.

Read/Compare

The READ/COMPARE function compares tape data with core memory data. It can be useful for searching and positioning a magnetic tape to a specific record, such as a label or leader, whose content is known in core memory, or to check a record just written. READ/COMPARE occurs in the forward direction only; CA and WC must be set. If there is a comparison failure, incrementing of the CA ceases, and the READ/COMPARE error bit is set in the status register. Tape motion continues to the end of the record; the MTF is then set and an interrupt occurs. If there has been a READ/COMPARE error, examination of the CA reveals the word that failed to compare.

Rewind

The high speed REWIND command does not require setting of the CA or WC. Density and parity settings are also ignored. The REWIND command rewinds the tape to loadpoint (BOT) and stops. Another unit may be selected after the command is issued and the rewind is in process. MTF is set, and an interrupt occurs (if the unit is selected) when the unit is ready to accept a new command. The selected unit's status can be read to determine or verify that REWIND is in progress.

Continued Operation

1. To continue operating in the same mode, the MTGO instruction is given before tape motion stops. The order of commands required for continued operation are as follows:

   a. LCM, if the command is to be changed.

   b. MTAF, will only clear MTF and EF flags since tape control will be in a Not Ready state.

   c. MTGO, if LCM requested an illegal condition, the EF will be set at this time.

2. To change modes of operation, either in the same or opposite direction, the MTCM command is given to change the mode and a MTGO command is given to request the continued operation of the drive. If a change in direction is ordered, the transport will stop, pause, and automatically start up again.

3. If the WRITE function is being performed, the only forward change in command that can be given is WRITE EOF,

4. If no MTGO instruction is given, the transport will shut down in the inter-record gap.

(Note: No flags will be set when the control becomes ready or the transport becomes ready, except if the REWIND command is present in the command register and the selected drive reaches BOT and is ready for a new command.)

5. If a WRITE (odd parity) command is changed to WRITE EOF, the parity is automatically changed to even.

(Note: Even parity will remain in the command register unless changed by a new command instruction, MTLC, which clears and loads the entire command register.)

Status or Error Conditions

Twelve bits in the magnetic tape status register indicate status or error conditions. They are set by the control and cleared by the program.

The magnetic tape status register bits are:

BIT*	FUNCTION (WHEN SET)
0	Error flag (EF)
1	Tape rewinding
2	Beginning of tape (BOT)
3	Illegal command
4	Parity error (Lateral or Longitudinal)
5	End of file (EOF)
6	End of tape (EOT)
7	Read/compare error
8	Record length incorrect WC = 0 (long) WC ≠ 0 (short)
9	Data request late
10	Bad tape
11	Magnetic tape flag (MTF) or job done

*The register bits are equivalent in position to the AC bits (i.e., $SR_0 = AC_0$, etc.).

MTF (SR11)

The MTF flag is set under the following conditions:

1. Whenever the tape control has completed an operation (after the EOR mark passes the read head).

2. When the selected transport becomes ready following a normal REWIND function.

These functions will also set the EF if any errors are present.

EOF (SR5)

End-of-file (EOF) is sensed and may be encountered for those functions which come under the heading of READ STATUS FUNCTION, i.e., SPACE, READ DATA, or READ/COMPARE and WRITE EOF. When EOF is encountered, the tape control sets EOF = 1. MTF is also set; hence, an interrupt**occurs and the EOF status bit may be checked.

**All references to interrupts assume the tape flags have been enabled to the interrupt (command register bit 9 = 1) and that the unit is selected.

EOT (SR6) and BOT (SR2)

End-of-tape (EOT) detection occurs during any forward command when the EOT reflective strip is sensed. When EOT is sensed, the EOT bit is set, but the function continues to completion. At this time the MTF is set (and EF is set), and an interrupt occurs.

Beginning-of-tape (BOT) detection status bit occurs only when the beginning-of-tape reflective strip is read on the transport that is selected.

When BOT detection occurs, and the unit is in reverse, the function terminates. If a tape unit is at load point when a REVERSE command is given, an illegal command error bit is set, causing an EF with BOT set. An interrupt then occurs.

Illegal Command Error (SR3)

The illegal command error bit is set under the following conditions:

1. A command is issued to the tape control with the control not ready.

2. A MTGO command is issued to a tape unit which is not ready, and the tape control is ready.

3. Any command which the tape control, although ready, cannot perform; e.g.:

a. WRITE with WRITE LOCK condition
b. 9-channel tape and incorrect density
c. BOT and SPACE REVERSE

Parity (SR4)

Longitudinal and lateral parity checks will occur in both reading and writing. The parity bit is set for either lateral or longitudinal parity failure. A function is not interrupted, however, until MTF is set. Maintenance panel indicators are available to determine which type of parity error occurred.

Read Compare Error (SR7)

When READ/COMPARE function is underway, SR7 is set to 1 for a READ/COMPARE ERROR (see earlier section on READ/COMPARE for further details).

Bad Tape (SR10)

A BAD TAPE ERROR indicates detection of a bad spot on the tape. Bad tape is defined as three or more consecutive missing characters followed by data, within the period defined by the READ SHUTDOWN DELAY. The error bit is set by the tape control when this occurs. MTF and interrupt do not occur until the end of the record in which the error was detected.

Tape Rewinding (SR1)

When a REWIND command has been issued to a tape unit and the function is underway, the tape rewinding bit is set in the control. This is a transport status bit, and any selected transport which is in a high speed rewind will cause this bit to be set.

Record Length Incorrect (SR8)

During a read or read/compare, this bit is set when the WC overflow differs from the number of words in the record. The EF flag is set.

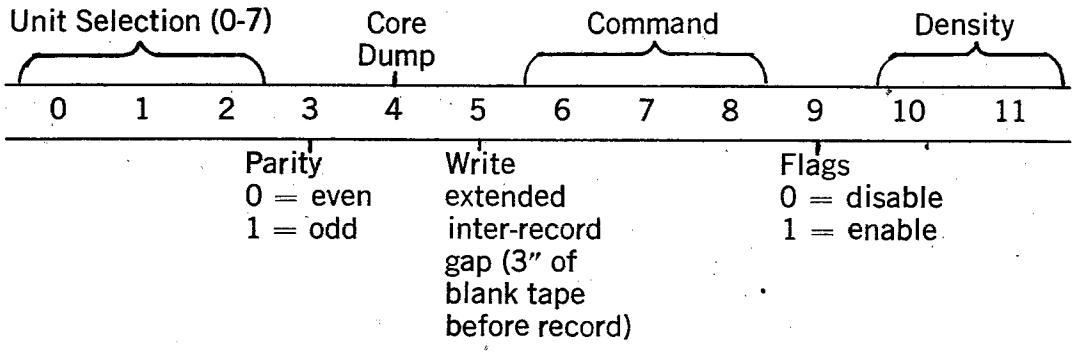
Data Request Late (SR9)

This bit can be set whenever data transmission is in progress. When the DATA FLAG causes a break cycle, the data must be transmitted before a write pulse or a read pulse occurs. If it does not, this error will occur, and data transmission will cease. The EF flag and bit 9 of the status register are set when the MTF is set.

Error Flag (SR0)

The ERROR FLAG (EF) is set whenever any error status bit is present at the time that MTF is set. However, when an ILLEGAL COMMAND is given, the EF is set and the MTF is not set.

Command Register Contents



Unit Selection

Unit	Unit Selection Bits 0	1	2
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Density Selection

Density	Density Bits	
200 BPI	0	0
556 BPI	0	1
800 BPI	1	0
800 BPI 9 channel	1	1

Command Selection

COMMAND	BITS 6	7	8
NO OP	0	0	0
Rewind	0	0	1
Read	0	1	0
Read/Compare	0	1	1
Write	1	0	0
Write EOF	1	0	1
Space Forward	1	1	1
Space Reverse	1	1	1

Magnetic Tape Function Summary

LEGEND: CA = Current Address Register = 32_8
WC = Word Count Register = 33_8
F = Forward
R = Reverse
DS = Density Setting
PR = Parity Setting
EN = Enable Interrupt

Function	Characteristics	Status of Error Types
NO-OP	CA: Ignored WC: Ignored DS: Ignored PR: Ignored EN: Ignored	Illegal BOT Tape Rewinding
SPACE FORWARD	CA: Ignored WC: 2's comp. of number of records to skip DS: Must be set PR: Must be set EN: Must be set	Illegal EOF Parity Bad Tape MTF, BOT, EOT
SPACE REVERSE	Same as Space Forward	Illegal EOF Parity Bad Tape BOT MTF
READ DATA	CA: Core Address — 1 WC: 2's comp. of number of words to be transferred DS: Must be set PR: Must be set EN: Must be set	Illegal EOF Parity Bad Tape MTF EOT Data Request Late Record Length Incorrect
WRITE DATA	Same as READ DATA	Illegal EOT Parity MTF Bad Tape Data Request Late
WRITE EOF	CA: Ignored WC: Ignored DS: Must be set PR: Must be set EN: Must be set	Same as WRITE DATA plus EOF

Function	Characteristics	Status of Error Types
READ/COMPARE	Same as READ DATA	Illegal EOF Read/Compare Error Bad Tape MTF EOT Data Late Record Length Incorrect
REWIND	CA: Ignored WC: Ignored DS: Ignored PR: Ignored EN: Must be set	Illegal Tape Rewinding MTF BOT

Magnetic Tape Transport, Type TU20 (7-channel)

The Type TU20 is a digital magnetic tape transport designed to be compatible with the Type TC58 Magnetic Tape Control. The transport operates at a speed of 45 inches per second and has three selectable densities: 200, 556, and 800 bpi. The maximum transfer rate is 36,000 six-bit characters per second. Standard seven-channel IBM-compatible tape format is used. The specifications for the unit are as follows:

FORMAT: NRZI. Six data bits plus one parity bit.

End and loadpoint sensing compatible with IBM 729 I-VI.

TAPE: Width of 0.5 inch. Length of 2400 ft. (1.5 mil.). Reels are 10.5 in., IBM-compatible, with file protect (WRITE LOCK) ring.

HEADS: Write-read gap of 0.300 in. Dynamic and static skew is less than 14 μsec.

TAPE SPECIFICATIONS: 45 IPS speed. Rewind time is less than 5 msec. Start distance is 0.080 in. (+0.035, —0.025 in.). Stop time is less than 1.5 msec. Stop distance is 0.045 in. ($\pm$0.015 in.).

DENSITY: 200, 556, and 800 BPI. Maximum transfer rate is 36 kHz.

TRANSPORT MECHANISM: Pinch roller drive; vacuum column tension.

CONTROLS: ON/OFF, ON LINE, OFF LINE, FORWARD, REVERSE, REWIND, LOAD, RESET.

PHYSICAL SPECIFICATIONS: Width of 22¼ in., depth of 27⅛ in., height of 69⅛. Weight — 600 lbs.

READ (READ/COMPARE) SHUTDOWN DELAY: 3.6 milliseconds.
WRITE SHUTDOWN DELAY: Approximately 4.5 milliseconds.

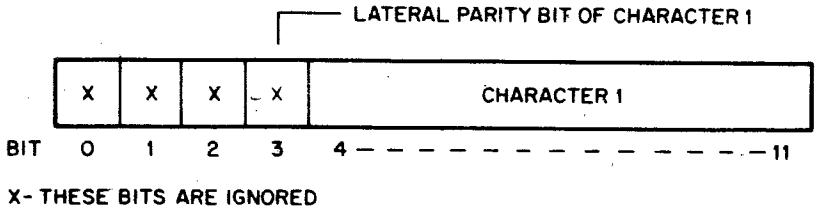
9-TRACK OPERATION

9- and 7-track transports may be intermixed on the Type TC58 control. When a transport is selected, it automatically sets the control for proper operation with its number of tracks.

Control of 9-track operation is identical to 7-track, except as noted below:

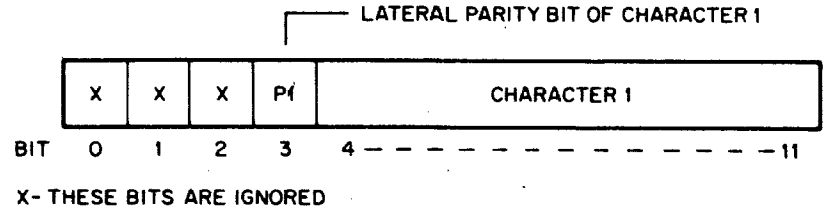
Write

A word in memory is written on tape with format shown below:



Read

A word is read into memory from tape with the format shown below:



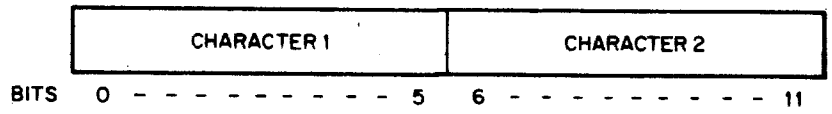
Read/Compare

A direct comparison of the characters on tape is made with those in memory. The parity bit is ignored, as are bits 0-3 in each memory word.

Core Dump Mode

This mode is used only with 9-track transports. It is entered by setting bit 4 of the command register.

Core dump mode permits the dumping of complete memory words in the form of two six-bit characters. The format is:



This is accomplished by only utilizing 7 of the 9 tracks on the tape.

Tape written in CORE DUMP MODE, must be READ (READ/COMPARE) in the same mode. These operations are the same as for a 7-track transport.

Magnetic Tape Transport, Type TU20A (9-channel)

The Type TU20A is a digital magnetic tape transport designed to be compatible with the Type TC58 Magnetic Tape Control. The transport operates at a speed of 45 inches per second and a density of 800 bpi. The maximum transfer rate is 36,000 eight-bit characters per second. Standard nine-channel

IBM-compatible tape format is used. The specifications for the unit are as follows:

FORMAT: NRZI. Eight data bits plus on parity bit.
End and loadpoint sensing compatible with IBM.

TAPE: Width of 0.5 inch. Length of 2400 ft. (1.5 mil.). Reels are 10.5 in., IBM-compatible, with file protect (WRITE LOCK) ring.

HEADS: Write-read gap of 0.150 in. Dynamic and static skew is less than 14 μsec.

TAPE SPECIFICATIONS: 45 IPS speed. Rewind time is less than 5 msec. Start distance is 0.080 in. (+0.035, —0.025 in.). Stop time is less than 1.5 msec. Stop distance is 0.045 in. ($\pm$0.015 in.).

DENSITY: 800 BPI. Maximum transfer rate is 36 kHz.

TRANSPORT MECHANISM: Pinch roller drive; vacuum column tension.

CONTROLS: ON/OFF, ON LINE, OFF LINE, FORWARD, REVERSE, REWIND, LOAD, RESET.

PHYSICAL SPECIFICATIONS: Width of 22¼ in., depth of 27⅛ in., height of 69⅛. Weight — 600 lbs.

READ (READ/COMPARE) SHUTDOWN DELAY: 3.6 milliseconds.

WRITE SHUTDOWN DELAY: Approximately 4.5 milliseconds.

DECTAPE SYSTEM

The DECtape system is a standard option for the PDP-8/I which serves as an auxiliary magnetic tape data storage facility. The DECtape system stores information at fixed positions on magnetic tape as in magnetic disk or drum storage devices, rather than at unknown or variable positions as is the case in conventional magnetic tape systems. This feature allows replacement of blocks of data on tape in a random fashion without disturbing other previously recorded information. In particular, during the writing of information on tape, the system reads format (mark) and timing information from the tape and uses this information to determine the exact position at which to record the information to be written. Similarly, in reading the same mark and timing information is used to locate data to be played back from the tape.

This system has a number of features to improve its reliability and make it exceptionally useful for program updating and program editing applications. These features are: phase or polarity sensed recording on redundant tracks, bidirectional reading and writing, and a simple mechanical mechanism utilizing hydrodynamically lubricated tape guiding (the tape floats on air and does not touch any metal surfaces).

DECtape Format

DECtape utilizes a 10-track read/write head. Tracks are arranged in five non-adjacent redundant channels: a timing channel, a mark channel, and three information channels. Redundant recording of each character bit on non-adjacent tracks materially reduces bit drop outs and minimizes the effect of

skew. Series connection of corresponding track heads within a channel and the use of Manchester phase recording techniques, rather than amplitude sensing techniques, virtually eliminate drop outs.

The timing and mark channels control the timing of operations within the control unit and establish the format of data contained on the information channels. The timing and mark channels are recorded prior to all normal data reading and writing on the information channels. The timing of operations performed by the tape drive and some control functions are determined by the information on the timing channel. Therefore, wide variations in the speed of tape motion do not affect system performance. Information read from the mark channel is used during reading and writing data, to indicate the beginning and end of data blocks and to determine the functions performed by the system in each control mode.

During normal data reading, the control assembles 12-bit computer length words from four successive lines read from the information channels of the tape. During normal data writing, the control disassembles 12-bit words and distributes the bits so they are recorded on four successive lines on the information channels. A mark channel error check circuit assures that one of the permissible marks is read in every six lines on the tape. This 6-line mark channel sensing requires that data be recorded in 12-line segments (12 being the lowest common multiple of 6-line marks and 4-line data words) which correspond to three 12-bit words.

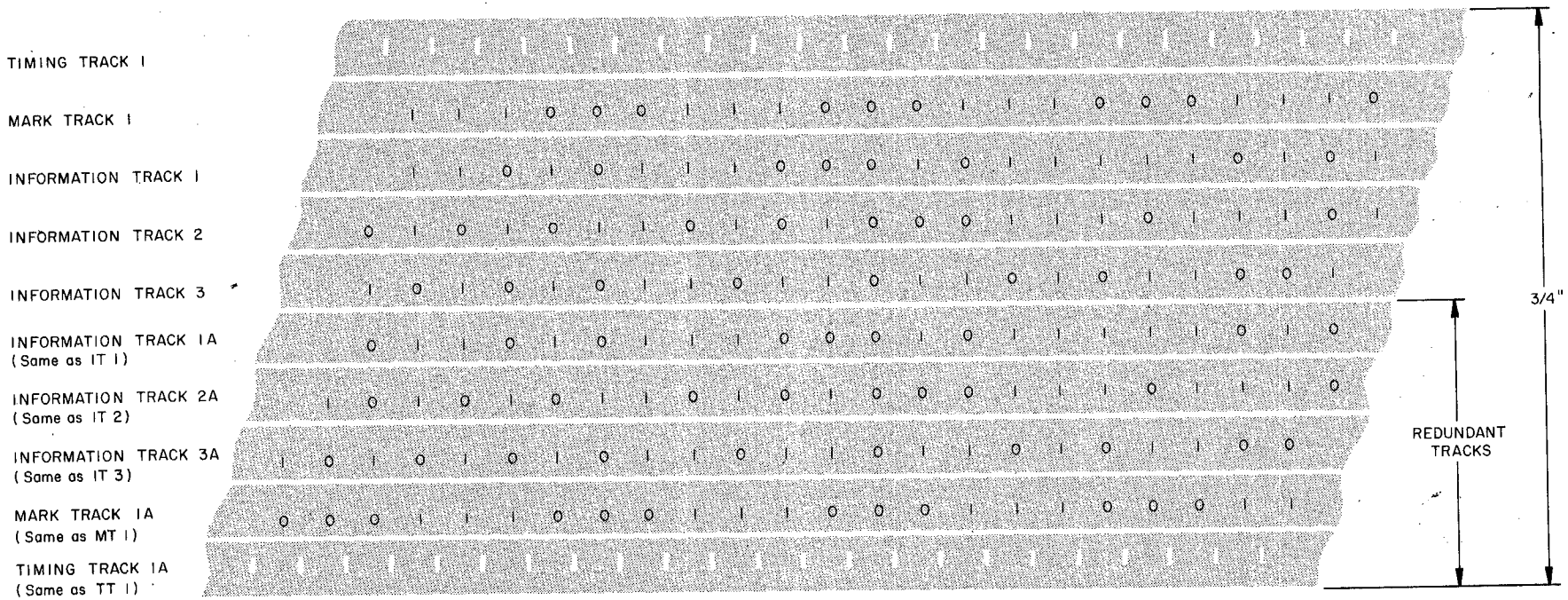


Figure 11. DECtape Track Allocations

[redacted]

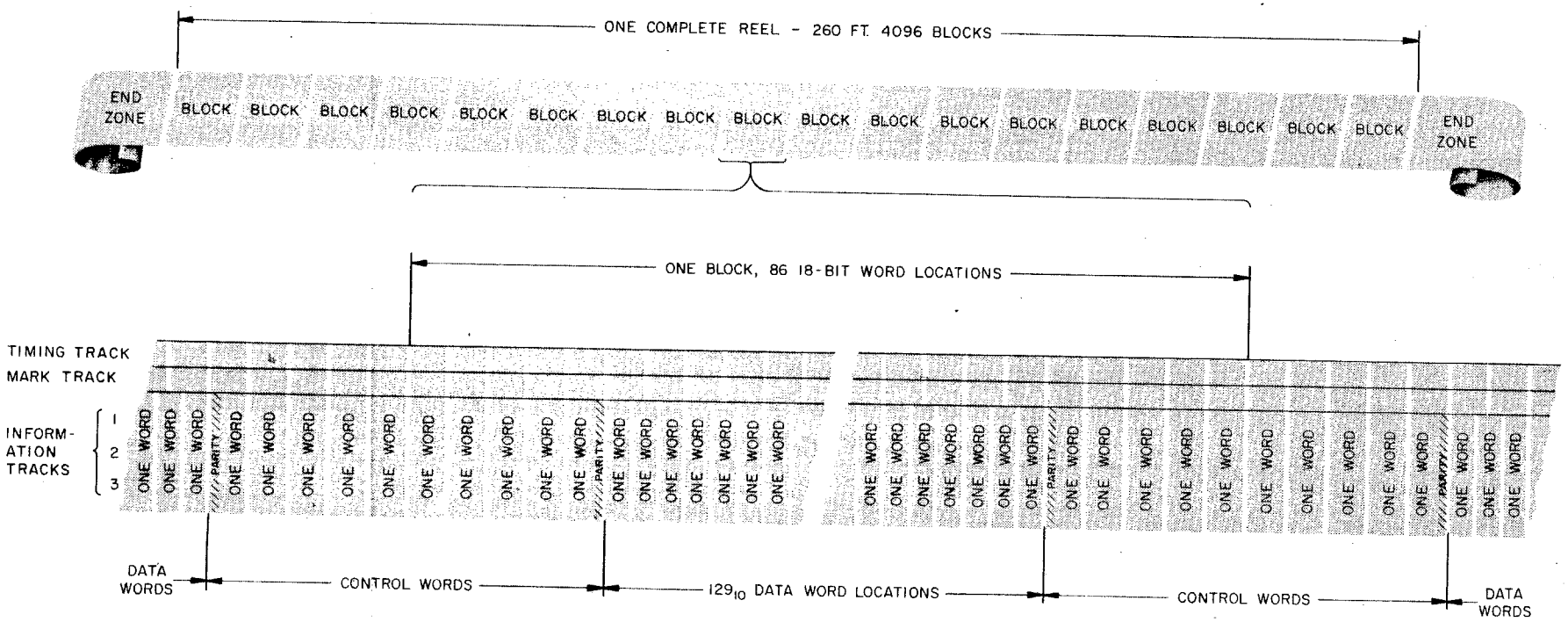


Figure 12. DECtape Mark Channel Format

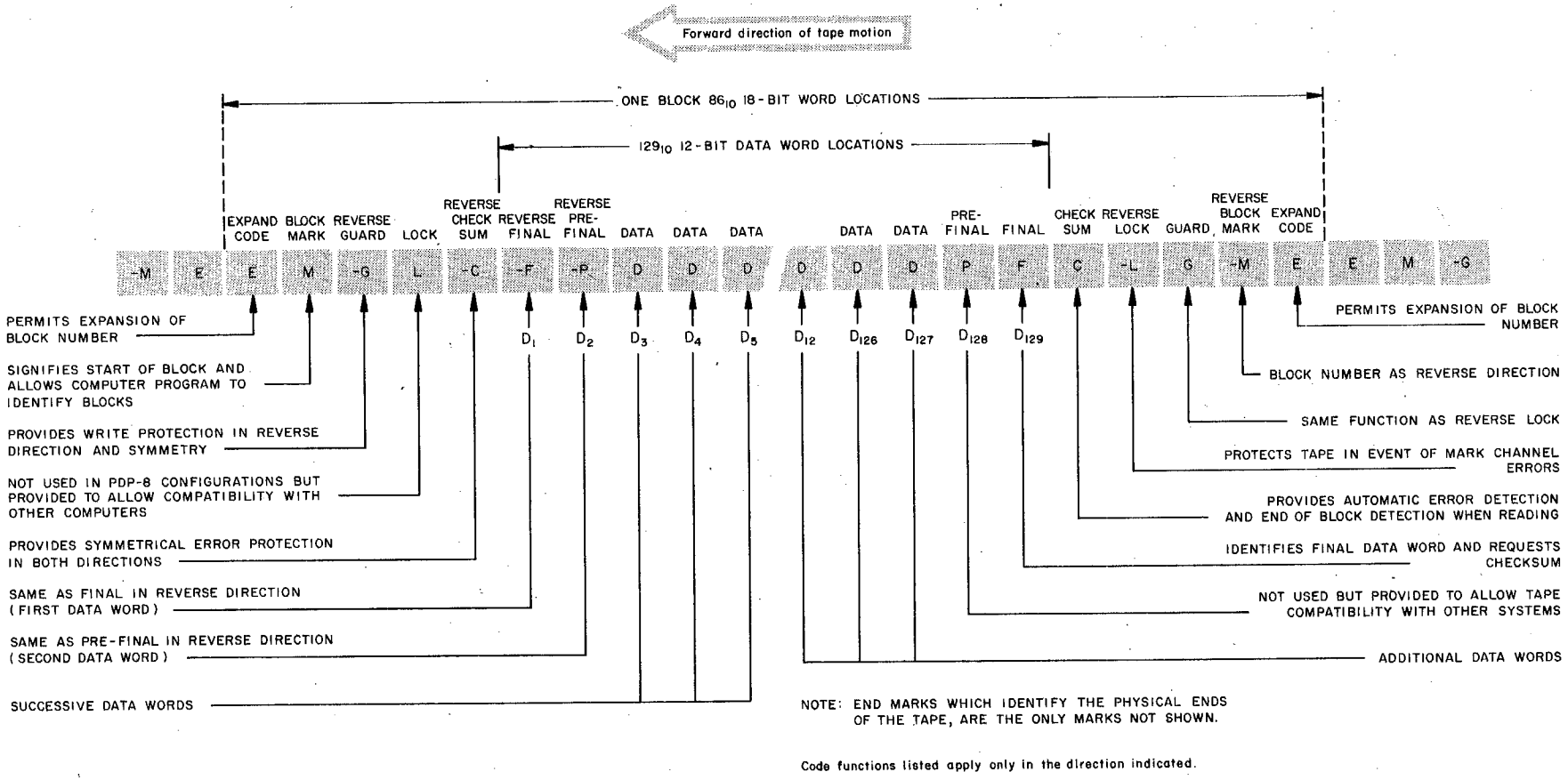


Figure 13. DECtape Control Word and Data Word Assignments

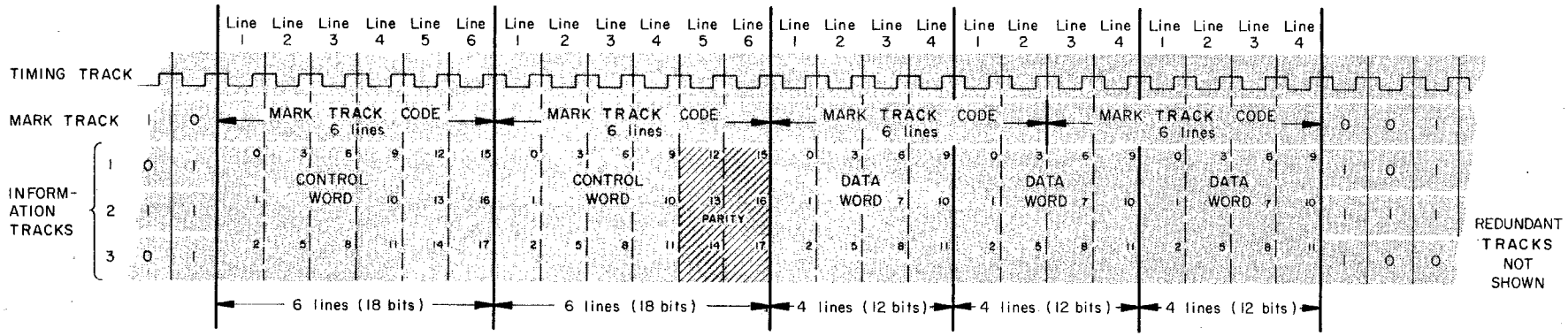


Figure 14. DECtape Format Details

A tape contains a series of data blocks that can be of any length which is a multiple of three 12-bit words. Block length is determined by information on the mark channel. Usually a uniform block length is established over the entire length of a reel of tape by a program which writes mark and timing information at specific locations. The ability to write variable-length blocks is useful for certain data formats. For example, small blocks containing index or tag information can be alternated with large blocks of data. (Software supplied with DECtape allows writing for fixed block lengths only.)

Between the blocks of data are areas called interblock zones. The interblock zones consist of 30 lines on tape before and after a block of data. Each of these 30 lines is divided into five 6-line control words. These 6-line control words allow compatibility between DECtape written on any of the DEC's 12-, 18-, or 36-bit computers. As used on the PDP-8/I, only the last four lines of each control word are used.

Block numbers normally occur in sequence from 1 to N. There is one block numbered 0 and one block N + 1. Programs are entered with a statement of the first block number to be used and the total number of blocks to be read or written. The total length of the tape is equivalent to 849,036 lines which can be divided into any number of blocks up to 4096 by prerecording of the mark track. The maximum number of blocks is determined by the following equation in which N_b = number of blocks and N_w = number of words per block (N_w must be divisible by 3).

$$N_b = \frac{212112}{(N_w + 15)} - 2$$

DECtape format is illustrated in Figures 11 through 14.

DECtape Transport (Type TU55) and DECtape Control (Type TC01)

A DECtape system configuration contains up to eight TU55 transports operated from one TC01 control. All data transfers occur between the computer and the control and are effected by the three-cycle data break facility. A 12-bit data buffer in the control synchronizes transfers between the TC01 and the PDP-8/I data break facility. Data read from four consecutive lines on tape by the transport are assembled into 12-bit words by a read/write buffer from the computer is disassembled by the read/write buffer and supplied to the transport for writing on four lines of tape.

Transfer of command and control signals between the computer and the control is effected by normal IOT instructions. Small registers and control flip-flops in the TC01 are joined to serve as two status registers for the transfer of command and control information with the PDP-8/I accumulator. Bit assignments of these registers are indicated in Figure 15 and Figure 16.

DECTAPE TRANSPORT (TYPE TU55)

The TU55 is a bidirectional magnetic-tape transport consisting of a read/write head for recording and playback of information on five channels of the tape. Connections from the read/write head are made directly to the external control which contains the read and write amplifiers.

The logic circuits of the TU55 control tape movement in either direction over the read/write head. Tape drive motor control is exercised completely through the use of solid state switching circuits to provide fast reliable operation. These switching circuits contain silicon controlled rectifiers which are controlled by normal DEC diode and transistor logic circuits. These circuits control the torque of the two motors which transport the tape across the head according to the established function of the device, i.e., go, stop, forward, or reverse. In normal tape movement, full torque is applied to the forward or leading motor and a reduced torque is applied to the reverse or trailing motor to keep proper tension on the tape. Since tape motion is bidirectional, each motor serves as either the leading or trailing drive for the tape, depending upon the forward or reverse control status of the TU55. A positive stop is achieved by an electromagnetic brake mounted on each motor shaft. When a stop command is given, the trailing motor brake latches to stop tape motion. Enough torque is then applied to the leading motor to take up slack in the tape.

Tape movement can be controlled by commands originating in the computer and applied to the TU55 through the TC01 DECtape Control, or can be controlled by commands generated by manual operation of rocker switches on the front panel of the transport. Manual control is used to mount new reels of tape on the transport, or as a quick maintenance check for proper operation of the control logic in moving the tape.

TU55 Dectape Transport Characteristics

Times given are typical but are not accurately controlled. Since DECtape is a fixed address system the programmer need not know accurately where the tape has stopped. To locate a specific point on tape he must merely start tape motion in search mode. The address of the block currently passing over the head will be automatically transferred to core where it can be compared with the desired block address and tape motion continued or reversed accordingly.

Start Time —	200 M Sec*
Stop Time —	200 M Sec*
Turn Around Time —	275 M Sec*

***Note** Also see control spec. These times are frequently lengthened by the particular control.

DECTAPE CONTROL TYPE TC01

The TC01 DECtape Control operates up to eight TU55 DECtape Transports. Binary information is transferred between the tape and the computer in 12-bit computer words approximately every 133⅓ microseconds. In writing, the control disassembles 12-bit computer words so that they are written at four successive lines on tape. Transfers between the computer and the control always occur in parallel for a 12-bit word. Data transfers use the three-cycle data break facility of the computer. As the start and end of each block of data are detected by the mark track detection circuits, the control raises a DECtape control flag (DTCF) which requests a computer program interrupt. The program interrupt is used by the computer program to determine the block number. When it determines that the forthcoming block is the one selected for a data transfer it establishes the appropriate read or write function. Each time a word is assembled or the DECtape system is ready to receive a word from the computer, the control raises a data flag (DF). This flag is connected to the computer data break facility to request a data break. Therefore, when each 12-bit computer word is assembled, the data flag causes a transfer via the three-cycle data break. By using the mark channel decoding circuits and

the data break in this manner, computation in the main computer program can continue during DECtape operations.

Four program flags in the control serve as condition indicators and request originators.

DECtape Flag (DT): This flag indicates the active/done status of the current function.

Data Flag (DF): This flag requests a data break to transfer a block number into the computer during a search function, or when a data word transfer is required during read or write function.

DECtape Control Flag (DTCF): This flag, when enabled by a binary 1 in bit 9 of status register A, requests a program interrupt if either the DECtape flag or the error flag is set and is connected to the instruction skip facility.
Error Flag (EF): Detection of any non-operative condition by the control sets this flag in status register B and stops (except for parity errors) the selected transport. The error conditions indicated by this flag are:

a. Mark Track Error: This error occurs any time the information read from the mark channel is erroneously decoded.

b. End of Tape: The end zone on either end of the tape is over the read head.

c. Select Error: This error occurs five microseconds after loading status register A to indicate any one of the following conditions:
   1. Specifying a unit select code which does not correspond to any transport select number, or which is set to multiple transports.
   2. Specifying a write function with the WRITE ENABLED/WRITE LOCK switch in the WRITE LOCK position on the selected transport.
   3. Specifying an unused function code (i.e. AC6-8 = 111).
   4. Specifying any function except read all with the NORMAL/WRTM/RDMK switch in the RDMK position.
   5. Specifying any function except write timing and mark track with the NORMAL/WRTM/RDMK switch in the WRTM position.
   6. Specifying the write timing and mark track function with the NORMAL/WRTM/RDMK switch in a position other than WRTM.

d. Parity Error: This error occurs during a read data function if the longitudinal parity over the entire data word, the reverse checksum, and the checksum is not equal to 1.

e. Timing Error: This error indicates a program fault caused by one of the following conditions:
   1. A data break did not occur within 66 microseconds (±30%) of the data break request.
   2. The DT flag was not cleared by the program before the control attempted to set it.
   3. The read data or write data function was specified while a data block was passing the read head.

INSTRUCTIONS

Instructions for a TC01/TU55 system are microprogrammed commands of the PDP-8/I IOT instruction and are defined as follows:

Read Status Register A (DTRA)

Octal Code: 6761
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of status register A is transferred into the accumulator by an OR transfer. The AC is not cleared before the transfer. The AC bit assignments are:

AC0-2 = Transport unit select number
AC3(0) = Forward
AC3(1) = Reverse
AC4(0) = Stop
AC4(1) = Go
AC5(0) = Normal mode
AC5(1) = Continuous mode

AC6-8 = 0 = Move function
AC6-8 = 1 = Search function
AC6-8 = 2 = Read data function
AC6-8 = 3 = Read all function
AC6-8 = 4 = Write data function
AC6-8 = 5 = Write all function
AC6-8 = 6 = Write timing and mark tracks function
AC6-8 = 7 = Unused (causes a select error if issued)
AC9(0) = DECtape Control Flag (DTCF) and error flag disabled from causing a program interrupt
AC9(1) = DECtape Control Flag (DTCF) and error flag enabled to cause a program interrupt.

Symbol: AC0-9 V Status Register A = > AC0-9

Clear Status Register A (DCTA)

Octal Code: 6762
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: Status register is cleared. The DECtape flag and error flags are undisturbed.
Symbol: 0 = > Status Register A

Load Status Register A (DTXA)

Octal Code: 6764
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The exclusive OR of the content of bits 0 through 9 of the accumulator is loaded into status register A, and bits 10 and 11 of the accumulator are sampled to control clearing of the error and DECtape flags, respectively. Loading status register A from AC0-9 establishes the transport unit select code, motion control, function, and enables or disables the DECtape control flag to request a program interrupt as described in the DTRA instruction. The sampling of AC10 and AC11 is as follows:

AC10(0) = Clear all error flags
AC10(1) = All error flags undisturbed
AC11(0) = Clear DECtape flag
AC11(1) = DECtape flag undisturbed

The accumulator is cleared at the end of this instruction.
Symbol: AC0-9 $\vee$ Status Register A $=>$ Status Register A
If AC10 = 0, then 0 $=>$ EF Flag
If AC11 = 0, then 0 $=>$ DT Flag
0 $=>$ AC

Skip on Flags (DTSF)

Octal Code: 6771
Event Time: 1

Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of both the error flag and the DECtape flag is sampled, and if any flag contains a binary 1, the content of the program counter is incremented by one to skip the next sequential instruction.
Symbol: If EF Flag = 1 V DT Flag = 1, then PC + 1 $=>$ PC

Read Status Register B (DTRB)

Octal Code: 6772
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of status register B is transferred into the accumulator by an OR transfer. The AC is not cleared before the transfer. The AC bit assignments are:

AC0 = Error flag (Error flag = mark track error V end of tape V select error V parity error V timing error)
AC1 = Mark track error
AC2 = End of tape
AC3 = Select error
AC4 = Parity error
AC5 = Timing error
AC6-8 = Memory field
AC9-10 = Unused
AC11 = DECtape flag

Symbol: AC V Status Register B $=>$ AC

Load Status Register B (DTLB)

Octal Code: 6774
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The memory field register portion status register B is loaded from the content of bits 6 through 8 of the accumulator. The accumulator is then cleared.
Symbol: AC6-8 $=>$ Memory Field, then 0 $=>$ AC

CONTROL MODES

The DECtape system operates in either the normal or continuous mode, as determined by bit 5 of status register A during a DTXA command. Operation in each mode is as follows:
Normal (NM): Data transfers and flag settings are controlled by the format of information on the tape.
Continuous (CM): Data transfers and flag settings are controlled by a word count (WC) read from core memory during the first cycle of each three-cycle data break; and by the tape format.

FUNCTIONS

The DECtape system performs one of seven functions, as determined by the octal digit loaded into status register A during a DTXA command. These functions are:

Move: Initiates movement of the selected transport tape in either direction.

Mark channel decoding is inhibited in this mode except for end of tape.

Search: As the tape is moved in either direction, sensing a block mark causes a data transfer of the block number. If the word count overflows (WCO) in either NM or CM, the DT flag is set and causes a program interrupt. After finding the first block number, the CM can be used to avoid all intermediate interrupts between the current and the desired block number. This makes a virtually automatic search possible.

Read Data: This function is used to transfer blocks of data into core memory with the transfer controlled by the tape format. In NM the DT flag is set at the end of a block and causes a program interrupt. In CM transfers stop when the word count overflows, the remainder of the block is read for parity checking, and then the DT flag is set.

Read All: Read all is used to read tape in an unusual format, since it causes all lines to be read. In NM the DT flag is set at each data transfer. In CM the DT flag is set when WCO occurs. In either case the DT flag causes a program interrupt.

Write Data: This function is used to write blocks of data with the transfer controlled by the standard tape format. After WCO occurs, zeros are written in all lines of the tape to the end of the current block. Then the parity checksum for the block is written. The DT flag rises as the in the read data function .

Write All: The write all function is used to write an unusual tape format (e.g., block numbers). The DT flag raisings are similar to the read all function.

Write Timing and Mark Track: This function is used to write on the timing and mark tracks. This permits blocks to be established or block lengths to be changed. The DT flag raisings are also similar to the read all function. This function is illegal unless a manual switch in the control is on.

PROGRAMMED OPERATION

Prerecording of a reel of DECtape, prior to its use for data storage, is accomplished in two passes. During the first pass, the timing and mark channels are placed on the tape. During the second pass, forward and reverse block mark numbers, the standard data pattern, and the automatic parity checks are written. These functions are performed by the DECTOG program. Prerecording utilizes the write timing and mark channel function and a manual switch on the control which permits writing on the timing and mark channels, activates a clock which produces the timing channel recording pattern, and enables flags for program control. Unless both this control function and switch are used simultaneously, it is physically impossible to write on the mark or timing channels. A red indicator lamp on the control lights when the manual NORMAL/WRTM/RDMK switch is in the WRTM position. Under these conditions only, the write register and write amplifier used to write on information channel 1 (bits 0, 3, 6, and 9) is used to write on the mark channel. This operation of prerecording need only be performed once for each reel of DECtape.

There are two registers in the TC01 DECtape Control that govern tape operation and provide status information to the operating program. Status register A (see Figure 15) contains three unit selection bits, two motion bits, the continuous mode/normal mode bit, three function bits, and three bits that control the flags. Status register B (see Figure 16) contains the three memory field bits and the error status bits. PDP-8/I IOT microinstructions are used to clear, read, and load these registers. In addition, there is an IOT skip instruction to test control status.

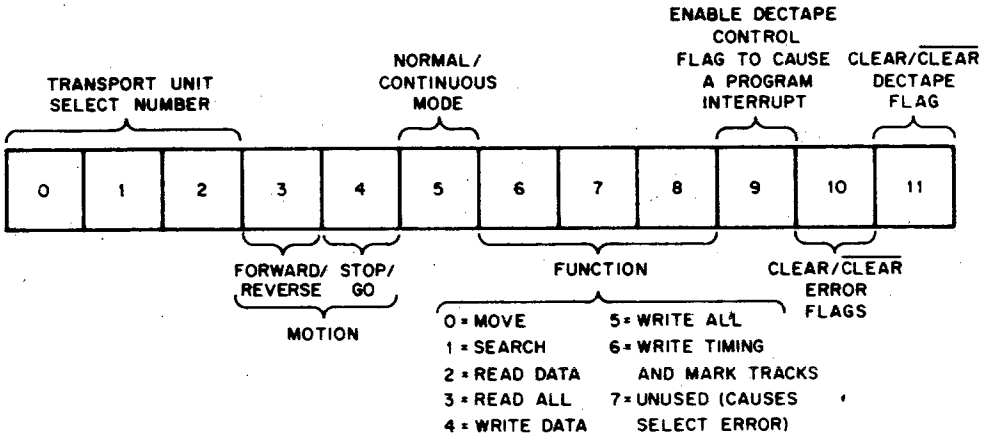


Figure 15. DECtape Status Register A Bit Assignments

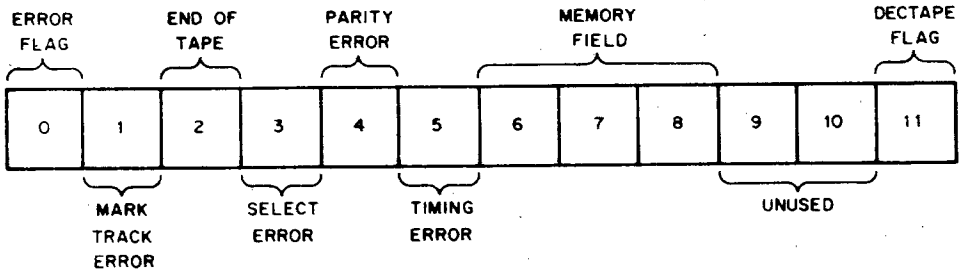


Figure 16. DECtape Status Register B Bit Assignments

Since all data transfers between DECtape and PDP-8/I memory are controlled by the data break facility, the program must set the word count (WC) and current address (CA) registers (locations 7754 and 7755 respectively) before a data break. After initiating a DECtape operation, the program should always check for error conditions (a program interrupt would be initiated if the error flag is enabled and if the program interrupt system is enabled). The DECtape system should be started in the search function to locate the block number selected for transfer and then, when the correct block is found, the transfer is accomplished by programmed setting of the WC, CA, and status register A.

When searching, the DECtape control reads only block numbers. These are used by the operating program to locate the correct block number. In NM, the DECtape flag is raised at each block number. In CM, the DECtape flag is raised only after the word count reaches zero. The current address is not incremented during searching and the block number is placed in core memory at the location specified by the content of the CA. Data is transferred to or from the PDP-8 core memory from locations specified by the CA register; which is incremented by one before each transfer.

When the start of the data position of the block is detected, the data flag is raised to initiate a data break request to the data break facility each time the DECtape system is ready to transfer a 12-bit word. Therefore, the main computer program continues running but is interrupted approximately every 133⅓ microseconds for a data break to transfer a word. Transfers occur between DECtape and successive core memory locations specified by the CA. The initial transfer address -1 is stored in the CA by an initializing routine. The number of words transferred is determined ony by tape format in NM, or by tape format and the word count in CM. At the conclusion of the data transfer the DT flag is raised and a program interrupt occurs. The interrupt subroutine checks the DECtape error bits to determine the validity of the transfer and either initiates a search for the next information to be transferred or returns to the main program.

During all normal writing transfers, a checksum (the 6-bit logical equivalence of the words in the data block) is computed automatically by the control and is automatically recorded as one of the control words immediately following the data portion of the block. This same checksum is used during reading to determine that the data playback and recognition take place without error.

Any one of the eight tape transports may be selected for use by the program. After using a particular transport, the program can stop the transport currently being used and select another transport, or can select another transport while permitting the original selection to continue running. This is a particularly useful feature when rapid searching is desired, since several transports may be used simultaneously. Caution must be exercised however, for although the original transport continues to run, no tape end detection or other sensing takes place. Automatic end sensing that stops tape motion occurs in all functions, but only in the selected tape transport.

The following is a list of timing considerations for programmed operations. (N_S = the number of block numbers to be read in the search function and continuous mode, counting through the one causing the WCO. Only the block number causing the WCO requests a program interrupt N_D = number of words transferred ÷ the number of words per block. If the remainder ≠ 0, use the next larger whole number. N_A = number of words transferred.)

Operation	Timing
Answer a data break request	Up to 66 microseconds, ±30%
Word transfer rate	One 12-bit word every 133 microseconds, ±30%
Block transfer rate	One 129-word block every 18.2 milliseconds, ±30%
Start time	375* milliseconds, ±20%
Stop time	375* milliseconds, ±20%
Turn around time	375* milliseconds, ±20%
Change function from search to read data for the current block after DT flag from block number	400 microseconds, ±30%
Change function from search to write data for current block after DT flag from block number	400 microseconds, ±30%
Change function from read data to search for the next block after DT flag from transfer completion	1000 microseconds, ±30%
Change function from write data to search for next block after DT flag from transfer completion	1000 microseconds, ±30%

*These times are typical but not accurately controlled.

Operation	Timing
DECtape flag rises	
in continuous mode	
Move function	Never
Search function	(N_S) x (18.2 milliseconds, ±30%)
Read data function	(N_D) x (18.2 milliseconds, ±30%)
Read all function	(N_A) x (133 microseconds, ±30%)
Write data function	(N_D) x (18.2 milliseconds, ±30%)
Write all function	(N_A) x (133 microseconds, ±30%)
Write T&M function	(N_A) x (133 microseconds, ±30%)
In normal mode	
Move function	Never
Search function	Every 18.2 milliseconds, ±30%
Read data function	Every 18.2 milliseconds, ±30%
Read all function	Every 133 microseconds, ±30%
Write data function	Every 18.2 milliseconds, ±30%
Write all function	Every 133 microseconds, ±30%
Write T&M function	Every 133 microseconds, ±30%

Software

Four types of programs have been developed as DECtape software for the PDP-8/I:

a. Subroutines which the programmer may easily incorporate into a program for data storage, logging, data acquisition, data buffering (queing), etc.

b. A library calling system for storing named programs on DECtape and a means of calling them with a minimal size loader.

c. System software which provides for storing, assembling and editing of programs on DECtape thereby greatly increasing the versatility and flexibility of the PDP-8/I.

d. Programs for preformatting tapes controlled by the content of the switch register to write the timing and mark channels, to write block formats, to exercise the tape and check for errors, and to provide ease of maintenance.

Program development has resulted in a series of subroutines which read or write any number of DECtape blocks, read any number of 129-word blocks as 128 words (one memory page), or search for any block (used by read and write, or to position the tape). These programs are assembled with the user's program and are called by a JMS instruction. The program interrupt is used to detect the setting of the DECtape flag, thus allowing the main program to proceed while the DECtape operation is being completed. A program flag is set when the operation has been completed. Thus, the program effectively allows concurrent operation of several input/output devices along with operation of the DECtape system. These programs occupy two memory pages ($400_8 = 256_{10}$ words).

The library system has the following features: First and perhaps foremost, the system leaves the state of the computer unchanged when it exits. Second, it calls programs by name from the keyboard and allows for expansion of the program file stored on the tape. Finally, it conforms to existing system conventions, namely, that all of memory, except for the last memory page (7600_8-7777_8), is available to the programmer. This convention ensures that the Binary Loader program (paper tape), and/or future versions of this loader, can reside in memory at all times.

The PDP-8/I DECtape library system is loaded by a 17_{10}-instruction bootstrap routine that starts at address 7600_8. This loader calls a larger program into the last memory page, whose function is to preserve on the tape, the content of memory from 6000_8 through 7577_8, and then load the INDEX program and the directory into those same locations. Since the information in this area of memory has been preserved, it can be restored when operations have been completed. The basic system tape contains the following programs:

a. INDEX: Typing this word causes the names of all programs currently on file to be typed out.

b. UPDATE: Allows the user to add a new program to the files. Update queries the operator about the program's name, its starting address, and its location in core memory.

c. GETSYS: Generates a skeleton library tape on a specified DECtape unit.

d. DELETE: Causes a named file to be deleted from the tape.

Starting with the basic library tape, the user can build a complete file of his active programs and continuously update it. One of the uses of the library tape may be illustrated as follows:

A program is written in PDP-8/I FORTRAN that is to be used repeatedly. The programmer may call the FORTRAN compiler from the library tape and with it, compile the program, obtaining the object program. The FORTRAN operating system may then be called from the library tape and used to load the object program. At this time the library program UPDATE is called, the operator defines a new program file (consisting of the FORTRAN operating system and the object program), and adds it to the library tape. As a result, the entire operating program and the object program are now available on the DECtape library tape.

The DECtape system software is permanently stored on DECtape, from which it can be rapidly loaded. Any systems programs such as the assembler's (XPAL and XMACRO), the Symbolic Editor (XEDIT), or the Binary Loader (XLOAD) can be loaded in less than one minute.

The system software uses a standard DECtape format. There are 128 (200_8) words per block and 1464 (2701_8) blocks, so the user has the remaining 1336 blocks for rapid access storage of his own programs.

The primary advantages for users are:

1. Efficient use of high-speed transfer rates between DECtape and core memory.
2. Symbolic programs may now be stored, edited, and assembled on DECtape, greatly increasing the versatility and flexibility of the PDP-8/I.
3. The computational workload can be more than doubled compared to high-speed paper tape systems.

User's programs are written exactly as before for assembly by the PAL or MACRO-8 assemblers. Using the Symbolic Editor, source programs are typed directly on to DECtape. After assembly, fast symbolic debugging can be done with DDT-8 — after loading the program Symbol Table into DDT with the symbolic loader, XSYM.

The Binary Loader (XLOAD) can load the assembled binary program directly from the DECtape for program execution. Source files, symbol tables, and program listings can be stored on DECtape and listed later, if desired. A duplicating program, XDUP, is available for copying programs.

This DECtape system also includes system calls to load any program from DECtape, to update or delete source files, and to restore the system for use by another programmer.

Although the system operates with one DECtape, a two-DECtape configuration is strongly recommended. This will permit duplication of programs and saving of back-up master tapes. In a single DECtape system, if the system library is accidentally clobbered, all is lost and cannot be replaced immediately because there is no means of recovery.

The last group of programs, called DECTOG, is a collection of short routines controlled by the content of the switch register. It provides for the recording

of timing and mark channels and permits block formats to be recorded for any block length. Patterns may be written in these blocks and then read and checked. Writing and reading is done in both directions and checked. Specified areas of tape may be "rocked" for specified periods of time. A given reel of tape may thus be thoroughly checked before it is used for data storage. These programs may also be used for maintenance and checkout purposes.

DATA COMMUNICATION SYSTEMS (TYPE 680)

A data communication system consists of a PDP-8/I computer with a Data Line Interface DL8/I option, a Serial Line Multiplexer Type 685, and other equipment connected to form a message switching system or to form a data link between serial data transmission equipment and a larger computer. As a message switching system, the 680 system transmits and receives data with up to 128 local or distant Teletype units. As a data link, the 680 system is an economical device for buffering, formatting, and transferring information between a computer and Teletype, or other serial processing equipment operating at one or more data speeds. Assuming only minor data handling before transmission to the larger computer, a 680 system can handle up to 128 5-bit Teletype lines at 50 baud. Although the 680 system programming has provision for handling only Teletype lines, programs to pack and unpack massages for other equipment are easily written.

Software for the 680 system is designed to concentrate Teletype data in serial bit format. Although Teletype format is assumed, other data transmission formats that present information in serial format can be used. Subroutines, as presently written, are designed for the 8-bit Teletype code, the 5-bit Teletype code, or a combination of both codes. They also handle mixed speeds on either 8-bit or 5-bit lines with minor changes. Full duplex lines are assumed, but the subroutines operate with half duplex lines, providing the user handles the expected echo.

A Data Communication System Type 680 hardware configuration varies according to the number, type, and distribution of the Teletype units it contains, and upon the use made of the system. Figure 17 shows the basic 680 system configuration, assuming 15 lines: eight local lines, and seven remote lines.

Teletype signals from remote stations are transmitted and received by a Telegraph Level Converter Type 683. Interface for local units is provided by a Teletype Connector Panel Type 682. Teletype signals for each station run from the 682 or 683 to a Serial Line Multiplexer Type 685. A Matricon Patchboard Type 684 also provides manual selection of channel connections between the 683 and 685. The 685 consists of a multiplexer for Teletype lines and a clock that causes a program interrupt at a rate eight times the line baud frequency. Single line connections are made between the 685 and the Data Line interface Type DL8/I, and between the 685 and the normal computer interface. The Type DL8/I option provides an output instruction to transfer Teletype information from the accumulator to the 685 and provides an input instruction to read Teletype information directly into the computer core memory from the 685. All Teletype information transfers occur serially, one bit at a time.

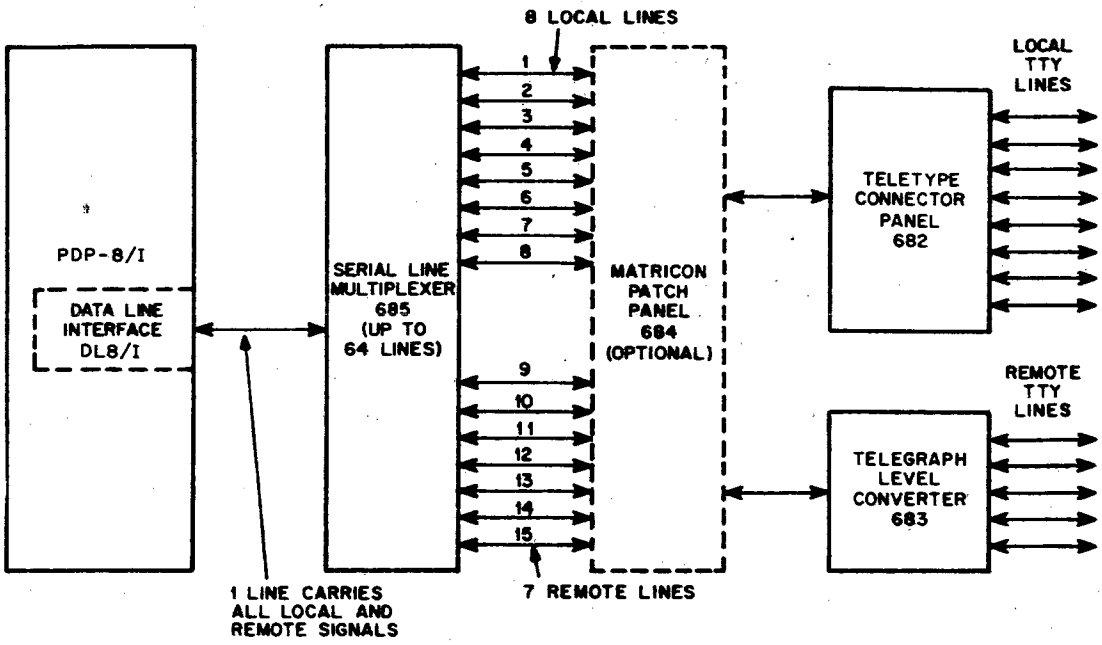


Figure 17. Data Communication System Block Diagram

In any serial data transmission system a word consists of an indication that character transmission is about to start, several bits that specify a character code, and an indication that the character is done. **Figure 17** shows the format of 11-unit code Teletype words as a typical word format used in serial data transmission. In such a system, the device receiving the word signal must determine the bit sampling time so that information is transferred reliably, even though the digital information signal is severely integrated (pulse rise and fall times increased and pulses rounded) due to transmission path impedance, and even though no synchronization is provided between sending and receiving units or between information on different lines. In addition, jitter (time displacement) of the information signal caused by the mechanical contact nature of the equipment originating the signal, must be considered when determining the strobe time.

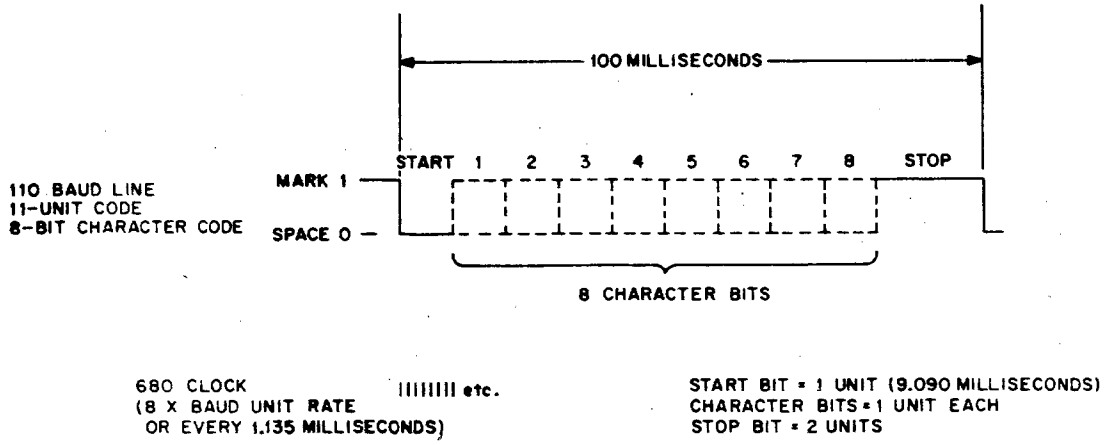


Figure 18. Typical Teletype Line Timing

The Data Communication System Type 680 uses a clock which operates at eight times the bit rate of information on the signal line to determine the sampling time. By counting pulses from this clock, strobe time in receiving and bit timing in transmitting can be controlled within 12.5 percent. Character transmission and reception in the 680 system is controlled by a combination of the hardware and software, providing the most flexibility and economy. This clock in the Serial Line Multiplexer Type 685 requests a program interrupt eight times during each character bit. The program interrupt subroutine counts the clock pulses and strobes a received bit after four clock pulses have occurred since the line became active, thus assuring that the bit is sampled after the middle of the pulse and within 12.5 percent of the center of the pulse. In like manner, clock pulses are counted by the program interrupt subroutine to transmit a bit after eight clock pulses have occurred.

Data Line Interface (DL8/I)

(The control circuitry for this device is located in the PDP-8/I central processor.)

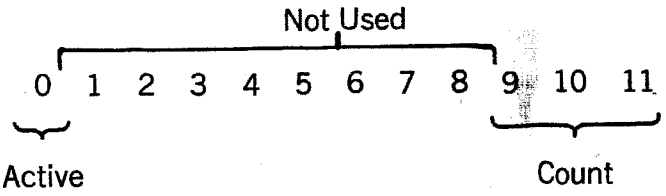
The Type DL8/I option of the PDP-8/I enables use of the computer with a Data Communication System Type 680. The Type DL8/I option controls and executes transmission and reception of Teletype information between the computer and the Type 680 system. Installation of a Type DL8/I option in a PDP-8/I system adds a Teletype IN (TTI) and a Teletype OUT (TTO) instruction to the instruction repertoire, and adds two major states to the processor major state generator. The Status (S) and Character (C) states are entered in executing the complex Teletype In instruction.

The TTI and TTO instructions transfer one bit of a Teletype character between the computer and the Serial Line Multiplexer Type 685. These instructions are executed in subroutines entered through the program interrupt subroutine. These subroutines are responsible for determining when a character is completely assembled in the character assembly word (CAW), and for any relocation or translation of assembled characters. Characters are always assembled so that the last bit transmitted shifts into the most significant bit of the CAW and preceding bits are loaded into less significant bits of the CAW, regardless of the Teletype code or transmission path being used.

Teletype In is a complex memory reference instruction which deals with the incoming Teletype line and with two core memory locations. The two locations addressed by the TTI instruction are the next two successive locations following it. These locations contain a line status word (LSW) and a character assembly word (CAW), respectively, so the following sequence is established:

Address	Content
Y	TTI
Y + 1	LSW
Y + 2	CAW

Bits in the LSW are assigned to record the active/inactive status of the line and serve as a real time clock which determines when line sampling should take place. The format of the LSW is:



The CAW stores partially assembled characters. Individual bits on the incoming line enter the most significant bit (bit 0) and are shifted towards the less significant bit (to the right) during the assembly process.
The TTI instruction is normally executed following a program interrupt caused by the clock in the multiplexer. Figure 19 shows the flow diagram of the TTI instruction.

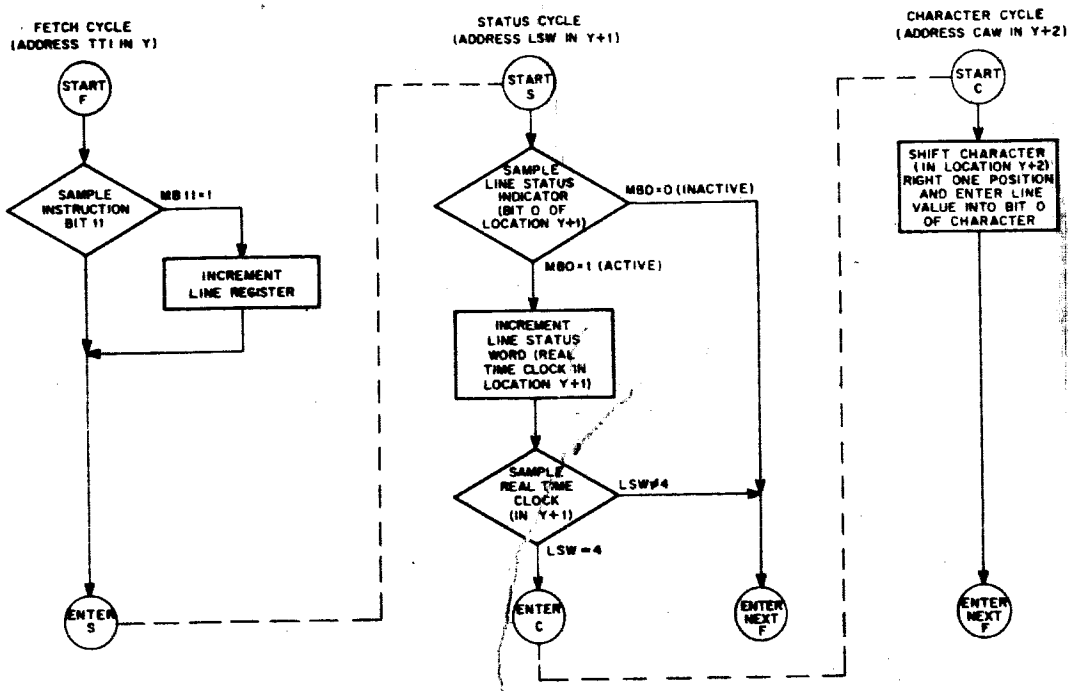


Figure 19. Teletype In Instruction Flow Diagram

Execution of the TTI instruction causes the next location in memory to be read and examined. This location contains the LSW. The first bit examined is the active bit (bit 0). If bit 0 contains a 0, indicating that the line was inactive when last tested, the current content of the line will be set into the active bit. That is, if a start bit is currently being received, the active indicator bit will be set to 1. If no start bit is being received it remains a 0. In either event the character assembly word is skipped over and the next instruction will be executed. Should examination of the active bit indicate that the line is already active, the count portion of the LSW is incremented by one and, unless the resulting count equals 4, the CAW is skipped. When the count becomes equal to 4, indicating that four clock interrupts have been received since the line first became active or that 4/8 of a bit time has elapsed so the center of the bit has been reached and it should be sampled. Thus the character assembly word is read, its content is shifted right one position, and the bit presently being received on the line is set into the leftmost position of the CAW. After the first bit has been received eight clock periods occur before the count is again equal to 4 and thus each bit in the serial train is sampled within 12.5% of the center of the bit.

The Teletype Out instruction affects only the content of the accumulator and the outgoing line. It is executed during a single memory cycle. It shifts the content of the accumulator one position to the right and transmits the least significant bit on the outgoing line. Since a bit is transmitted every time this instruction is executed it should be programmed to occur only after eight clock interrupts have been received since the last output.

These instructions are used only in subroutines and are not used in the main program. The following explanation of their use is for description only.

The Teletype In command brings the bits coming over the line into memory and assembles the bits into one Teletype character. The TTI command uses three memory locations as follows:

```
TTI
0       /status and counter word (LSW)
2000    /character assembly word (for 8-bit code) (CAW)
```

The program then returns to TTI + 3

The character assembly word is preset so that 1 appears in bit 11 when the entire character, including one stop bit, has been shifted in. The subroutines, finding a 1 in bit 11, assume that an entire character has been read, place the character in its own internal buffer together with the line number it came from, and reinitializes the TTI command by resetting the line status word to 0 and the character assembly word to the proper number. At each clock pulse the program only checks 1/8th of the lines (1/4 for 5-bit codes) for completion.

Unlike the TTO command, the TTI command is executed for all lines at each clock-produced program interrupt. However, once the incoming character is started (i.e., bit 0 of the LSW = 1) the first bit (the start code) is read at the fourth pulse and each succeeding bit is read at the eighth pulse thereafter, thus guaranteeing that the bit is read at the optimum time.

The Teletype Out command shifts the content of the accumulator right one position, sends the previous content of bit 11 to the Teletype line specified by the line select register, and brings a 0 into bit 0 of the accumulator. The program sequence to transmit a word from core memory to a Teletype unit might be as follows:

```
TAD   CHAR    /GET CHARACTER TO TRANSMIT
TTO           /SHIFT AND TRANSMIT ONE BIT
DCA   CHAR    /SAVE REMAINDER OF CHARACTER
```

This sequence assumes that the line select register has been loaded with the correct line number using commands for the Serial Line Multiplexer Type 685.

Serial Line Multiplexer Type 685

The 685 is simply a switch which allows the DL8/I to be connected to any one of 64 Teletype lines. To select a line, the accumulator is set to the number of the desired line, and its content is then transferred into the line select register of the 685 by an IOT command. The line select register (LSR) may be loaded at any time with a program-selected address, or can be incremented by a command which may be microprogrammed with the TTI or TTO instructions to scan all lines in numbered sequence. Incrementing is used for high-speed sequential scans. This unit also contains a flip-flop for each

outgoing line. This flip-flop is set or cleared by a TTO instruction and holds the line in the proper state until the next TTO instruction is executed.

INSTRUCTIONS

All instructions for the 680 system contain an operation code of 6, indicating that they are IOT commands. Commands which are associated with the 681 transfer one bit of a character with the computer and have a select code of 40. These commands are functionally memory reference instructions used to perform an input/output transfer operation. Commands associated with the line select register of the 685 use select codes 40 and 41. Commands associated with the clocks of the 685 use select code 42 through 45. All commands using select codes of 41 and 42 use the IOP pulses and are true IOT instructions. The instructions for the 680 system are:

Teletype Increment (TTINCR)

Octal Code: 6401
Event Time: Not applicable in the normal sense of IOT event times. However it can be considered event time 1, since it is executed before all other operations in the TTI or TTO commands with which it can be combined.
Indicators: IOT, FETCH

Execution Time: 1.5 microseconds when performed individually, or equal to the execution time of other commands when microprogrammed.

Operation: The content of the line select register (LSR) in the Serial Line Multiplexer is incremented by one to address the next sequentially numbered line unit. This operation occurs at T1 time of the Fetch cycle.
Symbol: LSR + 1 = > LSR

Teletype In (TTI)

Octal Code: 6402
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 3.0 or 4.5 microseconds
Operation: Three core memory locations are required by the TTI instruction. The first location contains the TTI instruction, and the two succeeding locations contain a line status word (LSW) and a character assembly word (CAW), respectively. Bit 0 of the LSW records the active/inactive status of the selected Teletype line, and bits 9 through 11 of the LSW serve as a real time clock to determine the bit assembly time for the CAW. Both of these words should be cleared prior to the first use of the TTI instruction in a subroutine. The TTI instruction checks the status of the selected line and the number in the real time clock. If the line is active and the clock indicates the center of a bit has passed, one bit of the Teletype line is shifted into the CAW.

The TTI instruction is executed in two or three computer cycles. The first cycle is in the Fetch state to read the instruction from core memory and to establish the next sequential core memory location as the address to be read during the next cycle. By placing a 1 in bit 11, this instruction can be microprogrammed to increment the content of the flip-flop line register of the Serial Line Multiplexer Type 685 during the Fetch cycle.

The second cycle is a Status state in which the LSW is read, the active/inactive status of the line is checked, the timing of the current bit is checked, and

(based on these conditions) the inactive status of the line is recorded in MB0 and the program advances to the next instruction, the real time clock count is incremented in the LSW and the program advances to the next instruction, or the real time clock count is incremented and the third cycle is initiated.

The active/inactive status of the Teletype line is checked by sampling the condition of bit 0 of the LSW. If MB0(0), indicating that the line is inactive (not transmitting a character) the LSW is shifted one position to the right in the MB, and the complement of the Teletype line is set into MB0. Therefore, if the line is now active, a 1 is set into MB0 and will be read during the Status cycle of the next TTI instruction. The program count is then incremented by one to skip over the CAW, the LSW is restored to core memory, the MB is cleared, and (providing no break request had been received) the Fetch state is entered to fetch the next instruction.

If the MB0(1) at the beginning of the Status cycle, the LSW is incremented by one to advance the real time clock and the LSW number is sampled. If LSW $\neq$ 3 it is too early to sample the active line so the program count is incremented to skip over the CAW, the LSW is restored to core memory, the MB is cleared, and the program advances to the Fetch state for the next instruction. If LSW = 4 after incrementation, the LSW is rewritten in memory and the major state generator (MSG) is set to the Character state to strobe the line into the CAW during the next cycle.

The third cycle is a Character state in which the CAW is read into the MB from core memory, the character is shifted right one position with the line bit being shifted into MB0, then the CAW is rewritten in memory. The program then advances to the Fetch state for the next instruction.
Symbol: Status state

If MB0(0), then line shifted into LSW and F = > MSG for next instruction.
If MB0(1), and MB $\neq$ 3, then LSW + 1 = > LSW and F = > MSG for next instruction.

If MB0(1) and MB = 3, then LSW + 1 = > LSW and C = > MSG to continue TTI instruction in next cycle.

Character state
Line shifted into CAW and F = > MSG for next instruction.

Teletype Out (TTO)

Octal Code: 6404
Event Time: Not applicable
Indicators: IOT, FETCH
Execution Time: 1.5 microseconds
Operation: This instruction must be preceded by a command sequence (such as CLA and TAD) that loads the AC with the character to be (or being) transferred to the external Teletype equipment. The TTO instruction clears the L, shifts the content of the AC and the L one position to the right, then transfers the bit contained in AC11 to the selected Teletype line.
Symbol:

0 = > L
L = > AC0 and ACj = > ACj + 1, then
AC11 = > Selected Line

Clear Line Select Register (TTCL)

Octal Code: 6411
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The line select register is cleared, so line 0 is addressed.
Symbol: 0 = > LSR

Load Line Select Register (TTSL)

Octal Code: 6412
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The line select register is set by an OR transfer from the content of bits 5 through 11 of the accumulator, then the accumulator is cleared.
Symbol: AC5-11 $\vee$ LSR = > LSR, then
0 = > AC

Read Line Select Register (TTRL)

Octal Code: 6414
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the line select register is loaded into bits 5 through 11 of the accumulator by an OR transfer.
Symbol: LSR V AC5-11 = > AC5-11

Skip on Clock 1 Flag (TTSKP)

Octal Code: 6421
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of clock 1 flag of the Serial Line Multiplexer is sampled, and if it contains a 1 (indicating that a clock pulse has occurred and the flag has been enabled to request a program interrupt) the content of the program counter is incremented by 1 to skip the next sequential instruction. If the skip occurs, clock 1 caused a program interrupt if the interrupt system was enabled when the clock pulse occurred.
Symbol: If Clock 1 Flag = 1, then PC + 1 = > PC

Turn On Clock 1 (TTXON)

Octal Code: 6424
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The CLOCK 1 ENABLE flip-flop is set and the clock 1 flag is cleared. When the CLOCK 1 ENABLE flip-flop is set the next clock pulse sets the clock 1 flag and requests a program interrupt.
Symbol: 1 = > Clock 1 Enable
0 = > Clock 1 Flag

Turn Off Clock 1 (TTXOFF)

Octal Code: 6422
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds

Operation: The CLOCK 1 ENABLE flip-flop is cleared and the clock 1 flag is cleared. When the CLOCK 1 ENABLE flip-flop is cleared the clock 1 flag can not be set by the clock, and can not request a program interrupt or be skipped upon. The clock is unaffected and continues to run, but all operations caused by clock pulses are disabled.

Symbol: 0 = > Clock 1 Enable
0 = > Clock 1 Flag

When the system handles multiple-baud frequencies additional clocks and instructions are provided. Instructions similar to TTSKP, TTXON, and TTXOF use select code 43 for clock 2, select code 44 for clock 3, and select code 45 for clock 4.

Software

Subroutines for the 680 system, as presently coded, occupy 400_8 core memory locations plus locations for internal buffering of the input and output characters and for the TTI instructions. In addition, autoindex registers and core memory locations in page 0 are required as specified in the following list:

8-Bit	5-Bit	5-Bit (2nd speed)	Meaning
TT8BGN	TT5BGN	TT4BGN	Beginning of subroutine
T8AX1	T5AX1	T4AX1	Autoindex register
T8AX2	T5AX2	T4AX2	Autoindex register
T8AX3			
T8AX3	T5AX3	T4AX3	Autoindex register
	T5AX4	T4AX4	Autoindex register (5-bit only)
TT8PG0	TT5PG0	TT4PG0	Start of area in page 0
T8O8F2	T5OBF2	T4OBF2	Start of 2nd output buffer (length = N)
T81BF	T51BF	T51BF	Start of input buffer (length = 2N)
T81N	T51N	T51N	Start of TT1 area (length = 3N + 1)
TTCHAR	TTCHAR	TTCHAR	Character area (appears only once)

The total amount of core memory used by 680 subroutines, including the tags and autoindex registers in page 0, is as follows:

422_8 + 7N (for 8-bit) or 438_8 + 7N (for 5-bit)

where N is the number of lines specified to the subroutines. Within limits, the programs can be stored anywhere in the PDP-8/I core memory.

If the 5-bit subroutines are being used all of the tags mentioned should substitute 5s for 8s shown. If both 8-bit and 5-bit systems are being used, both sets of subroutines are necessary and all tags and memory requirements must be duplicated for the second system. At present, coding is available for a single 8-bit system and for two different 5-bit systems to allow the programmer to assemble all of the necessary components with a main program at one time.

Percentages of machine time used in the average case for various types of systems are presented in the following list. Any additional features which may be required for the Teletype handling must be added to these times. The formulas for calculating these times are included so that times for systems with an intermediate number of lines or with combinations of lines can be calculated. For combined systems, add the percentages for each component.

Number of lines	8-Bit 110 Baud*	5-Bit 50 Baud**	5-Bit 75 Baud***
32	34.1%	20.0%	30.0%
64	57.7%	35.1%	52.7%
96	81.3%	50.3%	75.5%
128	104.9%	65.5%	98.3%

*Formula Used: Where N = the number of lines, the 8-bit subroutines require an average of 8.38N + 119.5 microseconds.

**Formula Used: Where N = the number of lines, the 5-bit subroutines require an average time of 11.85N +120 microseconds. Clock flags (at 50 baud) occur every 2500 microseconds.

***Formula Used: The percentages for 75 baud are merely 1.5 x 50 baud rate. Clock flags occur every 1667 microseconds.

For further information, refer to DEC Program Library documents DIGITAL-8-35-S-A and DIGITAL-8-35-S-B.

GENERAL PURPOSE MULTIPLEXED ANALOG-TO-DIGITAL CONVERTER SYSTEM (TYPE AF01A)

The Type AF01A General-Purpose Multiplexed Analog-to-Digital Converter combines a versatile, multi-purpose converter with a multiplexer to provide a fast, automatic, multichannel scanning and conversion capability. It is intended for use in systems in which computers sample and process analog data from sensors or other external signal sources at high rates. The Type AF01A option is used with the PDP-8/I to multiplex up to 64-analog signals and to convert the signals to binary numbers. Analog data on each of 64 channels can be accepted and converted into 12-bit digital numbers 420 times per second.*

Switching point accuracy in this instance is 99.975 per cent, with an additional quantization error of half the least significant bit (LSB). If less resolution and accuracy is required, all 64 channels can be scanned and the analog signals on them converted into 6-bit digital numbers 1420 times each second.**

Switching point accuracy in this case is 98.4 per cent, again with the additional quantization error of half the digital value of the LSB.

*Conversion rate = $[(35 + 2)(10^{-6})(64)]^{-1} = 420$ cycles/sec.
**Conversion rate = $[(9 + 2)(10^{-6})(64)]^{-1} = 1420$ cycles/sec.

A/D CONVERTER SPECIFICATIONS

The Type AF01A has a successive approximation converter that measures a 0 to −10 volt analog input signal and provides a binary output indication of the amplitude of the input signal. The characteristics of the A/D converter are as follows:

Accuracy and Conversion Times:	See Table 2 (includes all linearity and temperature errors)
Converter Recovery Time:	Zero

Input and Input Impedance:	0 to −10V at 10 megohms standard. Input scaling may be specified using the amplifier or sample and hold options (see Table 1)
Input Loading:	±1 μA and 125 pf for 0 to −10V input signal.
Output:	Binary number of 6 to 12 bits, with negative numbers represented in 2's complement notation. A 0V input gives a 4000_8; a −5V input a 0000_8 and a −10V (minus 1 LSB*) input gives 3777_8 number.

Provision is made for using the Type A400 Sample and Hold Amplifier (AH02 option) between the multiplexer output and A/D converter input to reduce the effective aperture to less than 150 nsec. The Type A400 may also be used to scale the signal input to accept ±10v, ±5v, or 0 to +10v. The Type A200 amplifier (AH03 option) may be substituted for the Type A400 to accomplish the same signal scaling without reducing the effective aperture. Both the AH02 and AH03 options may be used to obtain high input impedance and small aperture.

MULTIPLEXER SPECIFICATIONS

The multiplexer can include from 1 to 16 Type A121 Switch Modules. Each module contains four single-pole, high speed, insulated gate FET switches with appropriate gating. The Type A121 Switches are arranged as a 64-channel group of series-switch single-pole switches with a separate continuous ground wire for each signal input. The switched signal input wire and the continuous ground for each channel are run as twisted pairs to the input connectors mounted on the rear panel. The continuous grounds for all channels are terminated at the high quality ground of the AF01A System. Specifications (measured at input connector) are as follows:

Input Operating Signal Voltages:	+10V to −10V
Current:	1 mA
On Resistance	450 ohm (max)
Voltage Offset	0
"Off Leakage"	10 nA (max)
Capacitance	10 pf (max)
Speed	
10% Input to within 1 LSB* of output	2 μs
Operate Time	The time required to switch from one channel to another is 2 μs to within 1 LSB* of the final voltage. This time is preset within the control and starts when a set or index command is received.

*LSB—Least Significant Bit.

OPERATION

The Type AF01 System may be operated in either the random or sequential address modes. In the random address mode, the control routes the analog signal from any selected channel to the A/D converter input. In the sequential address mode, the multiplexer control advances its channel address by one each time an index command is received. After indexing through the maximum number of channels implemented, the address is returned to 0. When using sequential operation, the conditioning levels for random addressing are ignored.

The multiplexer switch settling time is preset within the control to initiate the conversion process automatically after a channel has been selected in either the random or sequential address mode. Two separate A/D Convert I/O Transfer Commands may also initiate one or more conversions on a currently selected channel.

A/D conversion times are increased by 2 μsec when multiplexer channels are switched to allow for settling time of the analog signal at the multiplexer output. Conversion times are increased an additional 3 μsec when AH03 is used. These times are added to the conversion times shown in Table 2 under selected channel conversion time, which is the only time required for each successive conversion on a selected channel.

When the Type AH02 Sample and Hold option is required, the multiplexer switch settling time and the sample and hold acquisition time are overlapped. The total conversion and switching time is increased by 10 μsec. (See A400 specifications.)

A/D CONVERTER/MULTIPLEXER CONTROLS

DESIGNATION	FUNCTION
WORD LENGTH:	Rotary switch used to select digital word length or conversion accuracy. Refer to Table 2 for corresponding conversion times.
POWER ON/OFF:	Applies 117 Vac power to internal power supplies.
CLR:	Clear multiplexer channel-address registers; i.e., selects analog channel 0 for conversion.
INDEX:	Advances multiplexer channel-address register by one each time it is depressed, enabling manual addressing of channels (up to 64) in sequential mode. Returns address to zero when maximum value is reached.
ADC:	Starts conversion of the analog voltage on the selected channel to a binary number when depressed.
A/D CONVERTER:	Indicates binary contents of A/D converter register.
MULTIPLEXER:	Indicates binary contents of multiplexer channel-address register.
POWER:	Indicates ON/OFF status.

TABLE 3. INPUT SIGNAL SCALING

CONFIGURATION	GAIN	INPUT SIGNAL	INPUT IMPEDANCE	BINARY OUTPUT	OPTION DESIGNATION
Standard		0	10 meg.	4000_8	STD
		−5	10 meg.	0000_8	
		−10	10 meg.	3777_8	
Sample & Hold	−1	+5	10 K	3777_8	AHO2
	−1	0	10 K	0000_8	
	−1	−5	10 K	4000_8	
Sample & Hold	−½	+10	10 K	3777_8	AHO2
	−½	0	10 K	0000_8	
	−½	−10	10 K	4000_8	
Amplifier	+1	+5	>100 meg.	4000_8	AHO3
	+1	0	>100 meg.	0000_8	
	+1	−5	>100 meg.	3777_8	
Amplifier	+½	+10	>100 meg.	4000_8	AHO3
	+½	0	>100 meg.	0000_8	
	+½	−10	>100 meg.	3777_8	
Amplifier and Sample & Hold	−1 or −½	+5 or 0 or −5	+10 or 0 or −10	>100 meg.	3777_8	AHO3 & AHO2
				>100 meg.	0000_8	
				>100 meg.	4000_8	

Note: Unipolar signals (0 to +5, or 0 to +10v) may also be specified with either the AHO3 or AHO2 option.

TABLE 4. SYSTEM CONVERSION CHARACTERISTICS**

		SELECTED CHANNEL (A/D)	RANDOM OR SEQUENTIAL (MPX. & A/D)	W/AHO3 AMP. (MPX. & A/D)	W/AHO2 SAMPLE & HOLD (MPX. & A/D)	W/AHO3 & AHO2 (MPX. & A/D)
WORD LENGTH (NO. OF BITS)	MAX SWITCHING POINT ERROR*	CONVERSION TIME (μSEC)	CONVERSION TIME (μSEC)**	CONVERSION TIME (μSEC)**	CONVERSION TIME (μSEC)**	CONVERSION TIME (μSEC)**
6	±1.6%	9.0	11.0 (9.5)	14.0 (11.0)	19.0 (14.0)	21.0 (18.0)
7	±0.8%	10.5	12.5 (11.0)	15.5 (12.5)	20.5 (15.5)	22.5 (19.5)
8	±0.4%	12.0	14.0 (12.5)	17.0 (14.0)	22.0 (17.0)	24.0 (21.0)
9	±0.2%	13.5	15.5 (14.0)	18.5 (15.5)	23.5 (18.5)	25.5 (22.5)
10	±0.1%	18.0	20.0 (18.5)	23.0 (20.0)	28.0 (23.0)	30.0 (27.0)
11	±0.05%	25.0	27.0	30.0	35.0	37.0
12	±0.025%	35.0	37.0	40.0	45.0	47.0

*±½ LSB for quantizing error.
**If system is to operate at less than 10 bits continuously, conversion times may be reduced to times shown in parentheses.

Programming

Programmed control of the converter/multiplexer by the PDP-8/I is accomplished with the IOT instructions listed below. PDP-8/I selects the converter/multiplexer with two device selection codes, depending upon whether conversion of multiplexing functions are being selected; 53_8 and 54_8. The converter/multiplexer interprets the device selection code to enable execution of the IOP command pulse generated by the IOT instruction.

Skip on A-D Flag (ADSF)

Octal Code: 6531
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The A-D converter flag is sensed, and if it contains a binary 1 (indicating that the conversion is complete) the content of the PC is incremented by one so that the next instruction is skipped.
Symbol: If A-D Flag = 1, then PC + 1 = > PC

Convert Analog Voltage to Digital Value (ADCV)

Octal Code: 6532
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: This time is a function of the accuracy and word length switch setting as listed in Table 2.
Operation: The A-D converter flag is cleared, the analog input voltage is converted to a digital value, and then the A-D converter flag is set to 1. The number of binary bits in the digital-value word and the accuracy of the word is determined by the preset switch position.
Symbol: 0 = > A-D Flag at start of conversion, then
1 = > A-D Flag when conversion is done.

Read A-D Converter Buffer (ADRB)

Octal Code: 6534
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The converted number contained in the converter buffer (ADCB) is transferred into the AC left justified; unused bits of the AC are left in a clear state, and the A-D converter flag is cleared. This command must be preceded by a CLA instruction.
Symbol: ADCB = > AC
0 = > A-D Converter Flag

Clear Multiplexer Channel (ADCC)

Octal Code: 6541
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The channel address register (CAR) of the multiplexer is cleared in preparation for setting of a new channel.
Symbol: 0 = > CAR

Set Multiplexer Channel (ADSC)

Octal Code: 6542
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The channel address register of the multiplexer is set to the channel specified by bits 6 through 11 of the AC. A maximum of 64 single-ended input channels can be used.
Symbol: AC 6-11 $=>$ CAR

Increment Multiplexer Channel (ADIC)

Octal Code: 6544
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The content of the channel address register of the multiplex is incremented by one. If the maximum address is contained in the register when this command is given, the minimum address (00) is selected.
Symbol: CAR + 1 $=>$ CAR

The converter/multiplexer may be operated by the computer program in either the random or sequential addressing mode. In the random addressing mode, the analog channel is selected arbitrarily by the program for digitizing and the resultant binary word is read into the accumulator. A sample program for the random addressing mode is as follows:

```
TAD ADDR        /YES — GET CHANNEL ADDRESS
ADSC            /AND SEND TO MULTIPLEXER
ADCV            /CONVERT A TO D
ADSF            /SKIP ON A/D DONE FLAG
JMP. —1         /WAIT FOR FLAG
ADRB            /AND READ INTO AC
```

In the sequential address mode, the program advances the multiplexer channel-address register to the next channel each time an analog value is converted and read into the accumulator.

Should the converter/multiplexer be operated in the interrupt mode, the computer will be signaled each time that a binary word is ready, enabling the system to use processor time more efficiently.

AMPLIFIER, SAMPLE AND HOLD OPTIONS FOR AF01A

AH03 AMPLIFIER OPTION

The AH03 consists of a DEC amplifier (part #1505379) mounted on an A990 Amplifier Board with appropriate scaling networks and gain trim and balance potentiometers.

Open loop gain	2×10^6
Rated output voltage	(@ 10 ma) $\pm$11v
Frequency response	
Unity Gain, small signal	10 mc
Full output voltage	300 kc
Slewing rate	30v/μsec
Overload recovery	200 μsec

Input voltage offset	Adjustable to 0
Avg vs temp	20 μv/°C
Vs supply voltage	15 μv/%
Vs time	10 μv/day
Input current offset	±2 na
Avg vs temp	0.4 na/°C
Vs supply voltage	0.15 na/%
Input impedance	
Between inputs	6 meg
Common mode	500 meg
Input voltage	±15v
Max common mode	±10v
Common mode rejection	20,000
Power	
Voltage	±15 to 16v
Current at rated load	35 ma

AH02 SAMPLE AND HOLD OPTION

A400 (standard gain options)	
Acquisition time to 0.01% (full-scale step)	<12 μsec
Aperture time	<150 nsec
Hold inaccuracy (droop)	<1 mv/msec
Temperature coefficient	0.1 mv/msec/°C
Gain (negative)	1.0 0.5
Input range (volts)	±5.0 ±10.0
Impedance	10K 10K
Output voltage	0 to −10v
Impedance	<1.0 ohm

GUARDED SCANNING DIGITAL VOLTMETER (TYPE AF04A)

DESCRIPTION

The Type AF04A is a guarded scanning digital voltmeter system, with wide dynamic range and high common-mode rejection, and fully capable of expansion to 1000 channels. The Type AF04A is used with a PDP-8/I computer to multiplex up to 1000 3-wire analog channels into a 6-decimal-digit (BCD) integrating digital voltmeter. Full scale ranges are from ±10mv to ±300v, with automatic ranging, 300 percent over ranging, and a usable 5 μv resolution. Guarded input construction and active integration assist in attaining an effective common-mode rejection of greater than 140 db at all frequencies. (Normal-mode rejection is infinite at multiples of power line frequency.)

This system is ideally suited for data acquisition or process monitoring where a wide range of signals requires large dynamic range. The 10-mv range has 0.001 percent resolution and, coupled with excellent noise rejection, allows accurate direct measurement of thermocouples, strain gauges, load cells, and other low-level transducers without additional amplification.

The AF04A Voltmeter, operated under program control, is capable of either random channel selection or sequential channel selection. The computer selects either program controlled ranging (for fastest speed) or autoranging, as well as the integration time of the integrating digital voltmeter (IDVM).

The digitized data, as well as the current channel address, is read by the computer in either two or three bytes.

A decimal display of the digitized value, including sign and decimal location, is continuously displayed on the front panel. The current channel number is also displayed. Front-panel controls on the digital voltmeter allow manual setting of all the programmed functions. A front-panel control allows continuous display of the internal secondary standard, which can be prewired to a particular channel for reference checking during normal operation. The AF04A Voltmeter System may be manually controlled, completely independent of the computer.

SPECIFICATIONS

Full scale ±	10mv, 100mv, 1v, 10v, 100v, 300v, and automatic ranging
Over ranging	300% on all but highest range
Resolution	5 μv (usable), 0.1 μv (LSB)
Accuracy (overall worst case with daily calibration at calibration temperature)	±0.004% of reading ±0.01% of full scale±5μv
Stability (RMS full scale and zero drift)	±0.006%/day
Temperature coefficient	
Full scale	±0.003% of reading/°C
Zero	±0.002% of full scale/°C (±0.006% of full scale/°C on 10mv and 1v range)
Line voltage stability	±0.0005%/10% change
Maximum common-mode voltage	±300v from power line ground
Common-mode rejection (166.6 msec integration period and 1000 ohm-source unbalance)	> 140 db at all frequencies
Normal-mode rejection	Infinite at multiples of line frequency
Input impedance	
10, 100, 1000 mv ranges	1000 meg/v
10, 100, 300v ranges	10 meg
Internal secondary standard	

Value	±1.000v
Accuracy	±0.002% traceable to N.B.S.
Stability	±0.005%/month
Temperature coefficient	negligible

SELECTED RESOLUTION

	0.001%		0.01%		0.1%	
DC Voltage Range	Maximum Reading	Resolution	Maximum Reading	Resolution	Maximum Reading	Resolution
10 mv	30.0000 mv	0.1 μv	030.000 mv	1 μv	0030.00 mv	10 μv
100 mv	300.000 mv	1 μv	0300.00 mv	10 μv	00300.0 mv	100 μv
1000 mv	3000.00 mv	10 μv	03000.0 mv	100 μv	003000. mv	1 mv
10v	30.0000v	100 μv	030.000v	1 mv	0030.00v	10 mv
100v	300.000v	1 mv	0300.00v	10 mv	00300.0v	100 mv
1000v*	0300.00v	10 mv	00300.0v	100 mv	000300.v	1v

*1000v range is scanner-limited to 300v peak maximum

SCANNING SPEED

(Programmed Range)

Resolution	Integration Time	Total Time	Scanning Speed
0.1%	1.6 msec	20 msec	50 ch/sec.
0.01%	16.6 msec	40 msec	25 ch/sec.
0.001%	166.6 msec	188 msec	5 ch/sec.

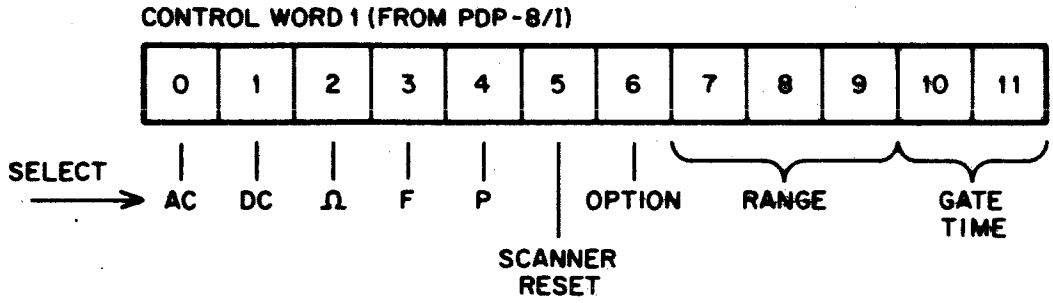
Scanning Speed (Auto Range)—Add 6-36 msec depending on per-channel voltage span.

INSTRUCTIONS

The I/O transfer (IOT) commands associated with the scanning digital voltmeter system are designed to minimize the computer overhead associated with this option while retaining maximum program controlled flexibility. The IOT instructions are:

Select Range and Gate (VSEL)

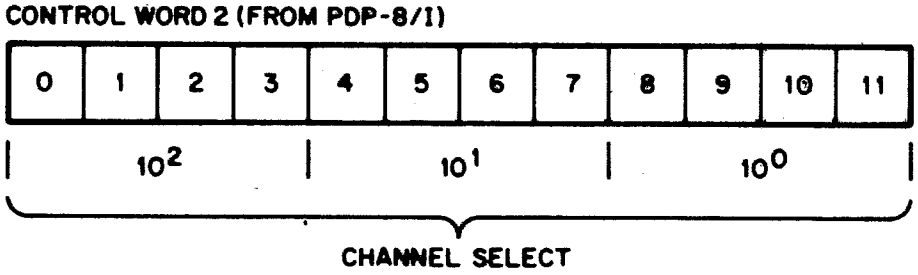
Octal Code: 6542
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of the accumulator are transferred to the AF04A control register as shown below:
Symbol: C(AC) = > C(VCR)



Control Word 1 only used if a range change is required.

Select Channel and Convert (VCNV)

Octal Code: 6541
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of the accumulator are transferred to the AF04A channel address register as shown below. The analog signal on the selected channel is automatically digitized.
Symbol: C(AC) = > C(VAR)



Index Channel and Convert (VINX)

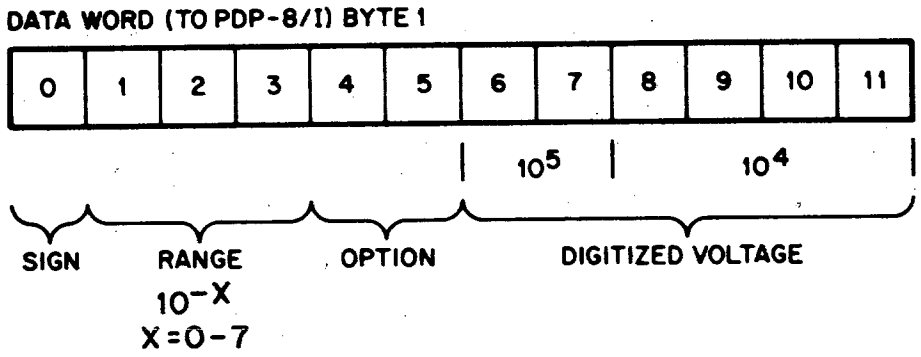
Octal Code: 6544
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The last channel address is incremented by one and the analog signal on the selected channel is automatically digitized. The contents of the control register is unchanged.
Symbol: VAR + 1 = > VAR

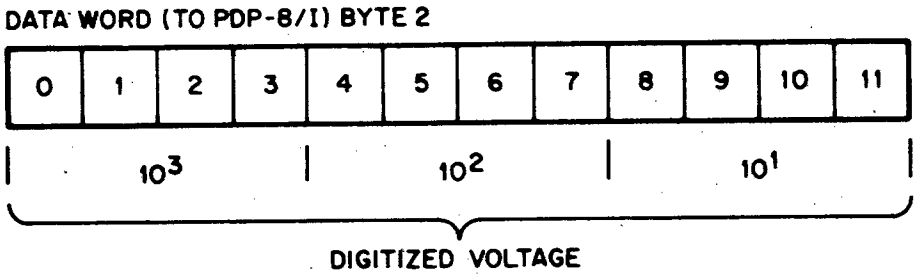
Skip on Data Ready (VSDR)

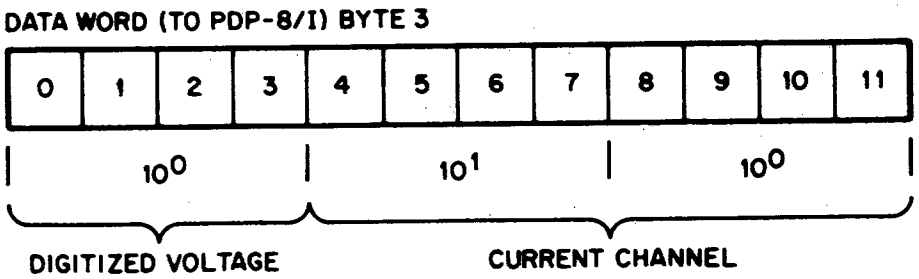
Octal Code: 6531
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: When the scanning voltmeter has selected a channel and digitized the analog signal, a data ready flag is set. This instruction is used to test for the data ready flag.
Symbol: If Flag = 1, then PC + 1 = > PC

Read Data and Clear Flag (VRD)

Octal Code: 6532
Event Time: 2
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The contents of the selected byte of the voltmeter output word is transferred to the accumulator and the data ready flag is cleared. The first data flag after the flag is set, is always byte 1 (see below). Subsequent bytes are program selected using the byte advance command.
Symbol: C(VOR) = > C(AC)







Data word 3 seldom required, all address and digitized data are in 8-4-2-1 BCD format.

Byte Advance (VBA)

Octal Code: 6534
Event Time: 3
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The total data word from the AF04A is 36-bits long. The first data word after the flag is set, is always the twelve most significant bits. The BYTE ADVANCE command requests the next twelve most significant bits. When the data is available, the data ready flag is set again. To select the twelve least significant bits, a second BYTE ADVANCE command is required. When the data is available, the data ready flag is set again.
Symbol: $C(VOR_{0-12}) = > C(VOR_{13-23})$
or $C(VOR_{13-23}) = > C(VOR_{24-35})$

Sample Current Channel (VSCC)

Octal Code: 6571
Event Time: 1
Indicators: IOT, FETCH, PAUSE
Execution Time: 4.25 microseconds
Operation: The analog signal on the current channel is digitized. This command is **not** required except when multiple samples are required on any channel. (Using this command on a preselected channel saves up to 10 milliseconds per sample.)
Symbol: None.

FREQUENCY AND PERIOD MEASUREMENT OPTIONS FOR AF04A

A separate input permits the IDVM to be used as a frequency counter capable of counting to 2mHz with selectable gate times of 1, 10, and 100 milliseconds, providing measurement resolution of 10Hz. Increased accuracy at low frequencies (to 10kHz with automatic 250% overranging) is accomplished with the period-measurement mode. This mode counts an internal frequency source for 1 , 10, or 100 periods of the frequency being measured, thereby providing increased full-scale accuracy. Period readout is in milliseconds.

Frequency and voltage measurements may be made within one scanning cycle by grouping all frequency inputs in one master or slave scanner and all voltage inputs in another master or slave scanner. The output of one scanner may then be connected to the frequency-input connector of the IDVM and the output of the other scanner to the voltage input. One of the optional control word bits is used to program the IDVM for frequency or period measurements.

SPECIFICATIONS

Frequency Measurements

Range: 10Hz to 2mHz

Sensitivity: 100mv rms or −1v pulses, at least 0.3 μsec wide at 50% points. 100v rms maximum working voltage.

Input Impedance: 22k ohms shunted by less than 1000 pf, including internal cabling.

Accuracy: ±1 count + time base accuracy

Time Base: 100 KHz crystal oscillator with initial accuracy of ±0.0005%, long-term stability ±0.0001%/wk; temp. coefficient ±0.0002%/°C.

Period Measurements

Range: 1, 10, and 100 period average. Input frequency from 10Hz to 25kHz sine wave or 0.1 pps. to 25,000 pps.

Sensitivity: 100 mv rms or —1v pulses, at least 0.3 μsec wide at 50% points. 100v rms maximum working voltage.

Input Impedance: 22k ohms shunted by less than 1000pf, including internal cabling.

Accuracy: ±1 count + time base accuracy + trigger error. Trigger error < ± 0.03% for 100mv rms sine wave with 40db signal-to-noise ratio.

Time Base: 100kHz crystal oscillator with initial accuracy of ±0.0005%, long-term stability ±0.0001%/wk; temp. coefficient ±0.0002%/°C.

Selected Resolution	0.001%		0.01%		0.1%	
Function	Maximum Reading	Resolution	Maximum Reading	Resolution	Maximum Reading	Resolution
Frequency	2000.00kHz	10Hz	02000.0kHz	100Hz	002000kHz	1kHz
Period	99.9999msec	0.1μs	999.999msec	1.0μs	9999.99msec	10μs

Additional AF04A Options

Information on the following options may be had from your nearest DIGITAL EQUIPMENT CORPORATION Office:

Frequency (period) measurements.
AC/ohms/DC Converter
Time-of-day clock.
Thumb-wheel data entry panel.
Thermocouple reference junctions.
Extended scanner for more than 1000 channels.
Special cabinet with roll-out drawer chassis accessibility.

CHAPTER 8
PDP-8/I INPUT/OUTPUT FACILITIES

Since the processing power of the computer depends largely upon the range and number of peripheral devices that can be connected to it, the PDP-8/I has been designed to interface readily with a broad variety of external equipment. The following chapters of this handbook define the interface characteristics of the computer to allow the reader to design and implement any electrical interfaces required to connect devices to the PDP-8/I. Chapters 9 and 10 functionally describe the logic circuit elements involved in programmed data transfers and data break transfers, respectively. Chapter 11 gives detailed information on digital logic circuits used for computer interfacing. Chapter 12 describes the design and construction of interface equipment. Chapter 13 describes additional interfacing techniques which demonstrate efficiency and flexibility of the PDP-8/I I/O facility. Chapter 14 lists connection point, module type, and module location, etc., for each interface signal; gives detailed loading and driving characteristics for each module in the computer interface; then presents some general rules and characteristics to be considered in selecting or designing electrical circuits to be connected to the PDP-8/I. Chapter 15 presents information for planning the installation of a basic PDP-8/I and the available standard optional equipment.

The simple I/O technique of the PDP-8/I, the availability of DEC's FLIP CHIP logic circuit modules, and DEC's policy of giving assistance wherever possible allow inexpensive, straightforward device interfaces to be realized. Should questions arise relative to the computer interface characteristics, the design of device interfaces using DEC modules, or installation planning, customers are invited to telephone the main plant in Maynard, Massachusetts, or any of the sales offices. Digital Equipment Corporation makes no representation that the interconnection of its circuit modules in the manner described herein will not infringe on existing or future patent rights. Nor do the descriptions contained herein imply the granting of license to use, manufacture, or sell equipment constructed in accordance therewith.

The basic PDP-8/I contains a processor and core memory composed of Digital's M Series TTL circuit modules. These circuits have an operating temperature exceeding the limits of 32°F to 130°F, so no air-conditioning is required at the computer site. Standard 115V, 60-CPS power operates an internal solid state power supply that produces all required voltages and currents. High-capacity, high-speed I/O capabilities of the PDP-8/I allow it to operate a variety of peripheral devices in addition to the standard Teletype keyboard/printer, tape reader, and tape punch. DEC options, consisting of an interface and normal data processing equipment, are available for connecting into the computer system. These options include a random access disc file,card equipment, line printers, magnetic tape transports, magnetic drums, analog-to-digital converters, CRT displays, and digital plotters. The PDP-8/I system can also accept other types of instruments or hardware devices that have an appropriate interface. Up to 61 devices requiring three programmed command pulses, or up to 183 devices requiring one programmed command pulse can be connected to the computer. One machine using the data break facility can be connected directly to the PDP-8/I or up to seven such machines can be connected through a Data Multiplex Type DMO1. Interfacing of any devices to the computer requires no modifications to the processor and can be achieved in the field.

Control of some kind is needed to determine when an information exchange is to take place between the PDP-8/I and peripheral equipment and to indicate the location(s) in the computer memory which will accept or yield the data. Either the computer program or the device external to the computer can exercise this control. Transfers controlled by the computer, hence under control of its stored program, are called programmed data transfers. Transfers made at times controlled by the external devices through the data break facility are called data break transfers.

Programmed Data Transfers

The majority of I/O transfers occur under control of the computer program. To transfer and store information under program control requires about six times as much computer time as under data break control. In terms of real time, the duration of a programmed transfer is rather small, due to the high speed of the computer, and is well beyond that required for laboratory or process control instrumentation.

To realize full benefit of the built-in control features of the PDP-8/I programmed I/O transfers should be used in most cases. Controls for devices using programmed data transfers are usually simpler and less expensive than controls for devices using data break transfers. Using programmed data transfer facilities, simultaneous operation of devices is limited only by the relative speed of the computer with respect to the device speeds, and the search time required to determine the device requiring service. Analog-to-digital converters, digital-to-analog converters, digital plotters, line printers, message switching equipment, and relay control systems typify equipment using only programmed data transfers.

Data Break Transfers

Devices which operate at very high speed or which require very rapid response from the computer use the data break facilities. Use of these facilities permits an external device, almost arbitrarily, to insert or extract words from the computer core memory, bypassing all program control logic. Because the computer program has no cognizance of transfers made in this manner, programmed checks of input data are made prior to use of information received in this manner. The data break is particularly well-suited for devices that transfer large amounts of data in block form, e.g., random access disc file, high-speed magnetic tape systems, high-speed drum memories, or CRT display systems containing memory elements.

Logic Symbols

Figure 20 defines the symbols used in the following chapters of this handbook to express signals and digital logic circuits.

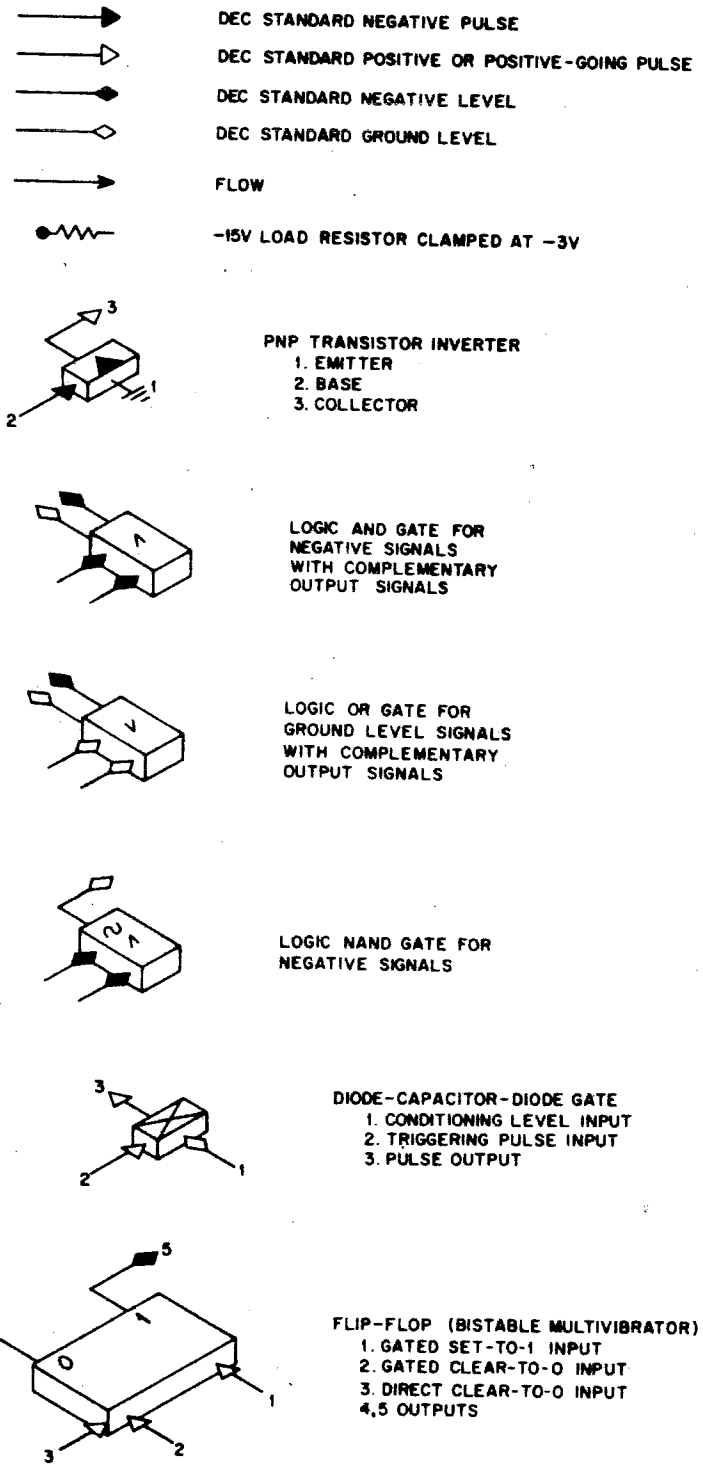
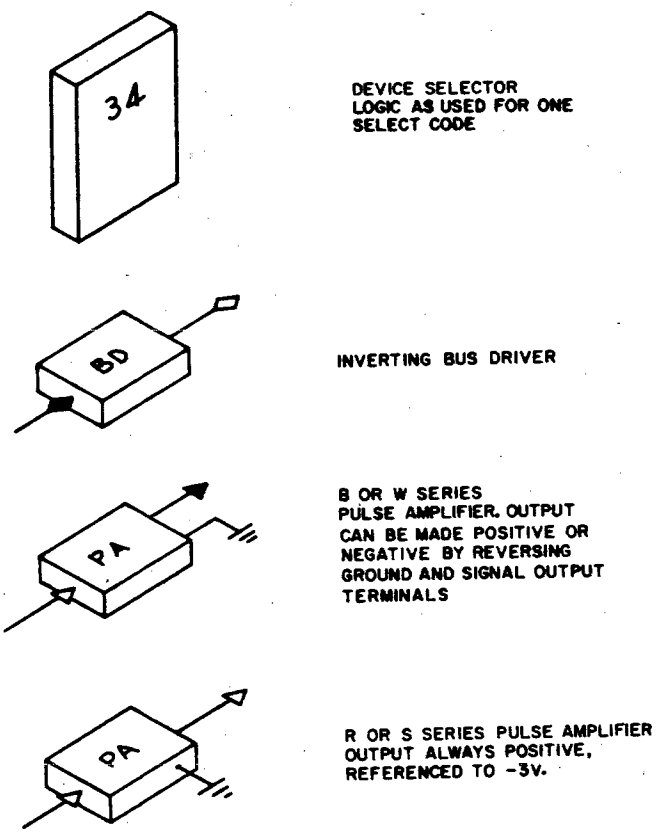


Figure 20. Logic Symbols



The PDP-8/I uses TTL logic internally. In order to discuss some of the internal logic pertaining to interfacing, it is necessary to understand the TTL symbology used in the 8/I. The figures are as follows:

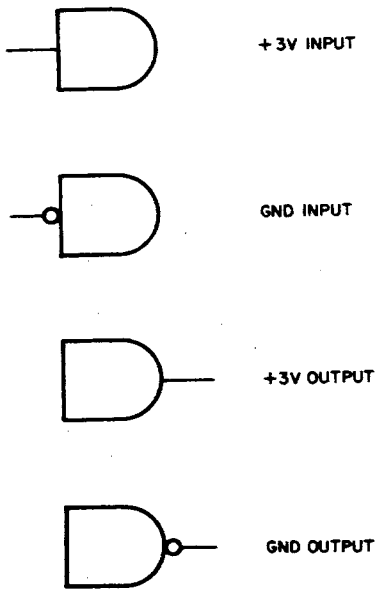


Figure 20. Logic Symbols (continued)

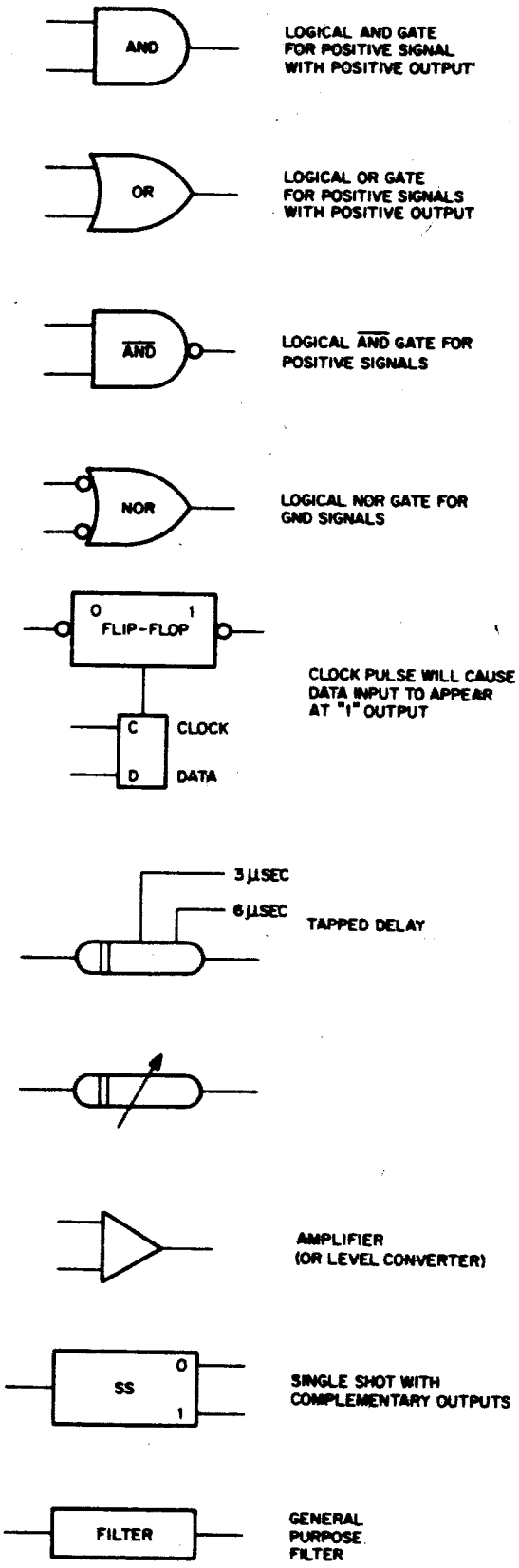


Figure 20. Logic Symbols (continued)

CHAPTER 9
PROGRAMMED DATA TRANSFERS

The majority of I/O transfers take place under control of the PDP-8/I program, taking advantage of control elements built into the computer. Although programmed transfers take more computer and actual time than do data break transfers, the timing discrepancy is insignificant, considering the high speed of the computer with respect to most peripheral devices. The maximum data transfer rate for programmed operations of 12-bit words is 148 kc when no status checking, end transfer check, etc., is done. This speed is well beyond the normal rate required for typical laboratory or process control instrumentation.

The PDP-8/I is a parallel-transfer machine that distributes and collects data in bytes of up to twelve bits. All programmed data transfers take place through the accumulator, the 12-bit arithmetic register of the computer. The computer program controls the loading of information into the accumulator (AC) for an output transfer, and for storing information in core memory from the AC for an input transfer. Output information in the AC is power amplified and supplied to the interface connectors for bussed connection to many peripheral devices. Then the program-selected device can sample these signal lines to strobe AC information into a control or information register. Input data arrives at the AC as pulses received at the interface connectors from bussed outputs of many devices. Gating circuits of the program-selected device produce these pulses. Command pulses generated by the device flow to the input/output skip facility (IOS) to sample the condition of I/O device flags. The IOS allows branching of the program based upon the condition or availability of peripheral equipment, effectively making programmed decisions to continue the current program or jump to another part of the program, such as a subroutine that services an I/O device.

The bussed system of input/output data transfers imposes the following requirements on peripheral equipment.

a. The ability of each device to sample the select code generated by the computer during IOT instructions and, when selected, to be capable of producing sequential IOT command pulses in accordance with computer-generated IOP pulses. Circuits which perform these functions in the peripheral device are called the device selector (DS).

b. Each device receiving output data from the computer must contain gating circuits at the input of a receiving register capable of strobing the AC signal information into the register when triggered by a command pulse from the DS.

c. Each device which supplies input data to the computer must contain gating circuits at the output of the transmitting register capable of sampling the information in the output register and supplying a pulse to the computer input bus when triggered by a command pulse from the DS.

d. Each device should contain a busy/done flag (flip-flop) and gating circuits which can pulse the computer input/output skip bus upon command from the DS when the flag is set in the binary 1 state to indicate that the device is ready to transfer another byte of information.

Figure 21 shows the information flow within the computer which effects a programmed data transfer with input/output equipment. All instructions stored in core memory as a program sequence are read into the memory buffer register (MB) for execution. The transfer of the operation code in the three most significant bits (bits 0, 1, and 2) of the instruction into the instruction register (IR) takes place and is decoded to produce appropriate control signals. The computer, upon recognition of the operation code as an IOT instruction, enters a 4.25 μsec expanded computer cycle and enables the IOP generator to produce time sequenced IOP pulses as determined by the three least significant bits of the instruction (bits 9, 10, and 11 in the MB). These IOP pulses and the buffered output of the select code from bits 3-8 of the instruction word in the MB are bussed to device selectors in all peripheral equipment. Figure 22 indicates the timing of programmed data transfers and Figure 23 shows the decoding of the IOT instruction.

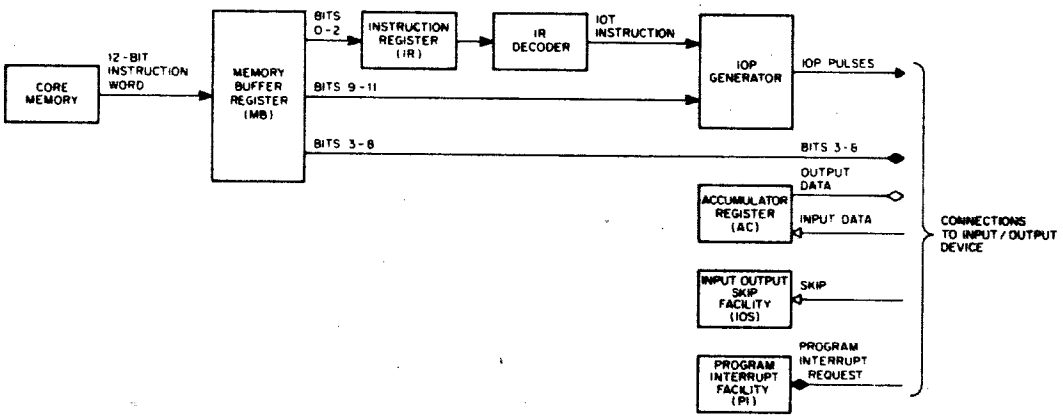


Figure 21. Programmed Data Transfer Interface Block Diagram

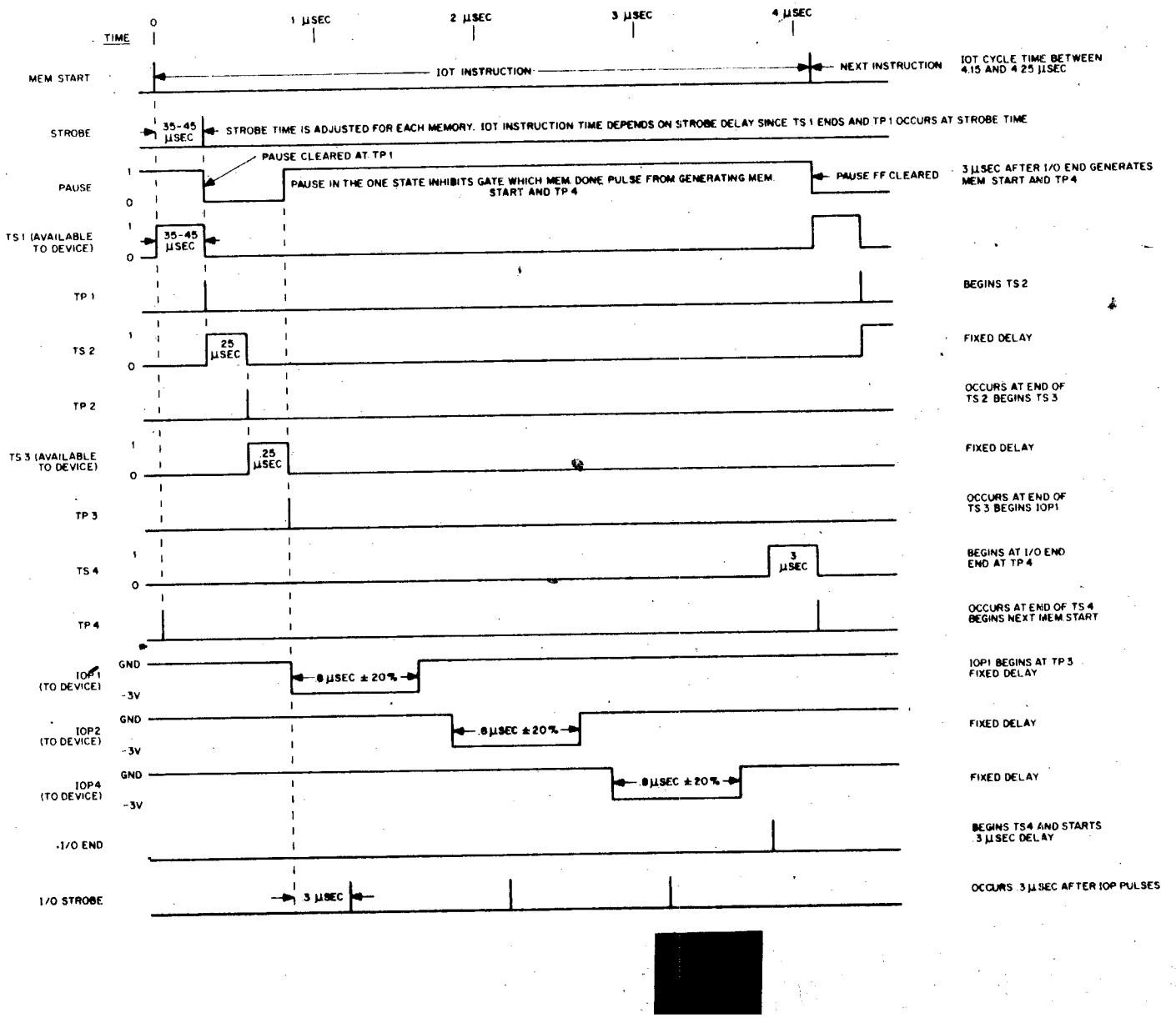


Figure 22. Programmed Data Transfer Timing

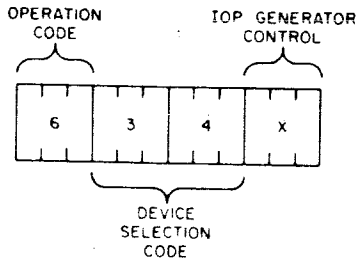


Figure 23. Typical IOT Instruction Decoding

Devices which require immediate service from the computer program, or which take an exorbitant amount of computer time to discontinue the main program until transfer needs are met, can use the program interrupt (PI) facility. In this mode of operation, the computer can initiate operation of I/O equipment and continue the main program until the device requests servicing. A signal input to the PI requesting a program interrupt causes storing of the conditions of the main program and initiates a subroutine to service the device. At the conclusion of this subroutine, the main program is reinstated until another interrupt request occurs.

Timing and IOP Generator

When the IR decoder detects an operation code of 6_8, it identifies an IOT instruction and the computer generates a slow cycle pulse. The Slow Cycle signal ANDS with TP3 to generate I/O Start and consequently generates 1 → Pause pulse. The computer enters the Pause state thereby disabling the normal timing generator of the processor. The logic circuits of the IOP generator are shown in Figure 24 to consist of three similar channels each consisting of a delay, gated flip-flop and output level converter. The I/O Start signal initiates the first delay and the operation of the other two channels is triggered by the pulse output of the delay in the previous channel. Series connection of the delays produces sequential operations of the three channels. The pulse output of the third channel delay restarts the normal timing generators of the processor. (IOT instructions associated with enabling and disabling the program interrupt facility and those for the Memory Extension Control Type MC-8/I, and the Data Line Interface Type DL-8/I inhibit generation of the Slow Cycle signal; thereby, preventing the computer to enter the Pause state. In these commands the IOP generator is inhibited so the normal timing pulses of the processor and special device selectors execute these instructions.)

Note: All cycle times of the PDP-8/I have a tolerance of ±20%.

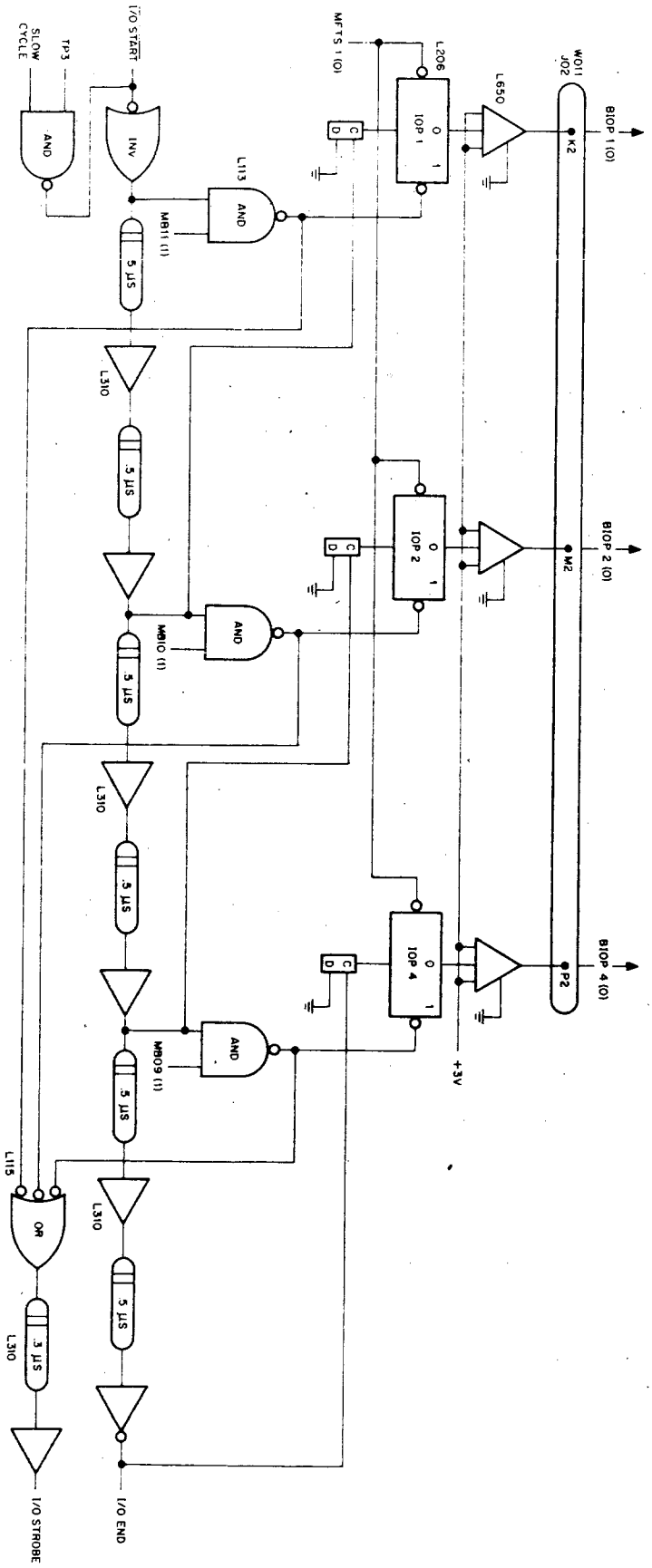


Figure 24. IOP Generator Logic

The AND gates associated with flip flops IOP 1, IOP 2, and IOP 4 sample the contents of MB 11 (1), MB 10 (1), and MB09 (1) respectively. The delay chain is initiated by I/O Start which occurs when its preceding AND gate is enabled by the Slow Cycle original and strobed by TP3. I/O Start sets the Pause flip flop which disables the normal cycling of the computer. The I/O Start pulse will set IOP 1 flip flop to a one if MB 11 (1) has enabled the input and gate and also starts the first delay chain. At the end of a microsecond delay IOP 1 flip flop is reset to zero and IOP 2 flip flop is set to a one if MB 10 (1) has enabled its input AND gate. This procedure continues until all three MB bits have been sampled and the appropriate IOP pulses have been generated. The 1 microsecond output pulses from the IOP flip flops are converted to DEC levels of zero and −3V. A Standard 8/I IOP pulse is a pulse which goes from ground to −3V for .8 microsecond and returns to ground. I/O STROBE PULSES are generated .3 microseconds after the beginning of each IOP pulse and are used, by the processor, for internally strobing the content input buss lines after they have been enabled by IOT pulses. The instruction bit that enables or disables generation of each IOP pulse, the corresponding number of the IOT pulses produced in the DS from the IOP pulse, and the event time for each IOP pulse is:

Instruction Bit	IOP Pulse	IOT Pulse	Event Time	Used Primarily For
11	IOP 1	IOT 1	1	Sampling Flags, Skipping.
10	IOP 2	IOT 2	2	Clearing Flags, Clearing AC.
9	IOP 4	IOT 4	3	Reading Buffers, Loading Buffers and Clearing Buffers.

Device Selector (DS)

Bits 3 through 8 of an IOT instruction serve as a device or subdevice select code. Bus drivers in the processor buffer both the binary 1 and 0 output signals of MB3-8 and distribute them to the interface connectors for bussed connection to all device selectors. Each DS is assigned a select code and is enabled only when the assigned code is present in the MB. When enabled, a DS regenerates IOP pulses as IOT command pulses and transmits these pulses to skip, input, or output gates within the device and/or to the processor to clear the AC.

Each group of three command pulses requires a separate DS channel (W103 module), and each DS channel requires a different select code (or I/O device address). One I/O device can, therefore, use several DS channels. Note that the processor produces the pulses identified as IOP 1, IOP 2, and IOP 4 and supplies them to all device selectors. The device selector produces pulses IOT 1, IOT 2, and IOT 4 which initiate a transfer or effect some control. Figure 25 shows generation of command pulses by several DC channels.

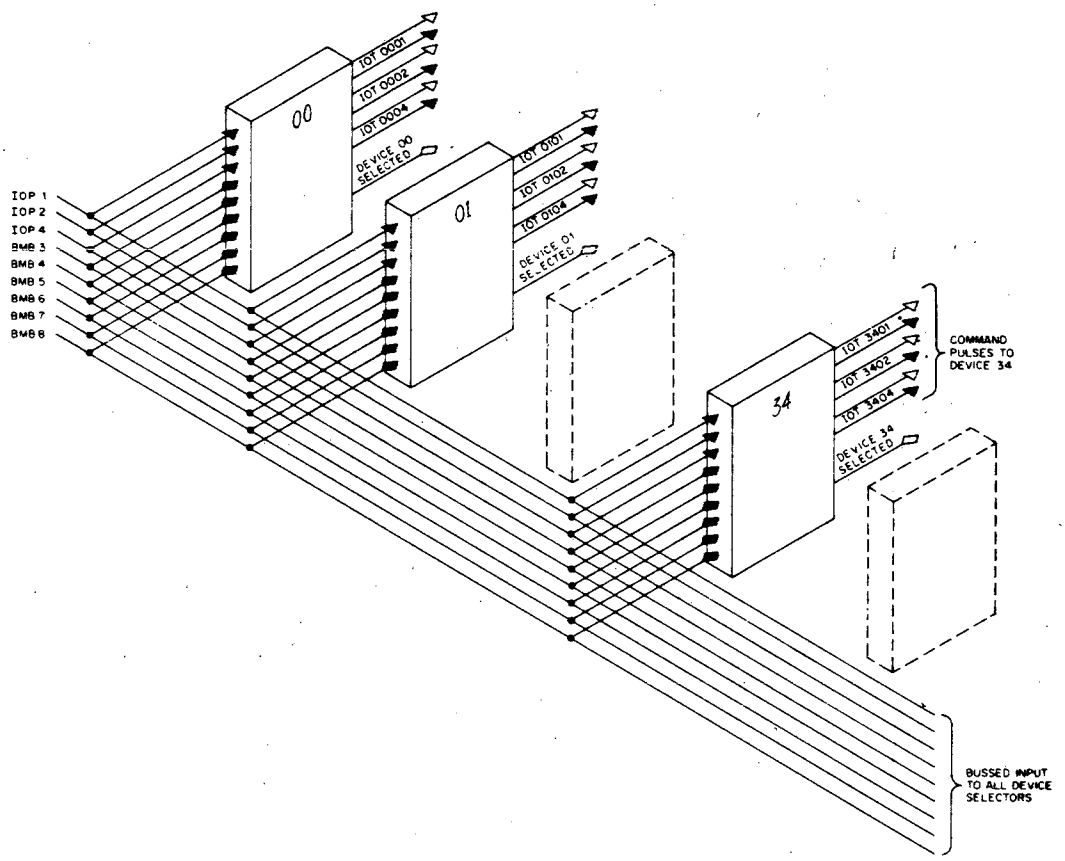


Figure 25. Generation of IOT Command Pulses by Device Selectors

The logical representation for a typical channel of the DS, using channel 34, is shown in Figure 26. A 6-input NAND gate wired to receive the appropriate signal outputs from MB3-8 for select code 34 activates the channel. In the DS module, the NAND gate contains 14 diode input terminals; 12 of these connect to the complementary outputs of MB3-8, and 2 are open to receive subdevice or control condition signals as needed. Either the 1 or the 0 signal from each MB bit is disconnected by removing the appropriate diode from the NAND gate when establishing the select code. The ground level output of the NAND gate indicates when the IOT instruction selects the device, and can therefore enable circuit operations within the device. This output also enables three gating inverters, allowing them to trigger a pulse amplifier if an IOP pulse occurs. The positive output from each pulse amplifier is an IOT command pulse identified by the select code and the number of the initiating IOP pulse. Three inverters receive the positive IOT pulses to produce complementary IOT output pulses. A pulse amplifier module can be connected in each channel of the DS to provide greater output drive or to produce pulses of a specific duration required by the selected device.

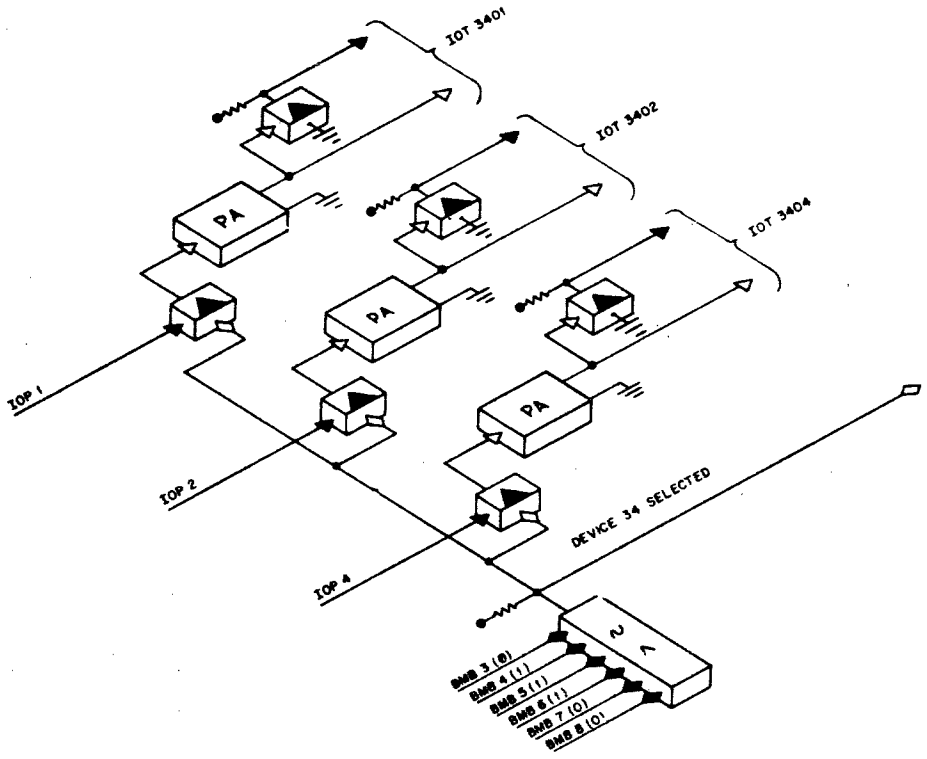


Figure 26. Typical Device Selector (Device 34)

Input/Output Skip (IOS)

Generation of an IOT pulse can be used to test the condition or status of a device flag, and to continue to or skip the next sequential instruction based upon the results of this test. This operation is performed by a 2-input AND gate in the device connected as shown in Figure 27. One input of the skip gate receives the status level (flag output signal), the second input receives an IOT pulse, and the output drives the computer IOS bus to ground when the skip conditions are fulfilled. When the IOS bus is driven to ground, the content of the program counter is incremented by 1 to advance the program count without executing the instruction at the current program count. In this manner an IOT instruction can check the status of an I/O device flag and skip the next instruction if the device requires servicing. Programmed testing in this manner allows the routine to jump out of sequence to a subroutine that services the device tested.

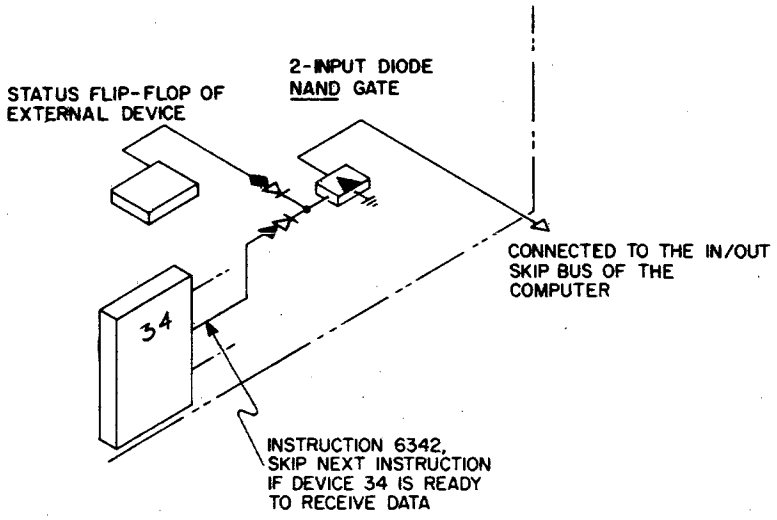


Figure 27. Use of IOS to Test the Status of an External Device

Assuming that a device is already operating, a possible program sequence to test its availability follows:

Address	Instruction	Remarks
.		
.		
.		
.		
100,	6342	/SKIP IF DEVICE 34 IS READY
101,	5100	/JUMP .−1
102,	5XXX	/ENTER SERVICE ROUTINE FOR /DEVICE 34
.		
.		
.		
.		

When the program reaches address 100, it executes an instruction skip with 6342. The skip occurs only if device 34 is ready when the IOT 6342 command is given. If device 34 is not ready, the flag signal disqualifies the skip gate, and the Skip pulse does not occur. Therefore, the program continues to the next instruction which is a jump back to the skip instruction. In this example, the program stays in this waiting loop until the device is ready to transfer data, at which time the skip gate in the device is enabled and the Skip pulse is sent to the computer IOS facility. When the skip occurs, the instruction in location 102 transfers program control to a subroutine to service device 34. This subroutine can load the AC with data and transfer it to device 34, or can load the AC from a register in device 34 and store it in some known core memory address.

Accumulator

The binary 1 output signal of each flip-flop of the AC, buffered by a bus driver, is available at the interface connectors. These computer data output lines are bus connected to all peripheral equipment receiving programmed data output information from the PDP-8/I. A direct-set terminal on each flip-flop of the AC is connected to the interface connectors for bussing to all peripheral equipment supplying programmed data input to the PDP-8/I. A pulse that drives the direct-set terminal to ground causes setting of an AC flip-flop to the binary 1 state. Output and input connections to the accumulator appear in Figure 28.

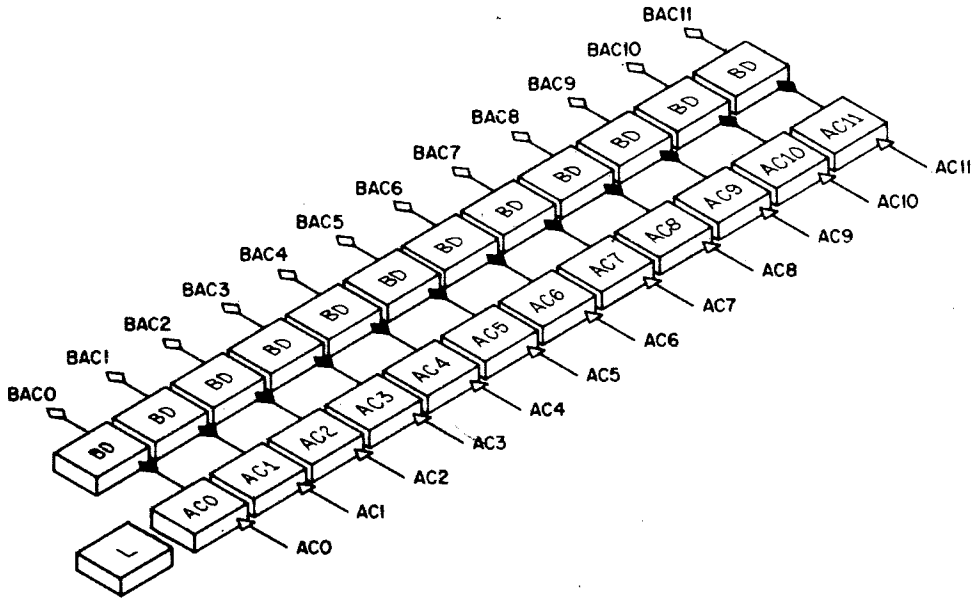


Figure 28. Accumulator Input and Output

Figure 28 illustrates the twelve bits of the accumulator and the link bit. The status of the link bit is not available to enter into transfers with peripheral equipment (unless it is rotated into the AC). A bus driver continuously buffers the output signal from each AC flip-flop. These buffered accumulator (BAC) signals are available at the interface connectors.

Input Data Transfers

When ready to transfer data into the PDP-8/I accumulator, the device sets a flag connected to the IOS. The program senses the ready status of the flag and issues an IOT instruction to read the content of the external device buffer register into the AC. If the AC is not cleared before the transfer, the resultant word in the AC is the inclusive OR of the previous word in the AC and the word transferred from the device buffer register.

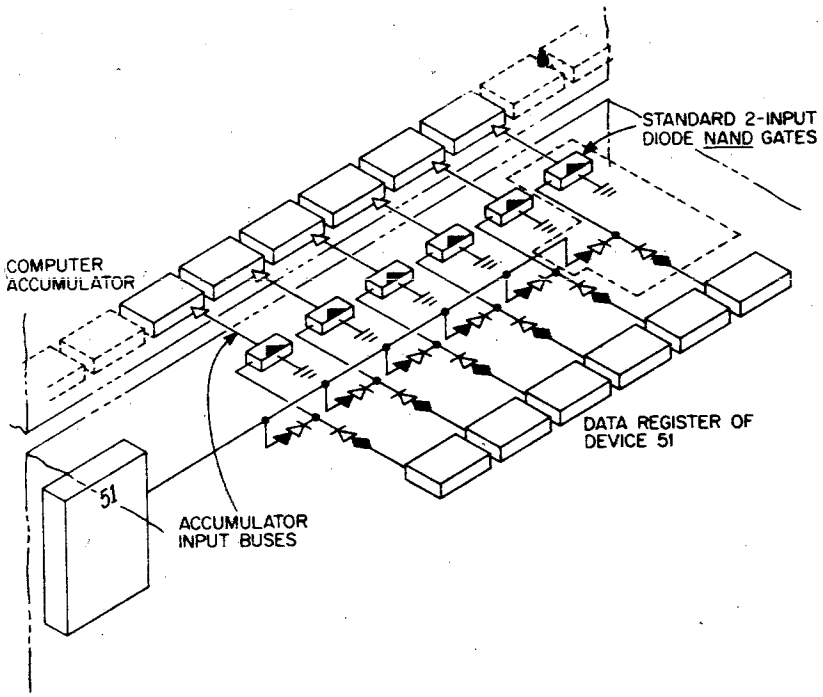


Figure 29. Loading Data into the Accumulator from an External Device

The illustration in Figure 29 shows that the accumulator has an input bus for each bit flip-flop. Setting a 1 into a particular bit of the accumulator necessitates grounding of the interface input bus by the standard DEC inverter. In the illustration, the 2-input AND gates set various bits of the accumulator. In this case an IOT pulse is AND combined with the flip-flop state of the external device to conditionally set 1's into the accumulator. (The program must include a clear AC command prior to loading in this manner; otherwise an inclusive OR takes place between the previous content of the accumulator and the content of the register being read.)

Following the transfer (possibly in the same instruction) the program can issue a command pulse to initiate further operation of the device and/or clear the device flag.

Output Data Transfers

The AC is loaded with a word (e.g., by a CLA TAD instruction sequence); then the IOT instruction is issued to transfer the word into the control or data register of the device by an IOT pulse (e.g., IOP 2), and operation of the device is initiated by another IOT pulse (e.g., IOP 4). The data word transferred in this manner can be a character to be operated upon, or can be a control word sampled by a status register to establish a control mode.

Since the BAC interface bus lines continually represent the status of the AC flip-flops, the receiving device can strobe them to sense the value in the accumulator. In Figure 30 a strobe pulse samples six bits of the accumulator to conditionally set an external 6-bit data register. Since this is not a jam transfer, it is necessary first to clear the external data register before setting 1's into it. The readin gates driving the external data register are part of the

external device and are not supplied by the computer. The data register can contain any number of flip-flops up to a maximum of twelve. (If more than twelve flip-flops are involved, two or more transfers must take place.) Obviously the clear pulse and the strobe pulse shown in Figure 30 must occur when the data to be placed in the external data register is held in the accumulator. These pulses therefore must be under computer control to effect synchronization with the operation or program of the computer.

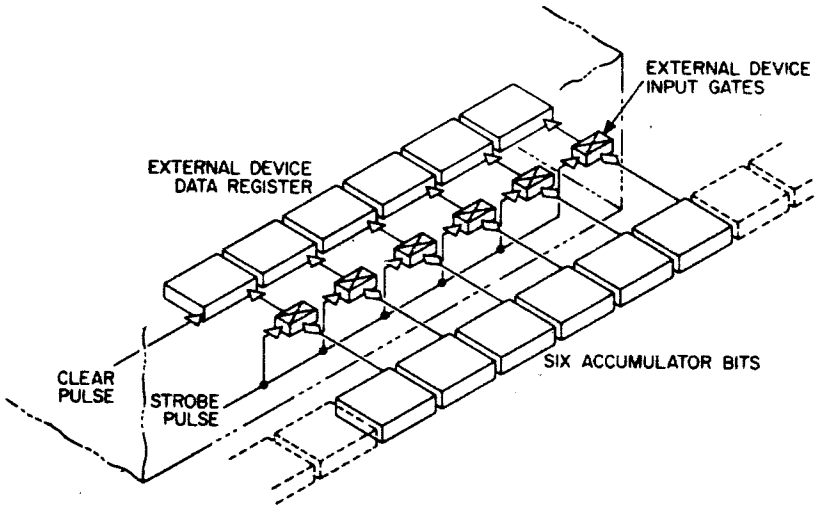


Figure 30. Loading a 6-Bit Word into an External Device from the Accumulator

Figure 31 illustrates the use of two of the pulses being gated by the device selector coded for "34." Pulse IOT 1 clears the data register and IOT 4 strobes the data from the accumulator into the data register. Note that the processor produces the IOP 1, IOP 2, and IOP 4 pulses and supplies them to all device selectors. The program-selected DS produces IOT 1, IOT 2, and IOT 4 pulses which initiate a transfer or effect some control. As indicated in Figure 31 this particular system adds two new microinstructions to the PDP-8/I repertoire. One generates a pulse to clear the data register of device number 34. The other microinstruction produces a pulse to load the data register of device number 34 with the content of the accumulator.

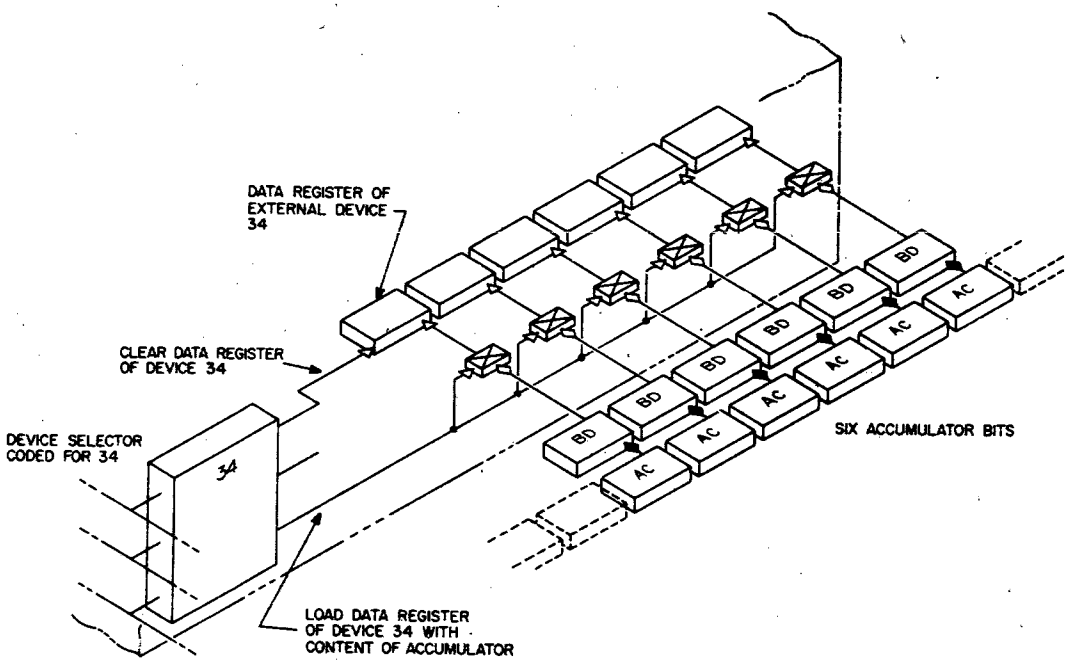


Figure 31. Use of a Device Selector for Activating and Controlling an External Device

The timing of the IOT cycle is shown in Figure 22. Note that the AC bus drivers are quiescent 400 nsec before the IOP 1 pulse occurs. Since FLIP CHIP DCD gates require a 400-nsec set-up time, the IOP 1 pulse cannot be used to load the content of the AC into an external buffer register having input DCD gates. If the device register has DCD gates, IOP 1 should be used to reset or clear registers, controls, or flags. The IOP 1 pulse can be used to read the content of the AC into an external device register that is equipped with input diode gates. IOP 2 or IOP 4 can be used to strobe the content of the AC through DCD gates if the lead lengths of the BAC lines and the pulse lines provide equivalent transmission delays. Only IOP 1 or IOP 2 (not IOP 4) can be used with the IOS facility.

Program Interrupt (PI)

When a large amount of computing is required, the program should initiate operation of an I/O device then continue the main program, rather than wait for the device to become ready to transfer data. The program interrupt facility, when enabled by the program, relieves the main program of the need for repeated flag checks by allowing the ready status of I/O device flags to automatically cause a program interrupt. When the program interrupt occurs, program control transfers to a subroutine that determines which device requested the interrupt and initiates an appropriate service routine.

In the example shown in Figure 32, a flag signal from a status flip-flop operates a standard inverter with no collector load. When the status flip-flop indicates the need for device service, the inverter drives the Program Interrupt Request bus to ground to request a program interrupt.

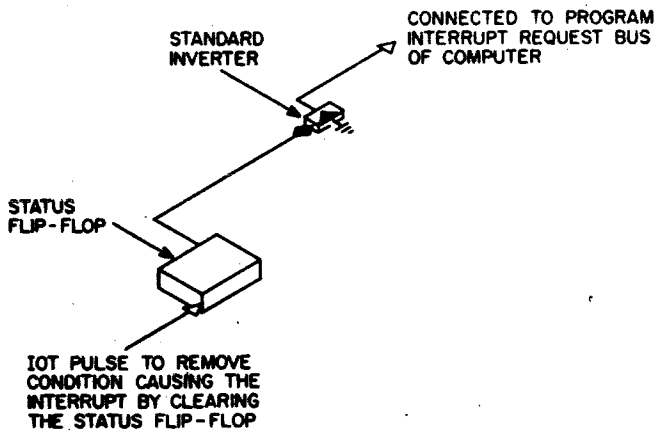


Figure 32. Program Interrupt Request Signal Origin

If only one device is connected to the PI facility, program control can be transferred directly to a routine that services the device when an interrupt occurs. This operation occurs as follows:

```
Tag     Address     Instruction          Remarks
        1000            .        /MAIN PROGRAM
        1001            .        /MAIN PROGRAM CONTINUES
        1002            .        /INTERRUPT REQUEST OCCURS
                    INTERRUPT OCCURS
        0000                     /PROGRAM COUNT (PC = 1003) IS
                                 /STORED IN 0000
        0001        JMP SR       /ENTER SERVICE ROUTINE
SR      2000            .        /SERVICE SUBROUTINE FOR
                        .        /INTERRUPTING DEVICE AND
                                 /SEQUENCE TO RESTORE AC, AND
        3001            .        /RESTORE L IF REQUIRED
        3002        ION          /TURN ON INTERRUPT
        3003        JMP I 0000   /RETURN TO MAIN PROGRAM
        1003            .        /MAIN PROGRAM CONTINUES
        1004            .
                        .
                        .
```

In most PDP-8/I systems numerous devices are connected to the PI facility, so the routine beginning in core memory address 0001 must determine which device requested an interrupt. The interrupt routine determines the device requiring service by checking the flags of all equipment connected to the PI and transfers program control to a service routine for the first device encountered that has its flag in the state required to request a program interrupt. In other words, when program interrupt requests can originate in numerous devices, each device flag connected to the PI must also be connected to the IOS.

Multiple Use of IOS and PI

In common practice, more than one device is connected to the PI facility. Therefore, since the computer receives a request that is the inclusive OR of requests from all devices connected to the PI, the IOS must identify the device making the request. When a program interrupt occurs, a routine is entered from address 0001 to sequentially check the status of each flag connected to the PI and to transfer program control to an appropriate service routine for the device whose flag is requesting a program interrupt. Figure 33 shows IOS and PI connections for three typical devices.

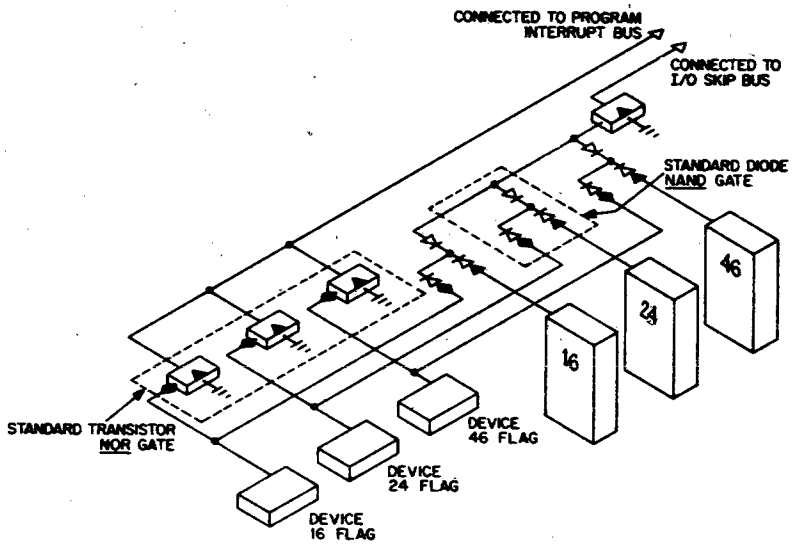


Figure 33. Multiple Inputs to IOS and PI Facilities

The following program example illustrates how the program interrupt routine determines the device requesting service:

Tag	Address	Instruction	Remarks
	1000	.	/MAIN PROGRAM
	1001	.	/MAIN PROGRAM CONTINUES
	1002	.	/INTERRUPT REQUEST OCCURS
		INTERRUPT OCCURS	
	0000		/STORE PC (PC = 1003)
	0001	JMP FLG CK	/ENTER ROUTINE TO DETERMINE
			/WHICH DEVICE CAUSED INTERRUPT
FLG CK		IOT 6341	/SKIP IF DEVICE 34 IS REQUESTING
		SKP	/NO — TEST NEXT DEVICE
		JMP SR34	/ENTER SERVICE ROUTINE 34
		IOT 6441	/SKIP IF DEVICE 44 IS REQUESTING
		SKP	/NO — TEST NEXT DEVICE
		JMP SR44	/ENTER SERVICE ROUTINE 44
		IOT 6541	/SKIP IF DEVICE 54 IS REQUESTING
		SKP	/NO — TEST NEXT DEVICE
		JMP SR44	/ENTER SERVICE ROUTINE 54
		.	
		.	
		.	

Assume that the device that caused the interrupt is an input device (e.g., tape reader). The following example of a device service routine might apply:

```
Tag         Instruction          Remarks
SR          DAC TEMP     /SAVE AC
            IOT XX       /TRANSFER DATA FROM DEVICE
                         /BUFFER TO AC
            DAC I 10     /STORE IN MEMORY LIST
            ISZ COUNT    /CHECK FOR END
            SKP          /NOT END
            JMP END      /END. JUMP TO ROUTINE TO HANDLE
                         /END OF LIST CONDITION
                 .
                 .
                 .
                         /RESTORE L AND EPC IF REQUIRED
            TAD TEMP     /RELOAD AC
            ION          /TURN ON INTERRUPT
            JMP I 0      /RETURN TO PROGRAM
```

If the device that caused the interrupt was essentially an output device (receiving data from computer), the IOT — then — DAC I 10 sequence might be replaced by a TAD I 10 — then — IOT sequence.

CHAPTER 10
DATA BREAK TRANSFERS

The data break facility allows I/O device to transfer information directly with the PDP-8/I core memory on a cycle-stealing basis. Up to seven devices can connect to the data break facility through the optional Data Multiplexer Type DM01. The data break is particularly well-suited for devices which transfer large amounts of information in block form.

Peripheral I/O equipment operating at high speeds can transfer information with the computer through the data break facility more efficiently than through programmed means. The combined maximum transfer rate of the data break facility is over 7.8 million bits per second. Information flow to effect a data break transfer with an I/O device appears in Figure 34.

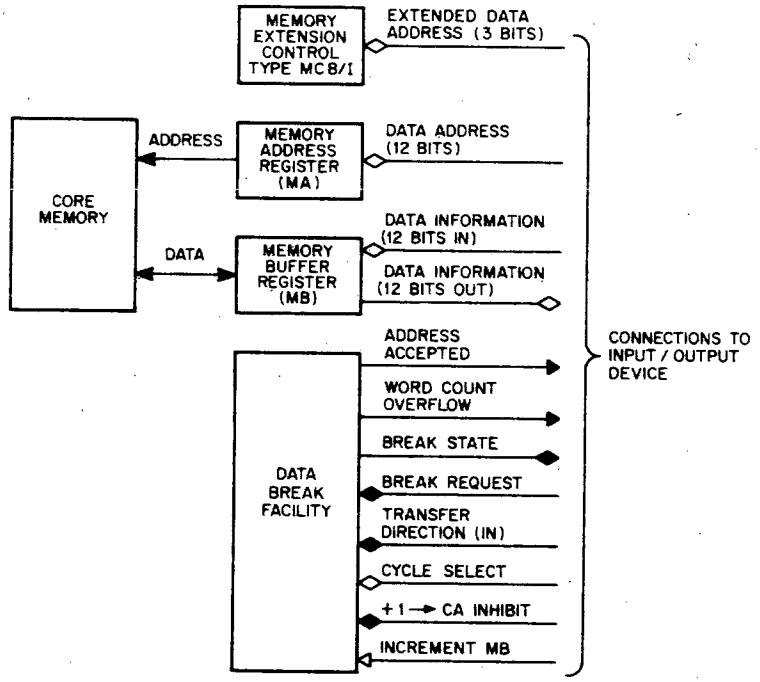


Figure 34. Data Break Transfer Interface Block Diagram

In contrast to programmed operations, the data break facilities permit an external device to control information transfers. Therefore, data-break device interfaces require more control logic circuits, causing a higher cost than programmed-transfer interfaces.

Data breaks are of two basic types: single-cycle and three-cycle. In a single-cycle data break, registers in the device (or device interface) specify the core memory address of each transfer and count the number of transfers to determine the end of data blocks. In the three-cycle data break two computer core memory locations perform these functions, simplifying the device interface by omitting two hardware registers.

In general terms, to initiate a data break transfer of information, the interface control must do the following:

a. Specify the affected address in core memory.

b. Provide the data word by establishing the proper logic levels at the computer interface (assuming an input data transfer), or provide readin gates and storage for the word (assuming an output data transfer).

c. Provide a logical signal to indicate direction of data word transfer.

d. Provide a logical signal to indicate single-cycle or three-cycle break operation.

e. Request a data break by supplying a proper signal to the computer data break facility.

Single-Cycle Data Breaks

Single-cycle breaks are used for input data transfers to the computer, output data transfers from the computer, and memory increment data breaks. Memory increment is a special output data break in which the content of a memory address is read, incremented by 1, and rewritten at the same address. It is useful for counting iterations or external events without disturbing the computer program counter (PC) or AC registers.

Input Data Transfers

Figure 35 illustrates timing of an input transfer data break. The address to be affected in core is normally provided in the device interface in the form of a 12-bit flip-flop register (data break address register) which has been preset by the interface control by programmed transfer from the computer.

External registers and control flip-flops supplying information and control signals to the data break facility and other PDP-8/I interface elements are shown in Figure 36. The input buffer register (IB in Figure 36) holds the 12-bit data word to be written into the computer core memory location specified by the address contained in the address register (AR in Figure 36). Appropriate output terminals of these registers are connected to the computer to supply ground potential to designate binary 1's. Since most devices that transfer data through the data break facility are designed to use either single-cycle or three-cycle breaks, but not both, the Cycle Select signal can usually be supplied from a stable source (such as a ground connection or a −3v clamped load resistor) rather than from a bistable device as shown in Figure 36.

Other portions of the device interface, not shown in Figure 36, establish the data word in the input buffer register, set the address into the address register, set the direction flip-flop to indicate an input data transfer, and control the break request flip-flop. These operations can be performed simultaneously or sequentially, but all transients should occur before the data break request is made. Note that the device interface need supply only static levels to the computer, minimizing the synchronizing logic circuits necessary in the device interface.

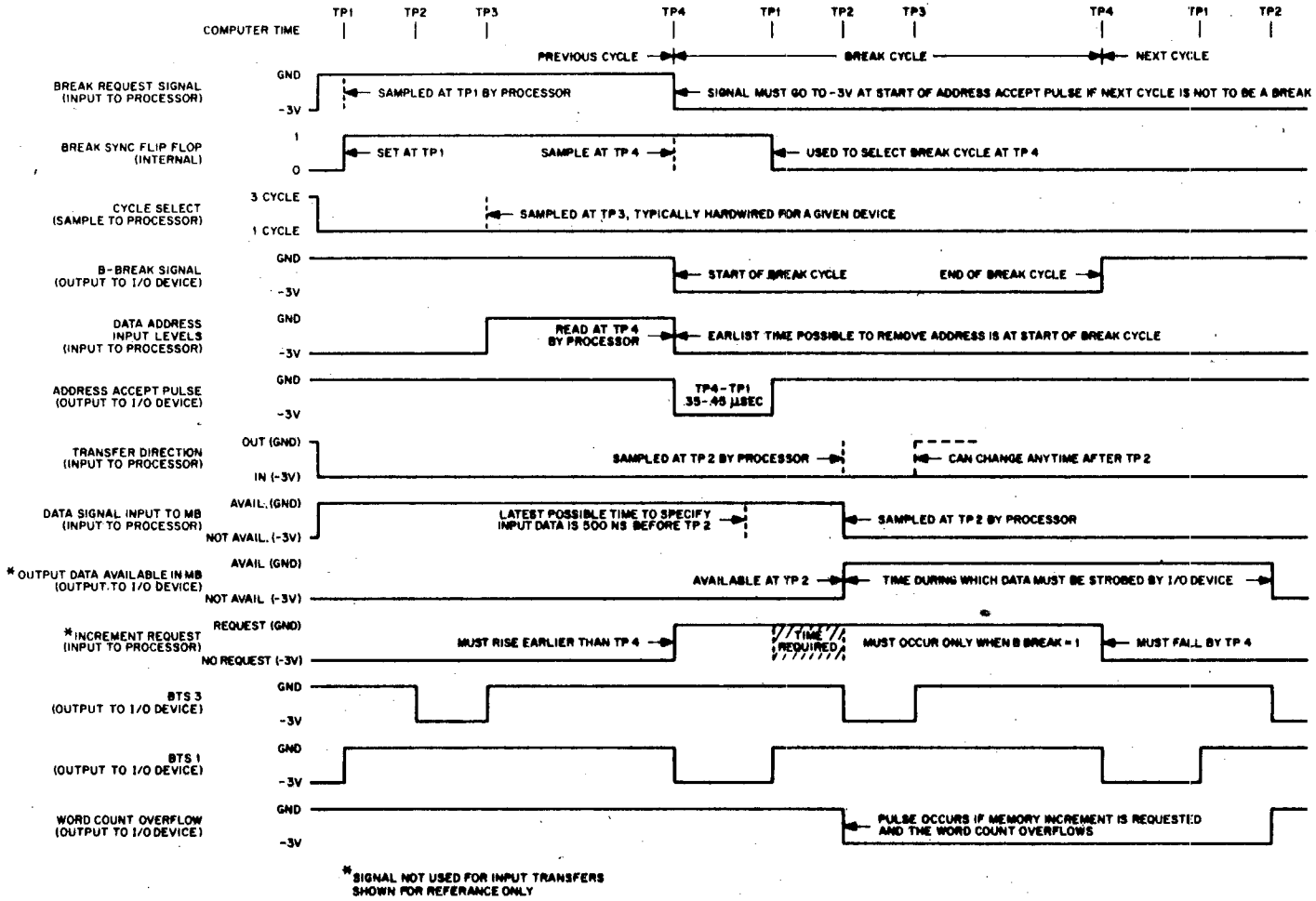


Figure 35. Single-Cycle Data Break Input Transfer Timing Diagram

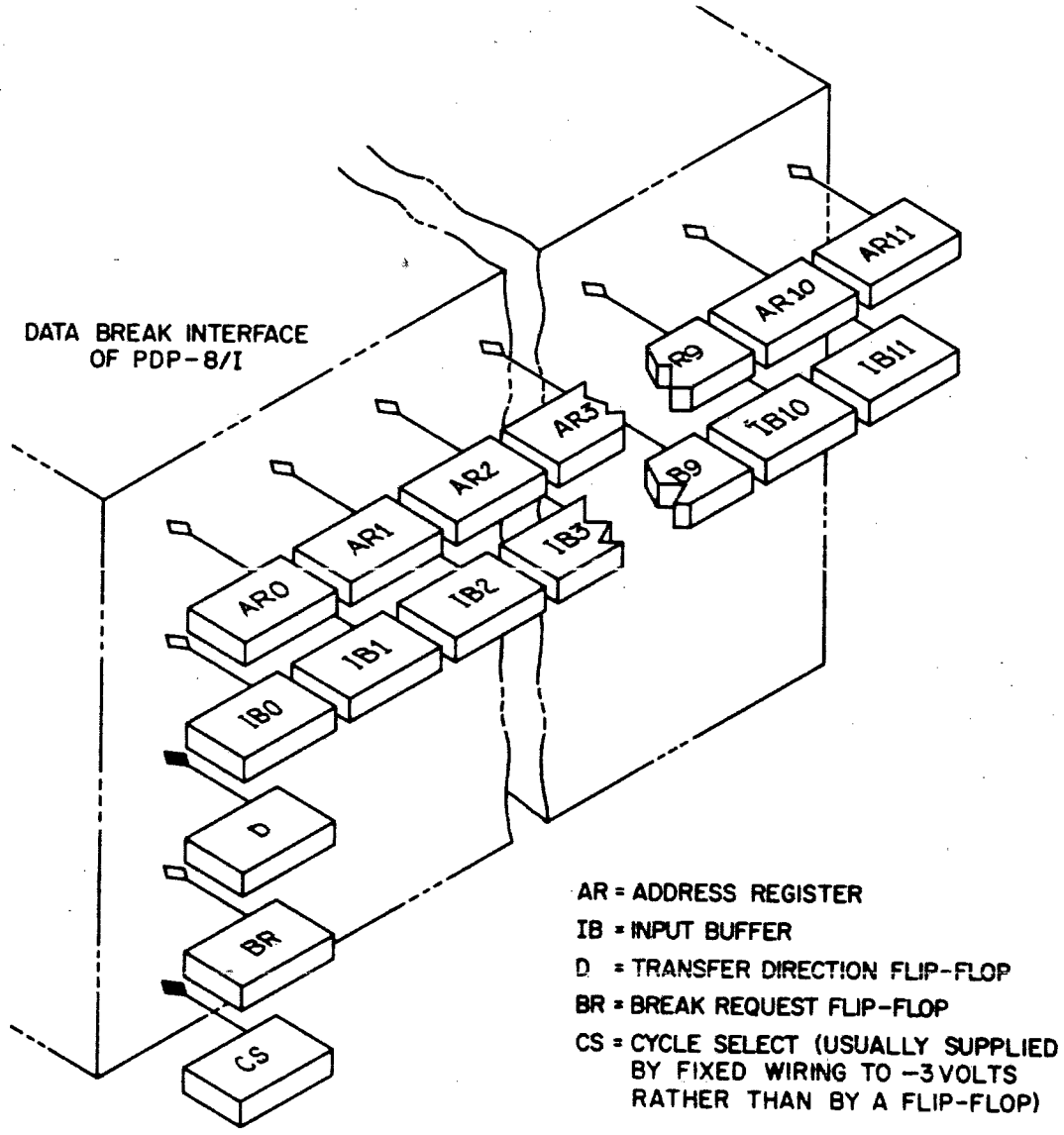


Figure 36. Device Interface Logic for Single-Cycle Data Break Input Transfer

When the data break request arrives, the computer completes the current instruction, generates an Address Accepted pulse (at TP4 time of the cycle preceding the data break) to acknowledge receipt of the request, then enters the Break state to effect the transfer (see Chapter 5 of this handbook for more details on data break operations performed by the computer). The Address Accepted pulse can be used in the device interface to clear the break request flip-flop, increment the content of the address register, etc. If the Break Request signal is removed before TP1 time of the data break cycle, the computer performs the transfer in one 1.5-μsec cycle and returns to programmed operation.

Output Data Transfers

Timing of operations occurring in a single-cycle output data break is shown in Figure 37. Basic logic circuits for the device interface used in this type of transfer are shown in Figure 38. Address and control signal generators are similar to those discussed previously for input data transfers, except that the Transfer Direction signal must be at ground potential to specify the output transfer of computer information. An output data register (OB in Figure 38) is usually required in the device interface to receive the computer information. The device, and not the PDP-8/I, controls strobing of data into this register. The device must supply strobe pulses for all data transfers out of the computer (programmed or data break) since circuit configuration and timing characteristics differ in each device.

When the data break request arrives, the computer completes the current instruction and generates an Address Accepted pulse as in input data break transfers. At TP4 time the address supplied to the PDP-8/I is loaded into the MA, the Break state is entered, and the MB is cleared. Not more than 450 nsec after TP4 (at TP2 time), the content of the device-specified core memory address is read and available in the MB. (This word is automatically rewritten at the same address during the last half of the Break cycle and is available for programmed operations when the data break is finished.) Data Bit signals are available as static levels of ground potential for binary 1's and −3v for binary 0's. The MB is cleared at TP2 time of each computer cycle, so the data word is available in the MB for approximately 1.5 μsec to be strobed by the device interface.

Generation of the strobe pulse by the device interface can be synchronized with computer timing through use of timing pulses BTS1 or BTS3, which are avaliable at the computer interface. In addition to a timing pulse (delayed or used directly from the computer), generation of this strobe pulse should be gated by condition signals that occur only during the Break cycle of an output transfer. Figure 39 shows typical logic circuits to effect an output data transfer. In this example the B Break signal and an inverted Transfer Direction signal are combined in a diode NAND gate to condition a diode-capacitor-diode gate. A buffered BTS1 pulse triggers the DCD gate to produce the strobe pulse. The BTS1 pulse determines the timing of the transfer in this example, since the input of the output buffer register has DCD gates. Conventional DCD gates require a minimum setup time of 400 nsec, which is adequately provided between the time when data is available in the MB and TS1 time.

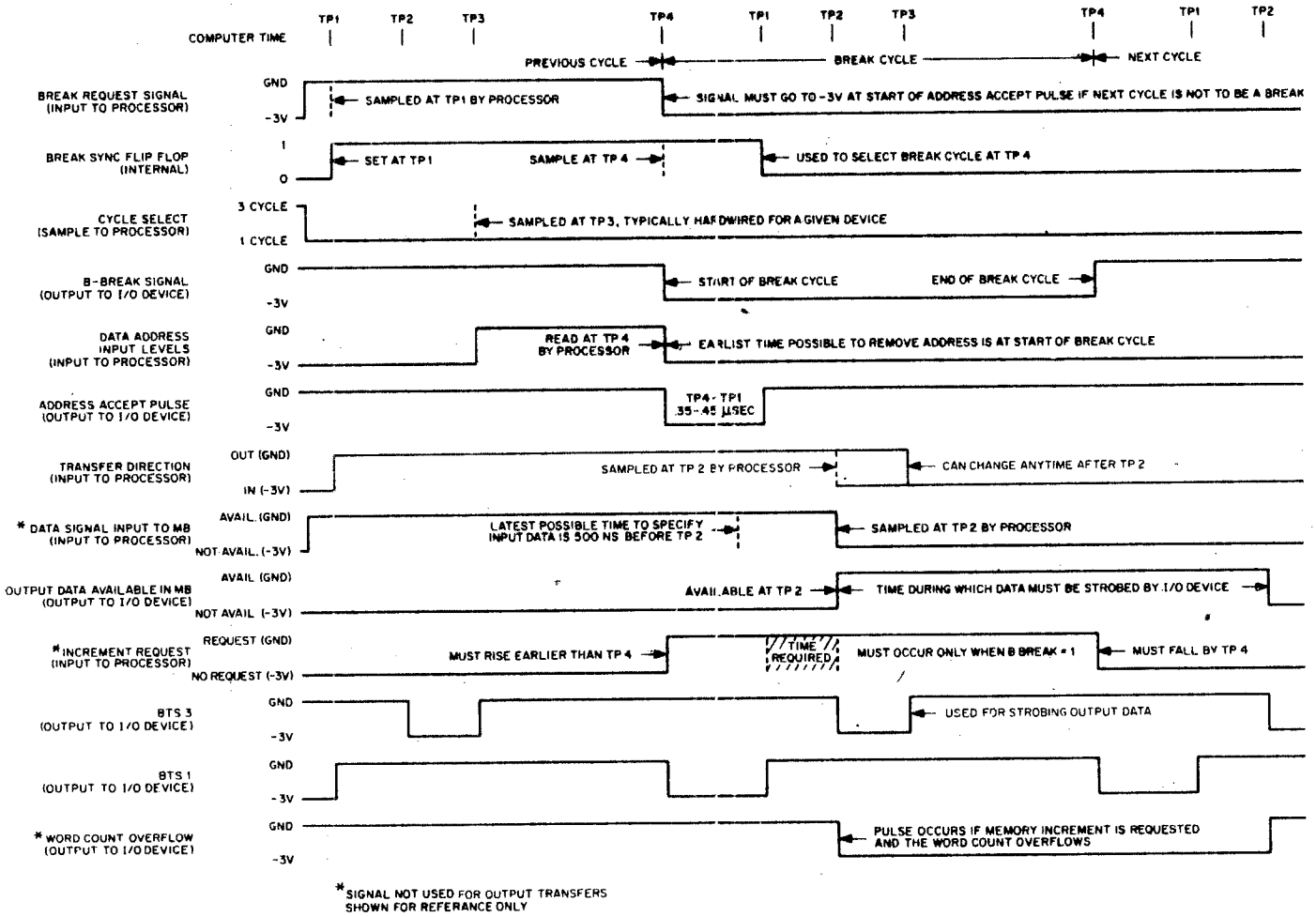


Figure 37. Single-Cycle Data Break Output Transfer Timing Diagram

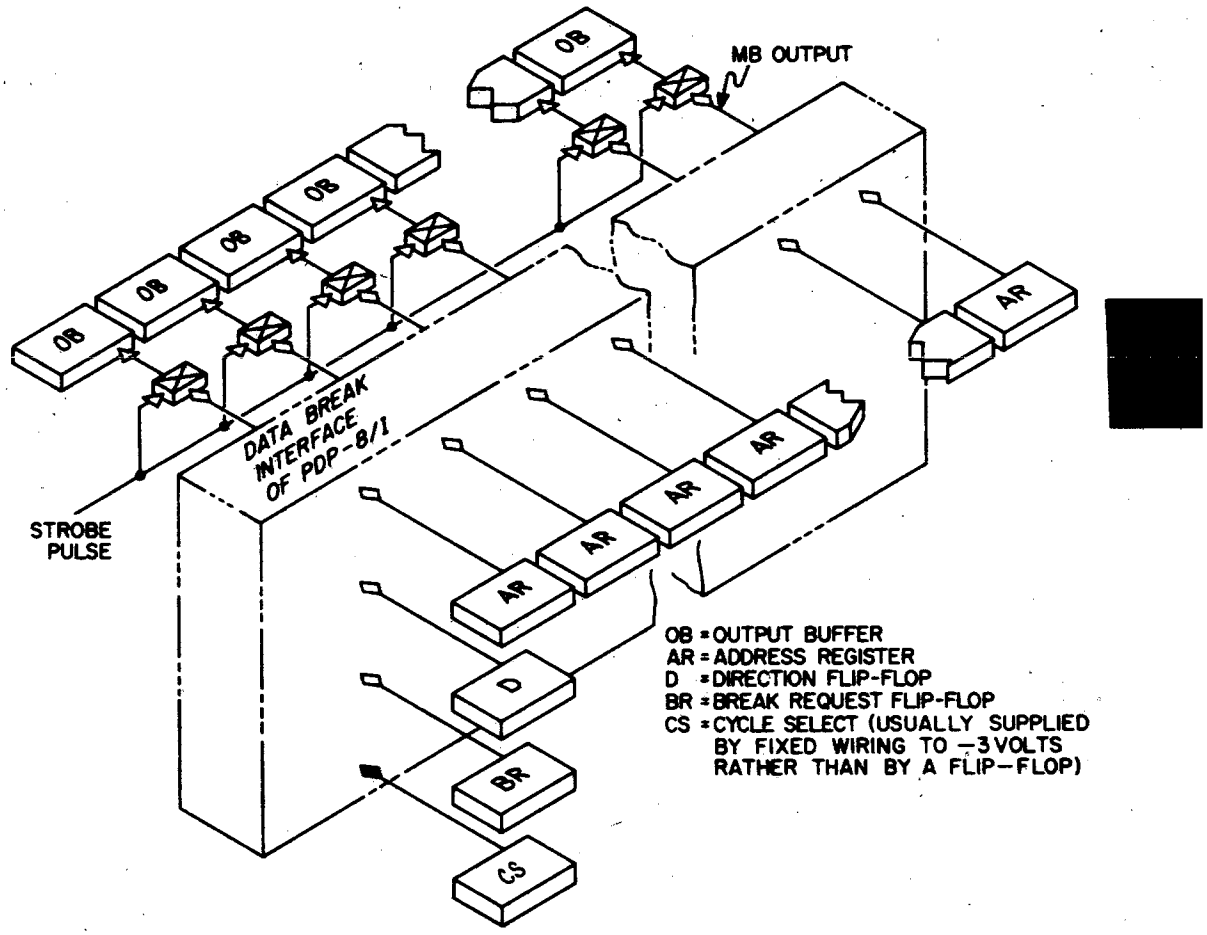


Figure 38. Device Interface Logic
for Single-Cycle Data Break Output Transfer

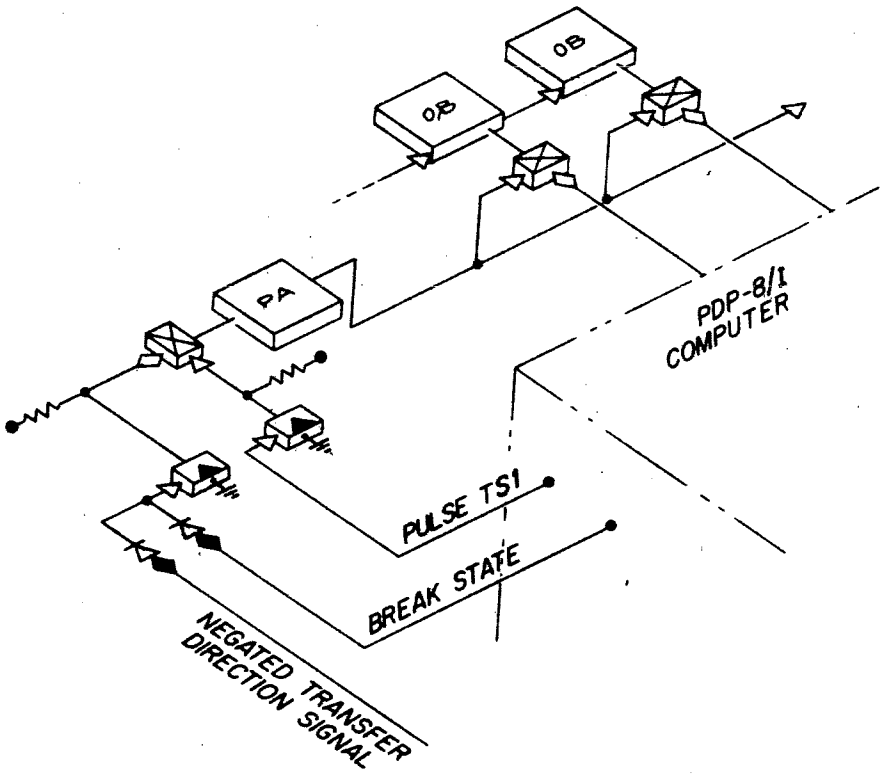


Figure 39. Device Interface for Strobing Output Data

By careful design of the input and output gating, one register can serve as both the input and the output buffer register. Most DEC options using the data break facility have only one data buffer register with appropriate gating to allow it to serve as an output buffer when the Transfer Direction signal is at ground potential or as an input buffer when the Transfer Direction signal is −3v.

Memory Increment

In this type of data break the content of core memory at a device-specified address is read into the MB, is incremented by 1, and is rewritten at the same address within one 1.5-μsec cycle. This feature is particularly useful in building a histogram of a series of measurements, such as in pulse-height analysis applications. For example, in a computer-controlled experiment that counts the number of times each value of a parameter is measured, a data break can be requested for each measurement, and the measured value can be used as the core memory address to be incremented (counted).

Signal interface for a memory increment data break is similar to an output transfer data break except that the device interface generates an Increment MB signal and does not generate a strobe pulse (no data transfer occurs between the PDP-8/I and the device). Timing of memory increment operations appear in Figure 40, and an example of the logic circuits used by a device interface appears in Figure 41.

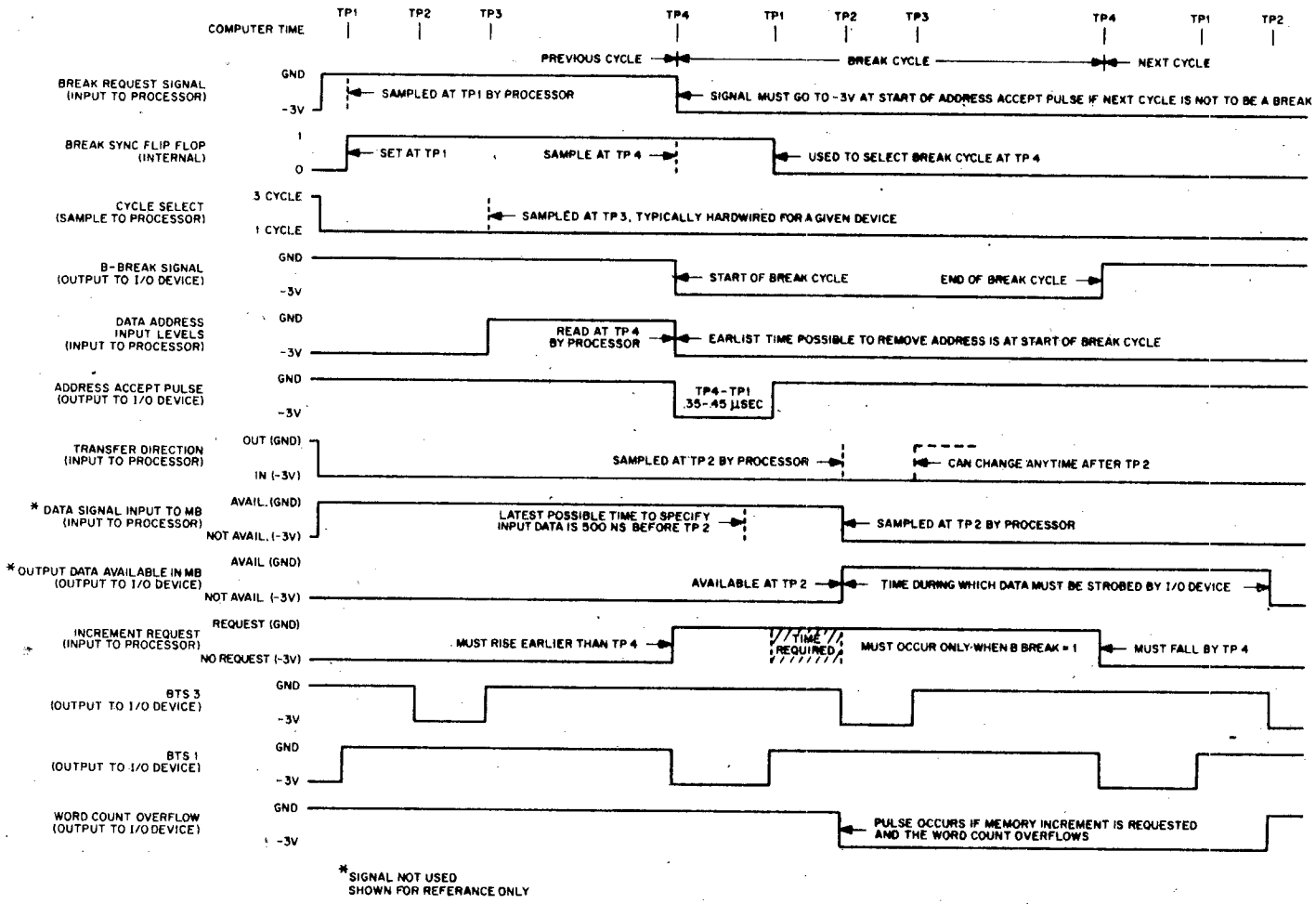


Figure 40. Memory Increment Data Break Timing Diagram

An interface for a device using memory increment data breaks must supply twelve Data Address signals, a Transfer Direction signal, a Cycle Select signal, and a Break Request signal to the computer data break facility as in an output transfer data break. In addition, a ground potential increment MB signal must be provided at least 250 nsec before TP2 time of the Break cycle. This signal can be generated in the device interface by AND combining the B Break computer output signal, the output transfer condition of the Transfer Direction signal, and the condition signal in the device that indicates that an increment operation should take place. When the computer receives this Increment MB signal, it forces the MB control element to generate a Count MB pulse at TP2 time to increment the content of the MB.

The device interface logic shown in Figure 41, samples the word count overflow signal to determine if word count overflowed when the data word is incremented. If overflow occurs this logic requests a program interrupt to allow the program to take some appropriate action, such as incrementing a core memory for numbers above 4096, stopping the test to compile the data gathered to the current point in the operation, reinitializing the addressing, etc. The logic in the figure uses the select code of programmed data transfer operation to skip on the overflow condition to determine the cause of a program interrupt, to clear the overflow flip-flop, and to clear the device flag. Note that the devices that use data break transfers almost always use programmed data transfers to start and stop operation of the device, to initialize registers, etc., and do not rely on data break facilities alone to control their operations.

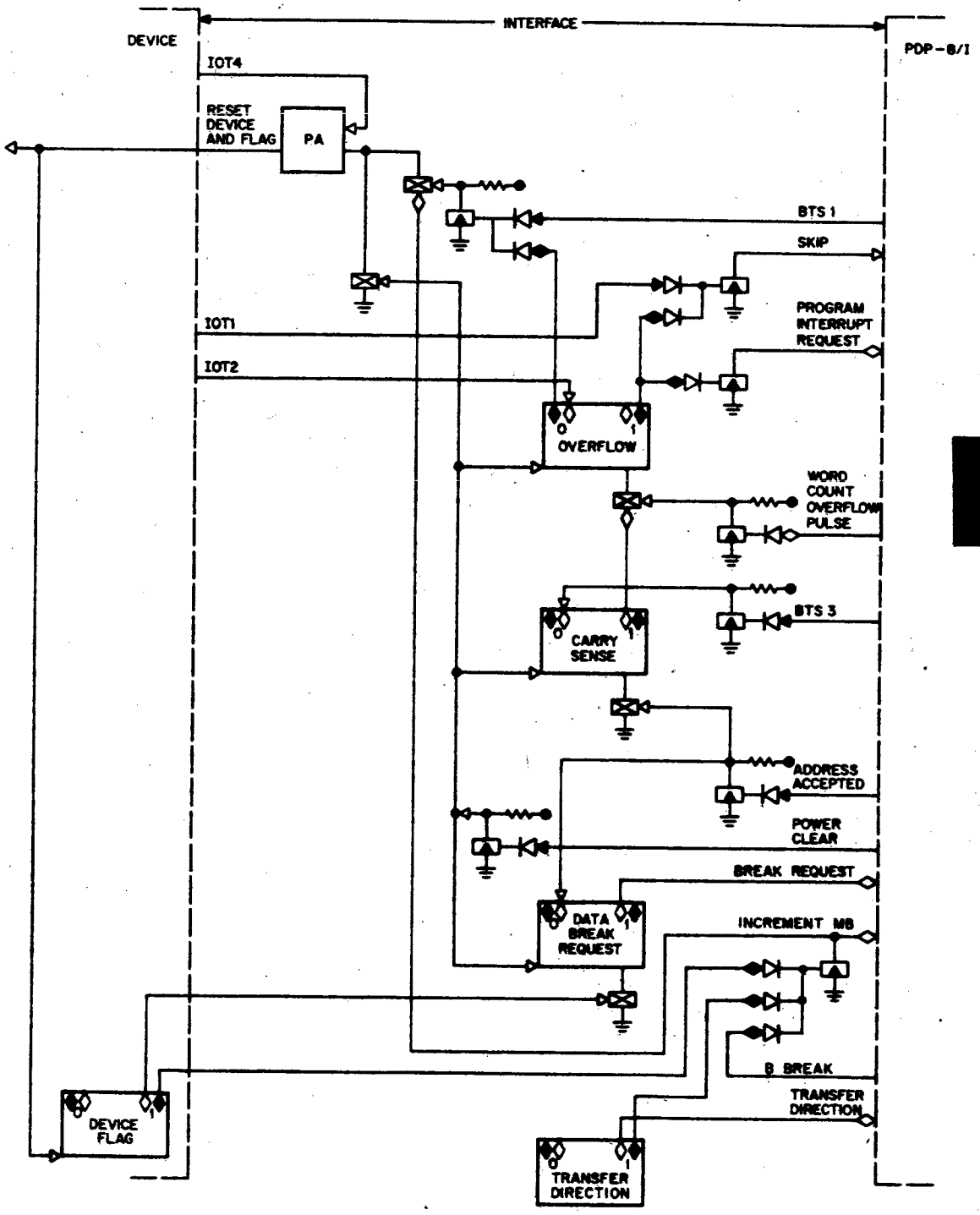


Figure 41. Device Interface Logic for Memory Increment Data Break

Three-Cycle Data Breaks

Timing of input or output 3-cycle data breaks is shown in Figure 42. The 3-cycle break uses the block transfer control circuits of the computer. The block transfer control provides an economical method of controlling the flow of data at high speeds between PDP-8/I core memory and fast peripheral devices, e.g., drum, disc, magnetic tape and line printers, allowing transfer rates in excess of 220 kc.

The three-cycle data break facility provides separate current address and word count registers in core memory for the connected device, thus eliminating the necessity for flip-flop registers in the device control. When several devices are connected to this facility, each is assigned a different set of core locations for word count and current address, allowing interlaced operations of all devices as long as their combined rate does not exceed 220 kc. The device specifies the location of these registers in core memory, and thus the software remains the same regardless of what other equipment is connected to the machine. Since these registers are located in standard memory, they may be loaded and unloaded directly without the use of IOT pulses. In a procedure where a device requests to transfer data to or from core memory, the three-cycle data break facility performs the following sequence of operations:

a. An address is read from the device to indicate the location of the word count register. This address is always the same for a given device; thus it can be wired in and does not require a flip-flop register.

b. The content of the specified address is read from memory and 1 is added to it before rewriting. If the content of this register becomes 0 as a result of the addition, a WC Overflow pulse will be transmitted to the device. To transfer a block of N words, this register is loaded with − N during programmed initialization of the device. After the block has been fully transferred this pulse is generated to signify completion of the operation.

c. The next sequential location is read from memory as the current address register. Although the content of this register is normally incremented before being rewritten, an increment CA Inhibit (+1 → CA Inhibit) signal from the device may inhibit incrementation. To transfer a block of data beginning at location A, this register is program initialized by loading with A-1.

d. The content of the previously read current address is transferred to the MA to serve as the address for the data transfer. This transfer may go in either direction in a manner identical to the single-cycle data break system.

The three-cycle data break facility uses many of the gates and transfer paths of the single-cycle data break system, but does not preclude the use of standard data break devices. Any combination of three-cycle and single-cycle data break devices can be used in one system, as long as a multiplexer channel is available for each. Two additional control lines are provided with the three-cycle data break. These are:

a. Word Count Overflow. A level change from GND to −3V, from TP2 to TP2, is transmitted to the device when the word count becomes equal to zero.

b. Increment CA Inhibit. When ground potential, this device-supplied signal inhibits incrementation of the current address word.

In summary, the three-cycle data break is entered similarly to the single-cycle data break, with the exception of supplying a ground-level Cycle Select signal to allow entry of the WC (Word Count) state to increment the fixed core memory location containing the word count. The device requesting the break supplies this address as in the one-cycle data break, except that this address is fixed and can be supplied by wired ground and −3v signals, rather than from a register. The sole restriction on this address is that it must be an even number (bit 11 = 0). Following the WC state a CA (Current Address) state is entered in which the core memory location following the WC address (bit 11 = 1 after PC + 1 = > PC) is read, incremented by one, restored to memory, and used as the transfer address (by MB = > MA). Then the normal B (Break) state is entered to effect the transfer.

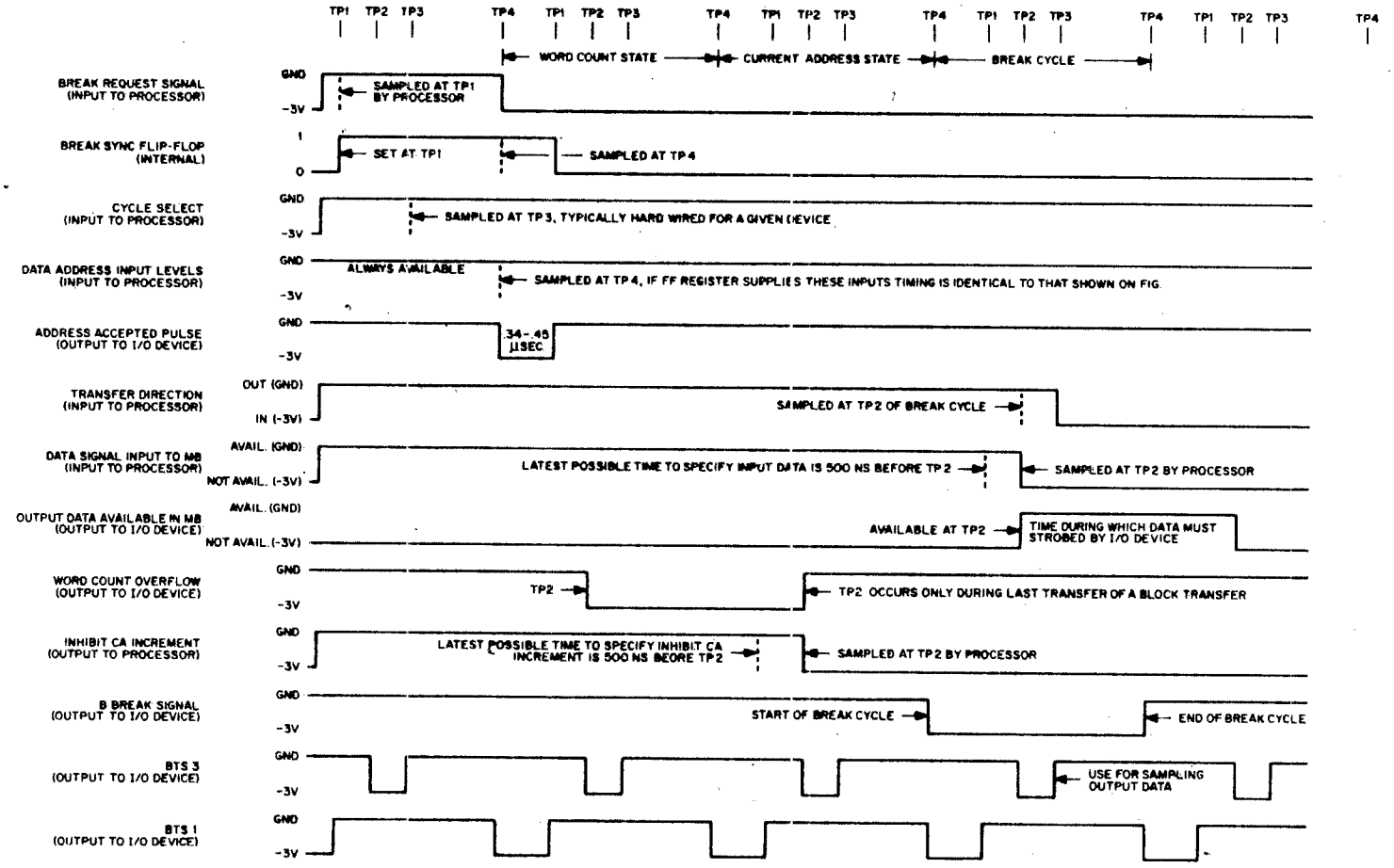


Figure 42. Three-Cycle Data Break Timing Diagram

CHAPTER 11

DIGITAL LOGIC CIRCUITS

PDP-8/I interfacing is constructed of Digital Flip Chip modules. The Digital Logic Handbook describes more than 100 of these modules, all of their component circuits, and the associated accessories; i.e., power supplies and mounting panels. The user should study this catalog carefully before beginning the design of a special interface.

The interface modules of the 8/I are the M506 and M650. All interface signals to the computer will use the M506. All interface signals from the computer will originate from the M650.

M506 Negative Input Converter

The M506 converts DEC levels of —3V and ground on the PDP-8/I I/O buss to positive levels of +3V and ground. All signals from the external devices to the computer are converted with this module. A —3V input will give a +3V output. A ground input will give a ground output. A 10MA clamped load is connected to each buss input pin.

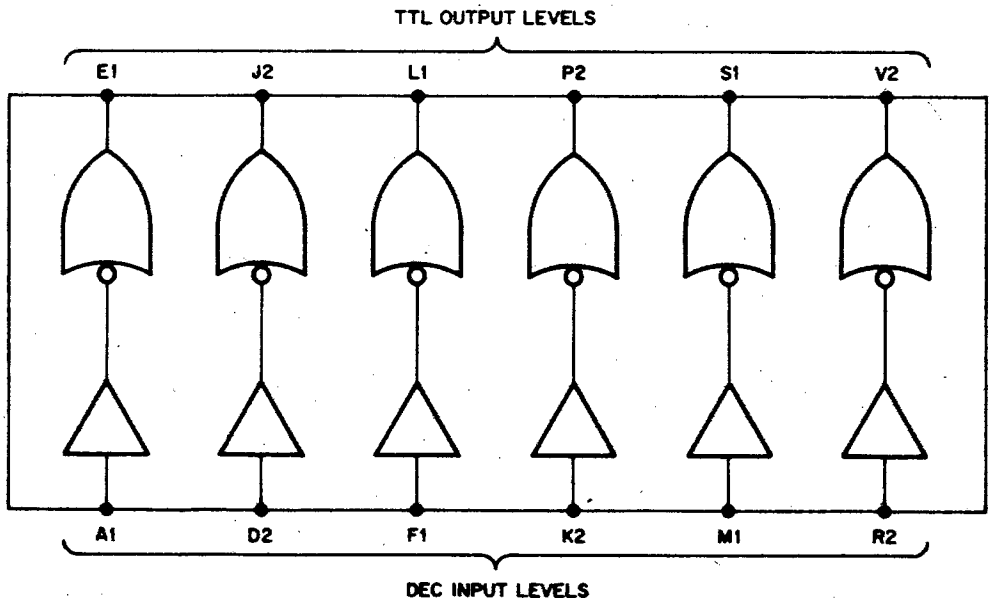
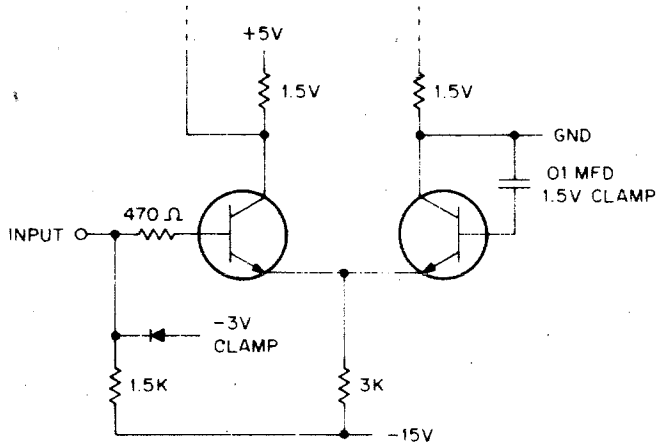


Figure 43. Logic Diagram (M506)



ALL TRANSISTORS ARE DEC 30098 (EIA 2N3009)
ALL DIODES ARE DEC 664 (EIA 1N3606)
ALL RESISTORS ARE 1/4 W, 5%

Figure 44. Input Circuit Schematic (M506)

M650 Output Converter and Buss Driver

The M650 converts PDP-8/I levels of +3V and ground to DEC levels of −3V and ground. All signals from the computer to the external devices originate from this module. The M650 contains three level converting buss drivers for driving heavy current loads to either ground or negative voltages. The buss drivers operate at frequencies up to 500KC with typical rise and fall times of 50 NSEC. The typical total transition times are 800 NSEC for output rise and 700 NSEC for output fall. The output to the interface connector drives 20MA of external load at either ground or −3V.

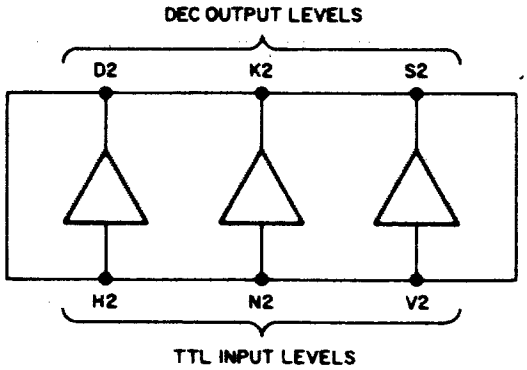


Figure 45. Logic Diagram (M650)

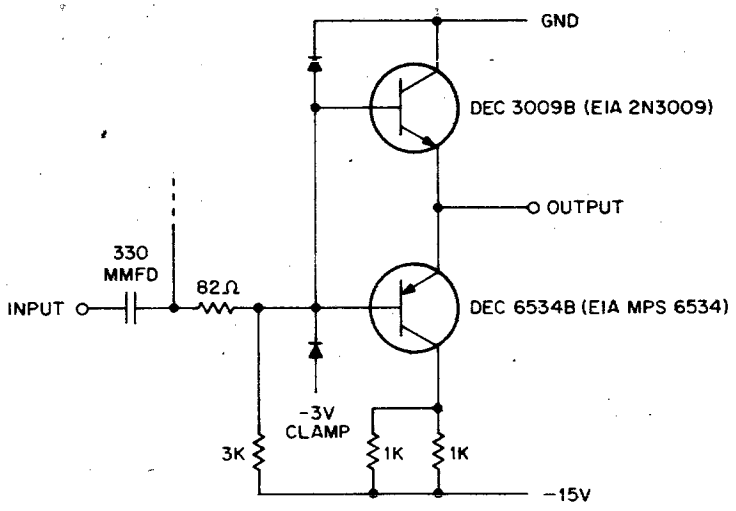


Figure 46. Output Circuit Schematic (M650)

Module Selection for Interface Circuits of Peripheral Equipment

Several Flip Chip modules are of particular interest in the design of equipment to interface to the PDP-8/I. Complete details on these and other Flip Chip modules can be found in the Digital Logic Handbook. Of particular interest to the interface designer are the following:

W103 DEVICE SELECTOR

The W103 selects an input/output device according to the code in the instruction word (being held in the memory buffer during the IOT cycle). Figure 47 shows the logic circuits of the W103 module.

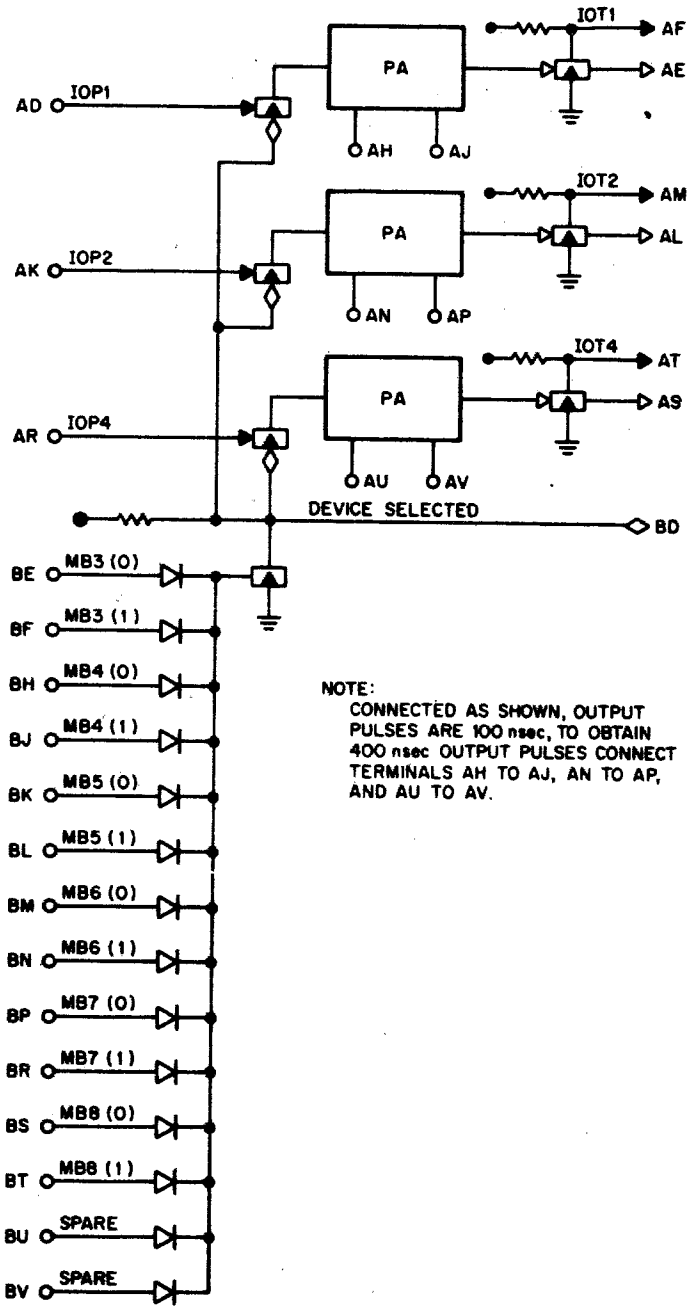


Figure 47. Device Selector W103 Logic Circuit

The twelve input diodes permit selection of any arbitrary 6-bit code, and decode the number held in MB bits 3 through 8. When the proper enabling code arrives at the diode gate, the three input gates driving the three pulse amplifiers are enabled and permit passage of the programmed IOP pulses. To establish a code on the module, the six unnecessary diodes are disabled by snipping one of their leads or removing them altogether. If MB bit 3 is a binary 1 to set up the correct code, the diode going to the binary 0 side of MB bit 3 is disabled. Two spare diodes are included for additional gating flexibility. Three pulse amplifiers produce R-series 100-nsec positive-going output pulses. Inverted pulse amplifier output pulses are provided for gates (such as the Type R111 Diode Gate) which require negative or negative-going pulses. Jumper terminals on each pulse amplifier establish a pulse duration of 400-nsec for the output pulse. It is recommended that the 400-nsec pulse duration be used when transmitting the pulse over long distances. The 400-nsec pulse also clears R-series flip-flops whose carry gates are permanently enabled. The positive pulse output of each pulse amplifier is rated 65 ma of external load at ground; the negative output is rated 15 ma at ground (when driving a load connected to −15v). These outputs are not designed to drive loads when at −3v (loads connected to ground). To drive this type of load, a clamped load resistor must be connected to the pulse amplifier output terminal to supply the current.

R111

The R111 Diode Gate contains three diode gates each connected to a transistor inverter. It is used, in devices interfaced to the PDP-8/I, for sampling device flags via the skip facility. (See Figure 27).

The gate operates as a NAND for negative inputs and as a NOR for ground inputs. Each gate has three input terminals: two are connected to diodes; a third is connected directly to the node point of the diode gate. The third terminal allows the number of input diodes to be increased by adding external diode networks such as the R001 or R002 modules. External diodes must be connected in the same direction as the diodes in the R111. Typical output total transition times are 60 nsec for rise and 50 nsec for fall.

Input signals to the diode terminals must be standard levels of ground or −3V, and must have a minimum duration of 100 nsec. Input load is 1 ma shared among the inputs at ground. Input signals to the node terminals accept only connections from Type R001 or R002 Diode Network modules, or their equivalents. The maximum combined length of all leads attached to a node terminal is 6 inches. Input signal load is similar to the diode input.

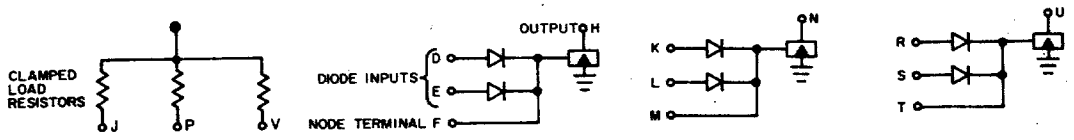
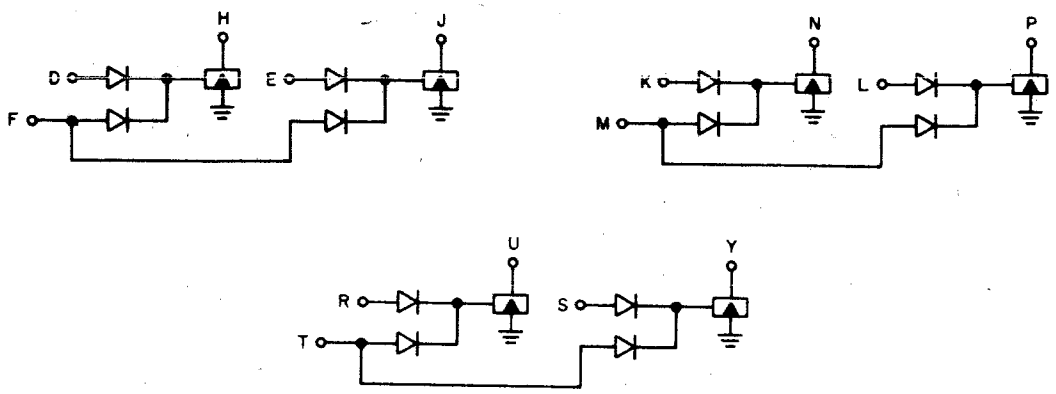


Figure 48. R111 Diode Gate

R123 DIODE GATE

This module contains six 2-input NAND gates for negative levels and is useful for transferring data into or out of the PDP-8/I accumulator. Standard DEC negative levels or 0.4 microsecond negative pulses such as those from the W103 Device Selector can be used as input signals. Input load per gate is 1 ma shared among the inputs at ground.



NOTES:
1. STROBE PULSE INPUT TO TERMINALS F, M, AND T WHICH ARE CONNECTED IN COMMON WHEN USED AS A BUS GATE
2. DATA BIT INPUTS TO TERMINALS D, E, K, L, R, AND S
3. TWO MODULES ARE REQUIRED TO STROBE A 12-BIT WORD

Figure 49. Diode Gate R123 Logic Circuit

Two R123 modules provide sufficient gating to transfer one 12-bit word into the accumulator. If more gates are needed to load the AC from several sources, the output terminals can be OR connected by bussing together additional gate collectors.

R203

The R203 contains three identical flip-flops and is frequently used as a device buffer for output data transfer. (See Figures 30 and 31).

Each Flip-flop has a direct clear and a DCD gate for conditional readin.

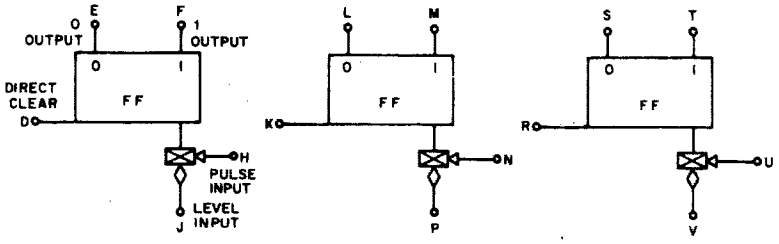


Figure 50. R203 Triple Flip-Flop

R141

The R141 AND/NOR Gate performs two levels of gating. The module contains a multiple-input diode gate with a transistor inverter for signal amplification. This module is used for multiple inputs to the IOS facilities. (See Figure 33). For negative input signals the R141 is seven 2- input AND gates which are NORed together. For ground input, it is seven 2- input OR gates NANDed together. The back to back diode circuits are possible because of an internal bias resistor connected to the input of each second stage diode.

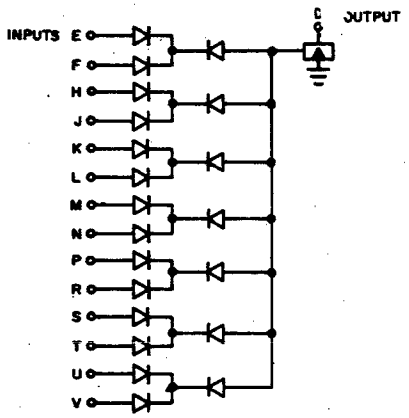


Figure 51. R141 AND/NOR Gate

CHAPTER 12
DESIGNING AND CONSTRUCTING INTERFACE EQUIPMENT

This section will provide the interface designer with additional information on design procedures, module layout, wiring, and cable selection. Additional help may be obtained from local DEC sales offices.

Physical:

The PDP-8/I was designed to provide the user maximum ease and flexibility in implementing special interfaces. External devices and interfaces are constructed and mounted outside of the basic machine, thereby, eliminating the necessity for modifications to the basic processor. All signals to and from the computer are carried on six coax cables.

To implement several devices the six cables parallel connect each peripheral in a serial type form. (See Figure 51).

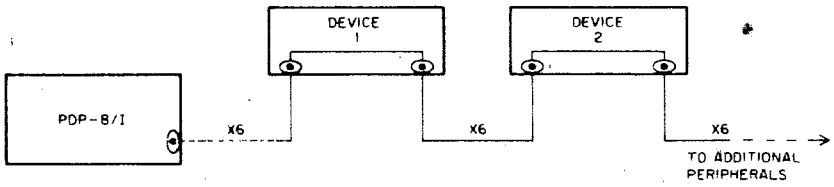


Figure 52. I.O. Bus Configuration

Module Layout

In general module layout is done based on the functional elements within a system and is primarily a matter of common sense.

Digital has, however, layout conventions for I/O cabling to extend devices. The interface designer may wish to use these conventions as a guide. The general rule is DO NOT DEAD END THE I/O BUS. This means that parallel connections should always be made at each device to handle possible future expansion.

CABLE LOCATION

1	2	3	4	5	6	7	8	9	10	11	32
AC0 TO AC8	AC9 TO AC11 IOP 1,2,3 T1,T2 PWR CLR	MB0 TO MB5	MB6 TO MB11	IM0 TO IM8	IM9 TO IM11 SKIP INT AC CLR RUN	DATA ADDR 0 TO 8	DATA ADDR 9 TO 11 ADDR ACK BK, REQ TRD	DATA BIT 0 TO 8	DATA BIT 9 TO 11	DATA BRIG EXT MEM	
	SAME ASSIGNMENTS AS ABOVE										

Module locations 1 through 6 (looking at wiring pin side) in an option mounting panel are reserved for program interrupt cable connections in (or out). Data Break information is assigned to locations 7 through 10, with location 11 available for data break use with extended memory. The lower mounting of the option mounting panel also has slots 1 through 11 reserved, exactly as mentioned above, for cable in (or out).

Cable Selection

The cable recommended for I/O interface connection is 9 conductor coax cable. This cable protects systems from radiated noise and cross talk between individual lines. Coax cable used and sold by Digital has the following nominal specs:

Zo = 95 ± 5
C = 13.75 PF/foot approx.
L = 124 Nhy/foot approx.
R = 0.21 ohm/foot nominal
V = 79% of velocity of light, approx.

Connector Selection

Of the many connectors available in the module product line several have particular application to I/O connections. Price and ordering information is available on these and other connectors in the Digital Logic Handbook. Of particular interest are the following:

W021 — This connector is the standard choice for PDP-8/I when filled with coax cable as shown below.

Connector Description

Single width module having 9 signal pins and 10 ground or shield pins. Pin allocation:

Signal Wires:

D E H K M P S T V

Shield Wires

C F J L N R U

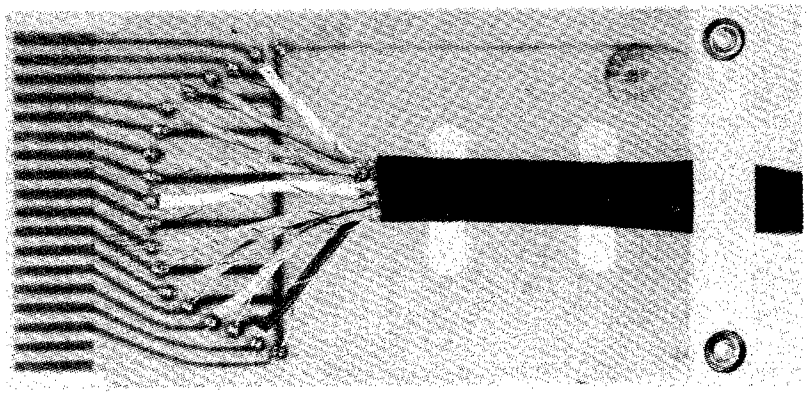


Figure 53.

W011 — Is a shorter version of the W021. It is used on the PDP-8/I or anywhere the cable has to bend sharply immediately after leaving the mounting panel.

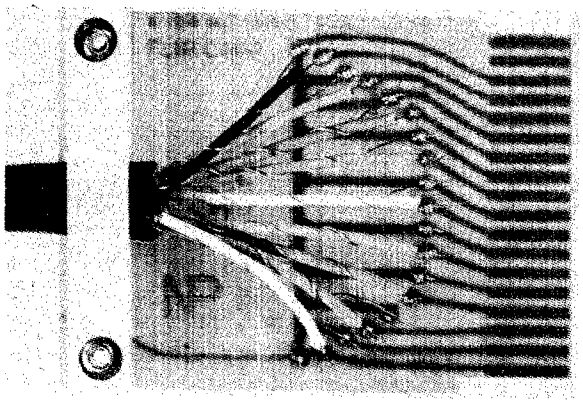


Figure 54.

W022 — The W022 is similar to the W021 except that it comes ready-mounted, 100-ohm, shunt termination resistors (½ watt).

This connector is especially suited for use in terminating coax cables carrying pulses from B684 Bus Drivers.

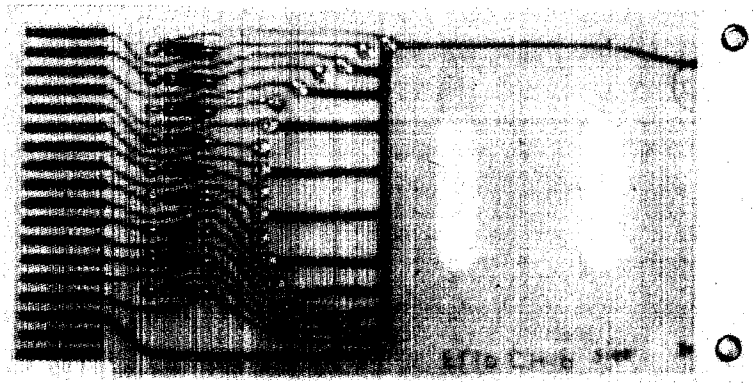


Figure 55.

W028 — In addition to all of the above features of the W021, the W028 provides the user with lugs on which to mount termination networks.

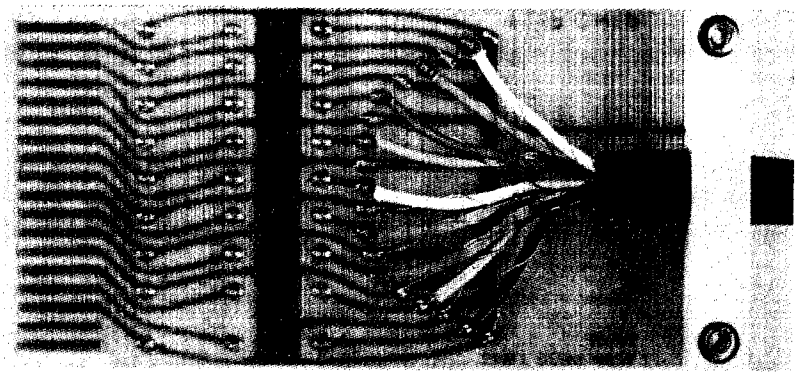


Figure 56.

This connector is well suited for I-O pulse or level transmission requiring small, special terminations such as parallel RC, high frequency RLC filters, non standard resistive terminations, etc.

A recommended cable configuration is coax cable with W011 on the processor end and W028 on the device end. Since termination requirements vary between installation, the W028 can be used as a straight through connector by connecting jumpers across the component terminals, in series with the signal; or, if necessary, with the appropriate terminating components inserted.

Wiring Hints:

These suggestions may help reduce mounting panel wiring time. They are not intended to replace any special wiring instructions given on individual module data sheets or in application notes. For fastest and neatest wiring, the following order is recommended.

(1) All power wiring (pins A, B, and C) and any horizontally bussed signal wiring. Use Horizontal Bussing Strips Type 932.

(2) Vertical grounding wires interconnecting each chassis ground with pin C grounds. Start these wires at the uppermost mounting panel and continue to the bottom panel. Space the wires 2 inches apart, so each of the chassis-ground pins is in line with one of them. Each vertical wire makes three connections at each mounting panel.

(3) All other ground wires. Always use the nearest pin C above the pin to be grounded, unless a special grounding pin has been provided in the module.

(4) All signal wires in any convenient order. Point-to-point wiring produces the shortest wire lengths, goes in the fastest, is easiest to trace and change, and generally results in better appearance and performance than cabled wiring. Point-to-point wiring is strongly urged.

The recommended wire size for use with the H800 Mounting Blocks and 1943 mounting panel is #24 for wire wrap, and #22 for soldering. The recommended size for use with H803 block is #30 wire. Larger or smaller wire may be used depending on the number of connections to be made to each lug. Solid wire and a heat resistant spaghetti (Teflon) are easiest to use when soldering.

ADEQUATE GROUNDING IS ESSENTIAL. IN ADDITION TO THE CONNECTIONS BETWEEN MOUNTING PANELS MENTIONED ABOVE, THERE MUST BE CONTINUITY OF GROUNDS BETWEEN CABINETS AND BETWEEN THE LOGIC ASSEMBLY AND ANY EQUIPMENT WITH WHICH THE LOGIC COMMUNICATES.

When soldering is done on a mounting panel containing modules, steps must be taken to avoid voltage transients that can burn out transistors. A battery- or air-operated tool is preferred, but the filter built into some line-operated tools afford some protection.

Even with completely isolated tools, such as those operated by batteries or compressed air, a static charge can often build up and burn out semiconductors. In order to prevent damage, the wire wrap tool should be grounded except when all modules are removed from the mounting panel during wire wrapping.

Cooling

The low power consumption of R-series modules results in a total of only about 25 watts dissipation in a typical 1943 Mounting Panel with 64 modules. This allows up to six panels of R-series modules to be mounted together and cooled by convection alone, if air is allowed to circulate freely. In higher-dissipation systems using modules in significant quantities from the W, B, and A series, the number of mounting panels stacked together must be reduced. For example, no more than three panels of B-series modules may be mounted together without forced-air cooling. In general, total dissipation from all modules in a convection-cooled system should be 150 watts or less (about 9-amp total current at —15v).

The regulating transformers used in most DEC power supplies have nearly constant heat dissipation for any loading within the ratings of the supply. Power dissipated within each supply will be roughly equal to half its maximum rated output power. If power supplies are mounted below any of the modules in a convection-cooled system, this dissipation must be included when checking against the 150 watt limit.

OCTAIDS AND PANELAIDS FOR INTERFACING TO THE PDP-8/I

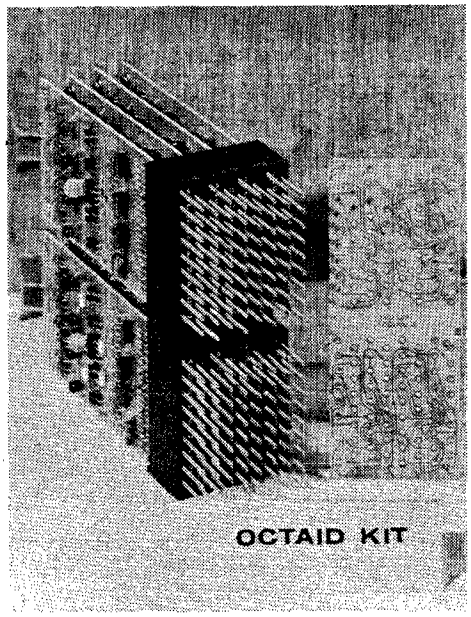
Digital's new OCTAID and PANELAID kits are designed to provide the logic user with an easy-to-assemble, time-saving group of components to achieve common logic functions, such as up-down counting, decoding digital-to-analog and analog-to-digital conversion, and computer interfaces. Standard FLIP-CHIP modules and connectors are used in conjunction with special purpose printed circuit interconnectors. Specification and prices can be readily obtained from the Digital Logic Handbook.

Of particular interest to the interface designer are the following octaids:

Kit D005 Input Buffer Interface
Kit D006 Output Buffer Interface
Kit C005 Bus Interface
Kit C006 Input/Output Buffer Register

The OCTAID series has up to eight standard FLIP-CHIP modules, and the PANELAID series has up to 64 modules. Each kit includes the necessary modules, connectors, and specially designed printed-circuit, back-panel wiring eliminating the necessity for hand-wiring. Since hand-wiring and trouble shooting are eliminated, a significant reduction in the amount of manufacturing time can be achieved.

Input/Output Buffer kits are designed to interface between Digital's PDP-8 or PDP-8/S computers and other OCTAID kits or specially designed systems. PANELAID kits, in general, can be interfaced directly to the PDP-8/I.



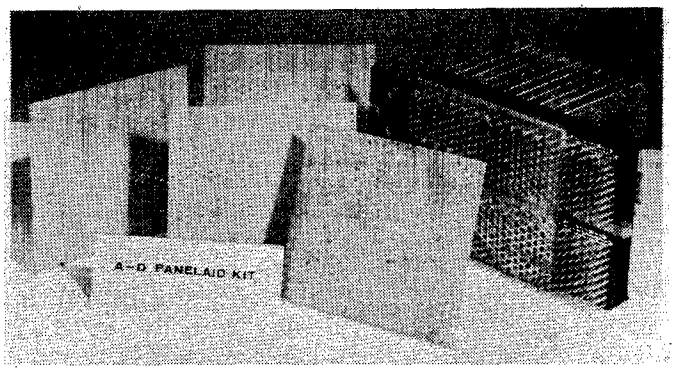


Figure 57.

IOT ALLOCATIONS

Below are listed the 64_8 IOT Device Codes and their allocations. In general, the customer can use any device code which is not currently being used by his particular computer. However, device numbers 30_8 to 37_8 have been allocated specifically for the customer's use. In general, when selecting device codes the following procedure should be used.

1. Use device numbers 30_8 to 37_8 since these do not have options assigned.

2. If more device locations are required, additional blocks should be selected which service options the user will most probably never use; such as, device numbers 40_8 to 47_8 which apply primarily to data communications systems.

This procedure should allow the designer sufficient device codes to implement his interface. For a basic machine, the designer can use all device codes with the exception of 00,03, and 04

In general, the rule is don't use device codes allocated to options which might be added to the equipment at a later date and you cannot use device codes which service options currently installed such as the ASR 33 Teletype.

IOT	OPTION
*00	Interrupt
01	High Speed Reader Type, PR8/I
02	High Speed Punch Type, PP8/I
*03	Teletype Keyboard/Reader
*04	Teletype Teleprinter/Punch
05	Display Type 30N and 338, VC8/I
06	Display Type 30N and 338, VC8/I
07	Display Type 30N and 338, VC8/I, Light Pen Type 370
10	Memory Parity Type MP8/I and Power Failure opt. KP8/I
11	Teletype System Type PT08
12	Teletype System Type PT08
13	Display Type 338
14	Display Type 338
15	Display Type 338
16	Display Type 338
17	Display Type 338
20	Memory Extension Type MC8/I
21	Memory Extension Type MC8/I
22	Memory Extension Type MC8/I
23	Memory Extension Type MC8/I
24	Memory Extension Type MC8/I
25	Memory Extension Type MC8/I
26	Memory Extension Type MC8/I
27	Memory Extension Type MC8/I
30	
31	
32	
33	
34	
35	

IOT	OPTION
36	
37	
40	Teletype System Type PT08 and 680
41	Teletype System Type PT08 and 680
42	Teletype System Type PT08 and 680
43	Teletype System Type PT08 and 680
44	Teletype System Type PT08 and 680
45	Teletype System Type PT08
46	Teletype System Type PT08
47	Teletype System Type PT08
50	Incremental Plotter Type VP8/I
51	Incremental Plotter Type VP8/I
52	Incremental Plotter Type VP8/I
53	General Purpose A/D Conv. and Multiplexer Type AF01A, AF02A, AF03A and AF04A Scanning Digital Voltmeter.
54	General Purpose A/D Conv. and Multiplexer Type AF01A, AF02A, AF03A and AF04A Scanning Digital Voltmeter.
55	D/A Converter Type AA01A
56	D/A Converter Type AA01A
57	D/A Converter Type AA01A, AC01A and AF04A Scanning Digital Voltmeter.
60	Serial Magnetic Drum Type 251 and RM08, 165B PDP 6 Interface and DF32 Disc file.
61	Serial Magnetic Drum Type 251 and RM08, 165B PDP 6 Interface and DF32 Disc file.
62	Serial Magnetic Drum Type 251, RM08 and DF32 Disc file.
63	Card Reader Types CR8/I and 451
64	
65	Automatic Line Printer Type 645
66	Automatic Line Printer Type 645
67	Card Reader Types CR8/I and 451
70	Automatic Mag. Tape Type TC58
71	Automatic Mag. Tape Type TC58
72	Automatic Mag. Tape Type TC58
73	Automatic Mag. Tape Type TC58
74	Automatic Mag. Tape Type TC58
75	
76	DECtape Controls Type TC01
77	DECtape Controls Type TC01

* These device locations are **not** available for customer options.

CHAPTER 13
INTERFACING TECHNIQUES

Until now we have been discussing the transfer of data words in and out of the machine and the use of PI and IOS as aids in servicing devices. Many applications do not necessitate the transfer of 12-bit data words as such; however, it is frequently necessary to monitor and control both electrical and mechanical signals from external devices. These signals could come in the form of serial pulse inputs or contact closures and could require serial output pulses or contact closures. The following discussion will give the interface designer some insight into various techniques of handling inputs and outputs of this type and demonstrate the efficiency and flexibility of the PDP-8/I I/O facility. All of the components discussed are standard off-the-shelf products which are specified in the Digital Logic Handbook C105.

In addition, your local sales office has qualified applications engineers available to help you in implementing these components in the successful design of your interface.

Counting Applications

Frequently data is available in the form of pulse trains or sequential events. The program periodically want to know the number of times these events have occurred during a specified period of time or to modify his program after a certain number of events have occurred. One method, of course, is to count these events in an external counter and periodically read into the Accumulator the contents of the counter. This would be a straightforward Data Input Transfer as previously discussed. This technique requires counting hardware and read in gates. For counts accumulated over a long period or at a slow rate this method is expensive. The alternate approach is to count these events in memory and is easily accomplished by using only the Program Interrupt and IOS facility.

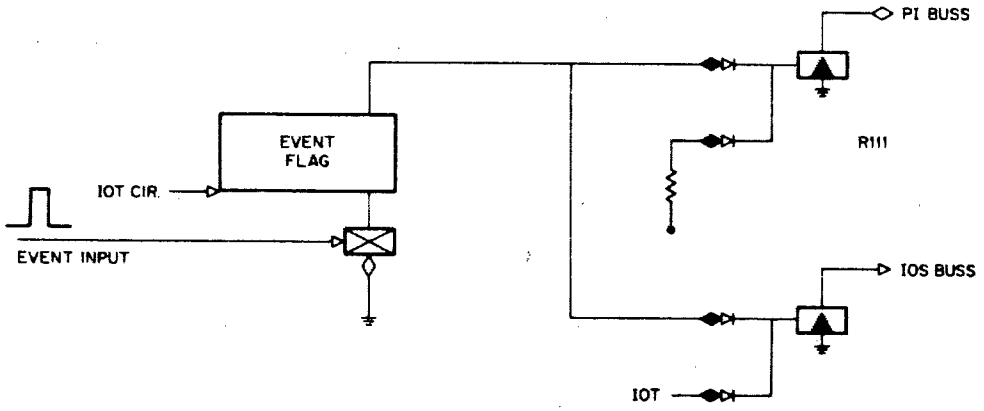


Figure 58.

At each occurrence of an event, an event flag is set thereby causing a program interrupt. After the program has recognized this event by testing the flag with the IOS facility, it issues the appropriate IOT to clear the flag, then adds "one" to the appropriate memory location. This technique allows the use of inexpensive core storage as opposed to hardware counters. Also counts of virtually any length can be accumulated by using several memory locations.

In deciding on whether to use the PI and IOS facility for counting events the designer must take into consideration the speed limitations of the machine and its program. Obviously, we could not count events occurring at a 1 megacycle rate since the time between events is only one microsecond and the computer cycle time is considerably less than that. On the other hand, events at rates of 50 cycles per second can easily be handled by the PI and IOS facility. For very high frequencies external hardware counters are necessary, and for very low frequencies the PI and IOS facility can be used. Frequently trade offs can be made between both methods. For example, if events are occurring at a frequency of between 500KC and 1MC are to be counted for a period of 15 seconds the total count storage requirement is 2^{24}. Since this is too fast for the PI and IOS facility, an external counter is required. However, if instead of building a 24-bit counter, we could build a 12-bit counter. This counter would overflow after 2^{12} counts and cause a Program Interrupt. Overflow counts would be counted in memory since the maximum input frequency to the computer would be $\frac{1 \times 10^6}{2^{12}}$ or approximately 250 cycles/sec. At the end of the 15 sec. period, the 12-bit external counter would be read into the AC as a Data Transfer. Thereby, accumulating the entire count in memory.

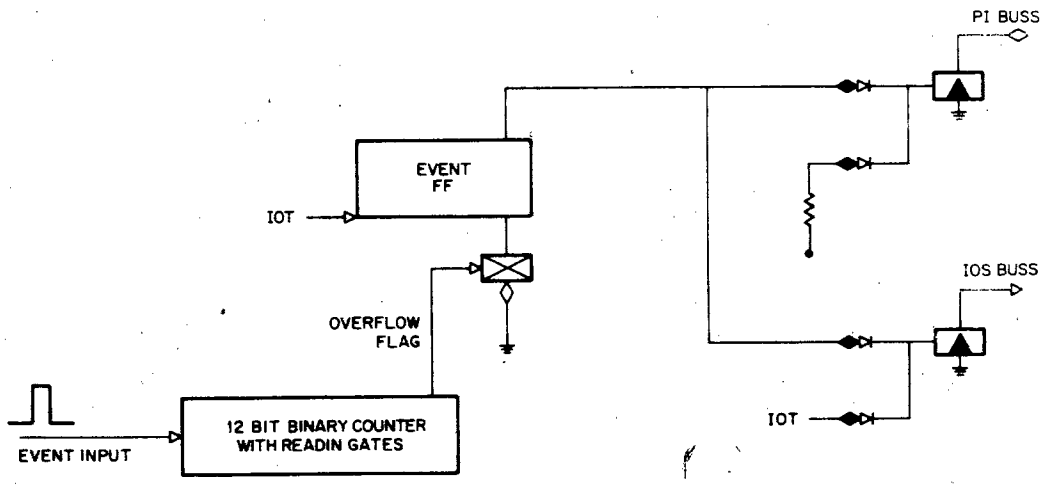


Figure 59.

CONTACT SENSING

Many interfacing applications require the sensing of contact closures. The requirements can be to either periodically sense the condition of a contact or to have the condition of the contact interrupt the computer, such as in an alarm circuit. Let us take the first case where we periodically want to test the conditions of a contact.

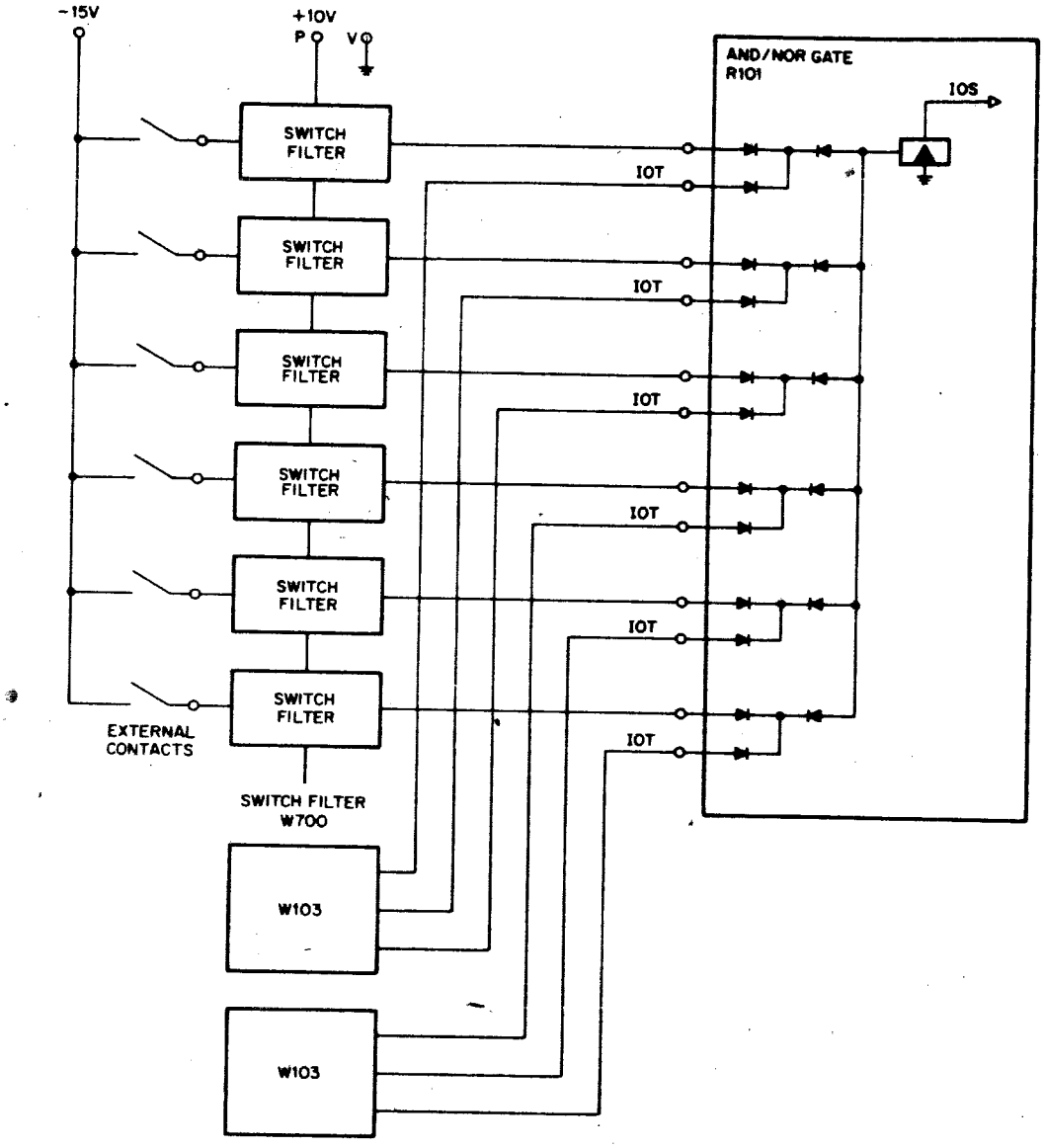


Figure 60.

In Figure 60, we have contact closure inputs which are conditions by W700 switch filters. When a contact is closed the switch filter output is —3V. This signal is sampled by a negative IOT pulse. If the contact is closed, an IOS pulse will be generated.

From this sample, it can be seen that sensing contact closures is relatively inexpensive except that here we require two W103 device selector cards in order to generate 6 separate IOT pulses for each contact.

When large numbers of contacts are used, a common technique is to sample contacts in blocks of 12. This technique saves hardware by greatly reducing the number of W103's required. (Refer to Figure 61.)

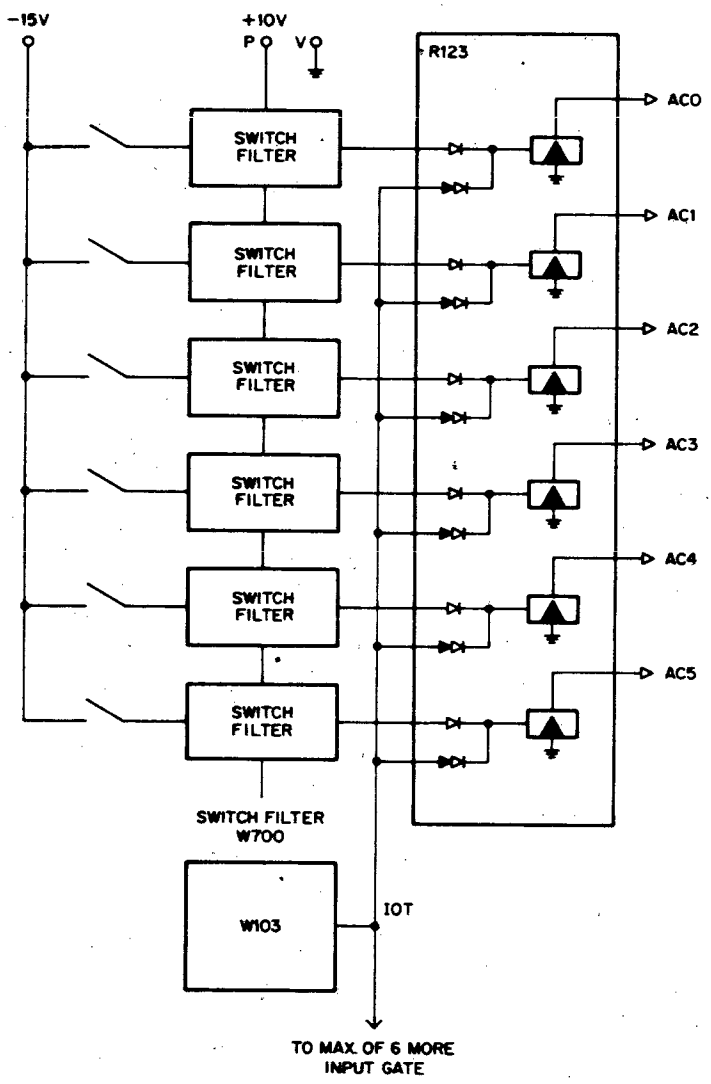


Figure 61.

It is obvious that this technique is essentially an Input Data Transfer using R123's. The accumulator will contain a 12-bit word after a transfer representing the condition of 12 contacts. Separating individual contacts can be accomplished by sequentially rotating each bit of the AC into the LINC and then testing the LINC for a one or a zero.

The second requirement is for the contact to interrupt the computer as in an alarm condition. In this situation, a W501 is used to condition the contact closure. The output of the W501 would be used to set a FF., such as we did when counting events (See Figure 58). These two techniques are identical in terms of hardware required. In this case, however, we would not count the event but would probably modify the program for this alarm condition.

SERIAL OUTPUTS

Serial outputs such as pulses required for stepping switches or stepper motors are easily implemented.

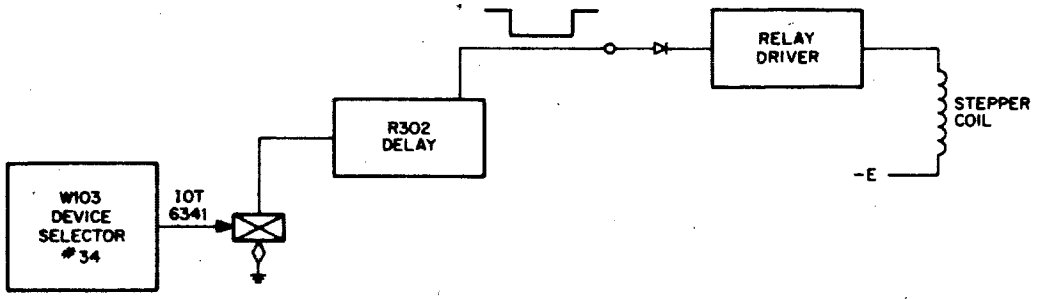


Figure 62.

By issuing the instruction 6341, a pulse is generated by the W103 device selector. This pulse can be made wider with a R302 adjustable delay. Current switching can be accomplished by selecting one of the W Series indicators, relay or solenoid drives. Complete specifications for these modules may be found in the Digital Logic Handbook.

BUFFERED RELAY OUTPUTS

Frequently applications require the control of external functions such as relays, motors or solenoids. Turning these devices on and off can be accomplished two ways. The first, is used primarily when only a few devices must be controlled: an example is shown below.

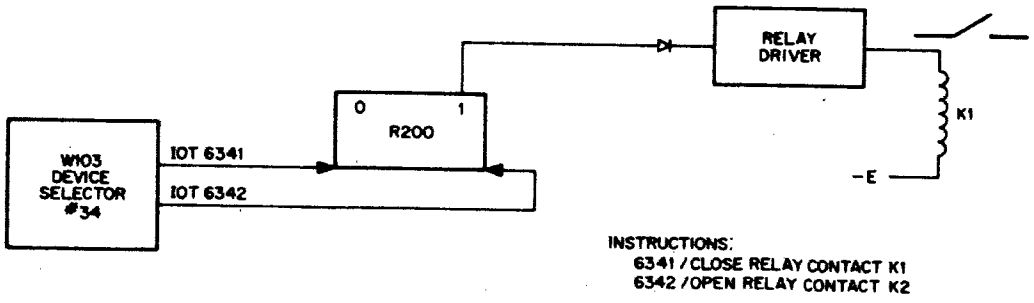


Figure 63.

Programming is simplified since the programmer has one instruction for closing contact K1 and one instruction for opening it. The R200 Flip-Flop, of course, acts as a buffer and will remain in its previous state until changed by an instruction.

It can be seen that implementing this particular interface will become expensive when large quantities of relay outputs are required, also this method will use up device codes quickly since two IOT instructions are required per contact. The alternate method, therefore, is frequently required when large numbers of contact closures are required. This method uses 12-bit accumulator words to control 12 relays at one time. For instance, if we wish to control a bank of 12 relays with only two IOT instructions the system would function as follows:

Each bit in the accumulator would represent the condition of one relay contact.

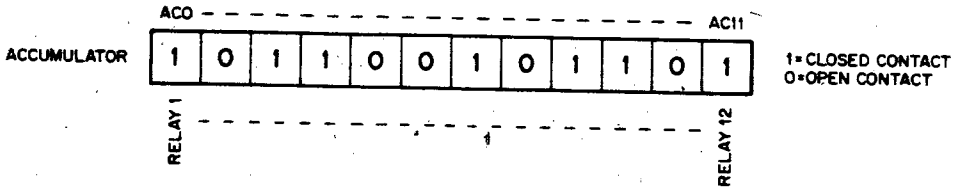


Figure 64.

By transferring this word into a 12-bit output buffer (standard data output transfer) and buffering each bit in the buffer with the appropriate relay driver, all 12 relay contacts may be modified at once using only one W103 device selector and only two IOT pulses.

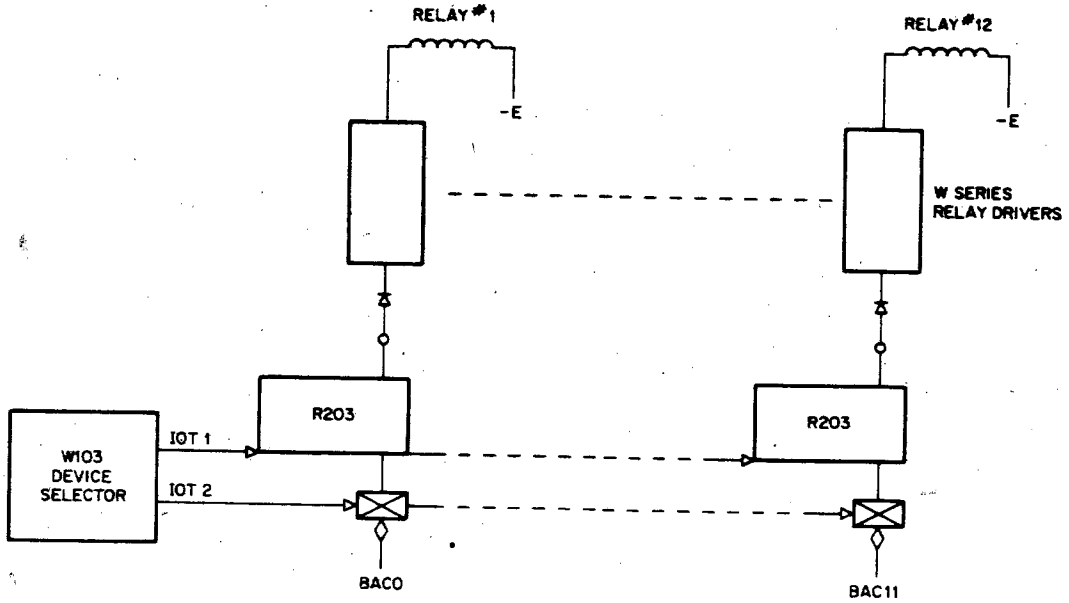


Figure 65.

CHAPTER 14
INTERFACE CONNECTIONS

All interface connections to the PDP-8/I are made at assigned module receptacle connectors in the mounting frame. Capital letters designate horizontal rows of modules within a mounting frame from top to bottom. Module receptacles are numbered from left to right as viewed from the wiring side (right to left from the module side). Terminals of a connector or module are assigned capital letters from top to bottom, omitting G, I, O, and Q.

The module receptacles and assigned use for interface signal connections are:

RECEPTACLE	SIGNAL USE
J05	AC 0-8 inputs
J07	Data Address 0-8 inputs
J09	Data Bit 0-8 inputs
J06	AC 9-11, Skip, Clear AC inputs, Interrupt Request, and Run output
J08	Data Address 9-11 inputs, Address Accepted, B Break outputs, and Memory Increment
J10	Data Bit 9-11 inputs, WC overflow, cycle select, and CA increment
H01	Address Extend 1, 2, 3 inputs and Data Field 0-2 outputs
J01	BAC 0-8 outputs
J03	BMB 0-5 outputs
J02	BAC 9-11, IOTs, BTS1, BTS3, and B Power Clear outputs
J04	BMB 6-11 outputs

Terminals C, F, J, L, N, R, and U of these receptacles are grounded within the computer and terminals D, E, H, K, M, P, S, T, and V carry signals. These terminals mate with Type W011 Signal Cable Connectors at each end of 93-ohm coaxial cable.

Interface connection to the PDP-8/I can be established for all peripheral equipment by making series cable connections between devices. In this manner only one set of cables is connected to the computer and two sets are connected to each device: one receiving the computer connection from the computer itself or the previous device; and one passing the connection to the next device. Where physical location of equipment does not make series bus connections feasible, or when cable length becomes excessive, additional interface connectors can be provided near the computer.

All logic signals passing between the PDP-8/I and the input/output equipment are standard DEC levels or standard DEC pulses. Logic signals have mnemonic names that indicate the condition represented by assertion of the signal. Standard levels are either ground potential (0.0 to −0.3v), designated by an open diamond (—◇) or are −3v (−3.0 to −4.0v), designated by a solid diamond (—◆). Standard pulses in the positive direction are designated by an open triangle (—▷) and negative pulses are designated by a solid triangle (-—▶).

The following tables present cable connections and logic circuit identification information for PDP-8/I interface signals. Computer input signals that must drive the interface bus to ground (data inputs to the AC, Clear AC, Skip, and Interrupt Request) must be connected to the collector of a grounded-emitter transistor, and so can be considered transistor-gated negative pulses () or levels ().

TABLE 5. PROGRAMMED DATA TRANSFER INPUT SIGNALS

Signal	Symbol	Interface Connection	Module Terminal	Module Type
AC 0		J05D2	J13A1	M506
AC 1		J05E2	J13D2	M506
AC 2		J05H2	J13F1	M506
AC 3		J05K2	J13K2	M506
AC 4		J05M2	J13M2	M506
AC 5		J05P2	J13R2	M506
AC 6		J05S2	J14A1	M506
AC 7		J05T2	J14D2	M506
AC 8		J05V2	J14F1	M506
AC 9		J06D2	J14K2	M506
AC 10		J06E2	J14M1	M506
AC 11		J06H2	J14R1	M506
Clear AC		J06P2	J15F1	M506
Interrupt Request		J06M2	J15D2	M506
Skip		J06K2	J15A1	M506

TABLE 6. PROGRAMMED DATA TRANSFER OUTPUT SIGNALS

Signal	Symbol	Interface Connection	Module Terminal	Module Type
BAC 0 (1)		J01D2	H07D2	M650
BAC 1 (1)		J01E2	H07K2	M650
BAC 2 (1)		J01H2	H07S2	M650
BAC 3 (1)		J01K2	H08D2	M650
BAC 4 (1)		J01M2	H08K2	M650
BAC 5 (1)		J01P2	H08S2	M650
BAC 6 (1)		J01S2	H09D2	M650
BAC 7 (1)		J01T2	H09K2	M650
BAC 8 (1)		J01V2	H09S2	M650
BAC 9 (1)		J02D2	H10D2	M650
BAC 10 (1)		J02E2	H10K2	M650
BAC 11 (1)		J02H2	H10S2	M650
BIOP 1		J02K2	H11D2	M650
BIOP 2		J02M2	H11K2	M650
BIOP 4		J02P2	H11S2	M650
BMB 3 (0)		J03K2	H14D2	M650
BMB 3 (1)		J03M2	H14K2	M650
BMB 4 (0)		J03P2	H14S2	M650
BMB 4 (1)		J03S2	H15D2	M650
BMB 5 (0)		J03T2	H15K2	M650
BMB 5 (1)		J03V2	H15S2	M650
BMB 6 (0)		J04D2	H16D2	M650
BMB 6 (1)		J04E2	H16K2	M650
BMB 7 (0)		J04H2	H16S2	M650
BMB 7 (1)		J04K2	H17D2	M650
BMB 8 (0)		J04M2	H17K2	M650
BMB 8 (1)		J04P2	H17S2	M650

TABLE 7. DATA BREAK TRANSFER INPUT SIGNALS

Signal	Symbol	Interface Connection	Module Terminal	Module Type
Data Address 0 (1)		J07D2	J16A1	M506
Data Address 1 (1)		J07E2	J16D2	M506
Data Address 2 (1)		J07H2	J16F1	M506
Data Address 3 (1)		J07K2	J16K2	M506
Data Address 4 (1)		J07M2	J16M1	M506
Data Address 5 (1)		J07P2	J16R2	M506
Data Address 6 (1)		J07S2	J17A1	M506
Data Address 7 (1)		J07T2	J17D2	M506
Data Address 8 (1)		J07V2	J17F1	M506
Data Address 9 (1)		J08D2	J17K2	M506
Data Address 10 (1)		J08E2	J17M1	M506
Data Address 11 (1)		J08H2	J17R2	M506
Data Bit 0 (1)		J09D2	J18A1	M506
Data Bit 1 (1)		J09E2	J18D2	M506
Data Bit 2 (1)		J09H2	J18F1	M506
Data Bit 3 (1)		J09K2	J18K2	M506
Data Bit 4 (1)		J09M2	J18M1	M506
Data Bit 5 (1)		J09P2	J18R2	M506
Data Bit 6 (1)		J09S2	J19A1	M506
Data Bit 7 (1)		J09T2	J19D2	M506
Data Bit 8 (1)		J09V2	J19F1	M506
Data Bit 9 (1)		J10D2	J19K2	M506
Data Bit 10 (1)		J10E2	J19M1	M506
Data Bit 11 (1)		J10H2	J19R2	M506
Break Request		J08K2	J15M1	M506
Transfer Direction		J08M2	J15R2	M506
Increment MB		J08T2	J20A1	M506
Cycle Select		J10K2	J20D2	M506
Increment CA		J08M2	J20F1	M506

*Direction is into PDP-8/I when signal is —3v, out of PDP-8/I when ground potential.

**The Increment MB input to the PDP-8/I must be the output of a gating circuit that enables generation of the ground level signal only when the B Break signal is present.

TABLE 8. DATA BREAK TRANSFER OUTPUT SIGNALS

Signal	Symbol	Interface Connection	Module Terminal	Module Type
BMB 0 (1)		J03D2	H13D2	M650
BMB 1 (1)		J03E2	H13K2	M650
BMB 2 (1)		J03H2	H13S2	M650
BMB 3 (1)		J03M2	H14K2	M650
BMB 4 (1)		J03S2	H15D2	M650
BMB 5 (1)		J03V2	H15S2	M650
BMB 6 (1)		J04E2	H16K2	M650
BMB 7 (1)		J04K2	H17D2	M650
BMB 8 (1)		J04P2	H17S2	M650
BMB 9 (1)		J04D2	H18D2	M650
BMB 10 (1)		J04T2	H18K2	M650
BMB 11 (1)		J04V2	H18S2	M650
B Break		J08P2	H20D2	M650
Address Accepted		J08S2	H20K2	M650
WC Overflow		J10P2	H19S2	M650

TABLE 9. MISCELLANEOUS INP SIGNALS

Signal	Symbol	Interface Connection	Module Terminal	Module Type
ADDR Extension 1		J11D2	ME8K, MC3K	S107, S151
ADDR Extension 2		J11E2	ME8H, MC3E	S107, S151
ADDR Extension 3		J11H2	ME8E, MC3J	S107, S151

TABLE 10. MISCELLANEOUS OUTPUT SIGNALS

Signal	Symbol	Interface Connection	Module Terminal	Module Type
B Run (1)		J06S2	H19D2	M650
TS3 (0)		J02S2	H12D2	M650
TS1 (0)		J02T2	H12K2	M650
Initialize		J02V2	H12S2	M650

MISCELLANEOUS INTERFACE SIGNALS

The following input and output signal connections are available for use with DEC equipment options or for use in special interface equipment designed by the customer.

ADDRESS EXTENSION INPUTS AND DATA FIELD OUTPUTS

When the Memory Extension Control Type MC-8/I is in the computer system, devices using the data break facility must supply a 12-bit data address and a 3-bit address extension. Conversely, the programmed transfer of an address to a register in a device that uses the data break occurs as a 12-bit word from the accumulator and a 3-bit data field extension from the MC-8/I.

The Address Extension 1-3 signals must be ground potential to designate a binary 1 and —3v to designate a binary 0.

B RUN OUTPUT SIGNAL

The binary 1 output of the RUN flip-flop flows to external equipment through the interface circuits. This signal is at —3v when the computer is performing instructions and is at ground potential when the program halts. Magnetic tape and DECtape equipment use this signal to stop transport motion when the PDP-8/I halts, preventing the tape from running off the end of the reel. The B Run signal is routed to the interface connector through a Type M650 Bus Driver module which can drive a 20-ma load.

BTS1 AND BTS3 OUTPUT PULSES

Two buffered timing pulse signals, designated BTS1 and BTS3, are supplied to I/O devices. These signals can synchronize operations in external equipment with those in the computer. The BTS1 and BTS3 pulse signals are derived from the TS1 and TS3 signals generated by the timing signal generator of the PDP-8/I. The Type M650 Level Converter standardizes the TS1 and TS3 pulses as negative pulses. The resulting (buffered) BTS1 and BTS3 pulses are supplied to the interface connections.

INITIALIZE OUTPUT PULSES

The Initialize pulses generated and used within the PDP-8/I are made available at the interface connections. External equipment uses these pulses to clear registers and control logic during the power turn-on period. Use of Initialize pulses in this manner is valid only when the logic circuits cleared by the pulses are energized before or at the same time the PDP-8/I POWER switch is turned on. Operating the KEY START switch also generates the Initialize pulses.

CHAPTER 15
INSTALLATION AND PLANNING

Space Requirements

Access space must be provided at the installation site to accommodate the PDP-8/I and peripheral equipment and to allow access to all doors and panels for maintenance.

The PDP-8/I is available in either a table top configuration or a rack mounted configuration. The rack mounted configuration and peripherals may be purchased completely installed in DEC cabinets or may be purchased unmounted for installation into the customer cabinet. In addition, detailed mounting information has been included for installation of the PDP-8/I into standard BUD and EMCOR racks.

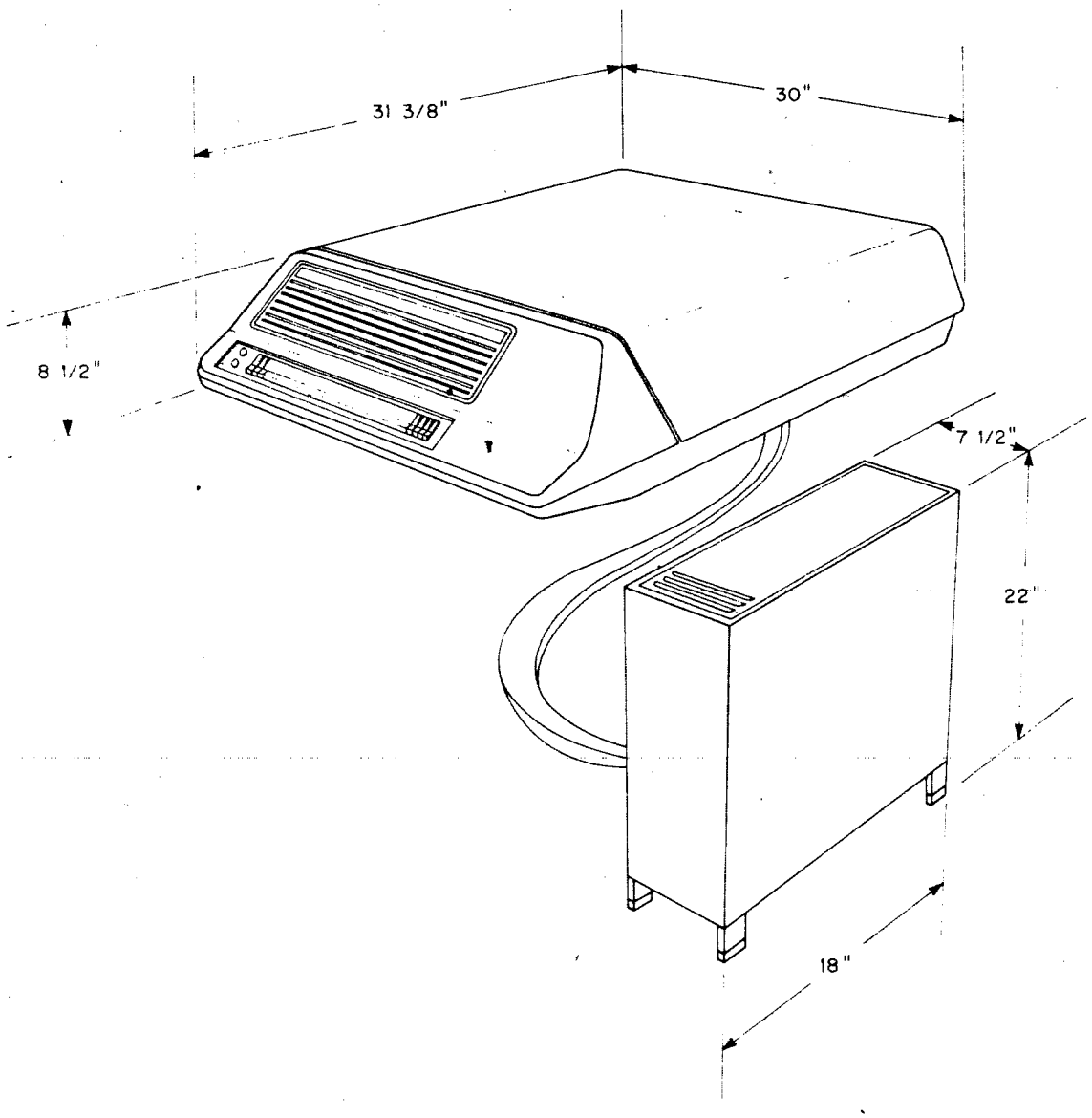


Figure 66. Table Top Mounted PDP-8/I Dimensions

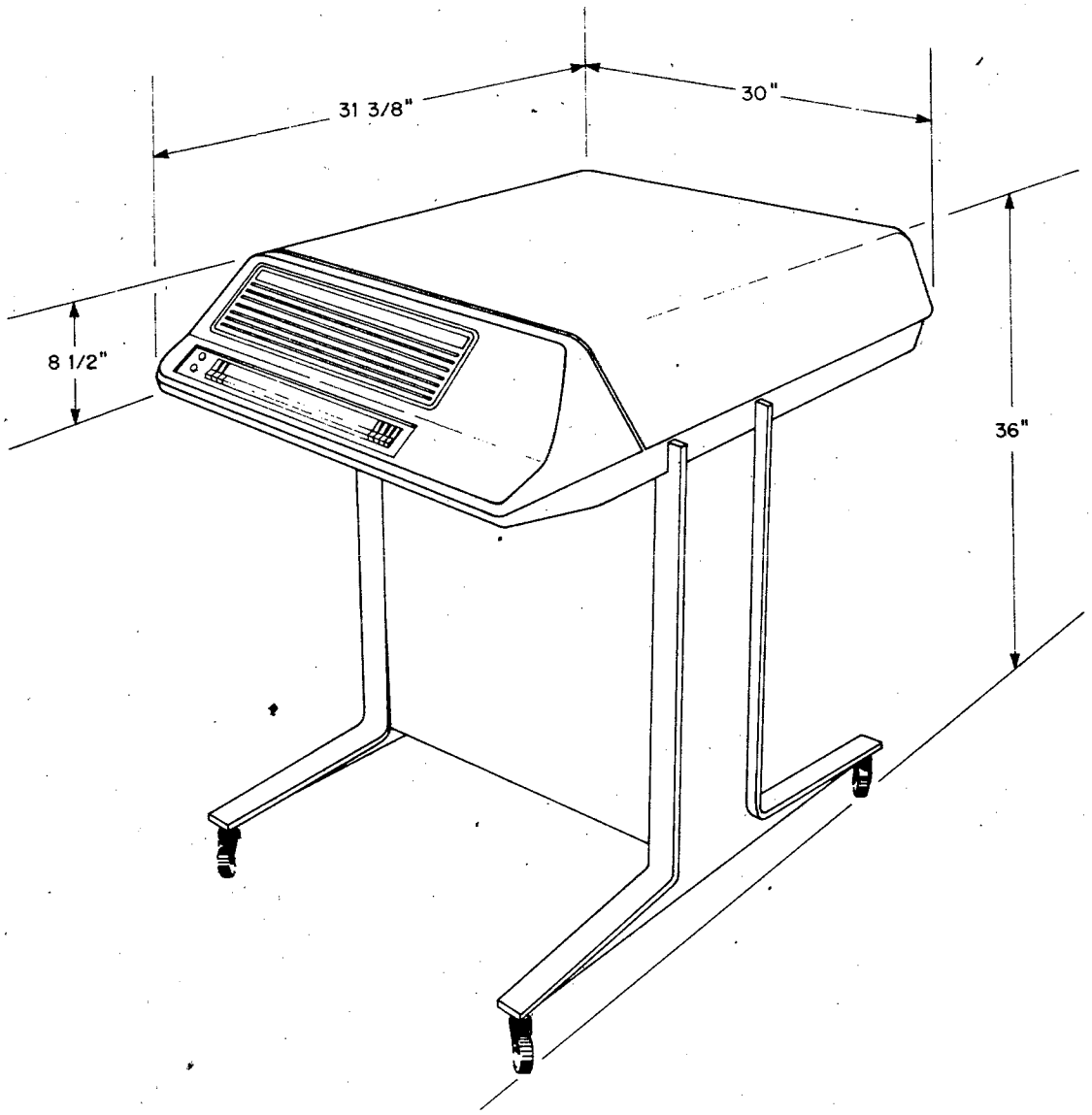


Figure 67. Pedestal Mounted PDP-8/I Dimensions

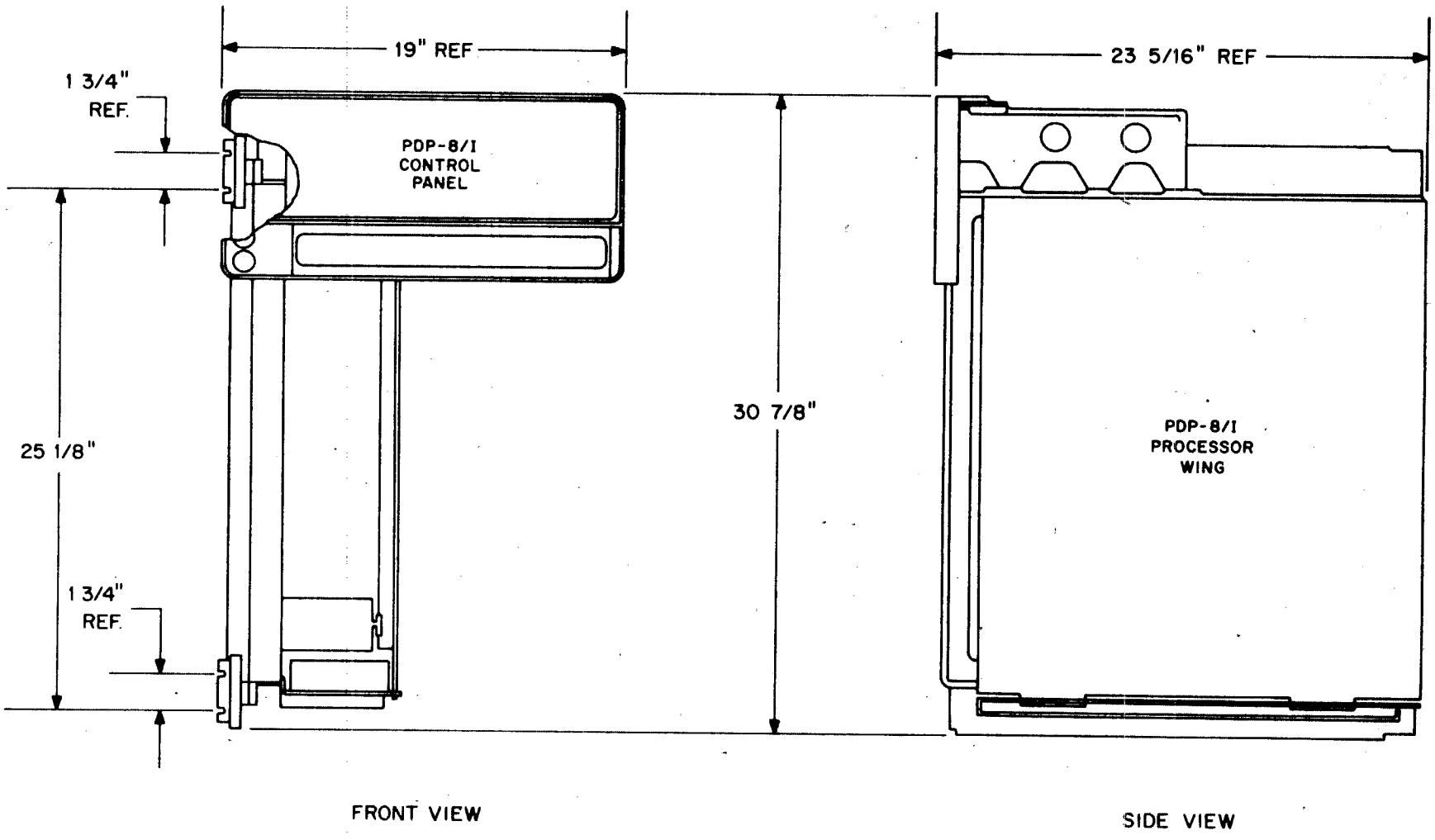


Figure 68. Rack Mounted PDP-8/I Dimensions

VIEW SHOWN WITHOUT FRONT DOOR (SUPPLIED BY BUD IF DESIRED)
AND WITHOUT LOWER COVER PANEL (SUPPLIED BY DEC)

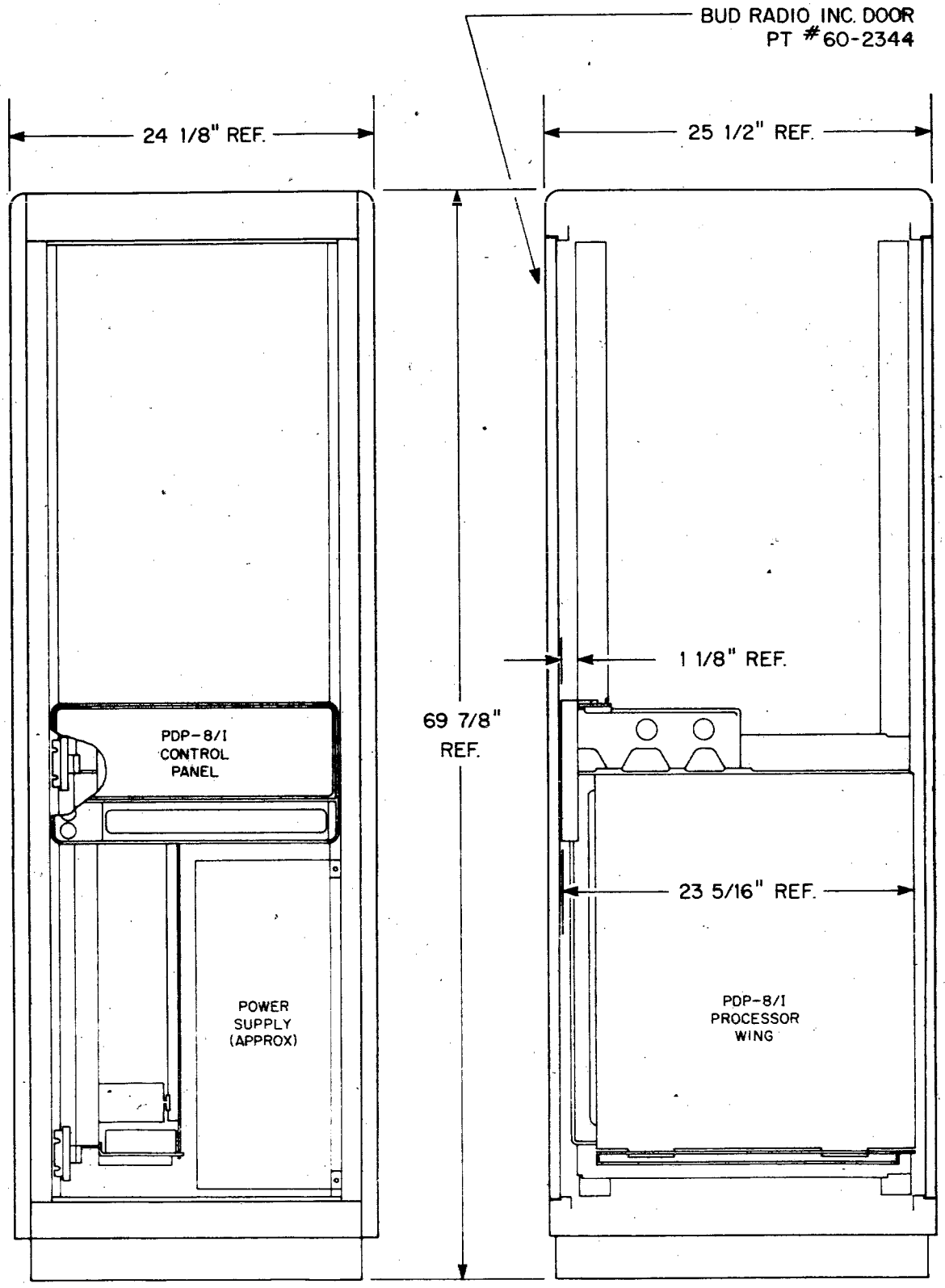


Figure 69. Installation BUD Cabinet PDP-8/I

VIEW SHOWN WITHOUT FRONT DOOR (SUPPLIED BY EMCOR IF DESIRED)
AND WITHOUT LOWER COVER PANEL (SUPPLIED BY DEC)

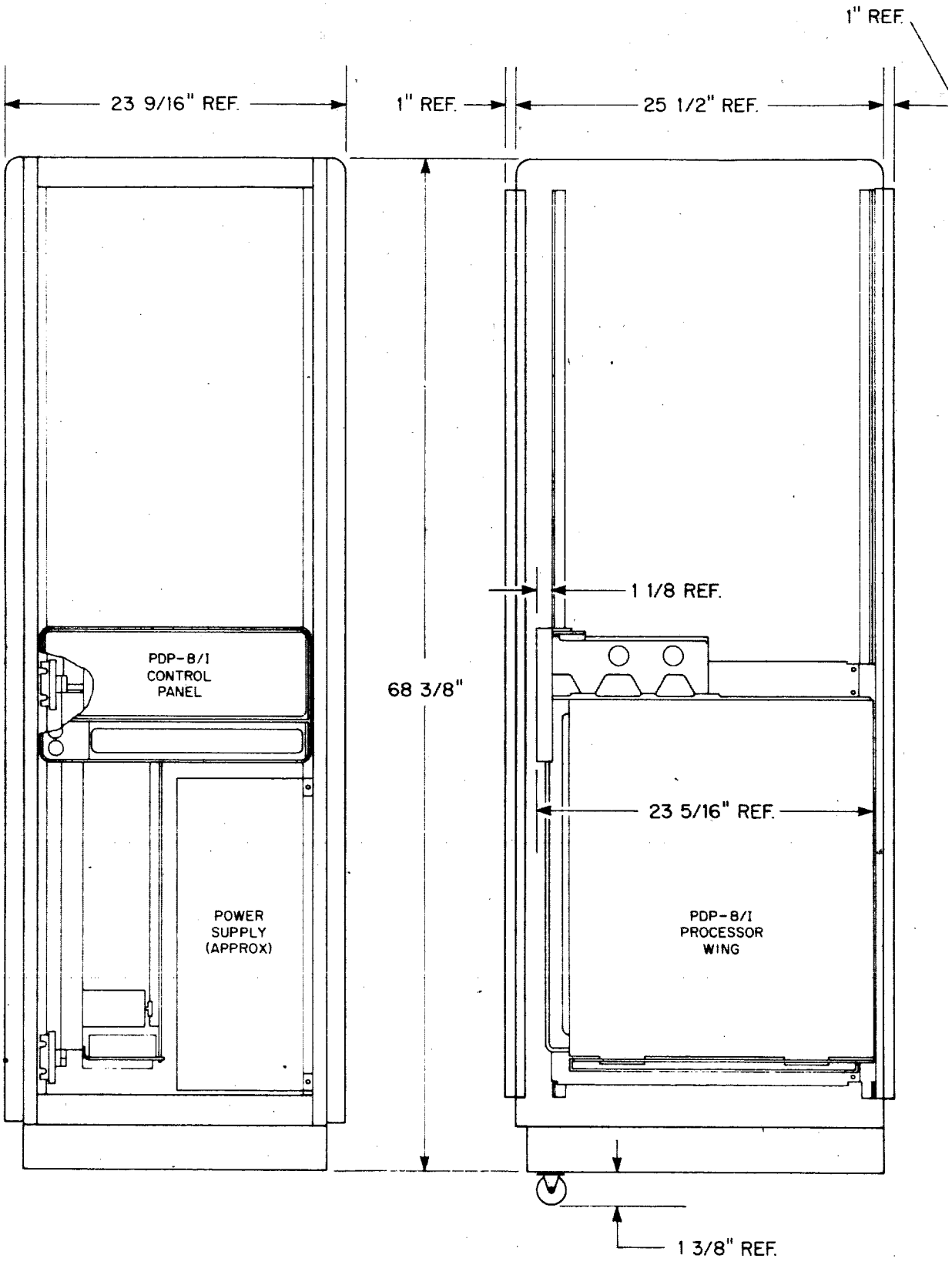


Figure 70. Installation Emcor Cabinet PDP-8/I

Minimum service clearance on all standard DEC computer cabinets is 8¾ inches at the front and 14⅞ inches at the back.

The standard Teletype automatic send receive set requires floor space approximately 22¼ inches wide by 18½ inches deep. Signal cable length restricts the location of the Teletype to within 18 inches of the side of the computer.

Environmental Requirements

Ambient temperature at the installation site can vary between 32 and 130 F (between 0 and 55 C) with no adverse effect on computer operation. However, to extend the life expectancy of the system, it is recommended that the ambient temperature at the installation site be maintained between 70 and 85 F (between 21 and 30 C).

During shipping or storing of the system, the ambient temperature may vary between 32 and 130 F (between 0 and 55 C). Although all exposed surfaces of all Digital cabinets and hardware are treated to prevent corrosion, exposure of systems to extreme humidity for long periods of time should be avoided.

Power Requirements

A source of 115v (±17v), 60-hz (±0.5 hz), single-phase power capable of supplying at least 15 amp must be provided to operate a standard PDP-8/I. To allow connection to the power cable of the computer, this source should be provided with a Hubbell 3-terminal, except for the basic table top PDP-8/I grounded-neutral flush receptacle (or its equivalent). A table-mounted PDP-8/I is provided with a 15-amp power plug; a rack-mounted PDP-8 has a 20-amp twist-lock plug; and systems that draw more than 20 amps use a 30-amp twist-lock plug.

Power dissipation of a standard PDP-8/I is approximately 780w, and the heat dissipation is approximately 2370 Btu/hr. Upon special request, a PDP-8/I can be constructed to operate from a 220v (±33v), 60-hz (±0.5 cps), single-phase power source or from a 100v (±15v), 50-hz (±0.5-hz), single-phase power source.

Cable Requirements

Nine-conductor coaxial cables with Type W011 Cable Connectors provide signal connection between the computer and optional equipment in free-standing cabinets. These cables are connected by plugging the W011 connectors into standard FLIP CHIP module receptacles.

All free standing cabinets require independent 115v receptacles. However, these units may be turned on or off or controlled from the PDP-8/I operator console.

Cables connect to cabinets through a drop panel in the bottom of cabinets. Subflooring is not necessary because casters elevate the cabinets from the floor to afford sufficient cable clearance.

Installation Procedure

During system check-out, customers are invited to visit the Maynard manufacturing facility to inspect and become familiar with their equipment. Computer customers may also send personnel to instruction courses on computer opera-

tion, programming, and maintenance conducted regularly in Maynard, Massachusetts.

DEC's engineers are available during installation and test for assistance or consultation. Further technical assistance in the field is provided by home office design engineers or branch office application engineers.

Table 9 gives installation data to be considered when installing a PDP-8/I. Table 10 lists space requirements. Figure 37, a typical PDP-8 system configuration, can be used as a guide for planning layout.

TABLE 11. INSTALLATION DATA

	Weight (lbs)	Dimensions			Service Clearance		Heat Dissipation Btu/hr	Current (amps)		Power Dissipation (kw)
		Height	Width	Depth	Front	Rear		Nom	Surge	
PDP-8/I Table Top	250	32	21-1/2	21-1/4	—	—	2660	7.5	—	0.78
PDP-8/I Rack Mount	250	31-1/4	19-5/8	21-7/8	22	—	—	7.5	—	0.78
Standard Cabinet CAB8B (Empty)	225	69-1/8	22-1/4	27-1/16	22	15	—	—	—	—
Teletype ASR-33	40	45	23	19	—	—	(included in Standard PDP-8/I)			
Serial Drum 251	500	69-1/8	22-1/4	27-1/16	9	15	1540	5	8	0.45
Card Reader CR8/I	25	8-1/4	18	10	—	—	270	0.57	1	0.06
TU20 & TU20A	400	69-1/8	22-1/4	27-1/16	21	21	4000	6.8	8	0.8
DECtape Transport TU55	35	10-1/2	19-1/2	9-3/4	9	—	410	1	2	0.11
Precision Display 30N	350	49	53	39	—	15	3140	8	8	0.9
Display 338	700	69-1/8	42	51	—	15	2700	10	10	0.8
Random Access Disc File DF32	75	10-1/2	19-1/2	21-1/4	—	—	1700	3	3	0.05
Serial Drum RM08	500	69-1/8	22-1/4	27-1/16	9	15	1540	5	8	0.45
Gen. Purpose A/D Converter and Multiplexer AF01A	55	8-11/16	19	19-1/2	—	—	188	.5	.5	0.06
Guarded Scanning Digital Voltmeter AF04A	300	69-1/8	24-1/4	27-1/16	22	15	2350	6.0	13.2	0.69

TABLE 12. SPACE REQUIREMENTS

Option	MOUNTING PANELS*		Remarks
	Logic	Other	
Memory Extension Control MC8/I	—	—	Mounts within standard PDP-8/I package
Memory Module MM8/I	2	—	Should be mounted in expander cabinet next to PDP-8/I
Automatic Multiply-Divide KE8/I	—	—	Mounts within standard PDP-8/I package
Memory Parity MP8/I	—	—	Mounts within standard PDP-8/I package
Automatic Restart KP8/I	—	—	Mounts within standard PDP-8/I package
Data Channel Multiplexer DM01	—	2	
Perforated Tape Reader PR8/I	—	10½ in. front panel	The control logic for these 5 options mount within standard PDP-8/I package
Perforated Tape Punch PP8/I	—	10½ in. front panel	
Oscilloscope Display VC8/I	—	10 in. front panel	
Incremental Plotter VP8/I	—	Table space needed for plotter	
Card Reader CR8/I	—	Table space needed for reader	
Light Pen 370	—	—	Logic and power supply included in VC8/I or 30N
DCS 680	—	—	
DL8/I	—	—	
685	2	—	
682	2	—	
Teletype System PT08	1	—	Handles up to 2 Teletypes or 2 Data-Phones
Magnetic Tape Control TC58	4	—	Controls up to 8 tape units (TU20 or TU20A) industry-compatible (or IBM-compatible)
DECtape Control TC01	3	—	Controls up to 8 TU55 DECtape Transports

*Mounting Panels are standard DEC 5¼ in. high logic panels.

NOTE: Power supplies for option logic are normally mounted on rear door of DEC cabinets. Customers using their own cabinets should allow additional space for power supplies.

SYSTEM CONFIGURATIONS

PDP-8/I systems are mounted in standard DEC cabinets. The basic cabinet contains the processor and power supply. There are three 10½ inch spaces above the PDP-8/I Control Panel on the basic cabinet. These spaces are numbered consecutively starting just above the control cabinet. Each space has options assigned in a fixed priority. See following figure.

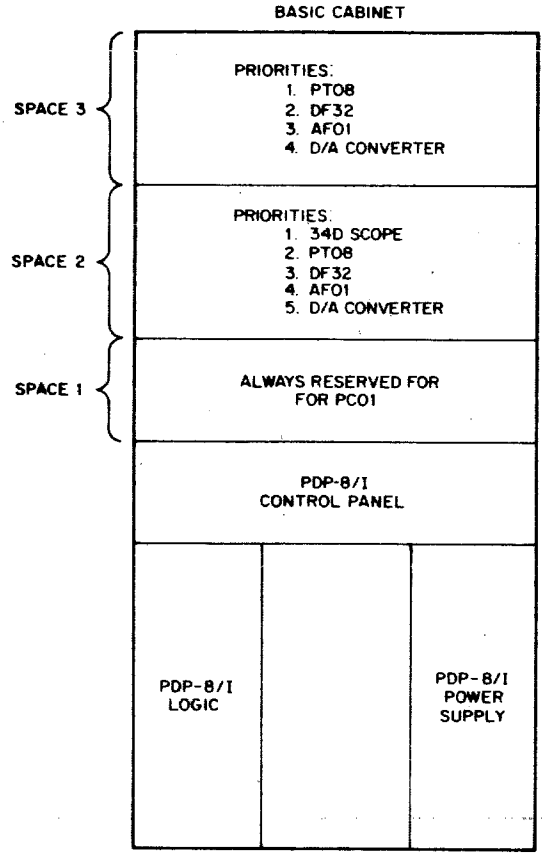


Figure 71. Priority Assortment, Basic Cabinet

When systems require additional cabinet space, an Option Cabinet must be added. This cabinet is mounted to the left of the Basic Cabinet (front view) with option priorities assigned per figure 72.

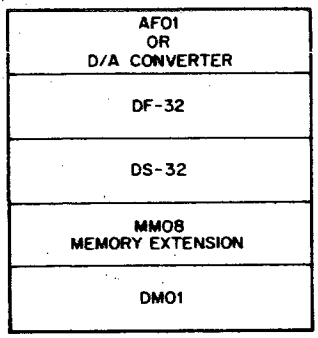


Figure 72.

Certain PDP-8/I options require separate cabinets. These options are as follows:

1. DECtape Cabinet: Can contain TC01, 3 TU55's, and DM01. Mounts to right of Basic Cabinet.
2. TC58 Cabinet: Can contain TC58 and DM01 at bottom. Mounts to right of Basic Cabinet.
3. TU20 Cabinet: Contains TU20. Mounts to right of Basic Cabinet.
4. Drum Cabinet: Contains RM08 or 251. Mounts to right of Basic Cabinet.
5. Communication System Cabinet: Contains 680 and 637. Mounts to left of Basic Cabinet.
6. Basic Typesetting Cabinet: Contains PDP-8/I, PA60A R/P interface, PA-61A R/P control, space for second PA-61A R/P control.
7. AFO4A Cabinet Contains Guarded Scanning Digital Voltmeter System (AFO4A)

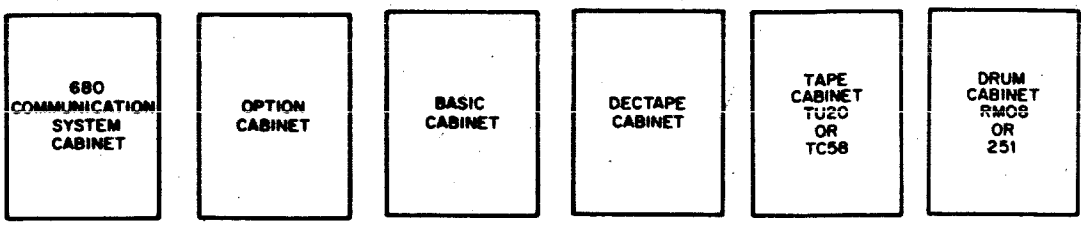


Figure 73. Cabinet configurations for expanded system

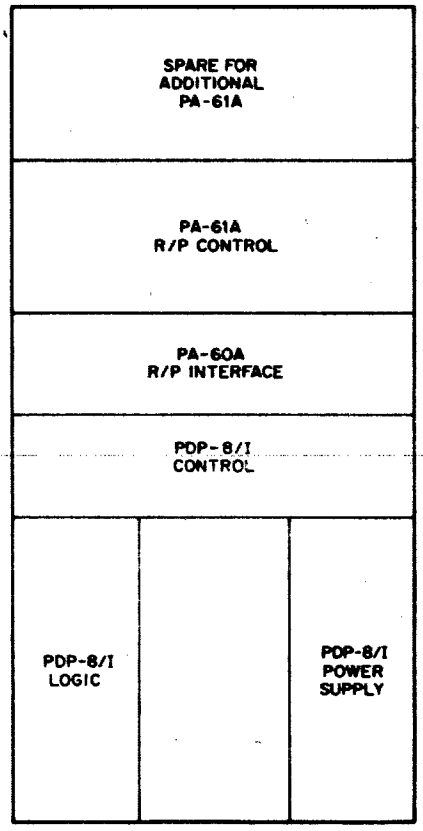


Figure 74. Basic Typesetting Cabinet

APPENDIX 1
PROGRAM ABSTRACTS

Family-of-8 Programs

The PDP-8/I is delivered to the user complete with an extensive selection of system programs and routines making the full data processing capability of the new computer immediately available to each user, eliminating many commonly experienced initial programming delays.

The programs described in these abstracts come from two sources, past programming efforts on the PDP-5, PDP-8, PDP-8/s and present and continuing programming effort on the PDP-8/I. Thus the programming system takes advantage of the many man-years of program development and field-testing by Digital computer users. There are over 1000 Family-of-8 systems in the field already.

Although in many cases PDP-8/I programs originated as PDP-5, PDP-8 and PDP-8/s programs, all utility and functional program documentation is issued anew, recursive format introduced with the Family-of-Eight computers. Programs, written by users of the PDP-5, the PDP-8, the PDP-8/s or the PDP-8/I, and submitted to the DECUS library (DECUS-Digital Equipment Corporation User's Society) are immediately available to Family-of-Eight users. Consequently users of all Family-of-Eight computers can take advantage of continuing program developments.

System Programs

DEC-08-ESAA-D
Symbolic Editor

The Symbolic Editor program is used to generate, edit, correct, and update symbolic program tapes using the tape teleprinter. With the Editor in memory, the user reads in portions of his symbolic tape, removes, changes, or adds instructions and operands, and gets back a new, complete, symbolic tape with errors removed. He can work through the program instruction by instruction, spot check it, or concentrate on new selections. The tape can contain either symbolic machine language, FORTRAN source statement, data, or text information. This program is available for use with either the 33ASR reader/punch or the high speed reader/punch.

DEC-08-AFAB-D
FORTRAN II Manual

One-pass FORTRAN compiler and operating system compiles FORTRAN source language statements into an object program tape. The operating system executes the program. This system contains the interpreter, arithmetic function subroutines, and input-output packages.

DEC-08-ASAA-D
PAL III (Program Assembler Language)

Symbolic machine language assembler. Converts programs code in symbolic machine language to binary machine language. The basic process performed by the Assembler is the substitution of numeric values for symbols, according to associations defined in the symbol table. In addition, the user may request

that the Assembler itself assign values to the user's own symbols at assembly time. These symbols are normally used to name memory locations, which may then be referenced by name. An assembly listing may be produced.

Digital-8-4-S
DDT

Dynamic Debugging Tape provides a means for on-line program debugging at the symbolic or mnemonic level. By typing commands on the console teleprinter, memory locations can be examined and changed, program tapes can be inserted, selected portions of the program can be run, and the updated program can be punched.

Digital-8-5-S
Floating-Point System

A Basic System
B Interpreter, I/O, I/O Controller
C Interpreter, I/O Functions
D Interpreter, I/O, I/O Controller, Functions
Includes Floating-Point Interpreter and I/O subsystems. Allows the programmer to code his problem in floating-point machine language.

Floating-point operations automatically align the binary points of operands, retaining the maximum precision available by discarding leading zeros. In addition to increasng accuracy, floating-point operations relieve the programmer of the scaling problems common in fixed-point operations. This system includes elementary function subroutines programmed in floating-point. These subroutines are sine, cosine, square root, logarithm, arctan, and exponential functions. Data being processed in floating-point is maintained in three words of memory (12-bit exponent, 24-bit mantissa). An accuracy of seven decimal places is maintained.

DEC-08-AFAZ-PB
FORTRAN Symbol Print

Loaded over the FORTRAN Compiler, this program lists the variables used and where they will be located in core. It also indicates the section of core not used by the compiled program and data.

DEC-08-SUAO-D
DECtape Programming Manual

The DECtape Library System is loaded by a 17_{10} instruction bootstrap routine that starts at 7600_8. This loader calls a larger program into the last memory page, whose function is to preserve on tape the contents of memory from 6000_8-7577_8, and then to load the INDEX program and the directory into those same locations. Since the information in this area of memory has been preserved, it can be restored when operations have been completed. The skeleton system tape contains the following programs:

INDEX—Typing this causes the names of all programs currently on file to be typed out.

UPDATE—Allows the user to add a new program to the files. UPDATE queries the operator about the program's name, its starting address, and its location in core memory.

GETSYS—Generates a skeleton library tape on a specified DECtape unit.

DELETE—Causes a named file to be deleted from the tape.

Starting the skeleton library tape, the user can build up a complete file of his active programs and continuously update it.

DEC-08-CMAA-D
MACRO-8

The MACRO-8 Symbolic Assembler accepts source programs written in symbolic language and translates them into binary form in two passes. MACRO-8 produces an object program tape (binary), a symbol table (for use with DDT), an octal symbolic assembly listing, and useful diagnostic messages. MACRO-8 is compatible with PAL III, and has the following additional features: user-defined macros; double precision integers, floating-point constants, arithmetic and Boolean operators, literals, text facilities, and automatic Link generation.

Digital-8-10-S
CALCULATOR

CALCULATOR is an equation evaluation routine. It differs from FORTRAN in that the function to be evaluated is entered via keyboard and calculated immediately upon termination of entry. Format control is provided so that computer results may be conveniently tabulated. Expressions causing the calling of common function subroutines are included.

Digital-8-11-S
DATAK

The DATAK system permits a complex, program-controlled data acquisition system to be adapted to a particular experimental environment through the use of a sophisticated and concise pseudo code. In addition to data-acquisition applications, DATAK furnishes the experimenter with a means of calibrating transducers and is a powerful aid in troubleshooting a complex data-gathering system. Paper tape output produced is acceptable as FORRAN input.

DEC-08-COAA-D(L)
ODT-II

ODT-II (Octal Debugging Tape) acts in debugging a program by facilitating communication with the program being run via the ASR 33 Teletypewriter. ODT-II features include register examinations and modification, control transfer, word searching, octal dumping, and instruction traps.

Digital-8-13-S
One-Dimensional Display and Analysis

The one-dimensional pulse-height analysis program is used to read in and analyze 1024-channel energy spectra data. The program receives and executes commands from the keyboard. These commands start and stop data taking and determine into which data region it goes, displays the data with markers, allows area of interest on the display screen to be expanded, integrates between markers, writes out data, punches out data, and controls background subtraction.

Digital-8-14-S
Multiparameter Display and Analysis

The two-dimensional pulse-height analysis program is used to read in and

analyze two-parameter energy and spectra data. The program receives and executes commands from the keyboard. These commands start and stop data taking, control the displays, and control wiring and punching of the data. The displays available are: isometric, vertical and horizontal slicing, differential and integral contours, and "twinkle box." The program is flexible with respect to the dimensions of the data matrix.

Digital-8-15-S
Oceanographic Analysis

This program represents the basic accepted physical oceanography method for the reduction of data concerning depth, temperature, and salinity measurements of the water column.

This program has been designed to allow the field oceanographer a rapid means of immediately calculating Sigma-T, anomaly of specific volume, and sound velocity following a Nansen cast whereby he may examine in detail the results of his endeavor, to determine not only the structure of the environment he has just sampled but also to check the validity of his measurements.

In addition to the above, an interpolation routine is incorporated into the program as well as a depth integration of the anomaly of specific volume.

Digital-8-16-S
Master Tape Duplicator/Verifier

The tape duplicator is a single-buffered read and punch program, utilizing the program interrupt. It computes a character count and checksum for each tape and compares with checks at the end of the tape.

Digital-8-35-S
A 680 5-Bit Character Assembly Subroutines
B 680 8-Bit Character Assembly Subroutines

These subroutines concentrate Teletype data by assembling serial-bit data into 5-bit (8-35-S-A) or 8-bit (8-35-S-B) characters and presenting the user with line number and character data. They also add start and stop bits and transmit characters serially. Full-duplex lines are assumed, but the subroutines will work the half-duplex if the user handles the expected echo.

Elementary Function Routines

The following routines are described in the Program Library Math Routines Manual (DEC-08-FFAA-D).

Square Root Subroutine-Single Precision

Forms the square root of a single-precision number. An attempt to take the square root of a negative number will give 0 for a result.

Signed Multiply Subroutine-Single Precision

Forms a 22-bit signed product from 11-bit signed multiplier and multiplicand.

Signed Divide Subroutine-Single Precision

This routine divides a signed 11-bit divisor into a signed 23-bit dividend

giving a signed 11-bit quotient and a remainder of 11 bits with the sign of the dividend.

Double-Precision Multiply Subroutine-Signed

This subroutine multiplies a 23-bit signed multiplicand by a 23-bit signed multiplier and returns with a 46-bit signed product.

Double-Precision Divide Subroutine-Signed

This routine divides a 23-bit signed divisor into a 47-bit signed dividend and returns with a 23-bit signed quotient and a remainder of 23 bits with the sign of the dividend.

Sine Routine-Double Precision

The Double-Precision sine subroutine evaluates the function Sin (X) for $-4<X<4$ (X is in radians). The argument is a double-precision word, 2 bits representing the integer part and 21 bits representing the fractional part. The result is a 23-bit signed fraction $-1<\text{Sin}(X)<1$.

Cosine Routine-Double Precision

This subroutine forms the cosine of a double-precision argument (in radians). The input range is $-4<X<4$.

Four-Word Floating-Point Package

This is a basic floating-point package that carries data as three words of mantissa and one word of exponent. Common arithmetic operations are included as well as basic input/output control. No functions are included.

Logical Subroutines

Subroutines for performing the logical operations of inclusive and exclusive OR are presented as a package.

Shift Right, Shift Left Subroutines (Single and Double Precision)

Four basic subroutines, shift right and shift left, each at both single and double precision, are presented as a package.

Logical Shift Routines

Two basic subroutines, shift right at both single and double precision, are presented as a package. The shifts are logical in nature.

Digital-8-21-F
Signed Multiply (Uses EAE) Single Precision

This subroutine forms a 22-bit signed product from an 11-bit signed multiplier and multiplicand using the Extended Arithmetic Element. It occupies less storage and takes less time to execute than its non-EAE counterpart, and it has the same calling sequence.

Digital-8-22-F
Signed Divide (Uses EAE) Single Precision

This subroutine divides a double-precision signed 22-bit dividend by a signed 11-bit divisor, producing a signed 11-bit quotient and a remainder of 11 bits having the sign of the dividend.

It makes use of the Extended Arthmetic Element instruction set and occupies less storage and takes less time to execute than its non-EAE counterpart. It has the same calling sequence except that the subroutine name is changed from DIVIDE to SPDIV.

Digital-8-23-F
Signed Multiply (Uses EAE) Double Precision

This subroutine multiplies a 23-bit, signed 2's complement binary number by a 23-bit signed 2's complement binary number, giving a 46-bit product with two signs on the higher order end. It makes use of the Extended Arithmetic Element instruction set and, because of this, occupies less storage and takes less time to execute than its non-EAE counterpart. Its calling sequence is comparable with the non-EAE version.

Digital-8-25-F
EAE Floating-Point Package

These packages perform the same tasks as the Floating-Point Packages (Digital-8-5-S A, B, C, D) except that certain routines have been speeded up by the use of the Extended Arithmetic Element.

For a detailed description of floating-point arithmetic and the Interpretive Floating-Point Packages, the reader is referred to Digital-8-5-S.

Utility Programs

Digital-8-0
Format for Program Documentation

With the advent of the PDP-8, Digital Equipment Corporation introduced a new, recursive format for program documentation. This format is used for routines and subroutines, such as utility and functional, but not necessarily for system programs.

This format and its use are described in this document.

Digital-8-1-U
Read-In-Mode Loader

The RIM Loader is a minimum-sized routine for reading and storing the information in Read-In-Mode coded tapes via the ASR 33 Perforated Tape Reader.

Digital-8-2-U
Binary Loader (33ASR, PR-8/I, MC-8/I Memory Extension)

The Binary Loader is a short routine for reading and storing the information in binary-coded tapes via the ASR 33 Perforated Tape Reader or by means of the Type PR-8/I High-Speed Perforated Tape Reader.

The Binary Loader will accept tapes prepared by the use of PAL (Program Assembly Language; see DEC-08-ASAA-D) or MACRO-8 (see DEC-08-CMAA-D). Diagnostic messages may be included on tapes produced when using either PAL or MACRO. The Binary Loader will ignore all diagnostic messages.

Digital-8-3-U
DECtape Library System Loader

The use of the DECtape Library System Loader is discussed. Certain conventions with respect to last page storage are established for this loader as well as for the Read-In-Mode and Binary Loaders.

DEC-08-PMPA-D
RIM Punch

This program provides a means of punching out the information as selected blocks of core memory as RIM-coded tape via the ASR 33 Perforated Tape Reader.

Digital-8-5-U-SYM
Binary Punch 33/PP-8/I

This program provides a means of punching out the information in selected blocks of core memory as binary-coded tape via the ASR 33 Perforated Tape Punch or via the High-Speed Punch PP-8/I.

Digital-8-6-U-SYM
Octal Memory Dump

This routine reads the console switches to obtain the upper and lower limits of an area of memory, then types on the Teletype an absolute address plus the octal contents of the first four words specified and repeats this until the block is exhausted, at which time the user may repeat the operation.

Digital-8-10-U
Binary-Coded-Decimal to Binary Conversion Subroutine

This basic subroutine converts unsigned binary-coded-decimal numbers to their equivalent binary values.

Digital-8-11-U
Double Precision BCD-to-Binary by Radix Deflation

This subroutine converts a 6-digit BCD number to its equivalent binary value contained in two computer words.

Digital-8-12-U
Incremental Plotter Subroutine

This subroutine moves the pen of an incremental plotter to a new position along the best straight line. The pen may be raised or lowered during the motion.

Digital-8-14-U
Binary to Binary-Coded-Decimal Conversion

This subroutine provides the basic means of converting binary data to binary-coded-decimal (BCD) data for typeout, magnetic tape recording, etc.

Digital-8-15-U-SYM
Binary-to-Binary-Coded-Decimal Conversion (Four Digit)

This subroutine extends the method used in Digital-8-14-U so that binary integers from 0 to 4095 in a single computer word may be converted to four binary-coded-decimal characters packed in two computer words.

Digital-8-17-U
EAE Instruction Set Simulator

This routine permits the automatic multiply-divide hardware option to be simulated on a basic PDP-8/I.

Digital-18-U-SYM
Alphanumeric Message Typeout

This is a basic subroutine to type messages packed in computer words. Two 6-bit characters are packed internally in a single word. All ASR 33 codes from 301 to 337 and from 240 to 277 (excepting 243 and 245) can be typed. The

typing of line feed (code 212) and carriage return (code 215) are made possible by arbitrarily assigning internal codes of 43 and 45, respectively, to represent these characters, thus preventing the output of ASCII codes 243 (#) and 245 (%).

Digital-8-19-U-SYM
Teletype Output Subroutines

A group of subroutines useful in controlling ASR 33 output is presented as a package. Provision is made for simulation of tabulation stops. The distance "tabbed" may be controlled by the user. Characters whose ASR 33 codes are in the groups 241 through 277, inclusive, and 300 through 337, inclusive, are legal. Space, carriage return then line feed, and tabulation are provided via subroutines.

Digital-8-20-U-SYM
Character String Typeout Subroutine

This basic subroutine types messages stored internally as a "string" of coded characters. All ASR 33 characters are legal.

Digital-8-21-U-SYM
Symbolic Tape Format Generator

The Format generator allows the user to create PDP-8 symbolic tapes with Formatting. It may be used to condense tapes with spaces by inserting tabs, or merely to align tabs, instructions, and comments.

Digital-8-22-U-SYM
Unsigned Decimal Print

This subroutine permits the typeout of the contents of a computer word as a 4-digit, positive, decimal integer.

Digital-8-23-U-SYM
Signed Decimal Print, Single Precision

This subroutine permits the typeout of the contents of a computer word as a signed twos complement number. If bit 0 of the computer word is a "1", the remaining bits represent a negative integer in twos complement form; if bit 0 equals "0", the remaining bits represent a positive integer. If the number is negative, a minus sign is printed; if positive, a space.

Digital-8-24-U-SYM
Unsigned Decimal Print, Double Precision

This subroutine permits the typeout of a double-precision integer stored in the usual convention for double-precision numbers, (see DEC-08-FFAA-D). The one exception is that all 24 bits are interpreted as magnitude bits (i.e. the bit "0" of the high-order word is not a sign bit). The typeout is in the form of a seven-digit, positive, decimal integer.

Digital-8-25-U-SYM
Signed Decimal Print, Double Precision

Ths subroutine permits the typeout of the contents of two consecutive computer words as one signed, double-precision, twos complement number. If bit 0 of the high order word is a "1," the remaining 23 bits represent a negative integer in twos complement form; if bit 0 equals "0," the remaining bits represent a positive integer. If the number is negative, a minus sign is printed; if positive, space.

DEC-08-SUAO-D
DECtape Programming Manual

Allows the programmer to read, write, or search DECtape using prewritten and tested subroutines. A series of subroutines which will read or write any number of DECtape blocks, read any number of 129-word blocks as 128 words (or one memory page), or search for any block (used by read and write, or to position the tape). These programs are assembled with the user program and are called by a jump to subroutine instruction. The program interrupt detects the setting of the DECtape (DT) flag, allowing the main program to proceed while the DECtape operation is being completed. A program flag is set when the operation is completed. The program thus effectively allows concurrent operation of several input/output devices with the DECtape.

Digital-8-28-U-SYM
Single Precision DBC & Input (ASR-33)
Signed or Unsigned

This routine accepts a string of up to four decimal digits (single precision for the PDP-8/I) from the Teletype keyboard and converts it to the corresponding 2's complement binary number.

The string may contain as legal characters a sign (+, −, or space) and the digits from 0-9. If the first legal character is not a sign, the conversion is unsigned. A back arrow (←) at any point in the string erases the current string and allows the operator to reenter the correct value. Any character after the first, other than another digit or back arrow causes the conversion to terminate and is found in location SISAVE within the subroutine.

Digital-8-29-U-SYM
Double Precision DBC & Input (ASR-33)
Signed or Unsigned

This routine accepts a string of up to eight decimal digits (double-precision for the PDP-8/I) from the Teletype keyboard and converts it to the corresponding 2's complement binary number.

The string may contain as legal characters a sign (+, −, or space) and the digits 0-9. If the first legal character is not a sign, the conversion is unsigned. A "back-arrow" (←) at any point in the string erases the current string and allows the operator to re-enter the value. Any character after the first, other than another digit or "back-arrow", causes the conversion to terminate and is found in location "DIDSAV" within the subroutine.

Digital-8-31-U
TC01 DECtape Subroutines

These subroutines provide the user with the ability to read, write and search using the TC01 tape system. The read and write subroutines transfer 128_{10} (one memory page) of the specified block (or blocks) although the standard block length is 129_{10} 12-bit words. Successive blocks are read (written) from (into) successive 128 word blocks of core. Provision is made for transfers to and from extended memories.

Digital-8-33-U
5/8 TOG (552)

This program is designed to write timing tracks, mark tracks, and block numbers onto a reel of DECtape providing the tape with the basic skeletal format necessary for its inclusion in any programmed DECtape system. The Formatter program also performs preliminary read-data and write-data checks

to assure the user that the tape produced can be reliably included in such an environment.

Digital-8-34-U
DECEX DECtape Exerciser
This program provides complete certification of the DECtape format produced.

Maintenance Programs

Maindec-08-D01A-D
Instruction Test Part 2A
This program is a test of memory reference instructions, operate instructions, and interrupt mode. An attempt is made to detect and isolate errors to their most basic faults and to the minimum number of logic cards.

Maindec-08-D02A-D
Instruction Test Part 2B
This program is a test of TWOS ADD (TAD) and ROTATE logic (RAL, RTL, RAR, RTR). Random numbers are used in the TWOS ADD portion of the test and sequential numbers are used in the ROTATE portion. Program control is dependent upon operator manipulation of four switches in the SWITCH REGISTER (bits 0, 1, 2, 3). Error information is normally printed out on the keyboard printer.

Maindec-08-D03A-D
Basic JMS and JMP Test
This is a diagnostic program for testing the JMP and JMS instructions of the PDP-8/I.

Maindec-08-D04A-D
Random JMP Test
This program tests the JMP instruction of the PDP-8/I. Most of memory is used as a JUMP field with a random number generator selecting each "JUMP FROM" and "JUMP TO" location.

Maindec-08-D05A-D
Random JMP-JMS Test
This is a diagnostic program to test the JMS instruction of the PDP-8/I. Random "FROM" and "TO" addresses are selected for each test. The JMP instruction is tested in that each test requires a JMP to reach the JMS.

Maindec-08-D07A-D
Random ISZ Test
This program is written to test the ISZ instruction of the PDP-8/I. An ISZ instruction is placed in a FROM location, and a TO location contains the OPERAND. Part 1 of the program selects FROM, TO, and OPERAND from a random number generator, with the option of holding any or all constant. Part 2 uses a fixed set of FROM, TO, and OPERAND numbers.

Maindec-08-D0AA-D
Instruction Test (EAE) Part 3A
This program is a test of the Extended Arithmetic Element Type KE8/I. The following instructions are tested: MQL, MQA, SHL, LSR, ASR, NMI, SCA. An attempt is made to detect and isolate errors to their most basic faults and to the minimum number of logic cards. Multiply and divide are tested by Maindec 08-D0BA-D.

Maindec-08-D0BA-D
Instruction Test (EAE) Part 3B

Divide overflow detection hardware and divide and multiply hardware are tested by using a pseudo random-number generator to produce the parameters for each test. A software simulated divided and multiply are used to test the results of the hardware divide and multiply.

Maindec-08-D1AA-D
Memory Power On/Off Test

This program is a Memory Data Validity Test to be used after a simulated power failure.

Maindec-08-D1BA-D
Memory Checkboard Test High/Low

Tests memory for core failure on half-selected lines under the worst possible conditions for reading and writing. It is used primarily for testing the operation of memory at marginal voltages.

Maindec-08-D1CA-D
Extended Memory Control Part 1

This program exercises and tests Extended Memory instructions CDF, CIF, RDF, RIF, RMF, and RIB, for proper operation. Basically, this program tests the control section of the memory. Data is tested by tests Maindec-08-D1BA-D and Maindec-08-D1DA-D.

Maindec-08-D1DA-D
Extended Memory Checkerboard Part 2

This program is a preliminary test for core memory failures on half-selected lines under worst-case conditions of reading and writing. It is used to test memory module X while running the program in memory module Y.

Maindec-08-D201-D
CR8/I (NCR) Card Reader Test

The program tests the CR8/I Card Reader logic for alpha and binary operation using binary and alpha card decks. It also tests control interrupt and timing.

Maindec-08-D2AA-D
Teletype Reader Test

Tests performance of the Teletype Model 33 Perforated Tape Reader using the reader to scan a closed-top test tape punched with alternating groups of character codes 000 to 377.

Each character is test for bits dropped or gained while reading; each group of characters is checked for characters missed entirely or read more than once.

Maindec-08-D2BA-D
Exerciser for the Teletype Paper Tape Reader

This is an exerciser program for the Teletype Paper Tape Reader. Test tapes are read in a random start-stop fashion and errors reported on the Teletype printer.

Maindec-08-D2CA-D
Teletype Punch Test

Punches a test tape in a predetermined pattern. The tape passes directly

from the Teletype punch to the Teletype reader, which checks the pattern for accuracy.

Maindec-08-D2DA-D
Teleprinter Test

The Teleprinter Test tests performance of the Teletype 33 Keyboard Printer. There are two parts to the test, selectable by the operator. The first part tests keyboard input by immediately causing the character typed to be printed for comparison. The second part tests continuous operation of the teleprinter by causing a line consisting of the ASCII character set to be repeatedly printed. The latter also tests for correct functioning of the interrupt after a character has been printed.

Maindec-08-D2EA-D
High Speed Reader Test

This program tests performance of the Type 750 High Speed Perforated Tape Reader and control by scanning a closed-loop test tape for transmission accuracy. The reader control is tested for correct operation with the PDP-8 interrupt system.

Maindec-08-D2FA-D
High Speed Reader Test (PR8/I)

This is a diagnostic program for the Digitronics 2500 and the PR8/I High Speed Paper Tape Readers. The program is divided into three parts, the first of which is a test tape generator that punches test tapes for parts two and three on the high speed punch. Part two is a series of specific tests with module isolation provided for error situations. Part three reads a preselected tape pattern with the choice of random or fixed block lengths and stalls between blocks.

Maindec-08-D2GA-D
High Speed Punch Test

This program consists of two separate tests. The first causes the High Speed Punch Type PP-8/I to produce a tape containing a sequence of "pseudo-random" character codes. This tape is checked for accuracy using either the high-speed reader or the Teletype reader.

In the second test, the character code represented by the setting SR_{4-11} is punched repeatedly. The switch setting may be changed while the test is running.

Maindec-08-D2HA-D
Typesetting Paper Tape Reader

This is a diagnostic program for the Paper Tape Reader PR68A using the PA60 Control logic. The program is divided into three parts, the first of which is a test tape generator that punches tapes to be used in parts 2 and 3. Part 2 is a series of specific tests with module isolation provided for error situations. Part 3 reads a preselected tape pattern with the choice of random or fixed block lengths and stalls.

Maindec-08-D2MA-D
Monroe Printer Test (MC 4000)

This is a test of the Monroe Printer and its associated control. Control failures are provided for by the use of error halts for operator notification. Data failures are detected by visual analysis of the printer output.

Maindec-08-D3BB-D
TC01 Basic Exerciser

The TC01 Basic Exerciser is a series of test programs that may be used to gain a high degree of confidence in the data handling ability of a TC01 DECtape Control and one to eight TU55 DECtape Transports. The Basic Exerciser consists of several basic routines that may be individually selected; each routine will operate on any configuration of one to eight drives. These routines include a Basic Motion Routine, Search Find All Blocks Test, Basic Search Routine, Start/Stop/Turnabout Test, Basic Write/Read Data Test with eight selectable patterns, and a Parity Generation and Checking Test. The operation of the Basic Motion Routine and the Basic Search Routine are controlled by keyboard input. Also, a Write Data Scope Loop, Read Data Scope Loop, and a Search Scope Loop are provided to keep the tape moving from end zone to end zone.

Maindec-08-D3RA-D
DECTREX 1-TC01 Random Exerciser

DECTREX 1 is a DECtape Random Exerciser for the TC01 DECtape control and any configuration of one to eight TU55 DECtape transports. Drive selection, tape direction, number of blocks, sequence of operation and patterns generated are by random selection. The DECtape functions exercised are search, read data and write data in normal and continuous modes, read all in continuous mode, and move.

Also included are a short series of processor tests that are executed while waiting for interrupts and during data breaks while searching, reading, and writing from DECtape.

Maindec-08-D5AA-D
RM08 Drum Test and Maintenance Compiler

This is a test and maintenance program designed to exercise the Type RM08 Drum. The test routines generated and executed by the compiler are specified by a pseudo program. This may be kept as an integral part of the compiler binary program tape, stored on a separate paper tape, or typed on-line for investigation of an observed malfunction. The SWITCH REGISTER allows testing of various size drums. Errors are indicated by printed messages, which may be suppressed if desired.

Maindec-08-D60B-D
338 Visual Buffered Display

This program tests modes 0-4 and 6 using selected number patterns. Visual identification and program error printouts are used to analyze 338 Display data mode problems.

Maindec-08-D6CA-D
Calcomp Plotter Test

This program tests the CALCOMP Plotter and its control. All control and plotting functions are tested.

Maindec 08-D6EA-D
Type 338 Display POP Test

This program is a test of POP instruction. On the 338 Display analyses of the Display Address Counter and Pushdown Pointer are printed upon error detection.

Maindec 08-D6FA-D
Type 338 Display PJMP Test

This program is a test of the PJMP instruction. On the 338 display analyses of the Pushdown Pointer, Display Address Counter, Status, Push Jmp Destination and Return Addresses are printed upon detection of an error in these areas. Uses Slide or Random address pattern.

Maindec-08-D6IA-D
Little Pictures for an 8

This program contains 12 individual 338 buffered display routines. The routines are selected to enable adjustment and validation of CRT Analog/Digital hardware.

Maindec-08-D8AA-D
196 Intercommunication Buffer

Interface Type 196 is divided into 3 parts, 196A, 196B, and 196C. Each part is designed for a different combination of computers, with which the 196 may be connected. The 196A is connected with a PDP-8 and another PDP-8. The 196B is connected with a PDP-8 and a PDP-7. The 196C is connected to a PDP-8 and a PDP-5. The Maindec for the 196 Communication Interface is also divided in parts A, B, and C. Each part contains two programs. The programs are labeled Processor A and Processor B. Processor A is always a PDP-8.

Maindec-08-D8BA-D
338 Push Button Test

This program exercises the push-button box and logic on the Type 338 Buffered Display. First, the IOTs that read and set the buttons are checked; then the display is initialized, and the push-button related instructions are executed and checked.

Maindec-08-D8CA-D
Data Test for 636B Communication Test

The Data Test checks accuracy of data transfers. A separate program tests IOTs. Both programs may be in memory at the same time.

Maindec-08-D8CJ-D
IOT Test for 636B Communication Test

The IOT test for Communication Interface Type 636B tests only input/output instructions. A separate program (see DEC-08-D8CA-D) tests accuracy of data transfers. Although the programs are separate, they may be in memory at the same time.

Maindec-08-D8DA-D
338 Character Generator Test

This program exercises the Character Generator VC38 on the Display 338. The test is visual and displays the following information:

THIS IS A TEST OF THE CHARACTER GENERATOR

DECUS LIBRARY PROGRAM ABSTRACTS

DECUS No. 5/8-1.1
BPAK — A Binary Input-Output Package

A revision of the binary package originally written by A. D. Hause of Bell Telephone Laboratories. With BPAK the user can read in binary tapes via the photoreader and punch them out via the Teletype punch. It may be used with any in-out device, but is presently written for the photoreader and Teletype punch. A simple modification converts BPAK so that it reads from the Teletype reader if the photoreader is disabled. In its present form it occupies locations 7600-7777.

DECUS No. 5-2.1
OPAK — An On-line Debugging Program for the PDP-5

A utility program which enables the user to load, examine, and modify computer programs by means of the Teletype. This program is a revision of the program written by A. D. Hause, Bell Telephone Laboratories. Extensive use of the program has suggested many refinements and revisions of the original program, the most significant additions being the word search and the break point. The standard version of OPAK is stored in 6200 to 7577 and also 0006. An abbreviated version is available (7000 to 7577, 0006) which is identical to the other except that it has no provision for symbolic dump. Both programs are easily relocated. Control is via Teletype, with mnemonic codes, (e.g. "B" for inserting breakpoint, "P" for proceed, etc.).

DECUS No. 5-3
BRL — A Binary Relocatable Loader with Transfer Vector Options for the PDP-5 Computer.

A binary loader program occupying 4640_8 to 6177_8 registers, also 160 to 177. It has two main functions:

1. It allows a PDP-5 operator to read a suitably prepared binary program into any page location in memory except the registers occupied by BRL.
2. It greatly simplifies the calling of programmed subroutines by allowing the programmer to use an arbitrary subroutine calling sequence when writing his program, instead of having to remember the location of the subroutines.

DECUS No. 5-4
Octal Typeout of Memory Area with Format Option

(Write-up and Listing Only)

DECUS No. 5-5
Expanded Adding Machine

Expanded Adding Machine is a minimum-space version of Expensive Adding Machine (DEC-5-43-D) using a table lookup method including an error space facility.

This is a basic version to which additional control functions can easily be added. Optional vertical or horizontal format, optional storage of intermediate result without reentry, fixed-point output of results within reason, and other features that can be had in little additional space under switch register control. (Write-up and Listing Only)

DECUS No. 5-6
BCD to Binary Conversion of 3-Digit Numbers

This program is based on DEC-5-4 and is intended to illustrate the use of alternative models in program construction.

While not the fastest possible, this program has one or two interesting features. It converts any 3-digit BCD-coded decimal number, $D_1D_2D_3$ into binary in the invariant time of 372 microseconds. Efficient use is made of BCD positional logic to work the conversion formula $(10D_1 + D_2)\ 10 + D_3)$ by right shifts in the accumulator. In special situations, it could be profitable to insert and initial test/exit on zero, adding 12 microseconds to the time for non-zero numbers.

(Write-up and Listing Only)

DECUS No. 5/8-7
Decimal to Binary Conversion by Radix Deflation on PDP-8

(Write-up and Listing Only)

DECUS No. 5-8
PDP-5 Floating Point Routines

Consists of the following routine:

1. Square Root — Binary Tape and Symbolic Listing
2. Sine-Cosine — Binary Tape only
3. Exponential — Binary Tape only

DECUS No. 5/8-9
Analysis of Variance PDP-5/8

An analysis of variance program for the standard PDP-5/8 configuration.

The output consists of:

A. For each sample:
   1. sample number
   2. sample size
   3. sample mean
   4. sample variance
   5. sample standard deviation

B. The grand mean

C. Analysis of Variance Table:
   1. The grand mean.
   2. The weighted sum of squares of class means about the grand mean.
   3. The degrees of freedom between samples.
   4. The variance between samples.
   5. The pooled sum of squares of individual values about the means of their respective classes.
   6. The degrees of freedom within samples.
   7. The variance within samples.
   8. The total sum of squares of deviations from the grand mean.
   9. The degrees of freedom.
   10. The total variance.
   11. The ratio of the variance between samples to the variance with samples.

This is the standard analysis of variance table that can be used with the F

test to determine the significance, if any, of the differences between sample means. The output is also useful as a first description of the data.

All arithmetic calculations are carried out by the Floating Point Interpretive Package (Digital-8-5-S)

DECUS No. 5-10
Paper Tape Reader Test

A test tape can be produced and will be continuously read as an endless tape. Five kinds of errors will be detected and printed out. The Read routine is in 6033-6040. Specifications: Binary with Parity Format — Length: registers in locations (octal): 10, 11, 4 through 67 (save 63, 64), and 6000-7777.

DECUS No. 5-11
PDP-5 Debug System

Purpose of this program is to provide a system capable of:

1. Octal dump 1 word per line.
2. Octal dump 10_8 words per line.
3. Modifying memory using the typewriter keyboard.
4. Clearing to zero parts of memory.
5. Setting to HALT codes part of memory.
6. Entering breakpoints into a program.
7. Initiating jumps to any part of memory.
8. Punching leader on tape.
9. Punching memory on tape in RIM format.
10. Punching memory on tape in PARITY format.
11. Load memory from tape in PARITY format.

DECUS No. 5-12
Pack-Punch Processor and Reader for the PDP-5

The processor converts a standard binary-format tape into a more compressed format, with two twelve-bit words contained on every three lines of tape. Checksums are punched at frequent intervals, with each origin setting or at least every 200 words.

The reader, which occupies locations 7421 to 7577 in the memory will load a program which is punched in the compressed format. A test for checksum error is made for each group of 200 or less and the program will halt on error detection. Only the most recent group of words need be reloaded. Read-in time is about ten per cent less than for conventional binary format, but the principal advantage is that little time is lost when a checksum error is detected, no matter how long the tape.

DECUS No. 5-13
PDP-5 Assembler for use in IBM 7094/7044

This program accepts IBM 7094/7044 symbolic programs punched on cards and assembles them for the PDP-5. An assembly listing is produced, and a magnetic tape is generated containing the program. This magnetic tape can be converted to paper tape and then read into the PDP-5 or it can be read directly into a PDP-5 with an IBM compatible tape unit. Cards are available.

DECUS No. 5/8-14
DICE Game for the PDP-5/8

Enables a user to play the game, DICE, on either the PDP-5 or PDP-8.

DECUS No. 5-15
ATEPO (Auto Test in Elementary Programming and Operation of a PDP-5 Computer)

The program will type questions or instructions to be performed by the operator of a 4K PDP-5. The program will check to see if the operator has answered the questions correctly. If this is the case, it will type the next question or instruction.

DECUS No. 5-16
Paper Tape Duplicator for PDP-5

The tape duplicator for the PDP-5 is a single buffered read and punch program utilizing the program interrupt. It computes a character count and checksum for each tape and compares with checks at the end of the tape. Checks are also computed and compared during punching.

DECUS No. 5/8-17
Type 250 Drum Transfer Routine For Use on PDP-5/8

Transfers data from drum to core (Read) or core to drum (Write) via ASR-33 Keyboard Control.

DECUS No. 5/8-18a
Binary Tape Disassembly Program

Dissassembles a PDP-5 or 8 program, which is on tape in BIN format. It prints the margin setting, address, octal contents, mnemonic interpretation (PAL) of the octal contents. A normal program or a program which uses Floating Point may be dissassembled

DECUS No. 8-19a
DDT-UP Octal-Symbolic Debugging Program

DDT-UP is an octal-symbolic debugging program for the PDP-8 which occupies locations 5600 through 7677. It is able to read a symbol table punched by PDPSYM and stores symbols, four locations per symbol, from 5577 down towards 0000. The mnemonics for the eight basic instructions and standard OPR and IOT group instructions are initially defined and the highest available location for the user is initially 5363.

From the Teletype, the user can symbolically examine and modify the contents of any memory location. DDT-UP allows the user to punch a corrected program in CBL format.

DDT-UP has a breakpoint facility to help the user run sections of his program. When this facility is used, the debugger also uses location 0005.

DECUS No. 5/8-20
Remote Operator FORTRAN System

Program modification and instructions to make the FORTRAN OTS version dated 2/12/65 operate from remote stations.

DECUS No. 5/8-21
Triple Precision Arithmetic Package for the PDP-5 and the PDP-8

An arithmetic package to operate on 36-bit signed integers. The operations are add, subtract, multiply, divide, input conversion, and output conversion. The largest integer which may be represented is $2^{35} - 1$ or 10 decimal digits. The routines simulate a 36-bit (3 word) accumulator in core location 40, 41, and 42 and a 36-bit multiplier quotient register in core locations 43, 44, and 45. Aside from the few locations in page 0, the routines use less core storage space than the equivalent double-precision routines.

DECUS No. 5-22
DECtape Duplicate

A DECtape routine for the PDP-5 to transfer all of one reel (transport 1) to another (transport 2). Occupies one page of memory beginning at 7400. The last page of memory is not used during the operation of the program, however, the memory from 1 to 7436 is used to set the DECtape reels in the proper starting attitude and is then destroyed during duplication. Duplication will commence after which both reels will rewind. Parity error will cause the program to halt with 0040 in the accumulator.

DECUS No. 5/8-23
PDP-5/8 Oscilloscope Symbol Generator

The subroutine may be called to write a string of characters, a pair of characters, or a single character on an oscilloscope. Seventy (octal) symbols in ASCII Trimmed Code and four special "format" commands are acceptable to this routine. The program is operated in a fashion similar to the DEC Teletype Output Package.

DECUS No. 5-24
Vector Input/Edit

Accepts input to a PDP-5 and allows both time-of-entry and post time-of-entry corrections.

DECUS No. 5-25
A Pseudo Random Number Generator

The random number generator subroutine, when called repeatedly, will return a sequence of 12-bit numbers which, though deterministic, appears to be drawn from a random sequence uniform over the interval 0000_8 to 7777_8. Successive numbers will be found to be statistically uncorrelated. The sequence will not repeat itself until it has been called over 4 billion times.

DECUS No. 8-26a
Compressed Binary Loader (CBL)

The CBL (Compressed Binary Loader) format in contrast to BIN format utilizes all eight information channels of the tape, thus achieving nearly 25% in time savings.

Whereas BIN tapes include only one checksum at the end of the tape, CBL tapes are divided into many independent blocks, each of which includes its own checksum. Each block has an initial loading address for the block and a word count of the number of words to be loaded.

The CBL loader occupies locations 7700 through 7777.

DECUS No. 8-26b
CBC (BIN to CBL) and CONV (CBL to BIN)

Two conversion programs which use the PDP-8 on-line Teletype to read a binary tape in one format and punch a binary tape in the other format. The conversion programs both ignore memory field characters so that the output is a tape for memory field 0.

DECUS No. 8-26c
XCBL — Extended Memory CBL Loader

XCBL is used to load binary tapes punched in CBL format into a PDP-8 with more than standard 4K memory. This loader occupies locations 7670 through 7777 of any memory field.

DECUS No. 8-26d
XCBL Punch Program

This program permits a user to prepare an XCBL tape of portions of a PDP-8 extended memory through the control of the keyboard of the on-line Teletype.

The program is loaded by the XCBL Loader

There are two versions of the program so that any section of memory may be punched:

LOW XCBL occupies 00000 - 00377 and its starting address is 00000

HIGH XCBL occupies 17200 - 17577 and its starting address is 17200.

The program may be restarted at the starting address at any time.

One option is provided according to the setting of bit 0 of the Switch Register. If bit 0 is a ONE, the operation of XCBL PUNCH is similar to that of DDT-UP (DECUS No. 8-19a).

DECUS No. 5/8-27 and 5/8-27a
Bootstrap Loader and Absolute Memory Clear

Bootstrap Loader inserts a bootstrap loading program in page 0 from a minimum of toggled instructions.

Absolute Memory Clear leaves the machine in an absolutely clear state and, therefore, cycling around memory obeying an AND instruction with location zero. Should not be used unless one plans to re-insert the loader program.

DECUS No. 5/8-28a
PAL III Modifications-Phoenix Assembler

This modification of the PAL III Assembler speeds up assembly on the ASR-33/35 and operates only with this I/O device. Operation is essentially the same as PAL III, except that an additional pass has been added, Pass 0. This pass, started in the usual manner but with the switches set to zero, reads the symbolic tape into a core buffer area. Subsequent passes then read the tape image from storage instead of from the Teletype.

DECUS No. 5/8-29
BCD to Binary Conversion Subroutines

These two subroutines improve upon the DEC supplied conversion routine. Comparison cannot be made to the DECUS-supplied fixed-time conversions. DECUS No. 5-6, because it is specified only for the PDP-5. One routine is designed for minimal storage, the other for minimal time. Both are fixed-time conversions; time specified is for a 1.5-μsec machine.

Minimal time routine: 73.6 μsec/32 locations

Minimal storage routine: 85 μsec/29 locations

DEC Routine: 64-237 μsec/37 locations

DECUS No. 5-30
GENPLOT—General Plotting Subroutine

This self-contained subroutine is for the PDP-5 with a 4K memory and a CALCOMP incremental plotter. The subroutine can move (with the pen in the up position) to location (x,y), make an "x" at this location, draw a line from this present position to location (x,y) and initialize the program location counters.

DECUS No. 5-31
FORLOT—FORTRAN Plotting Program for PDP-5

FORPLOT is a general-purpose plotting program for the PDP-5 computer in conjunction with the CALCOMP 560 Plotter. It is self-contained and occupies memory locations 0000_8 to 4177_8. FORPLOT accepts decimal data inputted on paper tape in either fixed or floating point formats. Formats can be mixed at will. PDP-5 FORTRAN output tapes are acceptable directly and any comment on these are filtered out.

DECUS No. 5/8-32a
Program to Relocate and Pack Program in Binary Format

Provides a means to shuffle machine language programs around in memory to make the most efficient use of computer store.

DECUS No. 5/8-33
Tape to Memory Comparator

Tape to Memory Comparator is a debugging program which allows comparison of the computer memory with a binary tape. It is particularly useful for detecting reader problems, or during stages of debugging a new program. Presently, uses high-speed reader, but may be modified for TTY reader.

DECUS No. 5-34
Memory Halt—A PDP-5 Program to Store Halt in Most of Memory

With Memory Halt and Opak, (DECUS No. 5-2.1), in memory, it is possible to store halt (7420) in the following memory locations:

0001 to 0005
0007 to 6177
7402 to 7403

DECUS No. 5/8-35
BCD to Binary Conversion Subroutine and Binary to BCD Subroutine (Double Precision)

This program consists of a pair of relatively simple and straightforward double-precision conversions.

DECUS No. 5-36
Octal Memory Dump Revised

The Octal Memory Dump on Teletype is a DEC routine (DEC-5-8-U) which dumps memory by reading the switch register twice; once for a lower limit and again for an upper limit. It then types an address, the contents of the program and the next three locations, issues a CR/LF, then repeats the process for the next four locations. This leaves the right two-thirds of the Teletype page unused. The 78_{10} instructions occupy two pages.

This revised routine uses the complete width of the Teletype page and occupies only one memory page, using less paper and two less instructions. Now an address and the contents of 15 locations are typed out before a carriage return.

Octal Memory Dump Revised has proved its value as a subroutine and/or a self-contained dump program when it is necessary to dump large sections of DECtape, magnetic tape (IBM compatible) or a binary formatted paper tape.

DECUS No. 5-37
Transfer II

For users who have more than one memory bank attached the PDP-5/8,

Transfer II may prove valuable in moving information from one field to another. When debugging, Transfer II enables a programmer to make a few changes in a new program and test it without reading in the original program again. Transfer II enables more extensive use of memory banks.

DECUS No. 5/8-38
FTYPE—Fractional Signed Decimal Type-In

Enables a user to type fractions of the form: .582, —.73, etc., which will be interpreted as sign plus 11 bits (e.g., $0.5 = 2000_8$). Subroutine reads into 300-3177 and is easily relocated, as it will work on any page without modifications.

DECUS No. 5/8-39
DSDPRINT, DDTYPE—Double-Precision Signed Decimal Input-Output Package

DSDPRINT, when given a signed 24-bit integer, types a space or minus sign, and then a 7-digit decimal number in the range —8388608 to +8388607. DDTYPE enables user to type in a signed decimal number in either single or double precision. These routines are already separately available, but the present subroutine package occupies only one memory page and allows for more efficient memory allocation. Located in 3000-3177, but will work on any page.

DECUS No. 5-40
ICS DECtape Routines (One-Page)

The routines will read or write from the specified DECtape unit and delay the program until all I/O is completed. The last block read will overflow the specified region and destroy one core location. Only standard 129 word DECtape blocks will be read or written. The routines will halt if an error occurs with the status bits in the AC.

DECUS No. 5-41
Breakpoint

This debugging routine has been reduced to a minimum operation. It is a mobile routine which can operate around any program that leaves an extra 30 cells of memory space.

Its function is to insert break points in any given location of the program being debugged, and to hold the contents of AC and Link. The programmer may examine any locations desired and then continue to the next breakpoint. It is presently located in 140_8-170_8, but may be easily relocated.

DECUS No. 5-42
Alphanumeric Input

With the Alphanumeric Input Package, any character may be read into the PDP-5 through either the Teletype or the high-speed reader. The characters are packed two/cell and stored in the address indicated in the switch register.

DECUS No. 5/8-43
Unsigned Octal-Decimal Fraction Conversion

This routine accepts a four-digit octal fraction in the accumulator and prints it out as an N-digit decimal fraction where N = 12 unless otherwise specified. After N digits, the fraction is truncated. Programs are included for use on the PDP-5 with Type 153 Automatic Multiply-Divide and the PDP-8 with Type 182 Extended Arithmetic Element.

Storage requirements: 55 Octal locations for the PDP-5. 47 Octal locations for the PDP-8.

DECUS No. 8-44
Modifications to the Fixed Point Output in the PDP-8 Floating Point Package (Digital 8-5-S)

The Floating Point Package (Digital 8-5-S) includes an Output Controller which allows output in fixed point as well as floating point format. This Output Controller takes the form of a certain number of patches to the "Floating Output E Format" routine, plus an additional page of coding.

Using the Calculator program (Digital 8-10-S), which includes the Floating Point Package, certain deficiencies were noted in the fixed-point output format, particularly the lack of any automatic rounding off. For example, the number 9, if outputted as a single digit, appears as 8. Modification attempts to provide automatic rounding off resulted in the Output Controller being completely rewritten with minor changes in the format.

This new version of the Output Controller is also in the form of patches to the Floating Output with an additional page of coding, thereby not increasing the size of the Floating Point Package.

The following summarizes this new version:

1. The number output is automatically rounded off to the last digit printed, or the sixth significant digit, whichever is reached first. Floating point output is rounded off to six figures since the seventh is usually meaningless.
2. A number less than one is printed with a zero preceding the decimal point (e.g., "+0.5" instead of "+.5").
3. A zero result, after rounding off, is printed as "+0" instead of "+".
4. The basic Floating Point Package includes the facility to specify a carriage return/line feed after the number using location 55 as a flag for this purpose. The patches for the Output Controller caused this facility to be lost. This version restores this facility.

DECUS No. 5/8-45
PDP-5/8 Remote & Time-Shared System

A time-shared programming system which allows remote stations immediate access to the computer and a wide selection of programs.

DECUS No. 5/8-46
PDP-5/8 Utility Programs

Consists of four programs (listed below) each of which may be selected via the teletypewriter. When the program is started, either by a self-starting binary loader or by manually starting the computer in address $200_{(8)}$, it is in its executive mode. In this mode, it will respond only to five keys and perform the following functions:

B—go to BIN to QK Converter Program
E—go to Editor Program
L—type a section of leader and stay in executive
P—go to Page Format Program
Q—go to QK to BIN Converter Program

DECUS No. 8-47
ALBIN—A PDP-8 Loader for Relocatable Binary Programs

ALBIN is a simple method for constructing relocatable binary formatted programs, using the PAL III Assembler. Allocation of these programs can be varied in units of one memory page (128_{10} registers.). When loading an ALBIN program, the actual absolute addresses of indicated program

elements (e.g., the keypoint of subroutines) are noted down in fixed program-specified location on page zero. In order to make a DEC symbolic program suitable for translation into its relocatable binary equivalent, minor changes are required, which, however, do not influence the length of the program. Due to its similarity to the standard DEC BIN loader, the ALBIN loader is also able to read-in normal DEC binary tapes. ALBIN requires 122_{10} locations, RIM loader included. Piling-up in core memory of ALBIN programs stored on conventional or DECtape can be achieved using the same method with some modifications.

DECUS No. 5/8-48
Modified Binary Loader MKIV

The Mark IV Loader was developed to accomplish four objectives:

1. Incorporate the self-starting format described in DECUS 5/8-27, ERC Boot.
2. Selected the reader in use, automatically, without switch register settings.
3. Enables a newly-prepared binary tape to be checked prior to loading by calculating the checksum.
4. Reduce the storage requirements for the loader so that a special program would fit on the last page of memory with it.

DECUS No. 8-49
Relativistic Dynamics

Prints tables for relativistic particle collisions and decay in the same format as the Oxford Kinematic Tables. It can be used in two ways:

1. Two-particle Collisions—Given the masses of incident, target, and emitted particles, the incident energy and centre-of-mass angles, the program calculates angles and energies of the emitted particles in the Lab frame. If the process is forbidden energetically, program outputs "E" allowing the threshold energy to be found.
2. Single Particle Decays—By specifying M2 = 0 (target), the problem will be treated as a decay, and similar tables to the above will be printed.

DECUS No. 5/8-50
Additions to Symbolic Tape Format Generator (DEC-8-21-U)

Performs further useful functions by the addition of a few octal patches. By making the appropriate octal patches via the toggles, the Format Generator can also format FORTRAN tapes, shorten tape by converting space to tabs, and convert the type of tape.

A short binary tape may be made and added on to the end of 8-21-U to "edit" an original tape that was punched off-line.

The rubout character will cause successive deletion of the previous characters until the last C. R. is reached but not removed. The use of "←" will cause the current line to be restarted. Thus an input tape may be prepared off-line without attention to format spacing, mistakes corrected as they occur, and finally passed through the Format Generator to create a correctly formatted, edited, and line-fed on either rolled or fanfold paper tape.

DECUS No. 5/8-51
Character Packing and Unpacking Routines

ASCII characters may be packed two to a word and recovered. Control

characters are also packable but are preceded by a 37 before being packed into the buffer. The two programs total 63_{10} words.

DECUS No. 8-52
Tiny Tape Editor

This Tiny Tape Character Editor fits in core at the same time as the PAL III or MACRO-8 assemblers. A tape may be duplicated at three speeds and stopped at any character for insertion or deletion. The toggle switches control the speed and the functions desired.

The program occupies 72_{10} registers.

DECUS No. 5/8-53
COPCAT

COPCAT is a tape to tape copy routine for PDP-5 and PDP-8 DECtape.

DECUS No. 5/8-54
Tic-tac-toe Learning Program—T^3

This program plays Tic-tac-toe basing its moves on stored descriptions of previously lost games. The main program is written in FORTRAN. There is a short subroutine written in PAL II used to print out the Tic-tac-toe board. The program comes already educated with about 32 lost games stored. Requires FORTRAN Object Time System.

DECUS No. 5/8-55
PALEX—An On-Line Debugging Program for PDP-5 and PDP-8

One problem with programs written in Program Assembly Language (PAL) for operation on a PDP-5/8 computer is the danger of an untested program being self-destructive, running wild, destroying other programs residing in memory such as loading programs. PALEX prevents any of the above unwanted operations from occurring while it gives the operator-programmer valuable debugging information and enables him to make changes in his program and try out the modified program. Once running, PALEX cannot be destroyed by any program or instruction in memory, the operator need not touch any manual console controls, and all required information is printed in easy-to-read format on the Teletype console.

DECUS No. 5/8-56
Fixed Point Trace No. 1

A minimum size monitor program which executes the users program one instruction at a time and reports the contents of the program counter, the octal instruction, the contents of the accumulator and link and the contents of the effective address by means of the ASR-33 Teletype. Storage Requirements: two pages.

DECUS No. 5/8-57
Fixed Point Trace No. 2

Similar to Fixed Point No. 1 except that the symbolic tape provided has a single origin setting instruction of (6000). Any four consecutive memory pages can be used, with the exception of page zero, by changing this one instruction.

DECUS No. 8-58
One-Page DECtape Routine (522 Control)

A general-purpose program for reading, writing, and searching of magnetic tape. This program was written for the Type 552 Control. It has many advantages over both the standard DEC routines and also over the DECUS

No. 5-46. The routines are one-page long and can be operated with the interrupt on or off. The DEC program delays the calling program while waiting for the unit and movement delays to time-out. This routine returns control to the calling program. This saves ¼ second every time the tape searches forward and half that time when it reverses. In addition, it will read and write block 0,. This program is an advantage over the previous one-page routines in that it allows interrupt operations, doesn't overflow by one location, interrupts the end zone correctly and not as an error, and provides a calling sequence identical to the DEC program.

DECUS No. 8-59
PALDT—PAL Modified for DECtape (552 Control)

When assembling programs, PALDT requires that the symbolic tape be read in only once. The program writes on the library tape itself after finding the next available block from the directory. During pass 0 the tape is read in using the entire user's symbol table. During passes 1, 2, 3, as much of the symbol table is used as possible. This means the fewest tape passes as possible. As an added advantage pass 0 ignores blank tape, leader-trailer, line feeds, form feeds, and rub outs; saving space. The whole program decreases the users symbol table by only three pages: one for the DECtape program above, one for pass 0, and one for the minimal length read in buffer.

DECUS No. 8-60
Square Root Function by Subtraction Reduction

A single precision square routine using EAE. This routine is usually faster than the DEC routine and can easily be modified for double precision calculation at only twice the computation time.

DECUS No. 8-61
Improvement to Digital 8-9-F Square Root

An improved version of the DEC Single Precision Square Root Routine (without EAE). Saves a few words of storage and execution is speeded up 12 per cent.

DECUS No. 8-62
High-Speed Reader Option for FORTRAN Compiler

Program modification that allows the PDP-8 FORTRAN Compiler to read source tapes through the high-speed reader, and punch on the ASR-33. The program is loaded in over the compiler. It can be punched on an extension of the compiler tape so that by depressing the CONTINUE key, it can be read in immediately following the compiler.

DECUS No. 5-63
SBUG-4

SBUG-4 allows the PDP-5 to execute one instruction of any given program at a time, returning to SBUG-4 following each instruction and printing out the contents of various registers. This permits following the path of a program which has gone astray or examining some defective operation.

DECUS No. 5/8-64
DECtape Programming System

This program provides rapid access to DEC software and utilizes routines through the use of DECtape. Programs may be stored, edited, assembled, listed, or executed without reliance upon paper tape.

May be used with both TC01 and 552 DECtape Controls.

DECUS No. 8-65
A Programmed Associative Multichannel Analyser

The program describes the use of a small computer as an associate analyser with special reference to the PDP-8. The advantages and limitations of the method are discussed in the write-up, and general program algorithms are presented.

DECUS No. 8-66
Editor Modified for DECtape

This program consists of modifications to the Digital 8-1-S Symbolic Editor to enable reading and writing on DECtape. This results in considerable time savings in assembling PAL programs since PAL has also been modified to accept the symbolic program directly from DECtape. The DECtape compatibility is also useful for storing text for later use and for regaining Editor memory space lost due to delete and change commands.

In addition, the overflow detection routine is now foolproof and results in a HALT.

Storage: Editor <0, 1461>

Modifications: <1462, 1502> <6376, 7177>

DECtape Routines: <7200, 7577>

Equipment: PDP-8 with EAE, ASR-33, DECtape

DECUS No. 8-67
PAL Modified for DECtape Input

This program consists of modifications to the Digital 8-3L-S PAL Assembly Program to enable it to obtain the symbolic program to be assembled from DECtape (in addition to paper tape), outputting the assembled program in the usual manner. (The symbolic program is written onto DECtape by use of the "Editor Modified for DECtape" Program.) The modification also makes it possible to assemble sections of programs in any order, and to intersperse sections or commands from the keyboard with those from DECtape. The resulting assembly is limited in speed mainly by the punching of the assembled program during Pass 2, and Pass 1 is speeded considerably. The modifications also include a tabulator interpreter, so that Pass 3 listings are produced in tabulated format.

Storage: PAL III <0, 3561> plus symbol table

Modifications: <6555, 7177>

DECtape Routines: <7200, 7577>

Equipment: PDP-8 with EAE, ASR-33, DECtape

DECUS No. 8-68
ALP Program

The ALP Program punches labels for paper tapes. When a key is stuck on the on-line Teletype keyboard, no echo is performed, but the PDP-8 outputs a few characters to the Teletype punch which form the outline of the character associated with the key.

The character outlines have a fixed width or 5 lines of tape, followed by 3 blank lines for separation between characters; all 8 columns of the paper tape are used to provide the maximum height of character outlines.

DECUS No. 5/8-69
LESQ29 and LESQ11

The purpose of the program is to fit the best sequences of parabolas to a given 400 point data curve in order to remove extraneous noise; rather than rely on a single 400 point parabola least squares fit to approximate a given data curve. Approximately 400 individual parabolas are computed as follows.

LESQ29

Data values 1 through 29 are subjected to a second order Least Squares fit. The median point of the resulting parabola (point #15) is then substituted for the original data value #15.*

A second parabola is then computed using data values 2 through 30. The median point of this parabola (point #16) is then substituted for point #16 of the original data curve.

This procedure is repeated until all data values have been replaced (except for the first and last 14 points which are excluded by the mechanics of the operation).

LESQ11

Process identical to LESQ29 except that an 11 rather than a 29 point smooth interval is used. First point replaced is point #6, and only the first and last 5 points are excluded from smoothing.

LESQ11 will preserve higher frequency data than LESQ29 for a given data curve with constant time between data points.

Minimum Hardware: 4K Memory PDP-5 or PDP-8, Teletypewriter (plotter, DECtape optional)

Other Programs Needed: Floating Point Package and appropriate data handling routines.

Storage Requirements: (LESQ11: 400-564; 700-716)
(LESQ29: 400-564; 700-751)

Execution Time: (PDP-5) LESQ11: 1 minute.
LESQ29: 2.5 minutes.

Restrictions: Positive integer data $<3777_8$; time between data points constant.

*See B. J. Power, R. N. Hagen, S. O. Johnson, "SPORT, A System for Processing Reactor Transient Data on the IBM-7040 Computer," pp. 4-8, AEC Research and Development Report (IDO-17078), Available from: The Clearinghouse for Federal Scientific and Technical Information, National Bureau of Standards, U. S. Department of Commerce, Springfield, Virginia.

DECUS No. 8-70
EAE Routines for FORTRAN Operating System (DEC-08-CFA3)

These are two binary patches to the FORTRAN Operating System which utilizes the Type 182 EAE hardware for single precision multiplication and normalization, replacing the software routines in FOSSIL (the operating system). The binary tape is loaded by the BIN Loader after FOSSIL has been loaded. Execution time of a Gauss-Jordan matrix inversion is reduced by approximately 30%.

Minimum Hardware: PDP-8 with Type 182 EAE

Other Programs Needed: FORTRAN Operating System (DEC-08-CFA3-PB) dated March 2, 1967.

DECUS No. 8-71
Perpetual Calendar

The program is designed as a computer demonstration. When a valid date is fed into the computer, the corresponding day of the week is typed out. If an invalid date is given, "YOU GOOFED, TRY AGAIN" is typed out. The program is based on the Gregorian Calendar and is, therefore, limited to years between 1500 and 4095. The upper limit being due to the computer's capacity.

Minimum Hardware: 4K storage, ASR-33 Teletype

Storage: 20-1333

DECUS No. 8-72
Matrix Inversion — Real Numbers

The program inverts a matrix, up to size 12 x 12, of real numbers. The algorithm used is the Gauss-Jordan method. A unit vector of appropriate size is generated internally at each stage. Following the Gauss sweep-out, the matrix is shifted in storage, another unit vector is generated, and the calculation proceeds.

Other Programs Needed: FORTRAN Compiler and FORTRAN Operating System.

Storage: This program uses essentially all core not used by the FORTRAN Operating System.

Execution Time: Actual computation takes less than 10 seconds. Data read-in and read-out may take up to five minutes.

DECUS No. 8-73
Matrix Inversion — Complex Numbers

The program inverts a matrix, up to size 6 x 6, of complex numbers. The algorithm used is the Gauss-Jordan method, programmed to carry out complex number calculations. A unit vector of appropriate size is generated internally. Following the Gauss sweep-out, the matrix is shifted, another unit vector is generated, and the calculation proceeds. The print-out of the matrices uses the symbol J to designate the imaginary part, e.g. A = a + jb.

Other Programs Needed: FORTRAN Compiler and FORTRAN Operating System.

Storage: This program uses essentially all core not used by the FORTRAN Operating System.

Execution Time: Actual computation takes less than 10 seconds. Data read-in and read-out may take up to five minutes.

DECUS No. 8-74
Solution of System of Linear Equations: AX = B, by Matrix Inversion and Vector Multiplication

This program solves the set of linear algebraic equations AX = B by inverting matrix A using a Gauss-Jordan method. When the inverse matrix has been calculated, it is printed out. At that point, the program requests the B-vector entries. After read-in of the B-vector, the product is computed and printed out. The program then loops back to request another B-vector, allowing the system to solve many sets of B-vectors without the need to

invert matrix A again. Maximum size is 8 x 8.

Other Programs Needed: FORTRAN Compiler and FORTRAN Operating System.

Storage: This program uses essentially all core not used by the FORTRAN Operating System.

Execution Time: Actual computation is less than 10 seconds. Data read-in and read-out may take up to five minutes.

DECUS No. 8-75
Matrix Multiplication — Including Conforming Rectangular Matrices

This program multiplies two matrices, not necessarily square but which conform for multiplication.

Other Programs Needed: FORTRAN Compiler and FORTRAN Operating System.

Storage: This program uses essentially all core not used by the FORTRAN Operating System.

Execution Time: Actual computation takes less than 10 seconds. Data read-in and read-out may take up to five minutes.

Author's comments regarding the four matrix routines: DECUS Nos. 8-72, -73, -74, -75.

"Each program has been written in FORTRAN for use on the basic PDP-8. The printed output of each program has been organized to efficiently and effectively use the Teletype, yet maintain a readable and meaningful output format. This has been done at the expense of optimizing storage requirements. Thus, each program may be changed to handle slightly larger cases by cutting down on the print-out information. Since the source programs are in FORTRAN, a user may readily change the program to suit his own requirements.

"Another common feature of each program is a print-out of the input data from core. This has been found desirable for checking that data was read in properly, and for purposes of having an accurate record of a given calculation. Also, since each program requires a large amount of data input, it is suggested that a data tape be made prior to running the program.

"The matrix inversion scheme used is straightforward and gives good results on run-of-the-mill matrices. However, error build-up is quite rapid on an ill-conditioned matrix. This is partly due to errors in the fifth and sixth decimal place caused by the floating point conversion at read-in time, and also to the limited mantissa carried in the PDP-8 floating point word."

DECUS No. 8/8S-76
PDP NAVIG 2/2

This program utilizes the output of the U. S. Navy's AN/SRN-9 satellite navigation receiver to obtain fixes on a PDP-8 or PDP-8/S. This program, except for some details of input and output, follows very closely NAVIG2 written for the IBM 1620 which in turn is derived from the TRIDON program written at the Applied Physics Laboratory of Johns Hopkins University for the IBM 7090.

PDP NAVIG 2/2 is written in PAL III for a 4096 core machine using the ASR-33. Floating point numbers using two 12-bit words as mantissa and

one 12-bit word as exponent are employed. The accuracy is slightly less than that using 7 decimal digits per word.

DECUS No. 8/8S-77
PDP-8 Dual Process System

The purpose of this system is to expedite the programming of multiprocessing problems on the PDP-8 and PDP-8/S. It maximizes both the input speed and the portion of real time actually used for calculations by allowing the program to run during the intervals between issuing I/O commands and the raising of the device flag to signal completion of the command. The technique also allows queuing of input data or commands so that the user need not wait while his last line is being processed, and so that each line of input may be processed as fast as possible regardless of its length. The system uses the interrupt facilities and has less than a 3% overhead on the PDP-8/S (about .1% on the PDP-8).

This method is especially useful for a slower machine where the problem may easily be calculation limited but would, without such a system, become I/O bound.

The program may also be easily extended to handle input from an A/D converter. Here, the input would be buffered by groups of readings terminated either arbitrarily in groups of N or by zero crossings.

The system requires 600_8 registers for two TTY's plus buffer space. Several device configurations are possible.

This program can increase the I/O to computation efficiency of some programs by 100%. It can do this even for a single Teletype. Each user will probably want to tailor the program to his individual needs.

DECUS No. 8-78
Diagnose: A Versatile Trace Routine for the PDP-8 Computer with EAE

This trace routine will track down logical errors in a program (the "sick" program). Starting at any convenient location in the "sick" program, instructions are executed, one at a time, and a record of all operations is printed out via the Teletype. To avoid tracing proven subroutines, an option is provided to omit subroutine tracing. The present routine is significantly more versatile than two other trace routines in the DECUS library (DECUS Nos. 8-56 and 8-57 — Biavati) for the PDP-8 in that it is able to trace "sick" programs containing floating-point, extended arithmetical, and a variety of input-output instructions. Diagnose is, however, at a disadvantage compared with Biavati's first routine (DECUS No. 8-56) in requiring more memory space (five pages as opposed to two); and compared with his second routine (DECUS No. 8-57) in not possessing the trace-suppression features of the latter. The mode of operation of Diagnose is quite different from that of the trace routines of Biavati.

Minimum Hardware: PDP-8 with EAE

Other Programs Needed: Floating Point Package needed for floating point tracing.

Storage: 5(4) pages of memory.

Miscellaneous: Program is relocatable.

DECUS No. 8-79
TIC-TAC-TOE (Trinity College Version)

This TIC-TAC-TOE game is programmed, using internal logic, so that the computer will either win or stalemate, but not lose a game. Either the player or the computer may choose to go first. At the termination of a game, the program restarts for the next game by typing anew the grid code to be followed.

DECUS No. 8-80
Determination of Real Eigenvalues of a Real Matrix

This is a two-part program for determining the real eigenvalues of a real-valued matrix. The matrix does not have to be symmetric. Part I uses the power method of iterating on an eigenvector to determine the largest eigenvalue of the matrix. Part II then deflates the matrix using the results of Part I so as to produce a matrix of order one less than that solved for in Part I. Part I can then be reloaded, and the next eigenvalue in line may be calculated. In this, all the real eigenvalues may be computed in order.

DECUS No. 8-81
A BIN or RIM Format Data or Program Tape Generator

This program enables a PDP-8 operator to generate tapes under Teletype control in RIM or PAL BIN format without formal assembly, assuming the operator knows the octal codes corresponding to each instruction. This is particularly useful when one is dealing with small programs for testing interface equipment or when making small modifications to large programs when one does not wish to spend time reassembling the whole program. Often during program debugging, changes are repeatedly toggled into core manually, which leaves no permanent record of the changes made and is prone to error. Tapes generated using this program can be appended to existing BIN or RIM tapes and can then be loaded with the original tape into core with the appropriate loader. Another use of this program is in the preparation of data tapes in RIM or BIN format so that data can be loaded straight into PDP-8 core via the usual leaders. The program also generates leader/trailer code and a checksum under program control.

Storage: Program occupies locations 6000-6077.

DECUS No. 8-82
Library System for 580 Magnetic Tape (Preliminary Version)

The system provides for storing program files (or other files) on the 580 Magnetic Tape with PDP-8, and recalling them at will without altering the state of the rest of the computer. In general principle, it is similar to the DECtape Library System, and the only effective storage requirement is the last page of memory.

At present, the system consists of three programs known as BOOTSTRAP 1, BOOTSTRAP 2, and the LIBRARY Routines.

Bootstrap 1 is a minimal loader program which resides in the last page of memory. Its function is to rewind the tape and load Bootstrap into the last page, automatically transferring control to it. Bootstrap saves the area of core to be used by the system as a record on the magnetic tape, loads the Library Routines into core, and transfers control to them.

The Library Routines comprise a Directory of the files on tape, an Input-Output package, enabling communication with the Teletype, and four

system programs:

LIst: Types out the names of files in the Directory

CAll: Transfers a file into core and exits

DUmp: Writes a file on tape, rewrites the Directory, and exits

EXit: Restores the computer to its original state, with Bootstrap 1 and BIN on the last page.

The magnetic tape subroutines and some control functions are included in Bootstrap 2. Each entry in the directory consists of three words: the name of the file, its first location in core, and the number of words it occupies. The capacity of the directory is 22_{10} entries.

DECUS No. 8/8S-83A and B
Octal Debugging Package (With and Without Floating Point)

This program is an on-line debugger which will communicate with the operator through the ASR-33 Teletype. It allows register examination and modification, octal dumping, binary punching, multiple and simultaneous breakpoints, starting a program, and running at a particular location with preset AC and link. ODP is completely relocatable at the beginning of all pages except page zero, and is compatible with the PDP-5, the PDP-8, and the PDP-8/S.

Requirements: The high version of ODP requires locations 7000-7577. The low version requires locations 0200-0777. All versions will require three pages. Also, location 0002 is used for a breakpoint pointer to ODP.

Equipment: The standard PDP-8 with ASR-33 Teletype is required. A high-speed punch is optional.

DECUS No. 8-84
One-Pass PAL III

This is a modification to Digital 8-3L-S. It is for use on an 8K PDP-8 with ASR 33. The principle of the modification is to store the incoming characters during Pass I into the memory extension and to take them from there during Pass 2 and 3. Source programs must be limited to 4095 characters. This modification can save about 40% of assembly time.

Operation of the program is the same as for PAL III except that the reading of the source program for Pass 2 and 3 need not be repeated. For these passes, one simply pressses CONTINUE after setting the correct switches.

Restriction: The program does not work with high-speed reader and punch.

DECUS No. 5/8-85
Set Memory Equal to Anything

This program will preset all locations to any desired settings. Thus, combining a memory clear, set memory equal to HALT, etc. into a single program. The program is loaded via the switch registers into core.

DECUS No. 8-86
High-Speed Reader Option for PDP-8 FORTRAN Compiler for use with DECtape-Stored Compiler

This program is for DECtape installations and utilizes the High-Speed Reader Routine in the DECtape Binary Loader. It is a modification to DECUS 8-82, High-Speed Reader Option. DECUS 8-82 is not usable by installations

equipped with DECtape because it utilizes part of the last page reserved for the DECtape loader.

The binary tape of this modification is loaded in over the FORTRAN Compiler, then the entire load put onto DECtape using the routine, "UPDATE."

DECUS No. 6/8-12
PDP-8 Assembler for PDP-6

Assembles PDP-8 programs written in PAL on a PDP-6 using any I/O devices.

APPENDIX 2
TABLES OF INSTRUCTIONS
PDP-8/I MEMORY REFERENCE INSTRUCTIONS

Mnemonic Symbol	Operation Code	Direct Addr. States Entered	Direct Addr. Execution Time (μsec)	Indirect Addr. States Entered	Indirect Addr. Execution Time (μsec)	Operation
AND Y	0	F, E	3.0	F, D, E	4.5	Logical AND. The AND operation is performed between the content of memory location Y and the content of the AC. The result is left in the AC, the original content of the AC is lost, and the content of Y is restored. Corresponding bits of the AC and Y are operated upon independently. ACj $\wedge$ Yj $=>$ ACj
TAD Y	1	F, E	3.0	F, D, E	4.5	Two's complement add. The content of memory location Y is added to the content of the AC in two's complement arithmetic. The result of this addition is held in the AC, the original content of the AC is lost, and the content of Y is restored. If there is a carry from AC0, the link is complemented. AC + Y $=>$ AC
ISZ Y	2	F, E	3.0	F, D, E	4.5	Increment and skip if zero. The content of memory location Y is incremented by one. If the resultant content of Y equals zero, the content of the PC is incremented and the next instruction is skipped. If the resultant content of Y does not equal zero, the program proceeds to the next

PDP-8/I MEMORY REFERENCE INSTRUCTIONS (continued)

Mnemonic Symbol	Operation Code	Direct Addr. States Entered	Direct Addr. Execution Time (μsec)	Indirect Addr. States Entered	Indirect Addr. Execution Time (μsec)	Operation
						instruction. The incremented content of Y is restored to memory. If resultant Y = 0, PC + 1 = > PC.
DCA Y	3	F, E	3.0	F, D, E	4.5	Deposit and clear AC. The content of the AC is deposited in core memory at address Y and the AC is cleared. The previous content of memory location Y is lost. AC = > Y 0 = > AC
JMS Y	4	F, E	3.0	F, D, E	4.5	Jump to subroutine. The content of the PC is deposited in core memory location Y and the next instruction is taken from core memory location Y + 1. PC + 1 = > Y Y + 1 = > PC
JMP Y	5	F	1.5	F, D	3.0	Jump to Y. Address Y is set into the PC so that the next instruction is taken from core memory address Y. The original content of the PC is lost. Y = > PC

PDP-8/I GROUP 1 OPERATE MICROINSTRUCTIONS

Mnemonic Symbol	Octal Code	Sequence	Operation
NOP	7000	—	No operation. Causes a 1.5 μsec program delay.
IAC	7001	3	Increment AC. The content of the AC is incremented by one in two's complement arithmetic.
RAL	7004	4	Rotate AC and L left. The content of the AC and the L are rotated left one place.
RTL	7006	4	Rotate two places to the left. Equivalent to two successive RAL operations.
RAR	7010	4	Rotate AC and L right. The content of the AC and L are rotated right one place.
RTR	7012	4	Rotate two places to the right. Equivalent to two successive RAR operations.
CML	7020	2	Complement L.
CMA	7040	2	Complement AC. The content of the AC is set to the one's complement of its current content.
CIA	7041	2, 3	Complement and increment accumulator. Used to form two's complement.
CLL	7100	1	Clear L.
CLL RAL	7104	1, 4	Shift positive number one left.
CLL RTL	7106	1, 4	Clear link, rotate two left.
CLL RAR	7110	1, 4	Shift positive number one right.
CLL RTR	7112	1, 4	Clear link, rotate two right.
STL	7120	1, 2	Set link. The L is set to contain a binary 1.
CLA	7200	1	Clear AC. To be used alone or in OPR 1 combinations.
CLA IAC	7201	1, 3	Set AC = 1.
GLK	7204	1, 4	Get link. Transfer L into AC 11.
CLA CLL	7300	1	Clear AC and L.
STA	7240	2	Set AC = −1. Each bit of the AC is set to contain a 1.

PDP-8/I GROUP 2 OPERATE MICROINSTRUCTIONS

Mnemonic Symbol	Octal Code	Sequence	Operation
HLT	7402	3	Halt. Stops the program after completion of the cycle in process. If this instruction is combined with others in the OPR 2 group the other operations are completed before the end of the cycle.
OSR	7404	3	OR with switch register. The OR function is performed between the content of the SR and the content of the AC, with the result left in the AC.
SKP	7410	1	Skip, unconditional. The next instruction is skipped.
SNL	7420	1	Skip if L $\neq$ 0.
SZL	7430	1	Skip if L = 0.
SZA	7440	1	Skip if AC = 0.
SNA	7450	1	Skip if AC $\neq$ 0.
SZA SNL	7460	1	Skip if AC = 0, or L = 1, or both.
SNA SZL	7470	1	Skip if AC $\neq$ 0 and L = 0.
SMA	7500	1	Skip on minus AC. If the content of the AC is a negative number, the next instruction is skipped.
SPA	7510	1	Skip on positive AC. If the content of the AC is a positive number, the next instruction is skipped.
SMA SNL	7520	1	Skip if AC $<$ 0, or L = 1, or both.
SPA SZL	7530	1	Skip if AC $>$ 0 and if L = 0.
SMA SZA	7540	1	Skip if AC $<$ 0.
SPA SNA	7550	1	Skip if AC $>$ 0.
CLA	7600	2	Clear AC. To be used alone or in OPR 2 combinations.
LAS	7604	1, 3	Load AC with SR.
SZA CLA	7640	1, 2	Skip if AC = 0, then clear AC.
SNA CLA	7650	1, 2	Skip if AC $\neq$ 0, then clear AC.
SMA CLA	7700	1, 2	Skip if AC $<$ 0, then clear AC.
SPA CLA	7710	1, 2	Skip if AC $>$ 0, then clear AC.

PDP-8/I EXTENDED ARITHMETIC ELEMENT MICROINSTRUCTIONS

Mnemonic Symbol	Octal Code	Sequence	Operation
MUY	7405	3	Multiply. The number held in the MQ is multiplied by the number held in core memory location PC + 1 (or the next successive core memory location after the MUY Command). At the conclusion of this command the most significant 12 bits of the product are contained in the AC and the least significant 12 bits of the product are contained in the MQ. Y × MQ = > AC, MQ.
DVI	7407	3	Divide. The 24-bit dividend held in the AC (most significant 12 bits) and the MQ (least significant 12 bits) is divided by the number held in core memory location PC + 1 (or the next successive core memory location following the DVI command). At the conclusion of this command the quotient is held in the MQ, the remainder is in the AC, and the L contains a 0. If the L contains a 1, divide overflow occurred so the operation was concluded after the first cycle of the division. AC, MQ ÷ Y = > MQ.
NMI	7411	3	Normalize. This instruction is used as part of the conversion of a binary number to a fraction and an exponent for use in floating-point arithmetic. The combined content of the AC and the MQ is shifted left by this one command until the content of AC0 is not equal to the content of AC1, to form the fraction. Zeros are shifted into vacated MQ11 positions for each shift. At the conclusion of this operation, the step counter contains a number equal to the number of shifts performed. The content of L is lost. ACj = > ACj − 1 AC0 = > L MQ 0 = > AC11 MQj = > MQj − 1 0 = > MQ11 until AC0 ≠ AC1
SHL	7413	3	Shift arithmetic left. This instruction shifts the combined content of the AC and MQ to the left one position more than the number of positions indicated by the content of core memory at address PC + 1 (or the next successive core memory location following the SHL command). During the shifting, zeros are shifted into vacated MQ11 positions. Shift Y + 1 positions as follows: ACj = > ACj − 1 AC0 = > L

PDP-8/I EXTENDED ARITHMETIC ELEMENT MICROINSTRUCTIONS (continued)

Mnemonic Symbol	Octal Code	Sequence	Operation
			MQ0 = > AC11 MQj = > MQj − 1 0 = > MQ11
ASR	7415	3	Arithmetic shift right. The combined content of the AC and the MQ is shifted right one position more than the number contained in memory location PC + 1 (or the next successive core memory location following the ASR command). The sign bit, contained in AC0, enters vacated positions, the sign bit is preserved, information shifted out of MQ11 is lost, and the L is undisturbed during this operation. Shift Y + 1 positions as follows: AC0 = > AC0 ACj = > ACj + 1 AC11 = > MQ0 MQj = > MQj + 1
LSR	7417	3	Logical shift right. The combined content of the AC and MQ is shifted left one position more than the number contained in memory location PC + 1 (or the next successive core memory location following the LSR command). This command is similar to the ASR command except that zeros enter vacated positions instead of the sign bit entering these locations. Information shifted out of MQ11 is lost and the L is undisturbed during this operation. Shift Y + 1 positions as follows: 0 = > AC0 ACj = > ACj + 1 AC11 = > MQ0 MQj = > MQj + 1
MQL	7421	2	Load multiplier quotient. This command clears the MQ, loads the content of the AC into the MQ, then clears the AC. 0 = > MQ AC = > MQ 0 = > AC
SCA	7441	2	Step counter load into accumulator. The content of the step counter is transferred into the AC. The AC should be cleared prior to issuing this command or the CLA command can be combined with the SCA to clear the AC, then effect the transfer. SC V AC = > AC
SCL	7403	3	Step counter load from memory. Loads complement of bits 7 through 11 of the word in

PDP-8/I EXTENDED ARITHMETIC ELEMENT MICROINSTRUCTIONS
(continued)

Mnemonic Symbol	Octal Code	Sequence	Operation
			memory following the instruction into the step counter. $\overline{MB}_{7-11} = > SC$ $PC + 2 = > PC$
MQA	7501	2	Multiplier quotient load into accumulator. The content of the MQ is transferred into the AC. This command is given to load the 12 least significant bits of the product into the AC following a multiplication or to load the quotient into the AC following a division. The AC should be cleared prior to issuing this command or the CLA command can be combined with the MQA to clear the AC then effect the transfer. MQ V AC $= >$ AC
CLA	7601	1	Clear accumulator. The AC is cleared during sequence 1, allowing this command to be combined with the other EAE commands that load the AC during sequence 2 (such as SCA and MQA). $0 = >$ AC
CAM	7621	1, 2	Clear accumulator and multiplier quotient. CAM = CLA LMQ.

BASIC IOT MICROINSTRUCTIONS

Mnemonic	Octal	Operation
Program Interrupt		
ION	6001	Turn interrupt on and enable the computer to respond to an interrupt request. When this instruction is given, the computer executes the next instruction, then enables the interrupt. The additional instruction allows exit from the interrupt subroutine before allowing another interrupt to occur.
IOF	6002	Turn interrupt off i.e. disable the interrupt.
High Speed Perforated Tape Reader and Control		
RSF	6011	Skip if reader flag is a 1.
RRB	6012	Read the content of the reader buffer and clear the reader flag. (This instruction does not clear the AC.) RB V AC 4-11 = > AC 4-11
RFC	6014	Clear reader flag and reader buffer, fetch one character from tape and load it into the reader buffer, and set the reader flag when done.
High Speed Perforated Tape Punch and Control		
PSF	6021	Skip if punch flag is a 1.
PCF	6022	Clear punch flag and punch buffer.
PPC	6024	Load the punch buffer from bits 4 through 11 of the AC and punch the character. (This instruction does not clear the punch flag or punch buffer.) AC 4-11 V PB = > PB
PLS	6026	Clear the punch flag, clear the punch buffer, load the punch buffer from the content of bits 4 through 11 of the accumulator, punch the character, and set the punch flag to 1 when done.
Teletype Keyboard/Reader		
KSF	6031	Skip if keyboard flag is a 1.
KCC	6032	Clear AC and clear keyboard flag.
KRS	6034	Read keyboard buffer static. (This is a static command in that neither the AC nor the keyboard flag is cleared.) TTI V AC 4-11 = > AC 4-11
KRB	6036	Clear AC, clear keyboard flag, and read the content of the keyboard buffer into the content of AC 4-11.
Teletype Teleprinter/Punch		
TSF	6041	Skip if teleprinter flag is a 1.
TCF	6042	Clear teleprinter flag.
TPC	6044	Load the TTO from the content of AC 4-11 and print and/or punch the character.
TLS	6046	Load the TTO from the content of AC 4-11, clear the teleprinter flag, and print and/or punch the character.

BASIC IOT MICROINSTRUCTIONS
(continued)

Mnemonic	Octal	Operation
Oscilloscope Display Type VC8/I and Precision CRT Display Type 30N		
DCX	6051	Clear X coordinate buffer.
DXL	6053	Clear and load X coordinate buffer. AC 2-11 = > YB
DIX	6054	Intensify the point defined by the content of the X and Y coordinate buffers.
DXS	6057	Executes the combined functions of DXL followed by DIX.
DCY	6061	Clear Y coordinate buffer.
DYL	6063	Clear and load Y coordinate buffer. AC 2-11 = > YB
DIY	6064	Intensify the point defined by the content of the X and Y coordinate buffers.
DYS	6067	Executes the combined functions of DYL followed by DIY.
Oscilloscope Display Type VC8/I		
DSB	6075	Set minimum brightness.
DSB	6076	Set medium brightness.
DSB	6077	Set maximum brightness.
DSB	6074	Zero brightness.
Precision CRT Display Type 30N		
DLB	6074	Load brightness register (BR) from bits 9 through 11 of the AC. AC 9-11 = > BR
Light Pen Type 370		
DSF	6071	Skip if display flag is a 1.
DCF	6072	Clear the display flag.
Memory Parity Type MP8/I		
SMP	6101	Skip if memory parity error flag = 0.
CMP	6104	Clear memory parity error flag.
Automatic Restart Type KP8/I		
SPL	6102	Skip if power is low.
Memory Extension Control Type MC8/I		
CDF	62N1	Change to data field N. The data field register is loaded with the selected field number (0 to 7). All subsequent memory requests for operands are automatically switched to that data field until the data field number is changed by a new CDF command.
CIF	62N2	Prepare to change to instruction field N. The instruction buffer register is loaded with the selected field number (0 to 7). The next JMP or JMS instruction causes the new field to be entered.
RDF	6214	Read data field into AC 6-8. Bits 0-5 and 9-11 of the AC are not affected.

BASIC IOT MICROINSTRUCTIONS (continued)

Mnemonic	Octal	Operation
RIF	6224	Same as RDF except reads the instruction field.
RIB	6234	Read interrupt buffer. The instruction field and data field stored during an interrupt are read into AC 6-8 and 9-11 respectively.
RMF	6244	Restore memory field. Used to exit from a program interrupt.
Data Communications Systems Type 680		
TTINCR	6401	The content of the line select register is incremented by one.
TTI	6402	The line status word is read and sampled. If the line is active for the fourth time, the line bit is shifted into the character assembly word. If the line is active for a number of times less than four, the count is incremented. If the line is not active, the active/inactive status of the line is recorded.
TTO	6404	The character in the AC is shifted right one position, zeros are shifted into vacated positions, and the original content of AC11 is transferred out of the computer on the Teletype line.
TTCL	6411	The line select register is cleared.
TTSL	6412	The line select register is loaded by an OR transfer from the content of AC5-11, then the AC is cleared.
TTRL	6414	The content of the line select register is read into AC5-11 by an OR transfer.
TTSKP	6421	Skip if clock 1 flag is a 1.
TTXON	6424	Clock 1 is enabled to request a program interrupt and clock 1 flag is cleared.
TTXOF	6422	Clock 1 is disabled from causing a program interrupt and clock 1 flag is cleared.
Incremental Plotter and Control Type VP8/I		
PLSF	6501	Skip if plotter flag is a 1.
PLCF	6502	Clear plotter flag.
PLPU	6504	Plotter pen up. Raise pen off of paper.
PLPR	6511	Plotter pen right.
PLDU	6512	Plotter drum (paper) upward.
PLDD	6514	Plotter drum (paper) downward.
PLPL	6521	Plotter pen left.
PLUD	6522	Plotter drum (paper) upward. (Same as 6512.)
PLPD	6524	Plotter pen down. Lower pen on to paper.
Serial Magnetic Drum System Type 251		
DRCR	6603	Load the drum core location counter with the core memory location information in the accumulator. Prepare to read one sector of information from the drum into the specified core location. Then clear the AC.

BASIC IOT MICROINSTRUCTIONS
(continued)

Mnemonic	Octal	Operation
DRCW	6605	Load the drum core location counter with the core memory location information in the accumulator. Prepare to write one sector of information into the drum from the specified core location. Then clear the AC.
DRCF	6611	Clear completion flag and error flag.
DREF	6612	Clear the AC then load the condition of the parity error and data timing error flip-flops of the drum control into accumulator bits 0 and 1 respectively to allow programmed evaluation of an error flag.
DRTS	6615	Load the drum address register with the track and sector address held in the accumulator. Clear the completion and error flags, and begin a transfer (reading or writing). Then clear the AC.
DRSE	6621	Skip next instruction if the error flag is a 0 (no error).
DRSC	6622	Skip next instruction if the completion flag is a 1 (sector transfer is complete).
DRCN	6624	Clear error flag and completion flag, then initiate transfer of next sector.

Serial Magnetic Drum System Type RM08

DRCR	6603	Load the drum core location counter with the core memory location information in the accumulator. Prepare to read one sector of information from the drum into the specified core location. Then clear the AC.
DRCW	6605	Load the drum core location counter with the core memory location information in the accumulator. Prepare to write one sector of information into the drum from the specified core location. Then clear the AC.
DRCF	6611	Clear completion flag and error flag.
DRES	6612	Clear the AC then load the condition of the parity error and data timing error flip-flops of the drum control into accumulator bits 0 and 1 respectively to allow programmed evaluation of an error flag. The contents of the drum sector counter are transferred into bits AC 6-11.
DRTS	6615	Load the drum address register with the track and sector address held in the accumulator. Clear the completion and error flags, and begin a transfer (reading or writing). Then clear the AC.
DRSE	6621	Skip next instruction if the error flag is a 0 (no error).
DRSC	6622	Skip next instruction if the completion flag is a 1 (sector transfer is complete).

BASIC IOT MICROINSTRUCTIONS (continued)

Mnemonic	Octal	Operation
DRFS	6624	Loads the drum field register with the contents of the accumulator bits 10 and 11. Loads the sector number register with the contents of the accumulator bits 0-5, to specify the number of sectors to be transferred. Loads the three most significant bits of the drum core location register (DCL_{0-2}) with the contents of the AC bits 6, 7, 8 to specify the core memory block to be used during the drum transfer.
Random Access Disc File (Type DF32)		
DCMA	6601	Clears memory address register, parity error and completion flags. This instruction clears the disc memory request flag and interrupt flags.
DMAR	6603	The contents of the AC are loaded into the disc memory address register and the AC is cleared. Begin to read information from the disc into the specified core location. Clears parity error and completion flags. Clears interrupt flags.
DMAW	6605	The contents of the AC are loaded into the disc memory address register and the AC is cleared. Begin to write information into the disc from the specified core location. Clears parity error and completion flags.
DCEA	6611	Clears the disc extended address and memory address extension register.
DSAC	6612	Skips next instruction if address confirmed flag is a 1. (AC is cleared.)
DEAL	6615	The disc extended address extension registers are cleared and loaded with the track data held in the AC.
DEAC	6616	Clear the AC then loads the contents of the disc extended address register into the AC to allow program evaluation. Skip next instruction if address confirmed flag is a 1.
DFSE	6621	Skips next instruction if parity error, data request late, or write lock switch flag is a zero. Indicates no errors.
DFSC	6622	Skip next instruction if the completion flag is a 1. Indates data transfer is complete.
DMAC	6626	Clear the AC then loads contents of disc memory address register into the AC to allow program evaluation.
Automatic Line Printer and Control Type 645		
LSE	6651	Skip if line printer error flag is a 1.
LCB	6652	Clear both sections of the printing buffer.
LLB	6654	Load printing buffer from the content of AC 6-11 and clear the AC.
LSD	6661	Skip if the printer done flag is a 1.

BASIC IOT MICROINSTRUCTIONS (continued)

Mnemonic	Octal	Operation
LCF	6662	Clear line printer done and error flags.
LPR	6664	Clear the format register, load the format register from the content of AC 9-11, print the line contained in the section of the printer buffer loaded last, clear the AC, and advance the paper in accordance with the selected channel of the format tape if the content of AC 8 = 1. If the content of AC 8 = 0, the line is printed and paper advance is inhibited.
DECtape Transport Type TU55 and DECtape Control Type TC01		
DTRA	6761	The content of status register A is read into AC0-9 by an OR transfer. The bit assignments are: AC0-2 = Transport unit select number AC3-4 = Motion AC5 = Mode AC6-8 = Function AC9 = Enable/disable DECtape control flag
DTCA	6762	Clear status register A. All flags undisturbed.
DTXA	6764	Status register A is loaded by an exclusive OR transfer from the content of the AC, and AC10 and AC11 are sampled. If AC10 = 0, the error flags are cleared. If AC11 = 0, the DECtape control flag is cleared.
DTSF	6771	Skip if error flag is a 1 or if DECtape control flag is a 1.
DTRB	6772	The content of status register B is read into the AC by an OR transfer. The bit assignments are: AC0 = Error flag AC1 = Mark track error AC2 = End of tape AC3 = Select error AC4 = Parity error AC5 = Timing error AC6-8 = Memory field AC9-10 = Unused AC11 = DECtape flag
DTLB	6774	The memory field portion of status register B is loaded from the content of AC6-8.
Card Reader and Control Type CR8/I		
RCSF	6631	Skip if card reader data ready flag is a 1.
RCRA	6632	The alphanumeric code for the column is read into AC6-11, and the data ready flag is cleared.
RCRB	6634	The binary data in a card column is transferred into AC0-11, and the data ready flag is cleared.
RCSP	6671	Skip if card reader card done flag is a 1.
RCSE	6672	Clear the card done flag, select the card reader and start card motion towards the read station, and skip if the reader-not-ready flag is a 1.
RCRD	6674	Clear card done flag.

BASIC IOT MICROINSTRUCTIONS (continued)

Mnemonic	Octal	Operation
Automatic Magnetic Tape Control Type TC58		
MTSF	6701	Skip on error flag or magnetic tape flag. The status of the error flag (EF) and the magnetic tape flag (MTF) are sampled. If either or both are set to 1, the content of the PC is incremented by one to skip the next sequential instruction.
MTCR	6711	Skip on tape control ready (TCR). If the tape control is ready to receive a command, the PC is incremented by one to skip the next sequential instruction.
MTTR	6721	Skip on tape transport ready (TTR). The next sequential instruction is skipped if the tape transport is ready.
MTAF	6712	Clear the status and command registers, and the EF and MTF if tape control ready. If tape control not ready, clears MTF and EF flags only.
— —	6724	Inclusively OR the contents of the command register into bits 0-11 of the AC.
MTCM	6714	Inclusively OR the contents of AC bits 0-5, 9-11 into the command register; JAM transfer bits 6, 7, 8 (command function).
MTLC	6716	Load the contents of AC bits 0-11 into the command register.
— —	6704	Inclusively OR the contents of the status register into bits 0-11 of the AC.
MTRS	6706	Read the contents of the status register into bits 0-11 of the AC.
MTGO	6722	Set "go" bit to execute command in the command register if command is legal.
— —	6702	Clear the accumulator.
General Purpose Converter and Multiplexer Control Type AF01A		
ADSF	6531	Skip if A/D converter flag is a 1.
ADVC	6532	Clear A/D converter flag and convert input voltage to a digital number, flag will set to 1 at end of conversion. Number of bits in converted number determined by switch setting, 11 bits maximum.
ADRB	6534	Read A/D converter buffer into AC, left justified, and clear flag.
ADCC	6541	Clear multiplexer channel address register.
ADSC	6542	Set up multiplexer channel as per AC 6-11. Maximum of 64 single ended or 32 differential input channels.
ADIC	6544	Index multiplexer channel address (present address + 1). Upon reaching address limit, increment will cause channel 00 to be selected.

BASIC IOT MICROINSTRUCTIONS (continued)

Mnemonic	Octal	Operation
Guarded Scanning Digital Voltmeter Type AF04A		
VSEL	6542	The contents of the accumulator are transferred to the AF04A control register.
VCNV	6541	The contents of the accumulator are transferred to the AF04A channel address register. Analog signal on selected channel is automatically digitized.
VINX	6544	The last channel address is incremented by one and the analog signal on the selected channel is automatically digitized.
VSDR	6531	Skip if data ready flag is a 1.
VRD	6532	Selected byte of voltmeter is transferred to the accumulator and the data ready flag is cleared.
VBA	6534	BYTE ADVANCE command requests next twelve bits, data ready flag is set.
VSCC	6571	SAMPLE CURRENT CHANNEL when required to digitize analog signal on current channel repeatedly.

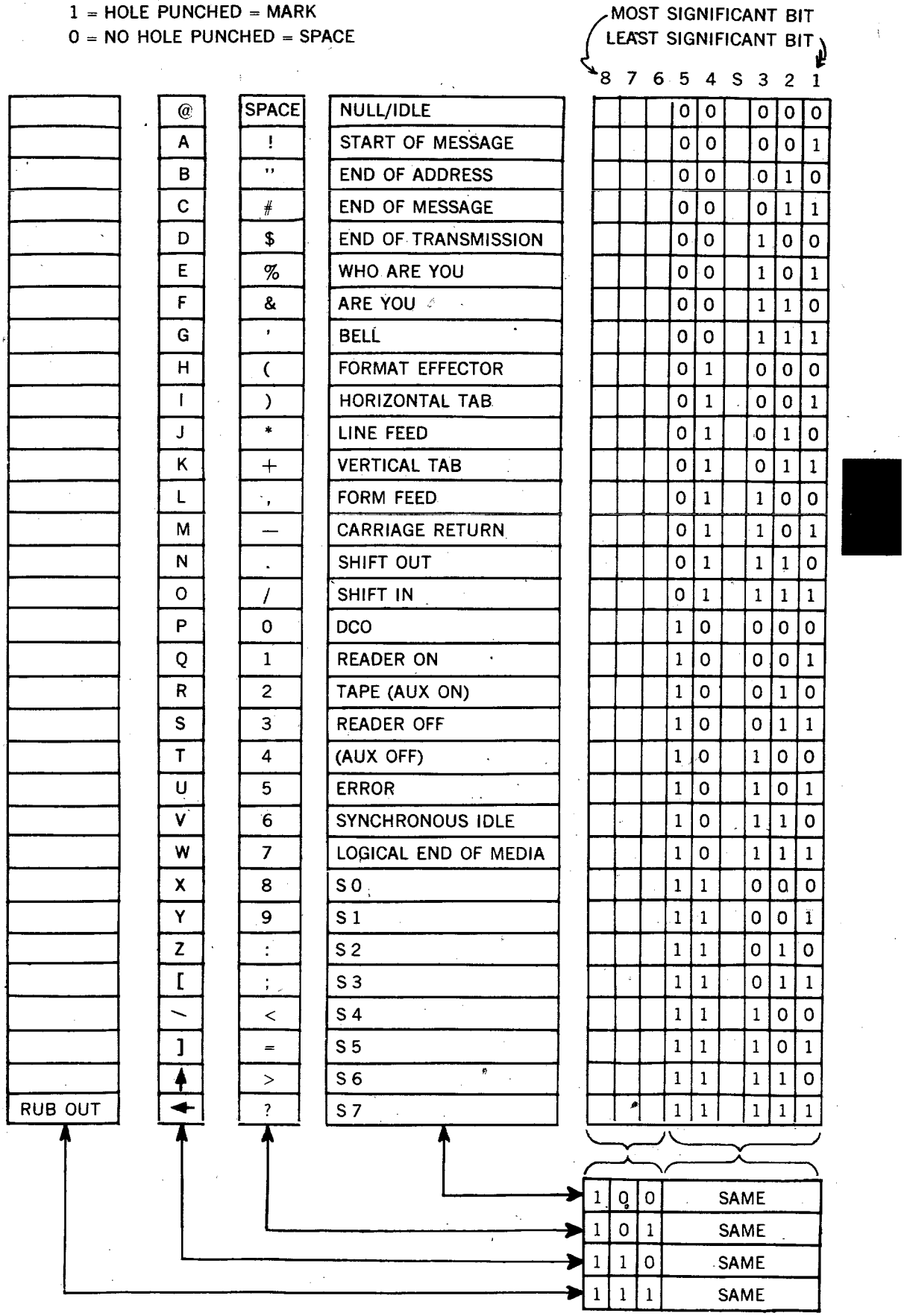
APPENDIX 3
TABLES OF CODES

MODEL 33 ASR/KSR TELETYPE CODE (ASCII) IN OCTAL FORM

Character	8-Bit Code (in octal)	Character	8-Bit Code (in octal)
A	301	!	241
B	302	"	242
C	303	#	243
D	304	$	244
E	305	%	245
F	306	&	246
G	307	'	247
H	310	(	250
I	311	)	251
J	312	*	252
K	313	+	253
L	314	,	254
M	315	–	255
N	316	.	256
O	317	/	257
P	320	:	272
Q	321	;	273
R	322	<	274
S	323	=	275
T	324	>	276
U	325	?	277
V	326	@	300
W	327	[	333
X	330	\	334
Y	331	]	335
Z	332	↑	336
		←	337
0	260		
1	261	Leader/Trailer	200
2	262	Line-Feed	212
3	263	Carriage-Return	215
4	264	Space	240
5	265	Rub-out	377
6	266	Blank	000
7	267	act-mode	375
8	270	escape	233
9	271		

MODEL 33 ASR/KSR TELETYPE CODE (ASCII) IN BINARY FORM

1 = HOLE PUNCHED = MARK
0 = NO HOLE PUNCHED = SPACE



CARD READER CODE

Card Code		Internal Code	Character
Zone	Num.		
—	—	01 0000	Blank
12	8-3	11 1011	.
12	8-4	11 1100	)
12	8-5	11 1101	]
12	8-6	11 1110	<
12	8-7	11 1111	- ←
12	—	11 0000	+
11	8-3	10 1011	$
11	8-4	10 1100	*
11	8-5	10 1101	[
11	8-6	10 1110	>
11	8-7	10 1111	&
11	—	10 0000	—
0	1	01 0001	/
0	8-3	01 1011	,
0	8-4	01 1100	(
0	8-5	01 1101	"
0	8-6	01 1110	#
0	8-7	01 1111	%
—	8-3	00 1011	=
—	8-4	00 1100	@
—	8-5	00 1101	↑
—	8-6	00 1110	'
—	8-7	00 1111	\
12	0	11 1010	?
12	1	11 0001	A
12	2	11 0010	B
12	3	11 0011	C
12	4	11 0100	D
12	5	11 0101	E
12	6	11 0110	F
12	7	11 0111	G
12	8	11 1000	H
12	9	11 1001	I

Card Code		Internal Code	Character
Zone	Num.		
11	0	10 1010	↑
11	1	10 0001	J
11	2	10 0010	K
11	3	10 0011	L
11	4	10 0100	M
11	5	10 0101	N
11	6	10 0110	O
11	7	10 0111	P
11	8	10 1000	Q
11	9	10 1001	R
0	8-2	01 1010	;
0	2	01 0010	S
0	3	01 0011	T
0	4	01 0100	U
0	5	01 0101	V
0	6	01 0110	W
0	7	01 0111	X
0	8	01 1000	Y
0	9	01 1001	Z
—	0	00 1010	0
—	1	00 0001	1
—	2	00 0010	2
—	3	00 0011	3
—	4	00 0100	4
—	5	00 0101	5
—	6	00 0110	6
—	7	00 0111	7
—	8	00 1000	8
—	9	00 1001	9
All other codes		00 0000	←

AUTOMATIC LINE PRINTER CODE

Character (ASCII)	6-Bit Code (in octal)	Character (ASCII)	6-Bit Code (in octal)
@	0	□	40
A	1	!	41
B	2	"	42
C	3	#	43
D	4	$	44
E	5	%	45
F	6	&	46
G	7	'	47
H	10	(	50
I	11	)	51
J	12	*	52
K	13	+	53
L	14	,	54
M	15	−	55
N	16	.	56
O	17	/	57
P	20	ø	60
Q	21	1	61
R	22	2	62
S	23	3	63
T	24	4	64
U	25	5	65
V	26	6	66
W	27	7	67
X	30	8	70
Y	31	9	71
Z	32	:	72
[	33	;	73
\	34	<	74
]	35	=	75
↑	36	>	76
←	37	?	77

APPENDIX 4

PERFORATED-TAPE LOADER SEQUENCES

READIN MODE LOADER

The readin mode (RIM) loader is a minimum length, basic, perforated-tape reader program for the 33 ASR. It is initially stored in memory by manual use of the operator console keys and switches. The loader is permanently stored in 18 locations of page 37.

A perforated tape to be read by the RIM loader must be in RIM format:

Tape Channel 8 7 6 5 4 S 3 2 1	Format
1 0 0 0 0 . 0 0 0	Leader-trailer code
0 1 A1 . A2 0 0 A3 . A4	Absolute address to contain next 4 digits
0 0 X1 . X2 0 0 X3 . X4	Content of previous 4-digit address
0 1 A1 . A2 0 0 A3 . A4	Address
0 0 X1 . X2 0 0 X3 . X4	Content
(Etc.)	(Etc.)
1 0 0 0 0 . 0 0 0	Leader-trailer code

The RIM loader can only be used in conjunction with the 33 ASR reader (not the high-speed perforated-tape reader). Because a tape in RIM format is, in effect, twice as long as it need be, it is suggested that the RIM loader be used only to read the binary loader when using the 33 ASR. (Note that PDP-8 diagnostic program tapes are in RIM format.)

The complete PDP-8/I RIM loader (SA = 7756) is as follows:

Absolute Address	Octal Content	Tag	Instruction I Z	Comments
7756,	6032	BEG,	KCC	/CLEAR AC AND FLAG
7757,	6031		KSF	/SKIP IF FLAG = 1
7760,	5357		JMP .−1	/LOOKING FOR CHARACTER
7761,	6036		KRB	/READ BUFFER
7762,	7106		CLL RTL	
7763,	7006		RTL	/CHANNEL 8 IN AC0
7764,	7510		SPA	/CHECKING FOR LEADER
7765,	5357		JMP BEG+1	/FOUND LEADER
7766,	7006		RTL	/OK, CHANNEL 7 IN LINK
7767,	6031		KSF	

Absolute Address	Octal Content	Tag	Instruction I Z	Comments
7770,	5367		JMP .−1	
7771,	6034		KRS	/READ, DO NOT CLEAR
7772,	7420		SNL	/CHECKING FOR ADDRESS
7773,	3776		DCA I TEMP	/STORE CONTENT
7774,	3376		DCA TEMP	/STORE ADDRESS
7775,	5356		JMP BEG	/NEXT WORD
7776,	0	TEMP,	0	/TEMP STORAGE
7777,	5XXX		JMP X	/JMP START OF BIN LOADER

Placing the RIM loader in core memory by way of the operator console keys and switches is accomplished as follows:

1. Set the starting address 7756 in the switch register (SR).
2. Press LOAD ADDRESS key.
3. Set the first instruction (6032) in the SR.
4. Press the DEPOSIT key.
5. Set the next instruction (6031) in the SR.
6. Press DEPOSIT key.
7. Repeat steps 5 and 6 until all 16 instructions have been deposited.

To load a tape in RIM format, place the tape in the reader, set the SR to the starting address 7756 of the RIM loader (not of the program being read), press the LOAD ADDRESS key, press the START key, and start the Teletype reader.

Refer to Digital Program Library document Digital-8-1-U for additional information on the Readin Mode Loader program.

BINARY LOADER

The binary loader (BIN) is used to read machine language tapes (in binary format) produced by the program assembly language (PAL). A tape in binary format is about one half the length of the comparable RIM format tape. It can, therefore, be read about twice as fast as a RIM tape and is, for this reason, the more desirable format to use with the 10 cps 33 ASR reader or the Type PR8/I High Speed Perforated Tape Reader.

The format of a binary tape is as follows:

LEADER: about 2 feet of leader-trailer codes.

BODY: characters representing the absolute, machine language program in easy-to-read binary (or octal) form. The section of tape may contain characters representing instructions (channels 8 and 7 not punched) or origin resettings (channel 8 not punched, channel 7 punched) and is concluded by 2 characters (channels 8 and 7 not punched) that represent a checksum for the entire section.
TRAILER: same as leader.

Example of the format of a binary tape:

Tape Channel 8 7 6 5 4 S 3 2 1	Memory Location	Content	Comments
1 0 0 0 0 . 0 0 0			leader-trailer code
0 1 0 0 0 . 0 1 0 0 0 0 0 0 . 0 0 0		0200	
0 0 1 1 1 . 0 1 0 0 0 0 0 0 . 0 0 0	0200	CLA	origin setting
0 0 0 0 1 . 0 1 0 0 0 1 1 1 . 1 1 1	0201	TAD 277	
0 0 0 1 1 . 0 1 0 0 0 1 1 1 . 1 1 0	0202	DCA 276	
0 0 1 1 1 . 1 0 0 0 0 0 0 0 . 0 1 0	0203	HLT	
0 1 0 0 0 . 0 1 0 0 0 1 1 1 . 1 1 1		0277	origin setting
0 0 0 0 0 . 0 0 0 0 0 1 0 1 . 0 1 1	0277	0053	
0 0 0 0 1 . 0 0 0 0 0 0 0 0 . 1 1 1		1007	sum check
1 0 0 0 0 . 0 0 0			leader-trailer code

After a BIN tape has been read in, one of the two following conditions exists:

a. No checksum error: halt with AC = 0

b. Checksum error: halt with AC = (computed checksum) − (tape checksum)

Operation of the BIN loader in no way depends upon or uses the RIM loader. To load a tape in BIN format place the tape in the reader, set the SR to 7777 (the starting address of the BIN loader), press the LOAD ADDRESS key, set SR switch 0 up for loading via the Teletype unit or down for loading via the high speed reader, then press the START key, and start the tape reader.

Refer to Digital Program Library document Digital-8-2-U-RIM for additional information on the Binary Loader program.

APPENDIX 5

SCALES OF NOTATION

2^x IN DECIMAL

x	2^x	x	2^x	x	2^x
0.001	1.00069 33874 62581	0.01	1.00695 55500 56719	0.1	1.07177 34625 36293
0.002	1.00138 72557 11335	0.02	1.01395 94797 90029	0.2	1.14869 83549 97035
0.003	1.00208 16050 79633	0.03	1.02101 21257 07193	0.3	1.23114 44133 44916
0.004	1.00277 64359 01078	0.04	1.02811 38266 56067	0.4	1.31950 79107 72894
0.005	1.00347 17485 09503	0.05	1.03526 49238 41377	0.5	1.41421 35623 73095
0.006	1.00416 75432 38973	0.06	1.04246 57608 41121	0.6	1.51571 65665 10398
0.007	1.00486 38204 23785	0.07	1.04971 66836 23067	0.7	1.62450 47927 12471
0.008	1.00556 05803 98468	0.08	1.05701 80405 61380	0.8	1.74110 11265 92248
0.009	1.00625 78234 97782	0.09	1.06437 01824 53360	0.9	1.86606 59830 73615

$10^{\pm n}$ IN OCTAL

10^n	n	10^{-n}	10^n	n	10^{-n}
1	0	1.000 000 000 000 000 000 00	112 402 762 000	10	0.000 000 000 006 676 337 66
12	1	0.063 146 314 631 463 146 31	1 351 035 564 000	11	0.000 000 000 000 537 657 77
144	2	0.005 075 341 217 270 243 66	16 432 451 210 000	12	0.000 000 000 000 043 136 32
1 750	3	0.000 406 111 564 570 651 77	221 411 634 520 000	13	0.000 000 000 000 003 411 35
23 420	4	0.000 032 155 613 530 704 15	2 657 142 036 440 000	14	0.000 000 000 000 000 264 11
303 240	5	0.000 002 476 132 610 706 64	34 327 724 461 500 000	15	0.000 000 000 000 000 022 01
3 641 100	6	0.000 000 206 157 364 055 37	434 157 115 760 200 000	16	0.000 000 000 000 000 001 63
46 113 200	7	0.000 000 015 327 745 152 75	5 432 127 413 542 400 000	17	0.000 000 000 000 000 000 14
575 360 400	8	0.000 000 001 257 143 561 06	67 405 553 164 731 000 000	18	0.000 000 000 000 000 000 01
7 346 545 000	9	0.000 000 000 104 560 276 41			

$n \log_{10} 2$, $n \log_2 10$ IN DECIMAL

n	$n \log_{10} 2$	$n \log_2 10$	n	$n \log_{10} 2$	$n \log_2 10$
1	0.30102 99957	3.32192 80949	6	1.80617 99740	19.93156 85693
2	0.60205 99913	6.64385 61898	7	2.10720 99696	23.25349 66642
3	0.90308 99870	9.96578 42847	8	2.40823 99653	26.57542 47591
4	1.20411 99827	13.28771 23795	9	2.70926 99610	29.89735 28540
5	1.50514 99783	16.60964 04744	10	3.01029 99566	33.21928 09489

ADDITION AND MULTIPLICATION TABLES

Addition | Multiplication

Binary Scale

$$0+0=0 \qquad 0+1=1+0=1 \qquad 1+1=10$$

$$0\times 0=0 \qquad 0\times 1=1\times 0=0 \qquad 1\times 1=1$$

Octal Scale

0	01	02	03	04	05	06	07
1	02	03	04	05	06	07	10
2	03	04	05	06	07	10	11
3	04	05	06	07	10	11	12
4	05	06	07	10	11	12	13
5	06	07	10	11	12	13	14
6	07	10	11	12	13	14	15
7	10	11	12	13	14	15	16

1	02	03	04	05	06	07	
2	04	06	10	12	14	16	
3	06	11	14	17	22	25	
4	10	14	20	24	30	34	
5	12	17	24	31	36	43	
6	14	22	30	36	44	52	
7	16	25	34	43	52	61	

MATHEMATICAL CONSTANTS IN OCTAL SCALE

$\pi = 3.11037\ 552421_8$	$e = 2.55760\ 521305_8$	$\gamma = 0.44742\ 147707_8$
$\pi^{-1} = 0.24276\ 301556_8$	$e^{-1} = 0.27426\ 530661_8$	$\ln \gamma = -0.43127\ 233602_8$
$\sqrt{\pi} = 1.61337\ 611067_8$	$\sqrt{e} = 1.51411\ 230704_8$	$\log_2 \gamma = -0.62573\ 030645_8$
$\ln \pi = 1.11206\ 404435_8$	$\log_{10} e = 0.33626\ 754251_8$	$\sqrt{2} = 1.32404\ 746320_8$
$\log_2 \pi = 1.51544\ 163223_8$	$\log_2 e = 1.34252\ 166245_8$	$\ln 2 = 0.54271\ 027760_8$
$\sqrt{10} = 3.12305\ 407267_8$	$\log_2 10 = 3.24464\ 741136_8$	$\ln 10 = 2.23273\ 067355_8$

APPENDIX 6

POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125
18 446 744 073 709 551 616	64	0.000 000 000 000 000 000 054 210 108 624 275 221 700 372 640 043 497 085 571 289 062 5
36 893 488 147 419 103 232	65	0.000 000 000 000 000 000 027 105 054 312 137 610 850 186 320 021 748 542 785 644 531 25
73 786 976 294 838 206 464	66	0.000 000 000 000 000 000 013 552 527 156 068 805 425 093 160 010 874 271 392 822 265 625
147 573 952 589 676 412 928	67	0.000 000 000 000 000 000 006 776 263 578 034 402 712 546 580 005 437 135 696 411 132 812 5
295 147 905 179 352 825 856	68	0.000 000 000 000 000 000 003 388 131 789 017 201 356 273 290 002 718 567 848 205 566 406 25
590 295 810 358 705 651 712	69	0.000 000 000 000 000 000 001 694 065 894 508 600 678 136 645 001 359 283 924 102 783 203 125
1 180 591 620 717 411 303 424	70	0.000 000 000 000 000 000 000 847 032 947 254 300 339 068 322 500 679 641 962 051 391 601 562 5
2 361 183 241 434 822 606 848	71	0.000 000 000 000 000 000 000 423 516 473 627 150 169 534 161 250 339 820 981 025 695 800 781 25
4 722 366 482 869 645 213 696	72	0.000 000 000 000 000 000 000 211 758 236 813 575 084 767 080 625 169 910 490 512 847 900 390 625

APPENDIX 7

OCTAL-DECIMAL CONVERSION

OCTAL-DECIMAL INTEGER CONVERSION TABLE

0000 to 0777 (Octal) | 0000 to 0511 (Decimal)

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

Octal Decimal
10000 - 4096
20000 - 8192
30000 - 12288
40000 - 16384
50000 - 20480
60000 - 24576
70000 - 28672

1000 to 1777 (Octal) | 0512 to 1023 (Decimal)

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

2000 (Octal) to 2777 (Octal)	1024 (Decimal) to 1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1509	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 (Octal) to 3777 (Octal)	1536 (Decimal) to 2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

4000 to 4777 (Octal)	2048 to 2559 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

5000 to 5777 (Octal)	2560 to 3071 (Decimal)

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

6000 to 6777 (Octal)	3072 to 3583 (Decimal)

Octal	Decimal
10000	4096
20000	8192
30000	12288
40000	16384
50000	20480
60000	24576
70000	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000 to 7777 (Octal)	3584 to 4095 (Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3596	3597	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	1022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

OCTAL-DECIMAL FRACTION CONVERSION TABLE

Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

OCTAL-DECIMAL FRACTION CONVERSION TABLE (continued)

Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL-DECIMAL FRACTION CONVERSION TABLE (continued)

Octal	Decimal	Octal	Decimal	Octal	Decimal	Octal	Decimal
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

PART IV: PRODUCT CATALOG

PDP COMPUTERS

PDP general-purpose digital computers are used for a wide variety of data processing and control functions. PDP's are constructed of highly reliable FLIP-CHIP digital circuit modules, and include built-in provisions for marginal checking. The resulting overall reliability has earned PDP's a reputation for trouble-free performance. An exceptionally varied line of input-output devices are available, and versatile facilities are provided in the computers to handle these and other devices.

A complete, well-documented package of programming aids accompanies each PDP computer. The package includes a FORTRAN compiler, a symbolic assembler, on-line debugging routines, an editor, and utility, arithmetic, and maintenance routines. Editing and on-line debugging programs use the same symbolic language as the assembly systems. This means that debugging is carried out in the same language as the program being debugged, eliminating the creation and reassembly of new symbolic tapes each time an error is found.

The arithmetic subroutines include a floating point package. Input-output subroutines are prepared for most of Digital's standard optional devices. Extensive maintenance routines are provided. Supporting these programming aids are free training courses at Digital and membership in DECUS, the Digital Equipment Computer Users Society. DECUS provides a means for users to exchange ideas and programs through regularly scheduled symposia. A library of fully documented programs is maintained.

PDP-8/I

The PDP-8/I is the newest member in Digital's Family-of-Eight computers. These include the PDP-8, PDP-8/S, DISPLAY-8, TYPESETTING-8, MULTI-ANALYZER-8, and the LINC-8.

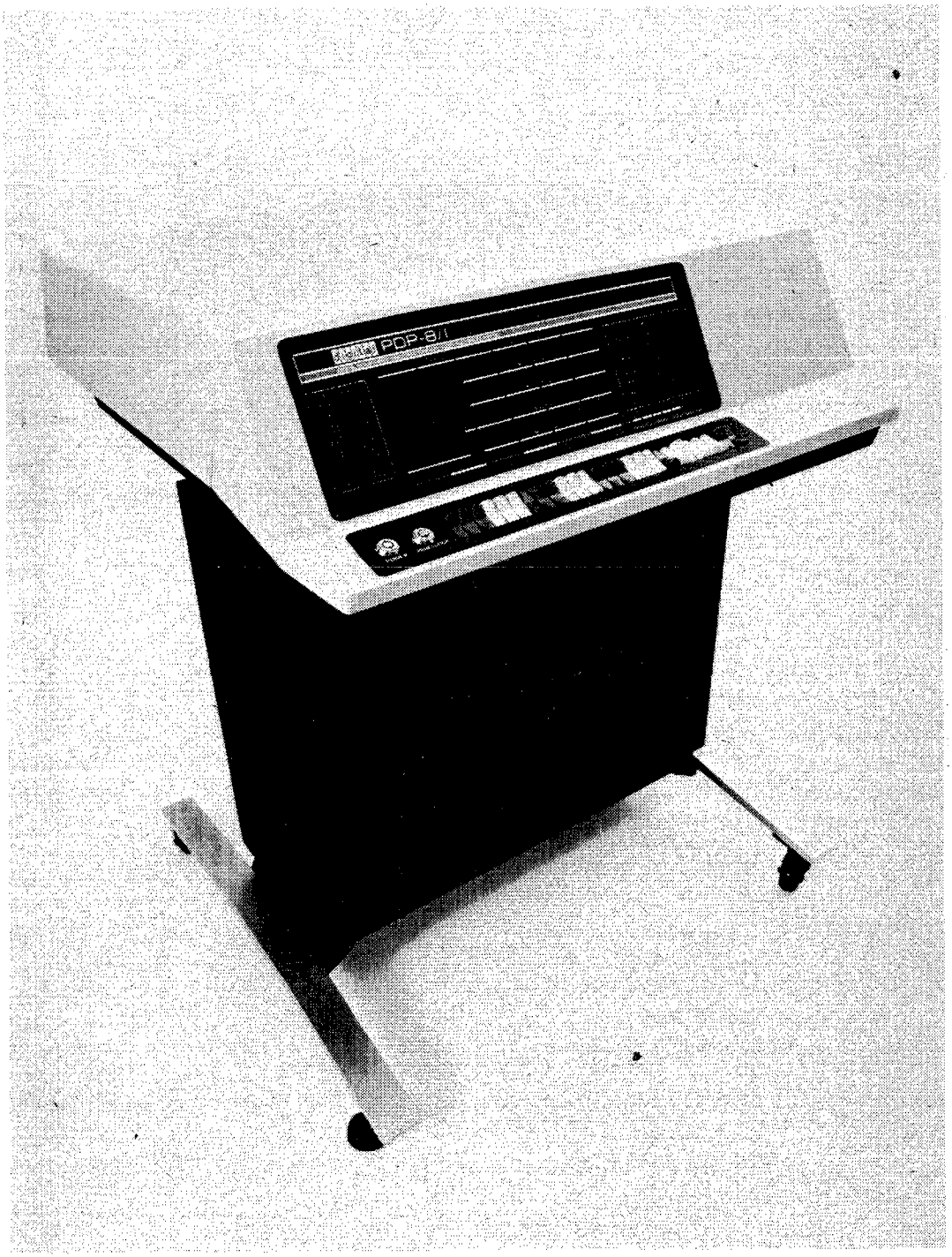
The PDP-8/I offers the power, speed, and expandability of the highly successful PDP-8, but at a significantly lower price. It provides a new ease of interfacing with a wide range of DEC peripherals, including the new random access disk file. It offers a programming system field-proven in nearly 2000 Family-of-Eight installations.

The basic PDP-8/I system features a 1.5 microsecond random access core memory and includes 4096 words of 12-bit ferrite core memory, with a plug-in capability of 8192 words in the basic machine; keyboard printer and tape reader punch. Pre-wiring is also included for a high speed paper tape reader and punch, a 100 card-per-minute card reader, an incremental plotter and a scope display as well.

In addition to Digital's new DECdisk, the PDP-8/I operates with a number of other optional devices such as DECtape, high speed perforated tape readers and punches, card equipment, a line printer, analog-to-digital converters, cathode ray tube displays and magnetic drum systems.

SPECIFICATIONS:

Word Length: 12 bits
Memory: 4096 to 32,768 words; cycle time 1.5 microseconds
Add Time: 3.0 microseconds
In-Out Transfer Rates: 7,992,000 bits per second
Standard I/O Devices: Printer-keyboard with paper tape punch and reader


PDP-8/I

PDP-8

The PDP-8 is a general-purpose, stored-program computer, featuring a 1.5 microsecond random access core memory, a fast arithmetic processor, and a buffered input-output control. These features combine to make the PDP-8 one of the most popular on-line computers for physics and biomedical analysis and process control. The PDP-8 is also used in large systems as a control element and as a training computer.

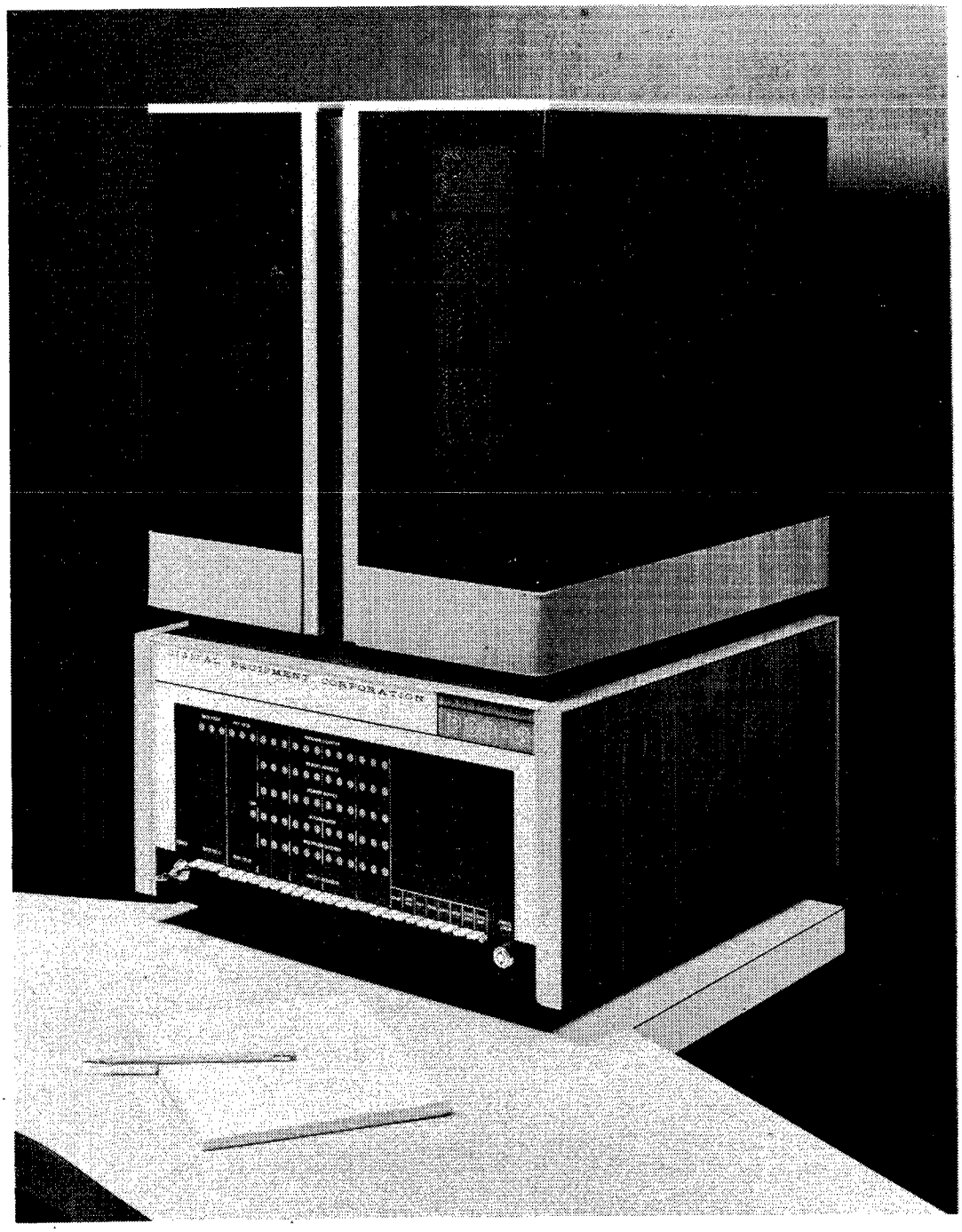
The PDP-8 is easy to install, maintain, and use, with comprehensive software, customer-tested in over 1000 installations. The basic system includes 4096 words of 12-bit ferrite core memory, keyboard-printer and tape reader-punch, eight auto-index registers, wired-in analog-to-digital converter, program interrupt, data interrupt, and indirect addressing.

A partial list of central processor options includes the Extended Arithmetic Element for high speed, double precision arithmetic; Memory Modules and Control for increasing memory size in increments of 4096 words to 32,768 words; a Data Channel Multiplexer providing direct memory access for seven external devices; a Serial Drum for storage of 65,536 to 262,144 words and a 32,768 word random access memory disc.

The applications success of the PDP-8 has led Digital to develop a series of computers based on the PDP-8 to meet a number of special needs, resulting in a unique family of small computer products. These include the DISPLAY 8, the LINC-8, the TYPESETTING-8, the MULTIANALYZER-8, the PDP-8/S, and the new PDP-8/I.

SPECIFICATIONS:

Word Length: 12 bits
Memory: 4096 to 32,768 words; cycle time 1.5 microseconds
Add Time: 3.0 microseconds
In-Out Transfer Rates: 7,992,000 bits per second
Standard I/O Devices: Printer-keyboard with paper tape punch and reader
Instructions: 49 with standard equipment, expandable to over 100 as optional equipment is added.



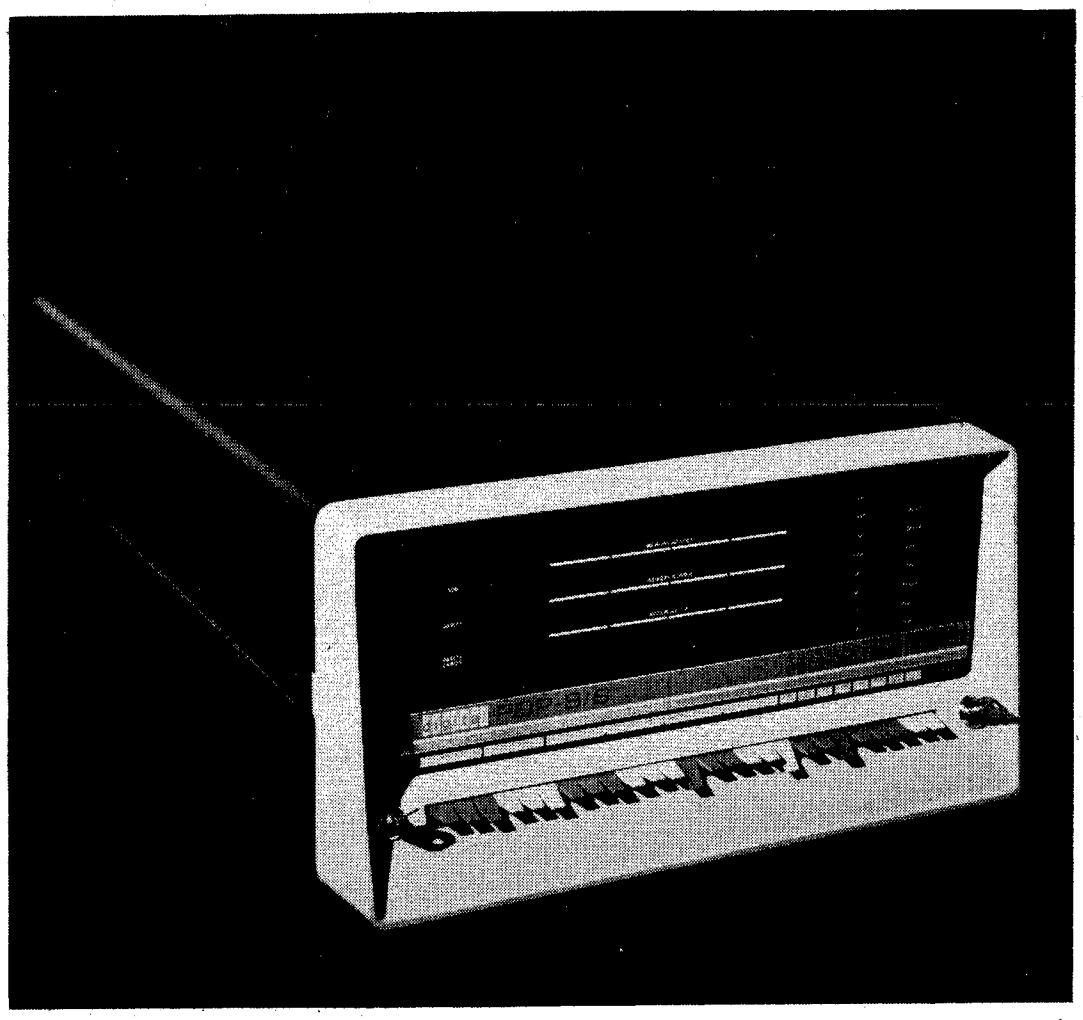
PDP-8/S

The PDP-8/S is the first full-scale, general-purpose, core-memory digital computer selling for under $10,000; it is designed for data handling and for controlling complex process system.

The PDP-8/S has the same size memory, the same input/output capabilities, the same extensive set of standard options as the PDP-8. Both use the same software. The difference between the two machines is in speed and physical size. The PDP-8/S adds in 36 microseconds compared with an add time of 3.0 microseconds for the PDP-8. The basic 12-bit-word PDP-8/S features an 8-microsecond, 4096-word, expandable core memory; a comprehensive software package, including FORTRAN; and an ASR-33 Teletype. Although the PDP-8/S combines a fully parallel core memory and input/output facility with a serial arithmetic unit, the machine appears to be fully parallel to the user. Flexible, high capacity, input/output capabilities of the computer operate a variety of peripheral equipment. In addition to the standard teletype and perforated tape equipment, the system can operate in conjunction with most of the optional devices offered in the PDP-8 family line. Equipment of special design is easily adapted for connection into the PDP-8/S system. The computer need not be modified to add peripheral devices.

SPECIFICATIONS:

Word length: 12 bits
Memory: 4096 to 32,768: cycle time 8.0 microseconds
Add Time: 36 microseconds
In-Out Transfer Rate: 1,500,000 bits per second
Standard I/O Devices: Printer-keyboard with paper tape punch and reader.



LINC-8

The LINC-8 is a computer-based system designed to control experiments and collect and analyze data in the laboratory. The system combines the features of the PDP-8 and the LINC computers, and allows the researcher to choose between the two programming systems available. The researcher simply uses one of the two consoles in the system. Typical biomedical applications for the new system are: arterial shock wave measurements in-phase triggering of stimuli from EEG alpha waves, processing of single-unit data from the nervous system. EKG processing, and operative conditioning applications.

Other applications for the LINC-8 include research in physics, chemistry, meteorology, oceanography, psychology, radiation, seismology, and acoustics.

The original LINC hardware and software were developed for on-line, real-time laboratory research under grants from the National Institutes of Health and the National Aeronautics and Space Administration. Development began at Massachusetts Institute of Technology and continued at Washington University in St. Louis.

The LINC-8 system includes: a built-in multiplexed analog to digital input facility, a relay register, dual digital LINCtape transports, an alphanumeric oscilloscope display and an ASR-33 teletypewriter. The LINC-8 takes advantage of the PDP-8's input/output bus for additional convenience in interfacing other laboratory instrumentation to the LINC-8 system.

With the LINC-8, the researcher has the option of using the LINC software which has been designed to allow the researcher to write his own programs after minimum instruction or he may use the more advanced PDP-8 programming system which includes FORTRAN. The LINC-8 system "talks" with researchers by displaying instructions and results on the oscilloscope display. Displays combine English language with data displays. To familiarize customers with the new system, Digital offers four courses in programming and maintenance of the LINC-8. These are included in the basic system purchase price.

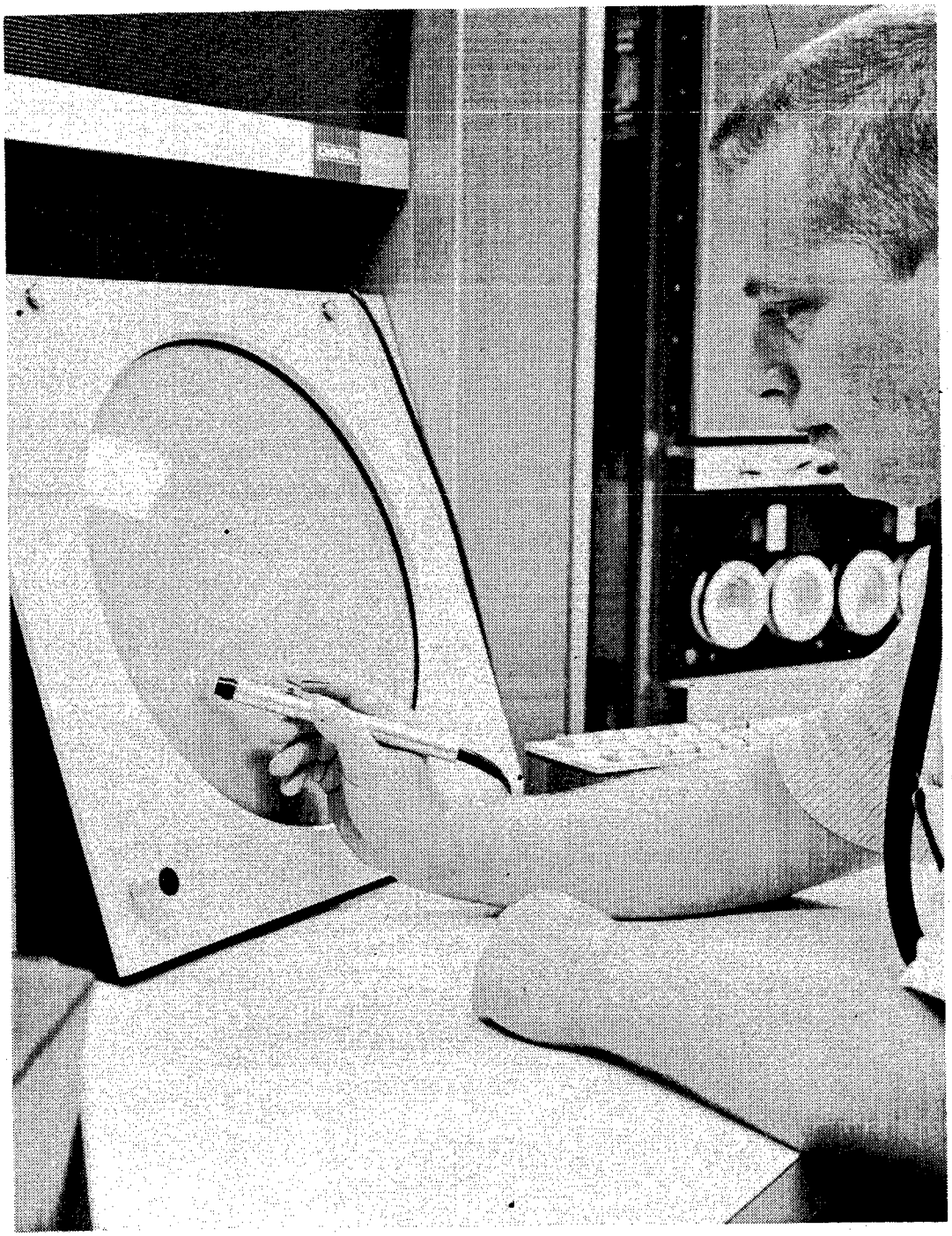
DISPLAY-8

The DISPLAY-8 (Type 338 Programmed Buffered Display) is an integrated cathode-ray-tube system containing its own general-purpose computer. It is capable of precisely displaying points, lines, and characters, and of performing extensive computation using the computer order code and a complete software package.

The computer is a PDP-8. It is fast enough to perform 2,000,000 additions per second while displaying 300,000 points, 600 inches of vector, or 700 characters flicker free at the same time. The highly flexible character generator produces alphabetical characters or special symbols, similar to those used on electronic circuit schematic, with equal ease.

The 338 can be used as a self-contained display system or as a buffered display station in a large computer system. The 338 can control interfaces to external data sources, such as the central computer in a large system, and can handle real time requests, such as data phone interrupts. The 338 can be programmed to view selected small areas of a large stored drawing: 10 by 10 inch window can be moved randomly about a 6 by 6 foot drawing for detailed examination and modification.

The system contains the following features for general purpose computations: An extensive software package that includes FORTRAN, symbolic assembler, debugging programs, floating point arithmetic, and display maintenance programs; 4096 words of core memory; program interrupt; and keyboard-printer and 10-hertz paper-reader punch. The 338 may be expanded using any standard PDP-8 plug-in units.



PDP-9

The PDP-9 is a stored-program, general-purpose digital computer, designed to handle a variety of on-line and real-time scientific applications calling for more computation power than offered by the PDP-8. The basic PDP-9 features a 2-microsecond add time; 8,192 words of 18 bit (plus optional parity bit) core memory; a real-time clock; a 300-character-per-second paper tape reader; a 50-character-per-second tape punch; and input-output teleprinter (Teletype Model KSR-33), Input/Output can be via programmed transfers, data channel transfers, or direct memory access. The maximum I/O transfer rate is 18,000,000 bits-per-second.

Single address instructions are used, with auto-indexing and one level of indirect addressing permitted. A single memory reference instruction can directly address any location in a block of 8,192 words of memory. PDP-9 has a Direct Memory Access channel plus four built-in Data Channels.

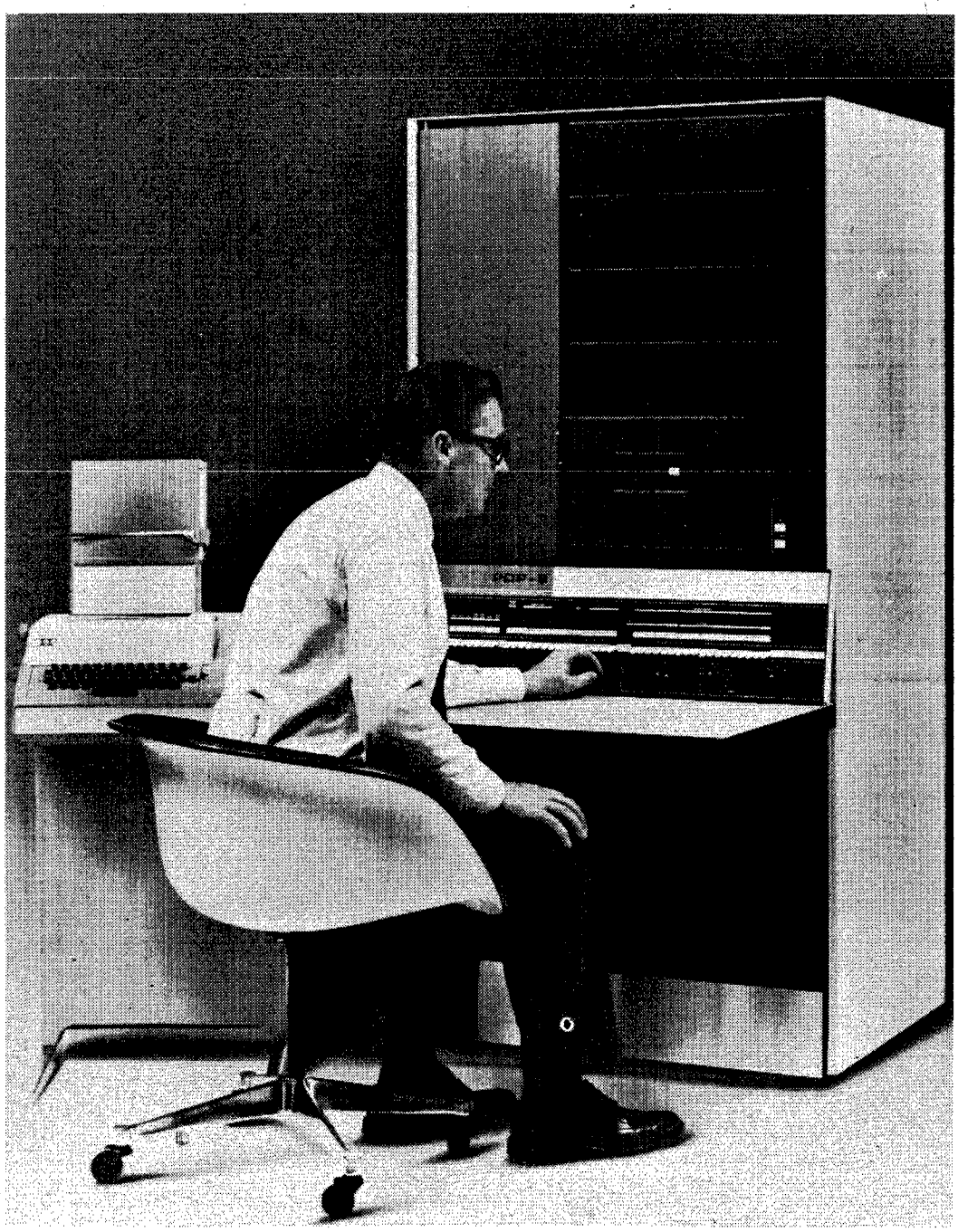
The memory can be expanded in 8,192-word increments to a total of 32,768 words. Mass storage devices, such as DECtape, IBM compatible magnetic tape, disks and drums are available as options for the PDP-9, as are a wide variety of other input-output devices and central-processor additions.

A comprehensive software package including FORTRAN IV, a MACRO Symbolic Assembler, a monitor system, and diagnostic routines is provided with the basic machine. With the modular software package, PDP-9 users can program in a device-independent environment to take full advantage of configurations with mass storage devices and central processor options.

Applications for the PDP-9 include its use in biomedicine, process control, chemical instrumentation, display processing, hybrid systems and data communications. A special configuration, the PDP-9 MULTIANALYZER, has been designed for physics applications.

SPECIFICATIONS:

Word length: 18 bits
Memory: 8,192 to 32,768 words in 8,192 word increments
Cycle time: 1.0 microseconds
Add Time: 2 microseconds
In-Out Transfer Rate: Up to 18,000,000 Bits per second
Standard I/O Devices: A 300 character-per-second paper tape reader, a 50 character-per-second paper tape punch and a 10 character-per-second KSR-33 teletype.
Options: DEC Tape, IBM Compatible magnetic tape, drums, CTRS, A/D converters, line printers, card readers, plotters, etc.



PDP-10

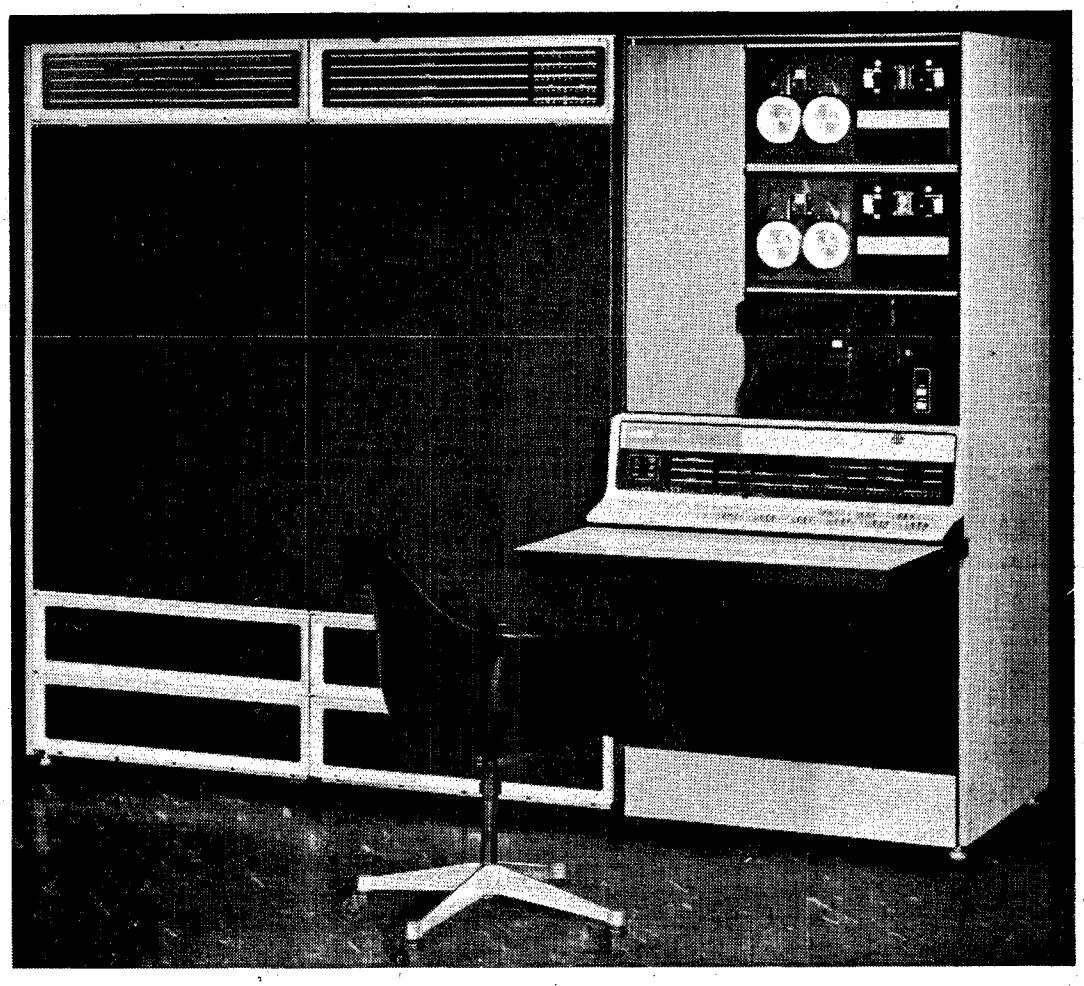
PDP-10 is an expandable, 36-bit computer system available in five configurations (PDP-10/10, 10/20, 10/30, 10/40, and 10/50) and offering optimum power and versatility in the medium price range.

The PDP-10 includes an extremely powerful processor with 15 index registers, 16 accumulators, and 8,192 words of 36-bit core memory, a 300-character-per-second paper tape reader, a 50-character-per-second paper tape punch, a console teleprinter, and a two-level priority interrupt subsystem. PDP-10/20 adds two DEC tapes. PDP-10/30 includes 16,384 words of memory and additional I/O devices. PDP-10/40 adds an extended order code and a memory protection and relocation feature. And PDP-10/50 permits swapping between 32,768 words or more of memory and fast access desk file via the multiplexer/selector channel, and includes multiprogramming time-sharing software.

The PDP-10 is designed for on-line and real-time applications such as physics and biomedical research, process control, as a departmental computation facility, in simulation and aerospace, chemical instrumentation, display processing and as a science teaching aid.

The software package includes real-time FORTRAN IV, a control monitor, a macro assembler, a context editor, a symbolic debugging program, an I/O controller, a peripheral interchange program, a desk calculator and library programs. All software systems assure upward compatibility from the standard 8,192 words of memory through the multiprogramming and swapping systems at both the symbolic and relocatable binary level.

PDP-10 features a 1-microsecond cycle time, a 2.1-microsecond add time, I/O transfer rates up to 7,200,000 bits per second and a modular, proven software package that expands to make full use of all hardware configurations. Memory can be expanded in 8,192 word increments to the maximum directly addressable 262,144 words.



GENERAL PURPOSE ANALOG-DIGITAL CONVERTER/MULTIPLEXERS

Digital offers a wide range of analog-digital and digital-analog converters from a 10-bit single-buffered D/A Converter contained on one double-width FLIP-CHIP™ Module card to a multiplexed integrating digital voltmeter with guarded reed relay scanner providing 140db of common mode rejection and expandable to 2,000 input channels.

Digital's analog-digital converter/multiplexer system is a combined unit with interface for the PDP-8, PDP-8/S and PDP-9 computers. The converter and multiplexer are available either as separate units or as a combined unit without computer interfacing. Optional equipment includes input amplifiers and sample and hold circuitry. The converter offers seven front panel selections of speed and word length. Maximum speed: 6 bits, 1.6% accuracy, 9 microseconds. Maximum accuracy: 12 bits, 0.025% accuracy, 35 microseconds. The multiplexer includes from one to 16 multiplexer switch modules, depending on the number of channels required. Any multiple of four channels may be selected to a maximum of 64. The time required to switch from one channel to another is 10 microseconds to within 1 millivolt of the final voltage. The multiplexer/converter combination is conveniently packaged in a single chassis 19 inches wide by 8-11/16 inches high by 19-1/2 inches deep.

DIGITAL also offers a new analog-digital converter for use with the PDP-8 or PDP-8/S computers to convert an analog input signal to a ten-bit binary number. The A/D Converter is a general purpose successive-approximation type with an accuracy of 0.1% of full scale ±1/2 LSB and a conversion time of 10 μsec. The converter includes cables for connection to the PDP-8 or PDP-8/S I/O Bus and a complete software package with IOT's and diagnostics.


A-D CONVERTER

INPUT-OUTPUT OPTIONS

MAGNETIC TAPE EQUIPMENT

DECtape, a unique fixed address magnetic tape system, allows on-line program debugging or high speed loading and readout. Density is 375 ± 60 bpi; tape speed is 80 ips with a 15 kc character rate. Reads and writes in both directions: redundant tracks allow less than one transient error in 10^{10} characters. Total storage, the equivalent of 4000 feet of perforated tape, is three million bits per reel.

Other magnetic tape systems include automatic and programmed controls and high or low density transports. Formats are IBM compatible at recording densities of 200, 556, and 800 bpi. Transfer rates range from 15 to 90 thousand characters per second. Transports include an electro-pneumatic design of high performance and low tape stress and wear.

RANDOM ACCESS DISC

A new DECdisc random access memory storage device significantly expands the memory capacity of the PDP-8/I, PDP-8, and PDP-8/S computers. The DF 32 has a capacity of 32,768 thirteen bit words (12 bits plus parity) with capability of expansion to 131,072 words. It is a fixed disc with one head per track. Transfer rate is 66 microseconds per 12 bit word. Average access time is 16.67 milliseconds.

MAGNETIC DRUM SYSTEMS

Drums provide auxiliary mass storage with direct access to memory. Sizes range from a 32,768 word drum to 262,144 words.

DISPLAY AND PLOTTING EQUIPMENT

Precision and incremental cathode ray tube displays convert digital data into graphic and tabular form. Light Pen detects plotted points to initiate computer action; Symbol Generator plots alphanumeric or special symbols in four sizes on scope face. Incremental Plotters give hard-copy graphs and histograms.

PRINTERS

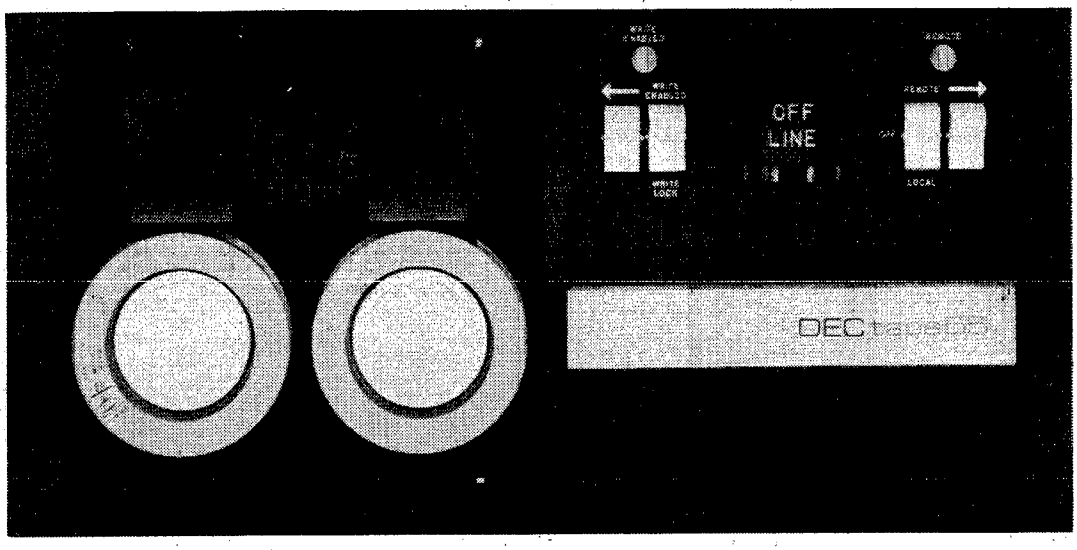
Automatic line printers produce hard-copy output data from 300 to 1000 lines per minute with 120 or 132 column lines and any of 64 characters per column. Teleprinters permit on line inputs and outputs from the computer console or remote stations at 10 characters per second. Character sets are ACSII.

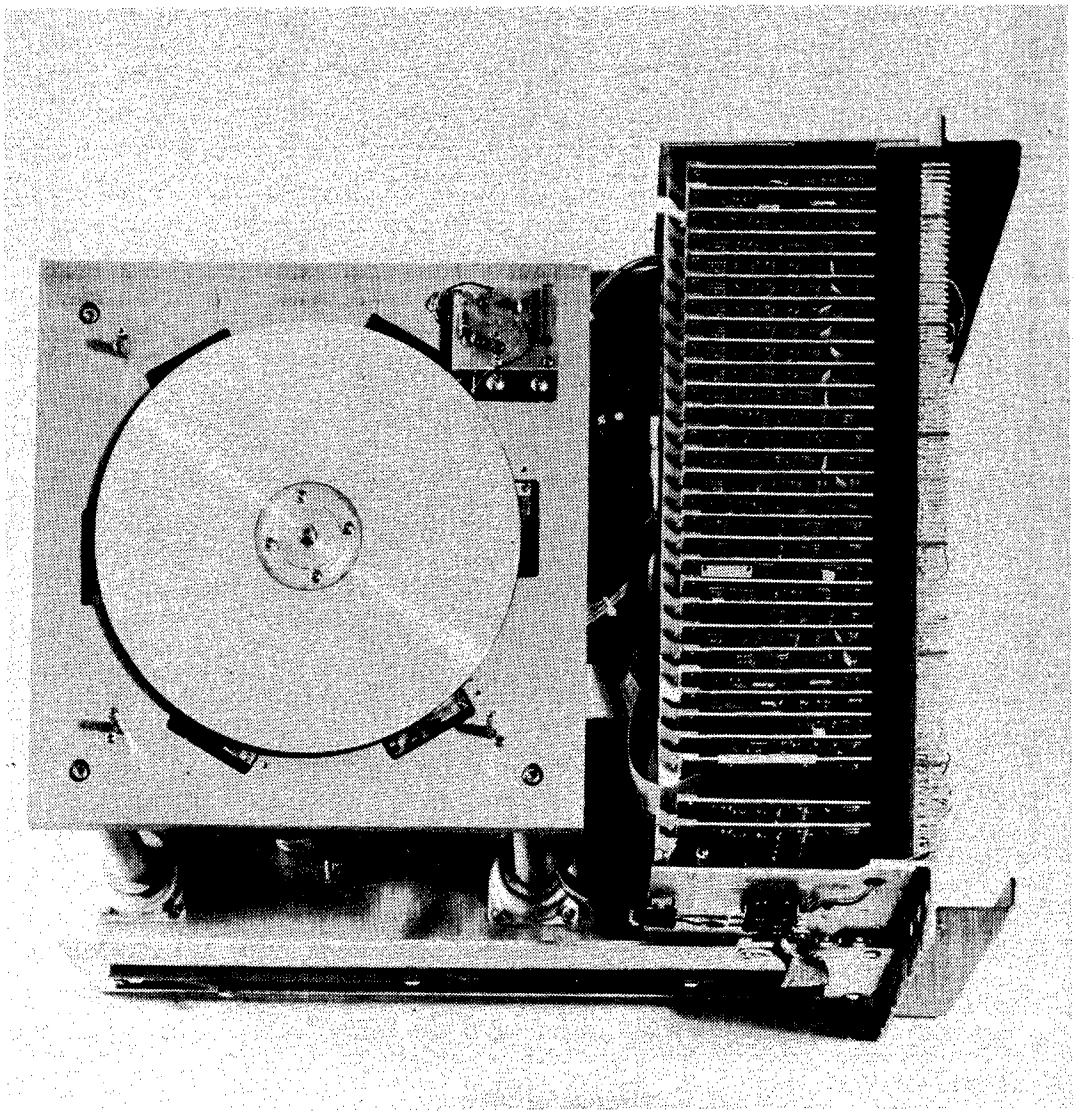
ANALOG-DIGITAL CONVERTERS

General purpose analog to digital converters offer seven front-panel selections of speed and word length. Maximum speed: 6 bits 1.6%. 9 microseconds. Maximum accuracy: 12 bits 0.025% 35 microseconds. Digital-to-analog equipment has maximum conversion time to an accuracy of one least significant bit of 2 μsec. Speeds may be limited by the repetition rate of the associated equipment.

PERFORATED TAPE AND CARD EQUIPMENT

Paper tape punches operate at 10 to 63 characters per second; readers at 10,300, and 400. Card punch controls permit operation at 100 or 300 cards per minute; card readers at 100, 200, or 800.


OFF
LINE
LOCAL



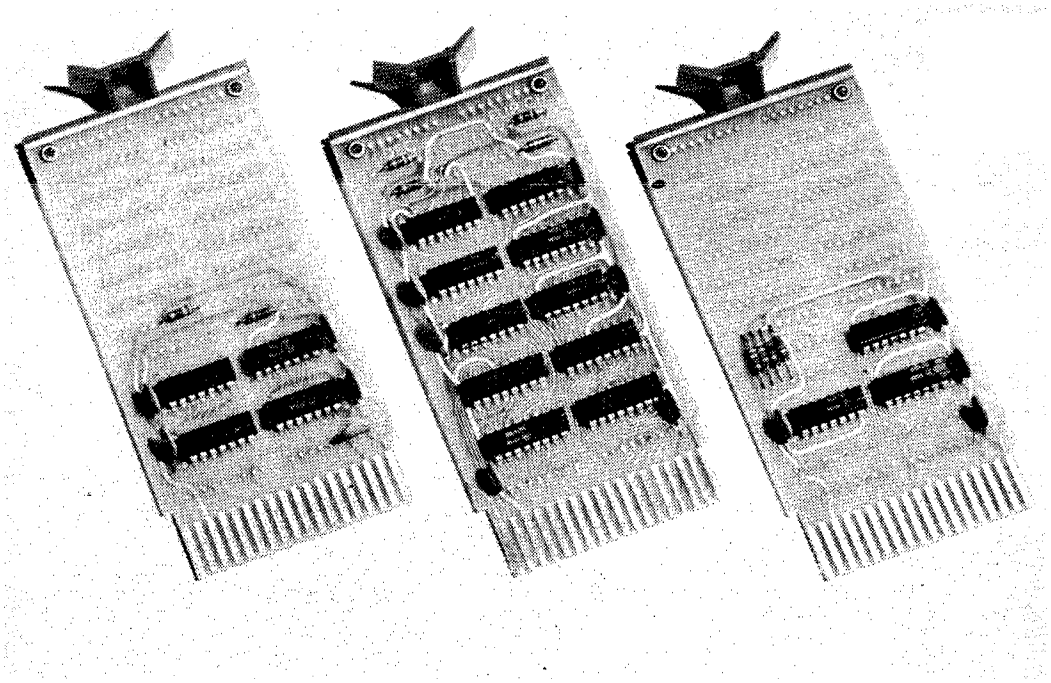
MODULE LINE

DIGITAL is one of the world's largest suppliers of digital circuit modules. These modules have been used in computers, interfaces, and special-purpose systems since 1958. The series includes basic logic modules and interface modules (DC to 10Mhz). DIGITAL also manufactures analog interface modules, a comprehensive line of high-speed TTL system modules (DC to 10 MHZ), and a line of low-speed modules with high noise immunity for use in industrial environment.

The M Series line of high-speed, monolithic integrated circuit logic modules feature TTL (transistor-transistor logic) integrated circuits for high speed, high fan-out and large capacitance drive capabilities coupled with excellent noise margins and a superior power-speed characteristic. M Series includes a full digital system complement of basic modules, designed with sufficient margins for reliable system operation at frequencies up to 6 MHz.

DIGITAL's new K series of Industrial Control modules are designed for process control or data acquisition applications where noise-immune logic is essential. The modules are deliberately slowed to an upper frequency range of 100 KHz with provision for reduction to 5 KHz for maximum noise immunity. K Series modules incorporate all-silicon diodes, transistors and integrated circuits.

DIGITAL's OCTAID and PANELAID kits are designed to provide the logic user with an easy-to-assemble, time-saving group of components to achieve common logic functions, such as up-down counting, decoding, digital-to-analog and analog-to-digital conversion, and computer interfaces. Standard FLIP-CHIP modules and connectors are used in conjunction with special purpose printed circuit interconnectors. Specific modules may be operated at frequencies up to 10 MHz.



NOTES

CUT ALONG DOTTED LINE

FOR MORE PRODUCT INFORMATION
INFORMATION REQUEST

☐ Please add my name to your mailing list to receive future technical bulletins and application notes as they are issued.

☐ Please forward any available information of the following application(s):

_____

_____

_____

☐ Please have a Digital engineer contact me for an appointment.

☐ I would like to discuss the following application(s):

_____

_____

_____

☐ Please send technical literature on the following Digital products:

☐ PDP-8/I ☐ PDP-9 ☐ PDP-8 ☐ PDP-8/S ☐ LINC-8

☐ CRT Displays ☐ Modules

☐ Other _____

Name _____ Position _____

Company _____

Dept. or Div. _____

Business _____

Street _____

City _____ State _____

ZIP Code No. _____ Telephone _____

FOLD HERE

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital EQUIPMENT CORPORATION

TECHNICAL PUBLICATIONS DEPT.
146 MAIN STREET
MAYNARD, MASS. 01754

DEC also has available a 400 page Digital Logic Handbook, a comprehensive reference manual for both the logic designer and the student. Over half the book is devoted to applications information and technical data. Special interest sections cover subjects such as analog-digital and digital-analog conversion techniques and logic training-breadboarding equipment. There are also detailed design specifications for more than 150 FLIP-CHIP Modules and accessories—the industry's most complete line of logic circuits.

Perhaps you have a friend who would like to receive a free copy of the Digital Logic Handbook or the Digital Small Computer Handbook. If so, please fill out the card below.

----------FOLD HERE----------

DIGITAL EQUIPMENT CORPORATION
Technical Publications Dept.
146 Main Street
Maynard, Mass. 01754

GENTLEMEN:

☐ Please send a free copy of The Digital Small Computer Handbook to:

☐ Please send a free copy of The Digital Logic Handbook to:

Name _____

Position _____

Company _____

Business _____

Street _____

Telephone _____

City _____

State _____ Zip Code No. _____

FOLD HERE

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital EQUIPMENT CORPORATION

TECHNICAL PUBLICATIONS DEPT.
146 MAIN STREET
MAYNARD, MASS. 01754

This new edition of DIGITAL'S Small Computer Handbook contains an expanded basic computer primer with four step-by-step examples of the use of small computers in scientific research and in process control. The book also contains three detailed User Handbooks, one for each of the Family-of-Eight general purpose computers — PDP-8, LINC-8, and the new PDP-8/S.

PRINTED IN U.S.A. 1750-3/67