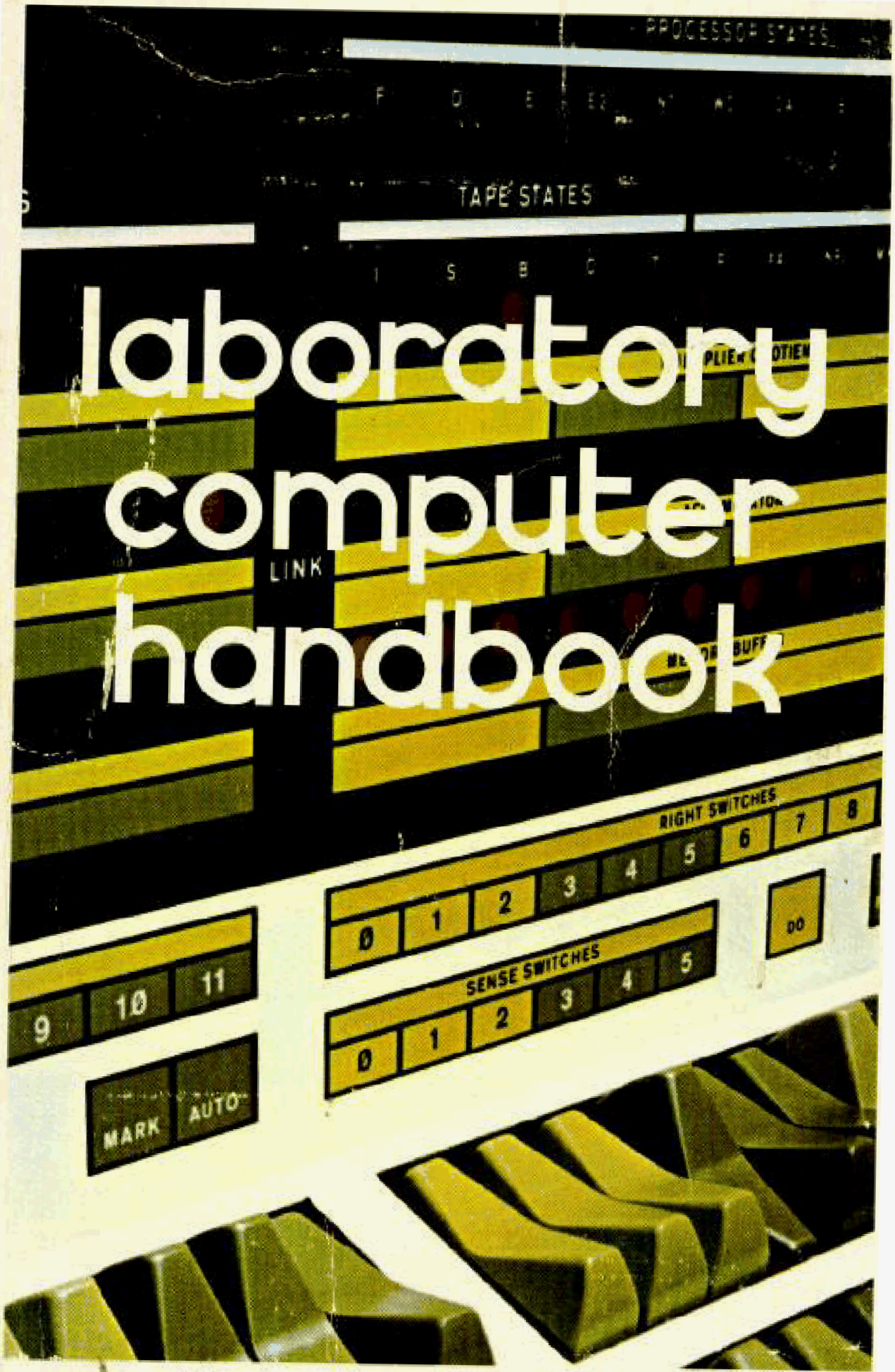
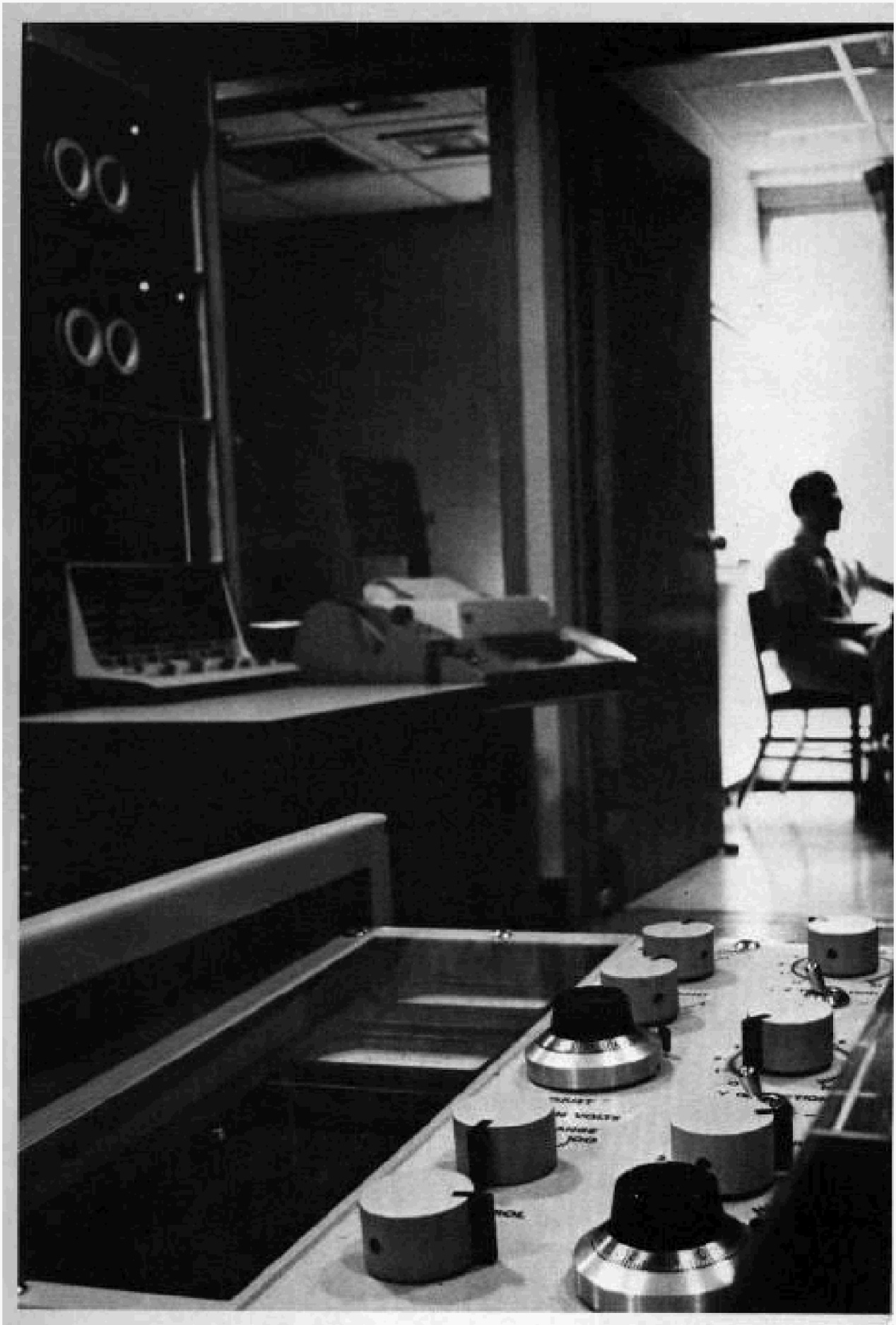
laboratory computer handbook

LINK



digital equipment corporation



The PDP-12 is designed for ease of use by laboratory researchers, yet retains total flexibility for experimenting with various data handling techniques.

digital

laboratory computer handbook

1971

digital equipment corporation

FOREWORD

The PDP-12 general purpose laboratory computer is a powerful partner to research and developmental activities; especially for storing, collating, combining, and analyzing laboratory data. It converses directly with the laboratory user while experiments take place. It displays results in time for him to vary experimental sequences based upon emerging results, giving him the power for creative problem solving, not mere data shuffling.

The laboratory user also gets long-term instrumentation economy. Economy and the ability to change data instrumentation tasks without changing the instrument. Economy in the ability to expand (including the size of the computer) by adding modules or options.

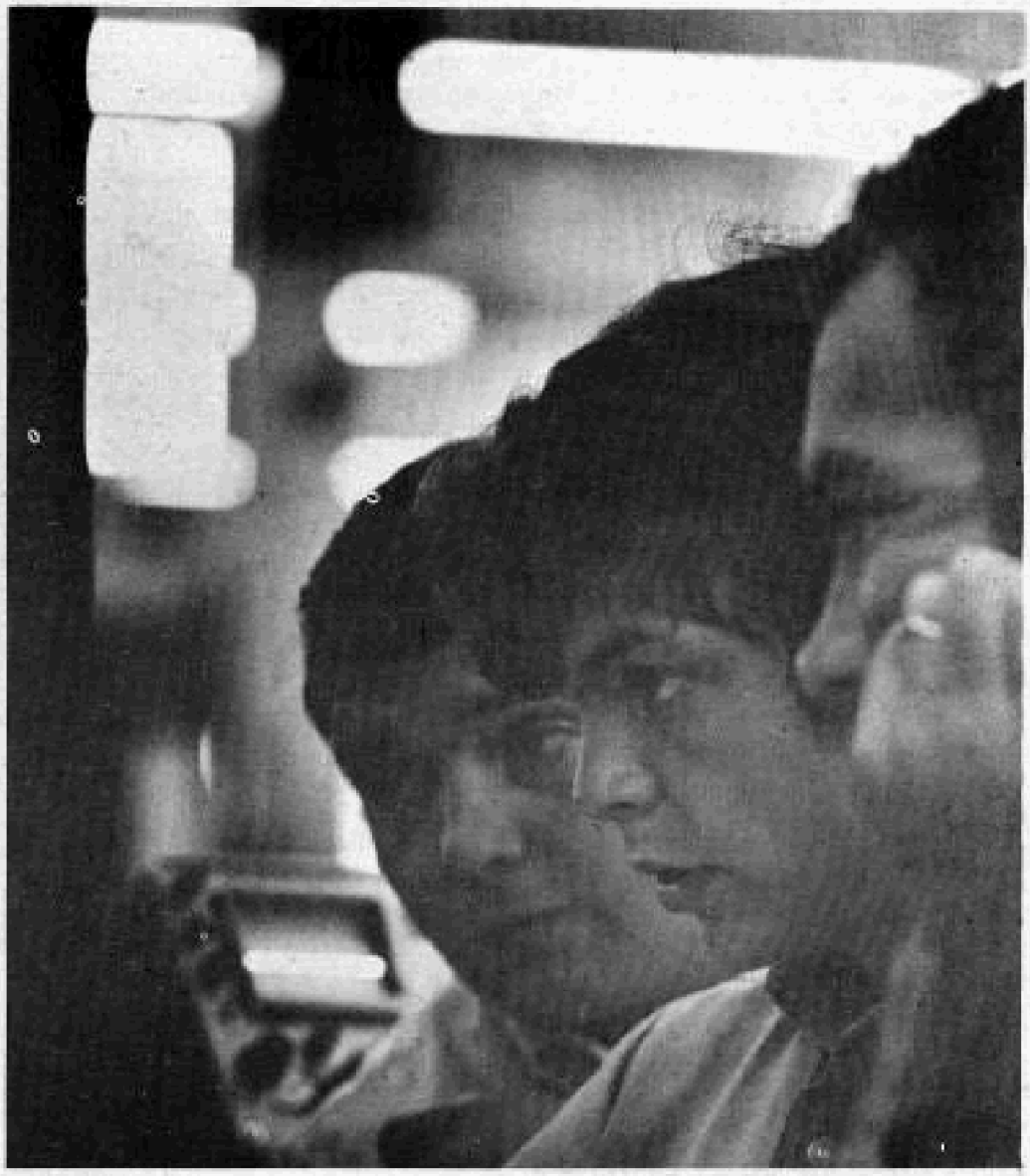
There is an investment to be made. Several weeks of diligent study of computer fundamentals to start. But, the payoff is great. Dividends come in terms of an entire career, and adding the computer discipline to your own, and the ability to do more and better laboratory work.

Digital Equipment Corporation has played a pioneering role in developing all purpose computers, particularly suited for the laboratory environment. Thousands of DEC computers are being used at universities, research and development centers, industry, pure and applied research, and physical, life, and behavioral sciences.

With development of the PDP-12, DEC has combined the features of three successful general purpose computers — the LINC, developed under National Institutes of Health and NASA grants, the popular PDP-8, and the LINC-8, into a single, complete data handling facility.

The PDP-12, described in this handbook, has become a standard in its field. It is the most modern of a family of 12-bit machines, designed specifically for the laboratory market place. It has the capability of utilizing programs written for the LINC, PDP-8, and LINC-8 with relatively minor modification.

Some of the programming excerpts used in this handbook are derived from several works written by persons outside of DEC. "Programming the LINC" LINC DE16, Section 2, Programming & Use, April 1965 by Mary Allen Wilkes and Wesley A. Clark, Washington University, St. Louis, Missouri. To the above individuals, as well as others, at the Computer Research Laboratory at Washington University, the National Institutes of Health, the National Aeronautics & Space Administration, and individual DEC computer users, we are greatly indebted.



Today's graduating engineer, scientist, and technician will encounter the computer in his professional life not simply as a computational aid, but as an integral part of an industrial system or scientific experiment.

INTRODUCTION

INTRODUCTION TO LABORATORY COMPUTERS

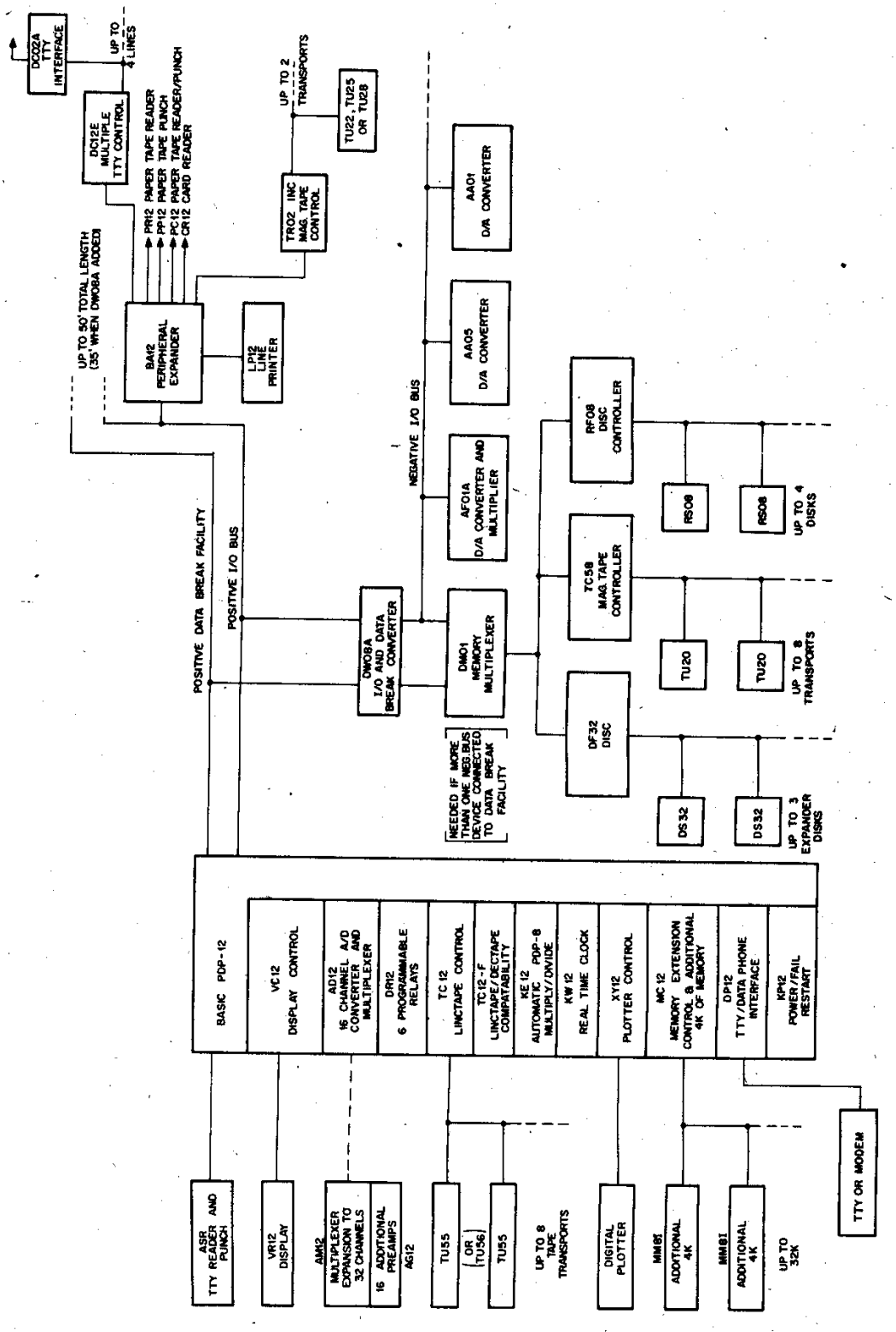
Until approximately 1963-1964, the concept of having a computer for use by researchers within the laboratory environment was almost unheard of, except in very special cases. Both the physical size and the price of such equipment was awesome, and it was highly unlikely that a single researcher, or small group of researchers could put forth a strong enough argument to justify the purchase of equipment that would cost several hundred thousand dollars. At that time, there was very little available, if any, that had the characteristics so necessary of a laboratory environment which is the requirement for "hands-on, on-line, interactive computing capability."

The use of a computer in a laboratory, represented a unique departure from the classical use of most computational equipment. In most cases, data from experiments was acquired off-line and then prepared for entry to the computer, typically via punched cards, paper tapes, or digital magnetic tapes. When time on a central computer was available, the data was analyzed by programs that had previously been prepared and finally the results of the experiment were available to the researcher. Typically, the "turnaround" time was at best several hours, and at worst several days. The use of the computer for the researcher was beginning to prove a valuable tool in the data reduction and analysis of his work, but the time required to give him the all important "feedback" about his experiment was still much too long. The optimum situation would be where the computer could be integrated into the experimental setup and the data could be acquired on-line and analyzed in the laboratory, and eventually feed back and control the experimental setup.

One of the early projects aimed at developing such a computer was the LINC program. LINC was a pseudonym for a *L*aboratory *IN*strument Computer and was designed primarily to fulfill the needs of the life and physical sciences researcher. The initial designs were aimed at fulfilling the following requirements:

1. Ease of interface to standard laboratory instrumentation, which included both analog and digital signals.
2. CRT display — for both graphical and numerical data display and results.
3. Auxiliary storage — for rapid access to both programs and data.
4. Ease of use — such that semi-skilled laboratory technicians could operate the equipment and run the experiments.

The fulfillment of these requirements together with improvements in the state of the art, and price/performance ratios were embodied in the LINC, LINC-8, and eventually the PDP-12 as manufactured by Digital Equipment Corporation. The present PDP-12 represents the third generation of this laboratory computer concept. As of mid-1970 there are over 600 such machines in use throughout the world in numerous laboratory environments.



PDP-12 Programmed Data Processor System, Functional Block Diagram

HOW THE PDP-12 FULFILLS THE LABORATORY REQUIREMENTS

Inherent in the definition of a laboratory computer system is the term "flexibility". It is mandatory that the user isn't required to change his instrumentation, but rather only experimental conditions. In this respect the laboratory computer as a laboratory instrument is quite unique. Its general purpose nature allows the user to reprogram it, handle a larger range of experimental conditions and environments without physical change to the hardware configuration.

"Flexibility" — The PDP-12 includes within its single central processor two distinct operating modes, each with its own complete instruction set. Like its predecessor, the LINC-8, the PDP-12 operates in one mode as a LINC (Laboratory INstrument Computer) and in the other mode as a PDP-8 computer — specifically a PDP-8/I. Both operating modes have equal status, and the computer may be stopped and started in either mode, and the programs may switch from one mode to the other at will. Computations in one mode are immediately available to programs operating in the other mode, plus only one set of processor registers are involved.

The basic memory capacity of the PDP-12 is 4096 (4K) 12 bit words and can be expanded to 32,768 (32K) words of 1.6 μsecond core storage.

The input/output facilities are available to the two operating modes of the PDP-12 in the following manner through LINC mode programming:

LINCtape — two tape transports controlled by a buffered subprocessor.

CRT Display — 6" x 9" screen, two intensification channels.

Analog Inputs — eight variable potentiometers, eight external inputs, expandable to 24.

Relay Buffer — six relays for control of external equipment.

In addition to these, the PDP-12 is also equipped with a positive logic PDP-8/I type input/output (I/O) bus, to which can be attached, all PDP-8 peripherals and options such as a high-speed paper tape reader and punch, as well as the standard ASR-33 Teletype.

Central Processor

The central processor contains all the logic and registers required to carry out the functions of both operating modes of the PDP-12. The central processor can best be described in terms of its active registers:

Accumulator (AC) 12 Bits — This register contains data being operated upon. Its contents may be shifted or rotated right or left; incremented, cleared, or complemented; stored in memory or added to the contents of a memory register; and logically or arithmetically compared with the contents of any memory register. The AC holds the sum after an addition, and part of the product after a multiplication. The AC is also involved in the transfer of data to and from various other registers outside the central processor.

Link (L) 1 Bit — The Link is an extension of the AC. When a carry occurs out of bit 0 of the AC during a 2's complement addition, the Link is complemented. It may be set or cleared independently of the AC, and

may be included (or not) in shifting and rotating operations performed on the contents of the AC.

Multiplier Quotient (MQ) 12 Bits — This register is used as a second arithmetic register for multiply and some rotate instructions. It is also used for the extended Arithmetic Option (KE 12) functions.

Program Counter (PC) 12 Bits — This register contains the address of the next instruction to be executed within the memory field selected by the Instruction Field Register (see below). In PDP-8 mode, the PC acts as a 12-bit counter; in LINC mode, it acts as a 10-bit counter.

Memory Address Register (MA) 12 Bits — This register contains the address for memory references. Whenever a core memory location is being accessed, either for reading or for writing, the MA contains the address of that location.

Instruction Register (IR) 12 Bits — This register contains the complete binary code of the instruction being executed.

Memory Buffer (MB) 12 Bits — All information passing between memory and any other register in the PDP-12 must go through the Memory Buffer Register, whether the transfer involves the central processor, an external device, or another memory register.

Instruction Field Register (IF) 5 Bits — This register selects the memory field containing the executable program. In LINC mode, it is used to designate one of up to thirty-two 1024-word segments. In PDP-8 mode, the three high-order bits of the IF are used to designate one of up to eight 4096-word fields.

Data Field Register (DF) Bits — This register selects the memory field containing data to be indirectly accessed by the memory reference instructions of a program. The fields are specified in each mode in the same way that the IF specifies the Instruction Field.

Memory

The principal unit of core memory is a module of 4096 (4K) 12-bit words. Additional 4K banks may be added, to a total of eight, or 32,768 words. Within each bank, the logical organization of memory depends on the operating mode. In LINC mode, the bank is divided into four 1024-word segments. At any given time, only two of these segments are active: the Instruction Field, which contains the executable program and the directly accessed data; and the Data Field, which contains only indirectly accessed data. Absolute addresses may be assigned and changed at will using the IF and DF described above.

In the PDP-8 mode, the memory field, which is the size of a 4K module, is divided into 32 pages of 128 words each. Within a single page, data may be accessed directly; between pages, indirect addressing must be used. If more than 4K of memory is provided, the IF and DF registers specify the active fields.

Operating Modes

The two operating modes, LINC and PDP-8, are independent of each other, though they may be combined and intermixed within a program. The user can run programs from the already-existing libraries for the PDP-8 family of computers including the LINC-8. Using the I-O Handler (PROGOFOP simulator) program provided with the PDP-12 basic software, most programs written for the LINC-8 can be run without modification. (Some LINC-8 programs may require slight changes). A complete software system designed for PDP-12 allows the programmer to assemble coding for either or both modes in a single program.

LINC Mode — In this mode, the instruction set of the classic LINC computer is implemented. In addition, several new provisions are available:

Extended Tape Addressing — This allows the programmer to transfer information between LINCtape and any section of core, removing the restriction to specific quarters of a given memory field. Other features include:

1. Tape Interrupt — which connects the tape processor status to the Program Interrupt.
2. No-pause — which permits the central processor to resume operation after initiating a tape transfer without waiting for completion.
3. Hold-motion — which allows a unit to remain in motion after deselection.

I/O Bus Access — In LINC mode the user has immediate access to those devices activated by LINC instructions A-D, DISPLAY, RELAYS, SENSE LINES, and TAPE. Any device connected to the I/O bus may be directly accessed from LINC mode programming by means of a special two-word instruction, in which the second word enables the bus and initiates the PDP-8 IOT timing chain. This second word is interpreted as a standard PDP-8 IOT instruction. The program continues to operate in LINC mode.

Special Functions — The LINC programmer may, by setting certain flip-flops: 1) change the size of characters displayed on the CRT; 2) enable the program trap, which intercepts certain LINC instruction codes; 3) disable interrupts from the ASR-33; 4) speed the sampling of analog inputs; and 5) clear the PDP-12 status by generating an I/O preset pulse.

PDP-8 Mode — In this mode, the user has available the entire PDP-8/I instruction set.

Interaction Between Modes — The user may switch from one mode to the other at will. In LINC mode, execution of the instruction *PDP* causes the processor to change immediately to PDP-8 mode operation, and all subsequent instructions are interpreted as PDP-8/I instruction. To switch from PDP-8 mode to LINC mode the IOT instruction *LINC* is used.

Input/Output Facilities and Display

As can be seen from Figure 1-1, there are two main paths for the transmision of data from the central processor or memory to peripheral

devices. One path, which is controlled by LINC mode programming, leads to the CRT display, LINCtape, A-D converter, and relays. The other path, which is the I/O bus, leads to the ASR-33 and to a large number of optional devices, such as: plotters, high-speed tape and card readers, disk storage, line printers, etc.

Display — The Cathode Ray Tube Display has a 58.5-square inch (6.5 x 9 inches) screen, on which individual points and whole characters may be displayed. The unit has two intensification channels, controlled by programming and by a switch on the display. Characters are plotted on a 4 point x 6 point matrix; a full character can be displayed with two instructions. Provision is made for displaying two sizes of characters.

Data Terminal — The data terminal provides a flexible means of receiving analog inputs and controlling the operation of external equipment not directly interfaced to the PDP-12.

Analog Inputs — Sixteen analog inputs feed a 10-bit A-D converter. A single LINC mode installation samples any of the 16 channels. A second set of sixteen inputs with preamplifiers, can be added to the basic facility.

Eight of the inputs, taken from phone jacks mounted on the Data Terminal Panel, are fed through preamplifiers to the converter. The remaining eight are taken from continuously-variable, ten-turn potentiometers, which are also mounted on the panel.

Relay Buffer — Six relays, mounted on the Data Terminal Panel, may be switched by means of a LINC mode instruction. The relays may be used to start and stop operations in external equipment. The status of the relays can be read back into the AC.

Auxiliary Scope Connector — A Blue Ribbon connector mounted on the Data Terminal Panel is wired to accept an auxiliary CRT for displaying information also sent to the screen of the built-in scope.

Sense Lines

These 12 digital sense lines may be individually tested with a LINC mode instruction.

LINCtape — The tape transports are controlled by a fully-buffered tape processor; once initiated by the LINC program, tape operations can be carried out independently of the central processor. Tapes normally are written and read in standard LINCtape format, though non-standard formats can be used. A special hardware option, TC12F, permits the use of all DECtape formats. In addition to the basic LINCtape commands, the PDP-12 also includes an Extended Operations facility, which allows, among other features, the transmission of data between tape and any program-defined area of memory, and the addition of TU55 or TU56 transports to a total of eight tape drives.

Input/Output (I/O) Bus — This connecting facility provides the control and data transmission path between the central processor and any peripheral devices that are attached to the bus. For some devices, such as: paper tape readers and punches, line printers, and incremental plotters, data is transferred via the accumulator (AC). Others, including

magnetic tape and disk, use the three-cycle Data Break for direct memory access. The I/O bus uses positive logic and accepts peripherals used with 8 family of computers. The processor is prewired to accept the following I/O bus options:

Extended Arithmetic Element (EAE), Type KE12

Programmable Real-time clock, Type KW12 A, B, or C

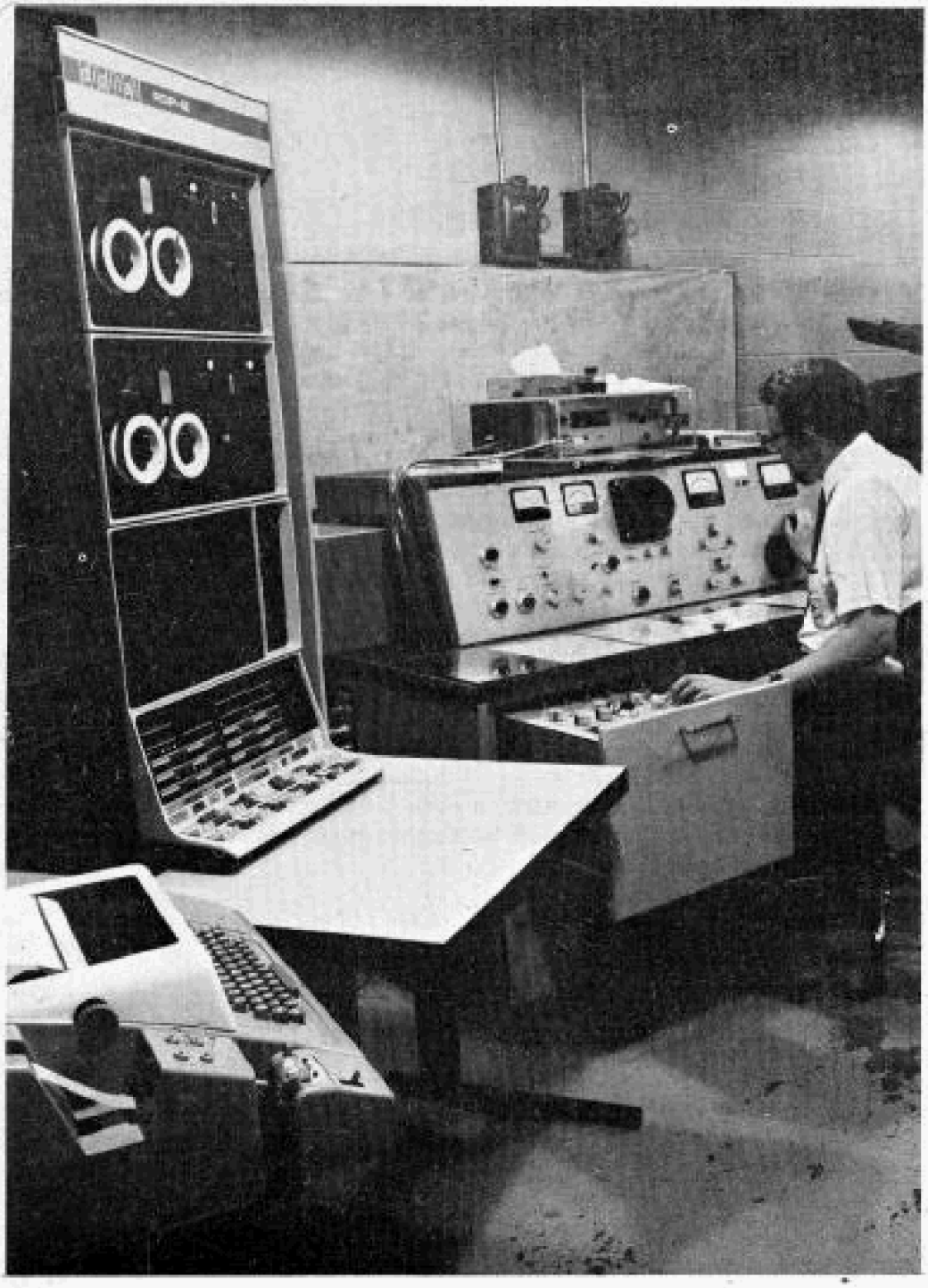
Incremental Plotter and control, Type XY12

TTY or Dataphone Interface, Type DP12

Many other devices can be added to the PDP-12 I/O bus with the inclusion of the BA12 Peripheral Expander and the DW08 I/O and Bus Converter. The Peripheral Expander allows the addition of high-speed paper tape reader and punch, card reader, and optional communication interfaces. The Bus Converter provides for the addition of disk and IBM-compatible magnetic tape storage, and A/D and D/A converters and associated multiplexers designed for the negative-logic PDP-8 I/O Bus.

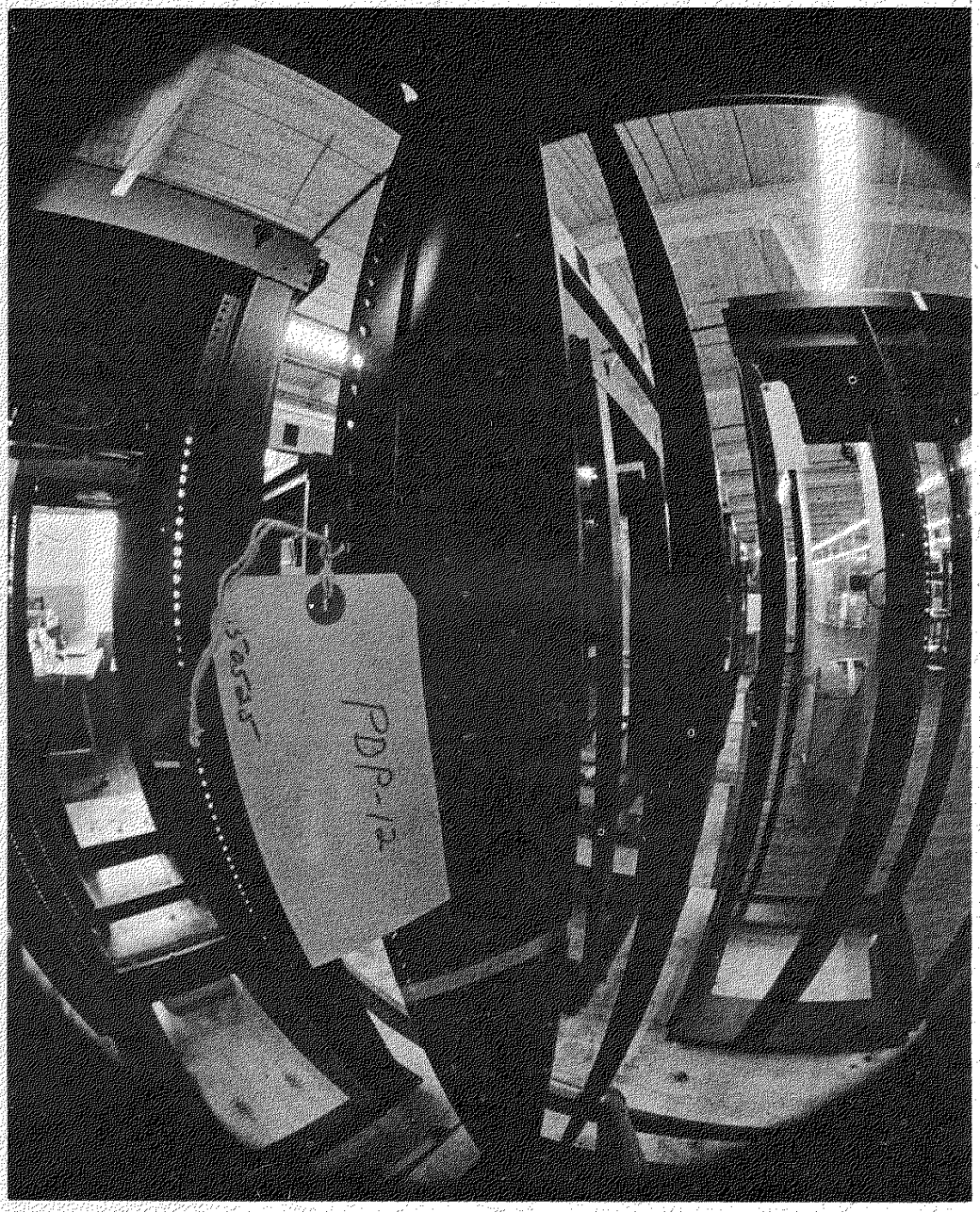
Keyboard/Printer (ASR-33) — An important means of direct communication between the user and the operating program is the ASR-33 Keyboard/Printer, standard on all configurations of the PDP-12. The ASR-33 is connected to the I/O bus, and can be accessed for input or output by programs in either operating mode. The ASR-33 is equipped with paper tape reader and punch; the reader and keyboard use the same input path and instructions, while the printer and punch use the same output path and instructions. The maximum transfer rate in either direction is 10 characters per second.

The ASR-33 has both full- and half-duplex capability. In full-duplex operation, data may be transmitted in both directions simultaneously, without interference. In half-duplex operation, data may be transmitted in only one direction at a time.

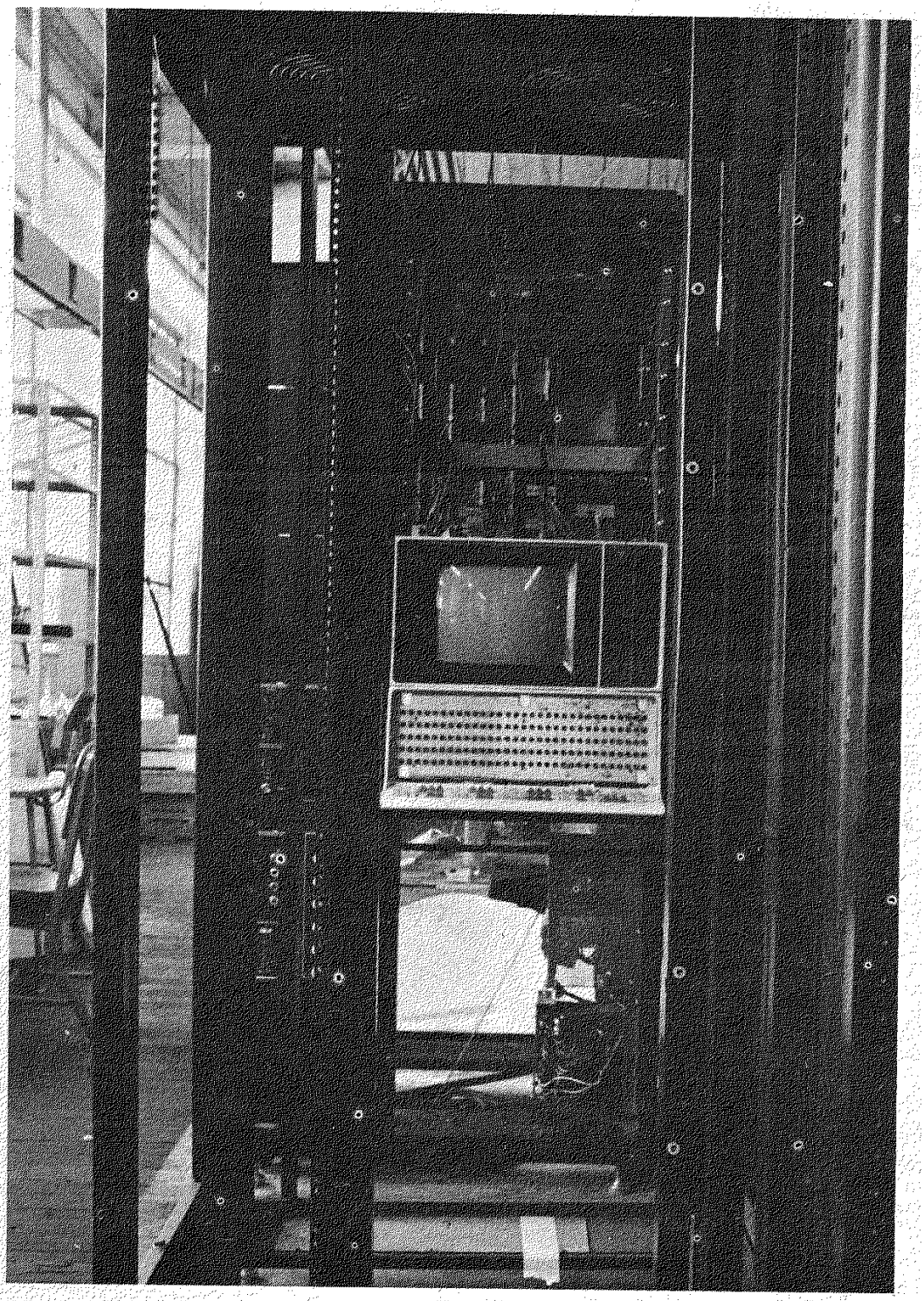


By simplifying program tasks, the PDP-12 frees users from the mechanics of program preparation to concentrate on more creative aspects of their work.

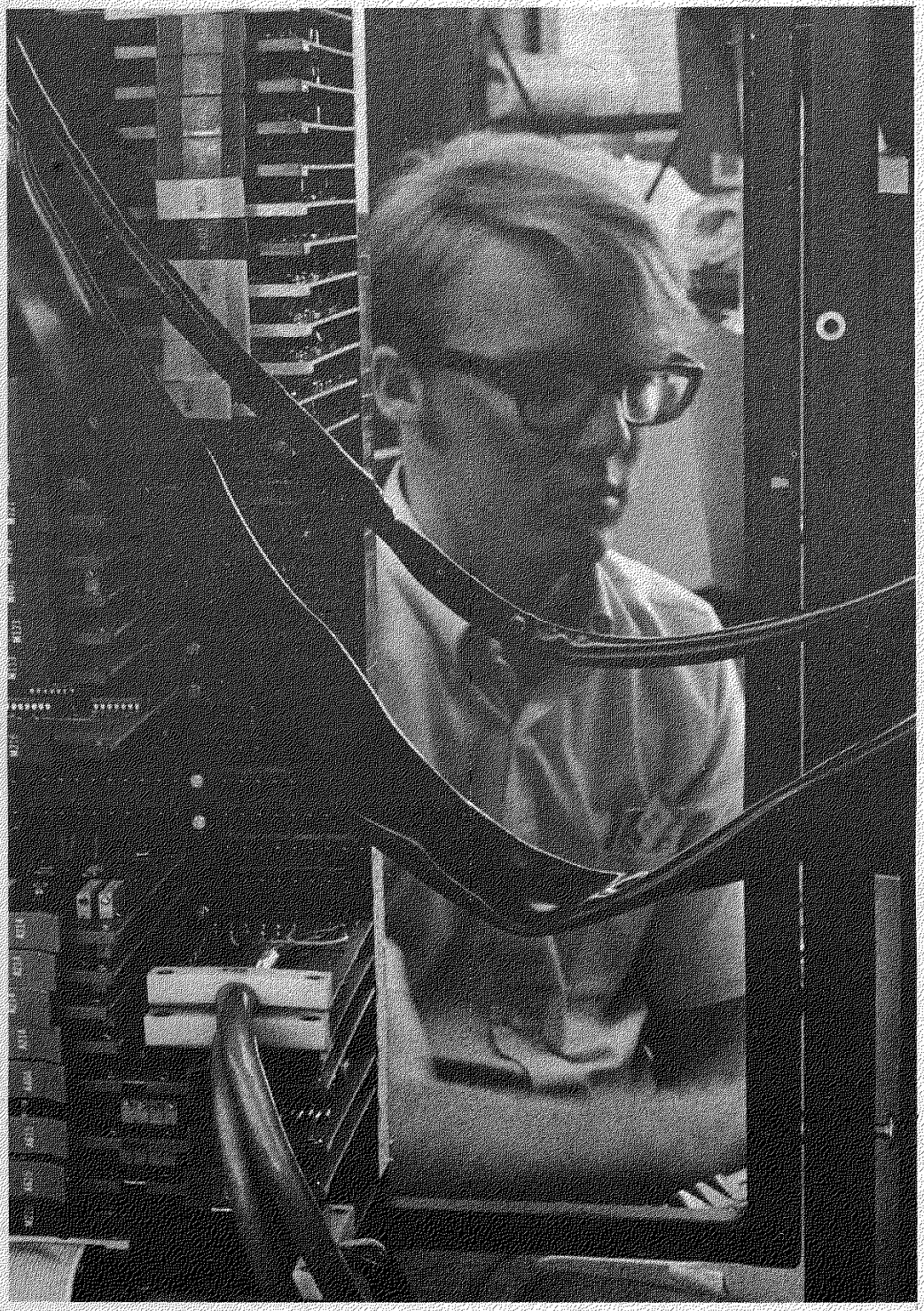
LAB COMPUTER HANDBOOK
TABLE OF CONTENTS



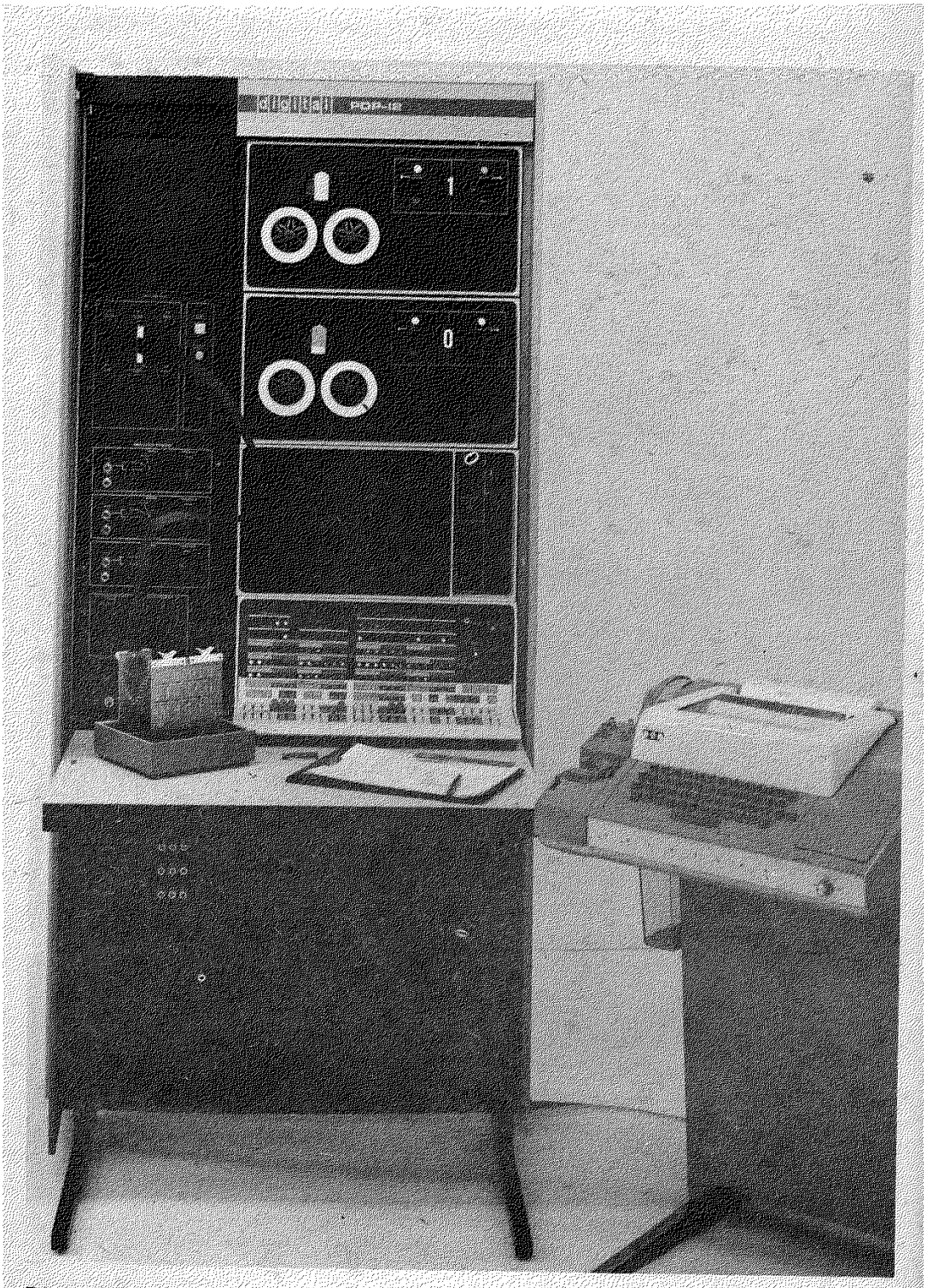
With over 10,000 computers installed worldwide, DEC stands second in the industry by the standard of total installations.



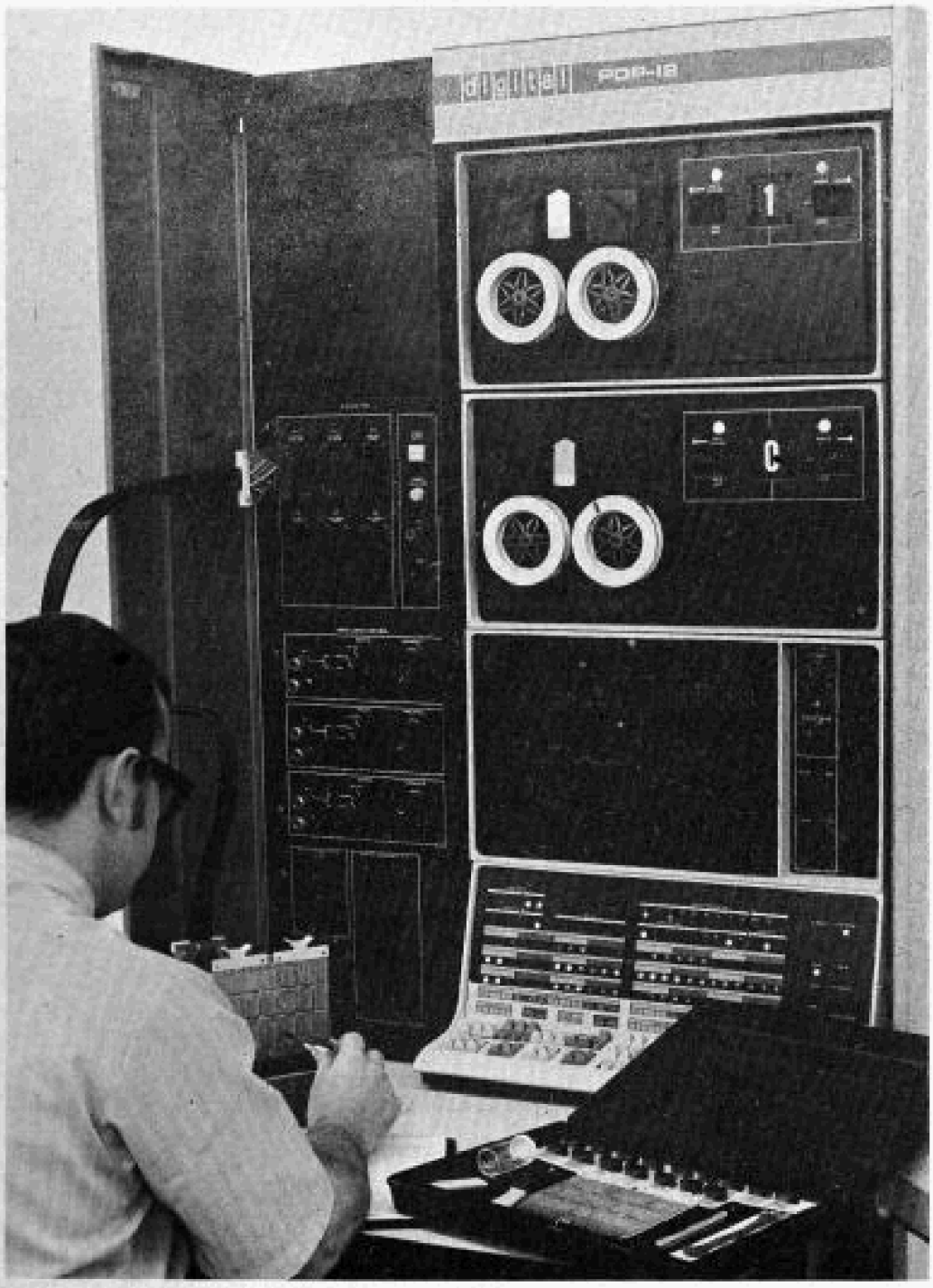
PDP-12 systems are designed with the modular approach in mind. Additional memory capacity, mass storage, and other peripherals may be installed in the field.



Field Service checks each computer before it leaves the factory, installs the system for you to make sure it works properly, and is ready to serve you if your computer ever needs help.



Digital's PDP-12 Computer System is uniquely configured for educating students in computer technology, programming and computer applications.



The PDP-12 is an approachable computer. Student "hands-on" interaction is encouraged by the easy-to-use nature of the system.

The floating-point processor increases the capability of the PDP-12 in performing complicated calculations especially on large blocks of data.

CHAPTER 1
BIO-MEDICAL SYSTEMS

USE OF THE PDP-12 IN ACQUISITION & ANALYSIS OF NEUROPHYSIOLOGICAL DATA

SIGNAL AVERAGING

For many years the study of evoked responses in electrophysiological research has been gaining in importance. The variable nature of evoked responses makes it difficult to obtain representative measurements from any one response; sometimes it is even impossible to determine visually whether the presentation of the stimulus has in any way affected the system. A common way to obtain stable measurements of responses involves averaging of the individual's responses to the identical stimuli. The advantages of averaging include being able to detect characteristic deflections in the waveform of the average response where these deflections are not visually detectable in any single response. Moreover, the average yields more meaningful measurements than those obtained from the individual response.

The PDP-12 offers a complete signal averaging program that enables the user to utilize the PDP-12 in the way he would a hardwired signal averager, and yet retain the flexibility of a general purpose laboratory computer.

Basically, the signal averaging program operates as follows:

The evoked response is sampled for a number of points following each stimulus. Samples taken at the same delay after the stimulus onset, are added, and their sum is stored. The PDP-12 utilizes two words for each of these data points, thereby guaranteeing there will be no overflow even after a possible 16,000 sweeps. After the desired number of sweeps (responses) has been added in this way, the average of the response is given by the computed sum times an appropriate scale factor. Since the coherent signal occurs in each response in the same way, whereas the noise varies randomly about a mean of zero, the coherent signal will tend to reinforce itself while the noise will decrease as more and more of these responses are added. The overall effect of the averaging process is the enhancement of the signal to noise ratio by approximately the square root of the number of sweeps that are averaged.

TIME INTERVAL HISTOGRAMS

The time interval histogram generated and displayed by the PDP-12 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 from single nerve cells by controlled stimuli. In neurological experiments, it provides an estimate of the relative firing rates of a single unit in response to a stimulus. In work done with gross electrodes, where the neuroelectric data is acquired from many cells, time interval histograms can be used to study the distribution of peak latencies from the on-set of the stimulus.

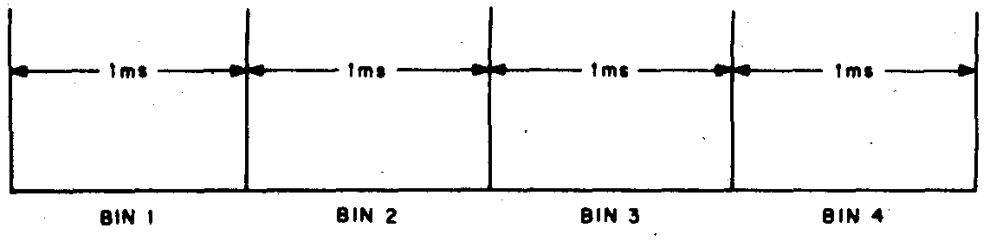
Another example of the value of the time interval histograms is in the study of heart action. From the standpoint of the computer, cardiovascular research has much in common with neurophysiological research. The basic data is often a time-voltage function produced by the system under study. Perturbations and the rhythmic activity of the heart may be detected and subjected to quantitative analysis by forming distributions of inter-beat intervals derived from the ECG (Electrocardiogram).

The Time-Interval Histogram Program

The following program shows you how to use the PDP-12 in the LINC mode of operation to accumulate the frequency distribution of time between events (the occurrence of a pulse) and to continuously display the distribution as an interval histogram.

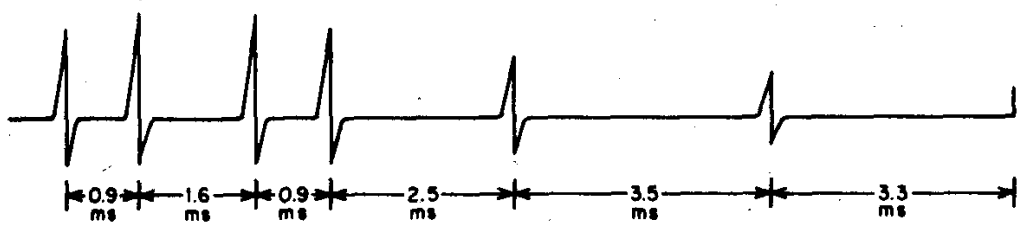
When an acceptable pulse is detected, the PDP-12 records the event by incrementing a location in its memory, called a "bin". Each bin contains the number of samples which occurred during the corresponding "time interval". The PDP-12 uses 512 bins to record the occurrence of up to 512 separate time intervals.

Suppose that we have a histogram with 4 horizontal graduates or bins of 1 millisecond intervals as shown.

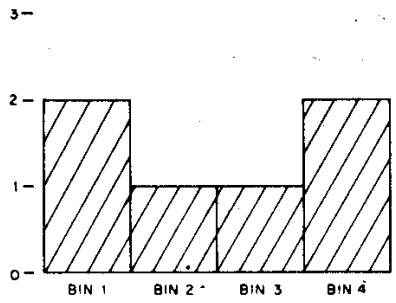


Four 1-Millisecond Interval Bins

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 were monitored over a period of time, a histogram of the pulse train would be as shown.



Pulse Train



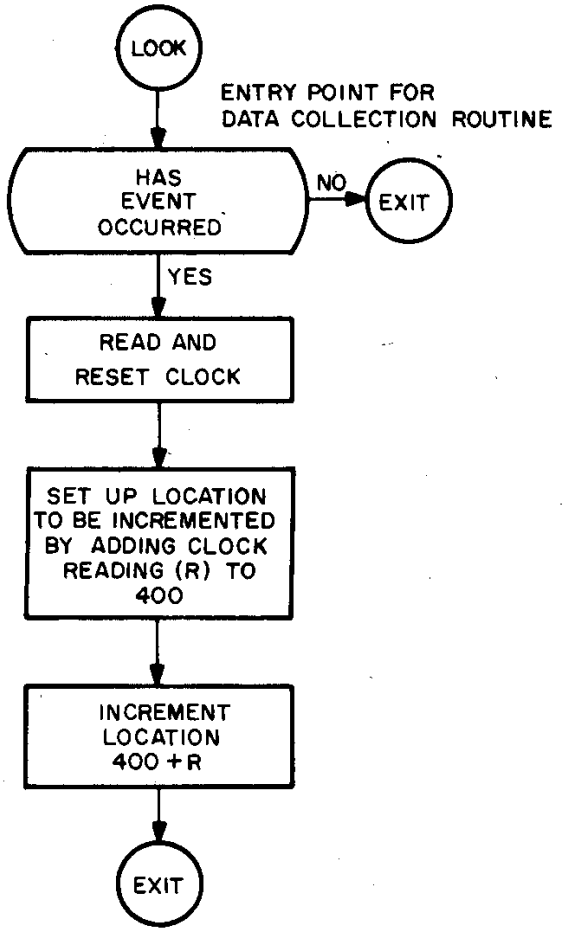
Pulse Train Histogram

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 PDP-12 programming example. We will retain the bin widths of 1 ms for this example.

We will also use the KW12A real-time clock to detect the event and record the time interval. The clock contains a digital counter with input detection circuits. An input event causes the contents of the counter to be saved in a buffer register and sets a flip-flop to "flag" the occurrence of an event. The program tests for the occurrence of the event and reads the digital buffer register into the accumulator and resets the flip-flop and the digital counter to 0. The digital counter is then free to start counting for the next event. Shown below is the clock initialization subroutine.

```
/CLOCK INITIALIZATION SUBROUTINE
INITI,    LDA          /I-O PRESET
          20
          ESF
          PDP          /GO TO -8 MODE
          PMODE
          CLA
          CLAB
          CLLR
          TAD          /
          TAD K0100
          CLLR
          CLA
          TAD K4300    /SET UP 1KC RATE AND RESET TO ZERO MODE
          CLLR
          CLA
          TAD K0003    /SET UP CHANNEL 3 ENABLE AND INTERRUPT
          CLEN
BEGIN,    CLSK         /SKIP ON EVENT FLAG
          JMP BEGIN
          LINC
          LMODE
          JMP GET
K0003,    0003
K4300,    4300
K0100,    0100
```

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 and subroutine listing for the explanation of the data collection subroutine.

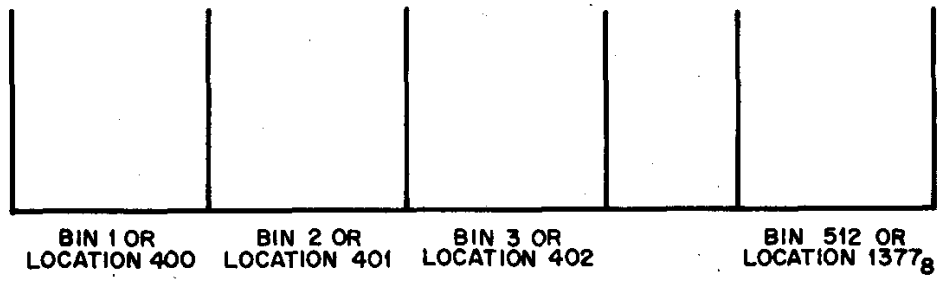


Data Collection Subroutine

```
/DATA COLLECTION SUBROUTINE

LOOK,   IOB
        CLSK        /IS THERE AN EVENT
        JMP 0       /NO, EXIT
        IOB
        CLSA        /CLEAR INPUT EVENT
        IOB
        CLBA        /YES, READ BUFFER
        ADD TEMP    /SET UP ADDRESS FOR INCREMENTING
        STC BTEMP   /TEMPORARILY STORE ADDRESS OF BIN TO
                     BE INCREMENTED
        LDA I       /PLACE 1 IN THE AC
```

```
            1
            ADM         /INCREMENT APPROPRIATE BIN
BTEMP,      0           /ADDRESS OF BIN TO BE INCREMENTED
                         STORED HERE
            JMP 0       /EXIT
TEMP,       400
            CLSA=6135
            CLSK=6131
            CLBA=6136
            CLCA=6137
```

The KW12A Clock is connected to the I/O bus. Thus, the first instruction in the data collection subroutine skips the next instruction if an event occurred; if the event did not occur, the next instruction JMP is executed, causing control to exit from this subroutine. Assuming an event occurs, we go to the IOB /CLSA, IOB /CLDA instructions which read the external clock and reset the flag so that we are prepared to start recording time and to note the next interval. After the execution of the CLDA, the clock reading is held in the accumulator. The next instruction, ADD TEMP, adds 400 to the accumulator which has the clock reading and then stores the results into symbolic locations BTEMP. Let us digress a moment to see the significance of this.



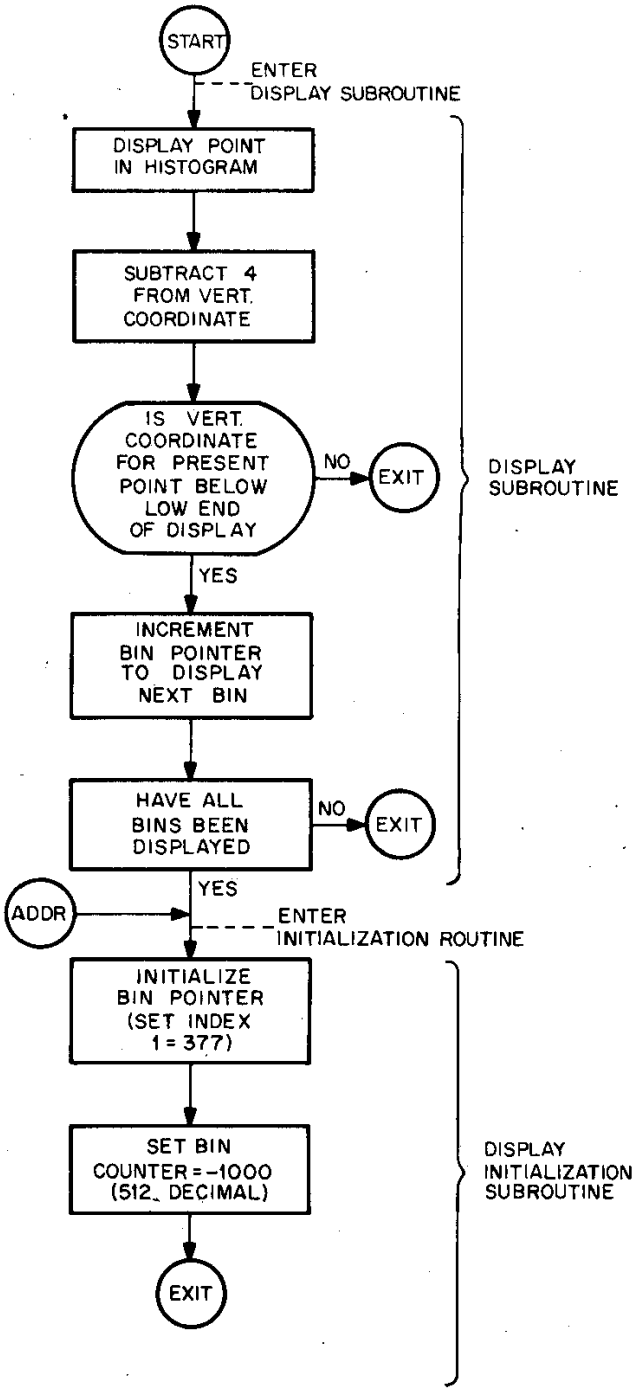
If we let location 400 be the address of bin 1, and 401 bin 2, etc., then by adding 400 to the clock reading and storing the results into symbolic location BTEMP, location BTEMP contains the address of the appropriate bin.

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 BTEMP 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 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 3 is set to 0 so that the histogram display starts at the left side of the CRT at a horizontal coordinate of 000.

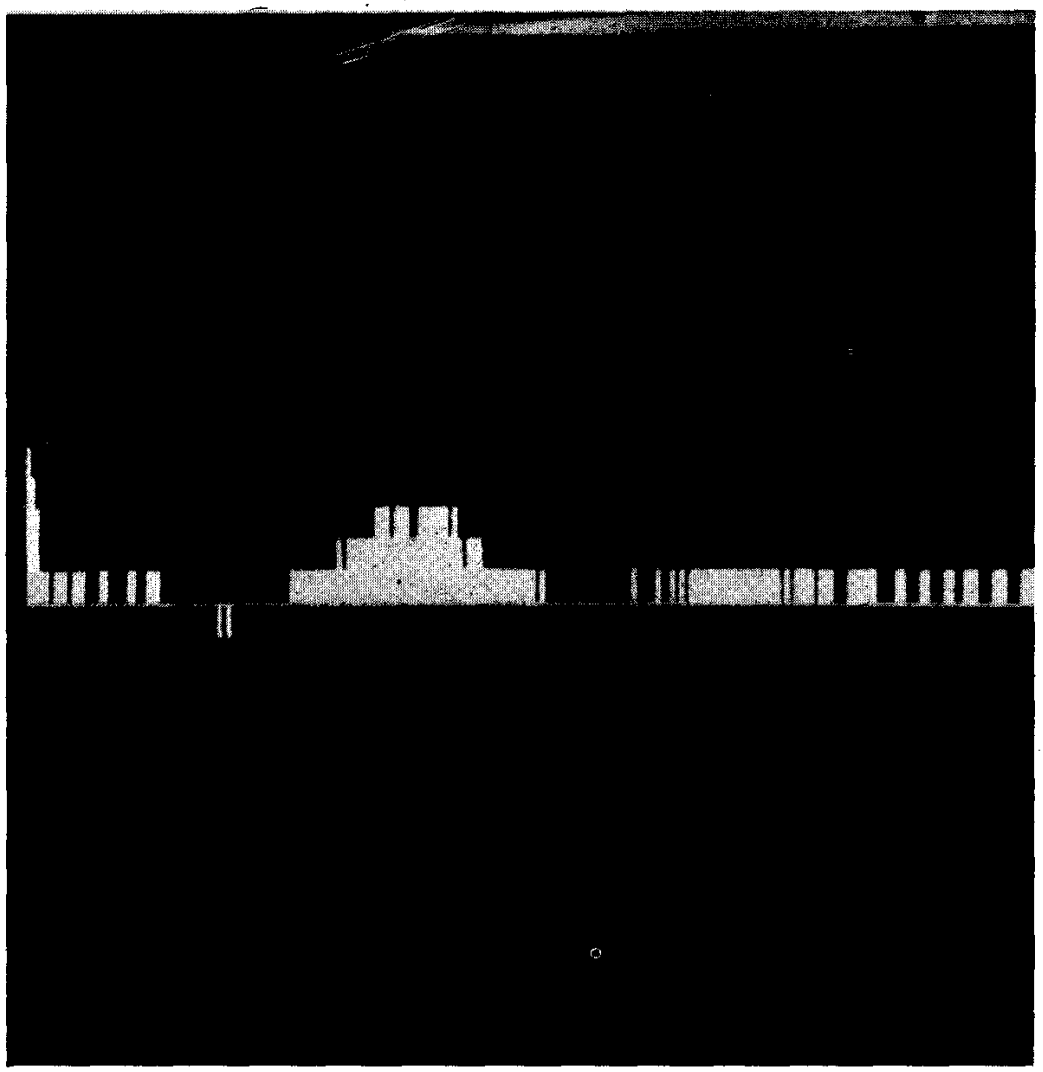


Display and Display Initialization Subroutines

```
/DISPLAY SUBROUTINE

START,    LDA          /VERT, COUNT TO THE AC
          VERT
          DIS 3        /PLOT POINT
          ADA I        /SUBTRACT 4 FROM THE AC


          -4
          STA          /STORE VERYIV/CAL COORDINATE FOR NEXT
                        DISPLAY
          VERT
          ADA I        /AC WILL BE NEG, IF NEXT DISPLAY IS BELOW
                        BOTTOM EDGE OF SCOPE
          377
          APO I        /SKIP IF ABOVE CASE IS TRUE
          JMP 0        /NO, EXIT
          LDA I 1      /LOAD COUNT OF VERTICAL COORD, FOR NEXT
                        BIN
          ADA I        /BIN COUNTS DISPLAYED FROM BOTTOM OF
                        SCOPE
          -377
          STA I        /AND STORE IN VERT
VERT,     0
          CLR          /CLEAR THE AC
          XSK I 3      /MOVE HORIZ. COORD.
          XSK I 2      /HAVE ALL BINS BEEN DISPLAYED?
          JMP 0        /NO, EXIT
/DISPLAY INITIALIZATION ROUTINE
ADDR,     SET I 1      /SET INITIAL TABLE ADDRESS
TABLE,    400          /DATA BEGINS IN 400
          SET I 2      /SET COUNTER FOR 512 BINS
          -1000
          SET I 3      /SET HORIZ. COORD.
          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 START, 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

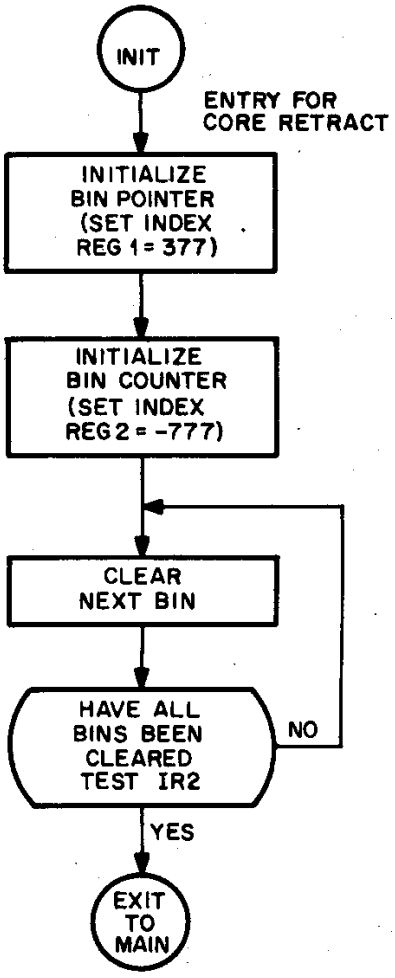
XSK i 3 which increments location 3 (the horizontal coordinate). To test for the end of the histogram the routine executes an XSK i 2 which adds 1 to location 2 skips the next instruction if bits 2 thru 11 of location 2 equal 1777_8. If all bins have not been displayed, we will not skip; therefore, we 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 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 the 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). 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 0 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 0 down so that we loop through the "accumulate data" subroutine.

CORE INITIALIZATION SUBROUTINE

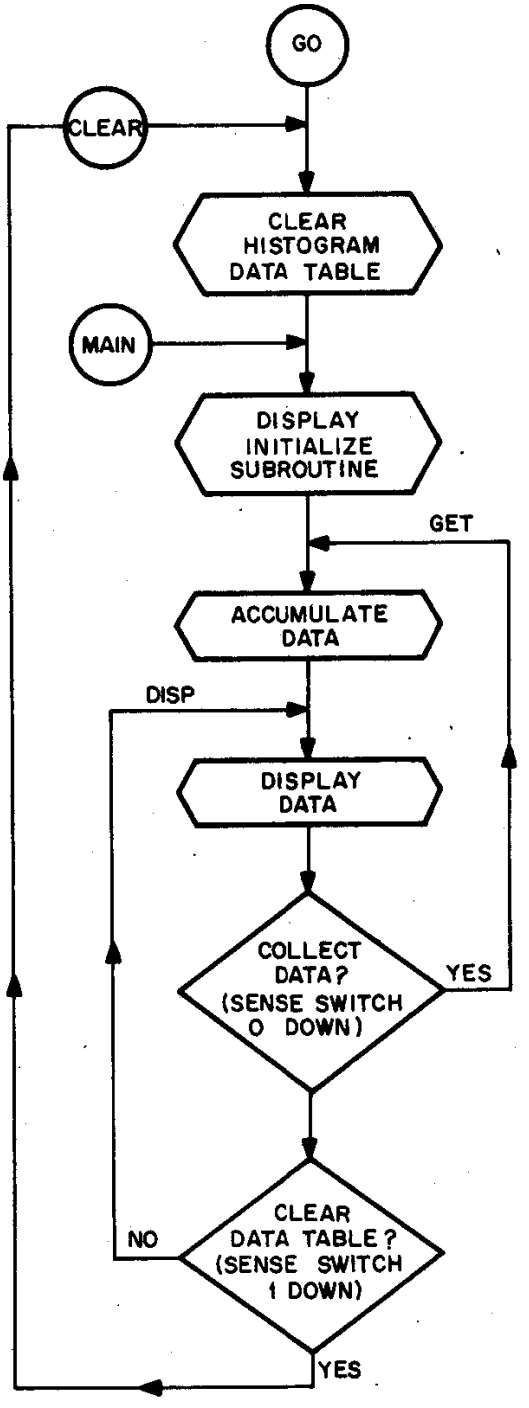
```
INIT,   SET I 1      /INITIALIZE POINTER TO FIRST LOCATION
        377           OF HISTOGRAM
        SET I 2      /SET NUMBER OF BINS
        -1000        /1000(8) = 512(10)
        CLR          /CLEAR ACCUMULATOR
LOOP,   STA I 1      /CLEAR NEXT BIN
        XSK I 2      /ADD 1 TO LOCATION 2; SKIP WHEN CON-
                      TENTS OF LOCATION 2 = 1777(8) (BITS 2
                      THROUGH 11)
        JMP LOOP     /JUMP BACKWARD TWO LOCATIONS (CON-
                      TINUE CLEARING)
        JMP MAIN     /RETURN TO MAIN PROGRAM TO SETUP
                      DISPLAY COUNTERS AND POINTERS
```



Core Initialization Subroutine

MAIN PROGRAM — TIME INTERVAL HISTOGRAM

```
CLEAR,   JMP INIT      /CLEAR TABLES
MAIN,    JMP ADDR      /SET UP COUNTERS AND POINTERS
GET,     JMP LOOK      /GET DATA
DISP,    JMP START     /DISPLAY ONE DATA POINT
         SNS 0         /SKIP IF SENSE SWITCH 0 IS UP
         JMP GET       /GET NEXT POINT
         SIS 1         /SKIP IF SENSE SWITCH 1 IS UP
         JMP DISP      /STATIC DISPLAY SELECTED
         JMP CLEAR     /0 AND 1 UP; RESTART
```



Main Program

The PDP-12 signal averager program digitizes, displays, and averages up to 4 channels of analog signals at rates selectable from 55 to 4095 microseconds per point per channel. The program is entirely core resident, and the user can make on-line selections and adjustments in the sampling rate, the number of sweeps, and the post-stimulus delay via PDP-12 keyboard and display scope. During the averaging operation, either the raw in-

put data or the averaged data is available for viewing on the CRT display, and the user at any time can switch between each of these waveforms by typing on the keyboard. To facilitate easier viewing, the averaged data can be contracted on the screen through the use of a keyboard command. The averaging process may be paused at any time during the operation and may be resumed thereafter.

When the averaging process has been completed (immediately after the total number of sweeps has been taken, up to 4095) it may be plotted out on an XY recorder that has been connected to the extension scope connector on the data terminal panel.

Also, upon completion of the average the data may be typed out on the Teletype. One entire channel or selective portions of a channel may be typed out through the use of appropriate keyboard command. If the data is scaled by a factor equal to the N where N is the number of sweeps, the typeout of data points will then be in millivolts calibrated as seen at the analog input. This is due to the fact that 2 raised to the scale factor power would be equal to the number of sweeps. Additional sets of sweeps can be added to the average already accumulated after a group of sweeps has been completed.

The data resident in core, after completion of an average, may be stored on LINC tape for future use and further analyses.

SIGNAL EDITING AND FREQUENCY ANALYSIS

One of the major problems in demonstrating certain predominant features in physiological signals is that "text book" examples are quite infrequent. Data must be laboriously searched for good examples of such phenomena. As an example of how the PDP-12 can aid in such a problem, let us take the case of study of electrosleep. Raw data can be collected by the program ADTAPE. After collection, the data can be visually inspected (by another program) for the desired features.

ADTAPE permits up to 16 A/D channels to be sampled consecutively. One or two of the channels can be displayed on the CRT display at any time during sampling simply by typing the number of the channel on the Teletype. Sampling rates up to 1KC and a maximum time per point of up to 40 seconds are acceptable. The signal to begin or end sampling can be given by means of a sense switch, external level, or clock channel. It is also possible to begin sampling after a pre-determined delay from the synchronizing signal and to terminate the sampling after a requested number of points have been collected (up to a maximum of 10,000). SAVE or NON-SAVE modes provide the option of storing the data on LINC tape. The entire setup procedure for this program utilizes interaction between the experimentor and the computer via the CRT display and the keyboard in a question and answer format.

At the conclusion of using the A/D Tape program, the EEG data we are concerned with will have been stored away starting at a specified tape block. At this time the program MAGSPY can be called down from the system's tape and used to visually scan the EEG data just acquired.

The MAGSPY program provides a "moving window" of the information that has been stored on any of the LINC tape units. A starting block can

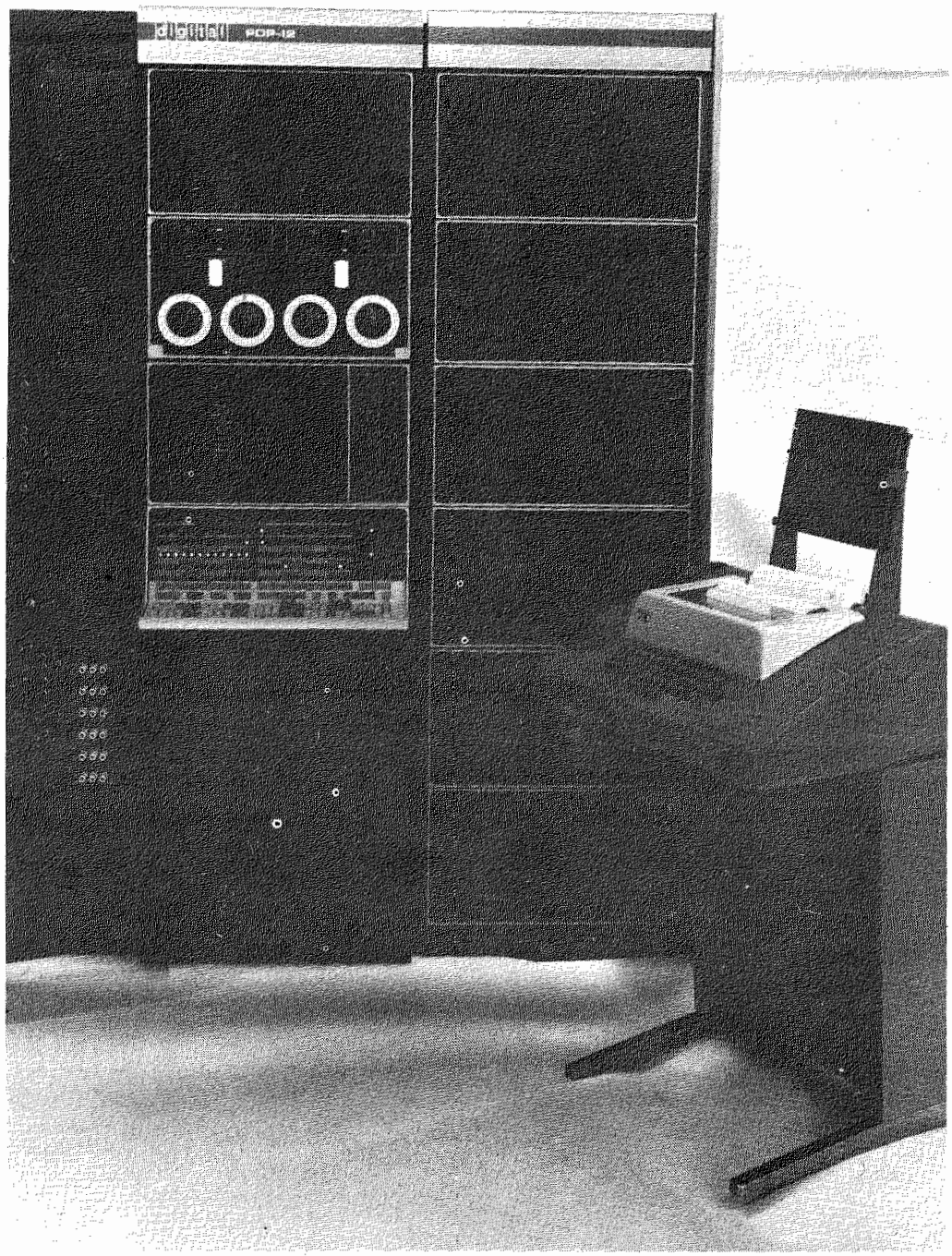
be specified and the data in that block will be displayed on the CRT and can be scanned in the forward or backward direction under control of a potentiometer on the data terminal panel.

The portions of the data containing the features of interest can be noted by block # and then be referred to at a later time for further analysis.

At this point, the user returns to the DIAL monitor and calls in the frequency analysis program he wishes to use. An example of one of these programs is FRQANA. FRQANA is a frequency analysis routine and performs both the fourier and inverse-fourier transform on 512 data points. It calculates the co-efficients of the sines and cosines for the real and imaginary components as well as the co-efficients of the power spectrum. These can be displayed on the CRT utilizing simple Teletype commands.

Having performed the frequency analysis of the information of interest, the data can be either photographed or plotted on an XY analog or incremental plotter. Comparisons may then be made and records established. The resolution of the frequency analysis is approximately the reciprocal of the time interval over which the data was sampled. For example, if the EEG was sampled at 256 points per second, a 512 sample record would have taken two seconds to acquire and therefore the resolution would be approximately one-half Hertz.

The use of the new signal processing software presently available on the PDP-12 will enhance the user's ability to do still more handling. The reader is urged to scan Chapter 8 and to familiarize himself with the software and how it can be applied to his tasks.



The PDP-12 is the world's best known laboratory computer system, it is ideally suited to direct interfacing of analytical instruments and general signal processing tasks.

CHAPTER 2
CHEMISTRY SYSTEMS

INTRODUCTION

The PDP-12 is a flexible laboratory instrument computer. It is ideally suited for interfacing to analytical instrumentation. The functions usually performed by adding various peripherals to a digital processor are built into the basic PDP-12. Analog input and output, digital input and output, and a scope-based operating system make this the most approachable computer for the scientist. Large libraries of programs and spectra are easily manipulated with the unique LINCtape* system.

The PDP-12/LDP (Laboratory Data Processor) is unrivaled in the research laboratory for qualitative and quantitative analysis, simulations, reporting, calculations, and automating instruments. The benefits accrued are analysis speed, accuracy, instrument performance enhancement, and record keeping. Interactively massaging data and adjusting parameters result in more profitable research techniques. The cost of the in-laboratory computer-based system is no more than a typical instrument's cost, while it provides immediate benefits.

The outputs of instruments, from the computer's point of view, are generally very much alike, differing in only superficial ways. Virtually every instrument creates some form of line spectrum where $y = f(x)$. The only real difference between a mass spectrometer and IR spectrophotometer is data rate. Consequently, one can design an instrument computer to be completely general.

However, the data rate consideration divides instruments into two broad categories: high and low speed. For example, a gas chromatograph/mass spectrometer or a pulsed nuclear magnetic resonance spectrometer instrument would require that the computer be dedicated when running. DIGITAL's computer-based, analytical instrument packages are designed to handle these basic application areas.

This comprehensive, computer-based system will handle the widest variety of laboratory instruments, without special interfacing. Typically, interfacing will entail only signal conditioning units. These pre-engineered kits allow the most straightforward connections and optimum system performance.

PDP-12 systems are designed with the modular approach in mind. Adding capacity and peripherals may be accomplished in the field. Software systems are designed to incorporate all DEC peripherals.

The PDP-12/LDP system will handle eight slow instruments; e.g., 4 G.C., 2 I.R., 2 NMR, simultaneously and independently. Data is stored, in raw form, directly on LINCtape for subsequent reduction and interactive analyses. Alternatively, the system may be sequentially dedicated to a very fast instrument such as a G.C.-mass spec. In this case, because of

*Addressable magnetic tape, each spool having a 128,000 or 229,000 word capacity costing $15 or less.

the large quantities of generated data, some on-line data reduction takes place before tape storage. Also, because of the routine, yet complex series of operations, these programs have been written by DEC to handle the complete analysis job automatically. So, the PDP-12/LDP system may be used as a roll around laboratory instrument computers to complement every instrument in the laboratory. Or, several machines may be permanently connected and used sequentially.

HARDWARE DESCRIPTION

General Configuration

PDP-12A LINC Computer-Based System includes:

central processor
4K words core memory
409,600 words tape storage
17.8 x 22.9 cm CRT Display
up to 24 analog inputs — each has differential preamplifier
solid state multiplexer
2 analog outputs
12 digital inputs
6 relay outputs

MC-12 Additional 4K Words Core Memory

KW12A Real Time Clock includes:

400 kHz crystal
3 Schmidt trigger inputs

AIP-12 Analytical Instrument Package includes:

Choice of A/D converters
solid state multiplexer for 16 channels, differential inputs
amplifier per channel, selection of many ranges
6 digital instrument inputs 8 digits each)
BCD to binary converter
4 sample and hold circuits
direct memory access
up to 50 kHz data rate
data break multiplexer
100 nanosecond aperture time
50 megohm input impedance

FPP-12 Floating Point Processor includes:

extended addressing capability
speed increases of 10 to 15 times
simultaneity of calculation and data acquisition

This computer-based laboratory system is extremely powerful because it is built with four separate processors: CPU, floating point processor, A/D processor, and tape processor. This allows for many levels of simultaneity, or overlapped I/O. For example, system software accepts data, smooths it, stores it on tape, and displays the real time data on the scope simultaneously. This is not possible in a system without individual mini-processors.

AIP-12 — Analytical Instrument Package

The AIP-12 is the prime component of the PDP-12/LDP system. It is a versatile instrument input device coupled directly to the PDP-12's core memory. It also acts as the master unit for the FPP-12 in the system, precluding the requirement for an external, separate multiplexer.

Choice of analog signal resolution is either 12 bits (1 part in 4096) sufficient for NMR or spectrophotometers or 15 bits (1 part in 32764) sufficient for mass spectrometers or gas chromatographs. Since each of the 16 channels are available with an individual, differential amplifier, the connected instruments may have very different characteristics from one another.

The PDP-12/LDP has four separate sample-and-hold circuits, with 100 nano-seconds aperture time. It provides hard-wiring instrument priority through the connectors supplied. Fast instruments may be sampled at rates up to 50 kHz.

Input impedance is 50 megohms with 1000 to 1 common mode rejection over the ±8 volt input range. Standard differential input range is ±2 volts. Overloads of ±35 volts are tolerated. Higher overloads are fused.

Plug-in receptacles are available for up to six separate digital instruments each having eight digits. A BCD to binary converter is built in so that all instrument inputs, analog or digital, look identical to the software, providing extreme versatility in data handling and manipulation. Levels on these inputs are standard TTL (0 and 3 volts).

AIP-12 Instruction Set

6301	SCH	A channel select word is transferred from the accumulator to registers in the AIP. The accumulator is cleared. If the channel ID code from the right half of the accumulator is 01 or 10-17 (octal) the bits are decoded to direct the data from a subsequent LCH instruction to the proper register. If the ID code is 20-57 the S,B and E bits from AC00, 01 and 02 are executed. The S bit, when set, directs data from the addressed channel into an auxiliary core storage area, rather than the main buffer. The E bit, when set, enables external synchronization pulses to trigger sampling of the addressed channel.
6302	LCH	Data from the accumulator is transferred to the working register addressed by the previous SCH or RCZ instruction. The accumulator is cleared. If the previously addressed channel was other than 01 or 10-17 no transfer occurs.

6305	RCZ	The number Z, contained in the right half of the accumulator before executing this instruction, is transferred to the channel select register in the AIP. The contents of the register in AIP addressed thereby is echoed back into the accumulator. Channels consisting of two words will echo back their second word if AC00=1. If channel numbers greater than 37 are addressed by this instruction, zeroes will be echoed back. Note that when channels 10 or 12 are addressed by this instruction, the true current address is read back, rather than the starting addresses of the core data buffers as loaded by IOT instructions 6301 and 6302.
6306	SEF	The next instruction is skipped when either the nonexistent channel or lost data flags are set and the status of the flags is placed placed in AC01 and AC00 respectively. If neither flag is set, the accumulator is cleared.
6307	SBF	The next instruction is skipped when any of the four buffer flags are set and the status of these flags is placed in AC00-AC03. If no flags are set, the accumulator is cleared. These flags indicate when segment boundaries are crossed in the main and auxiliary core data buffers and when the end of these buffers is reached. Each buffer returns to its starting address automatically upon reaching its end.

List of Channel ID Codes (Octal)

00	Output buffer of 13 or 15 bit ADC.
01	Q register (for comments or labels to data buffer).
02	Output of digital instrument multiplexer.
10	Main buffer current address.
11	Main buffer word count.
12	Auxiliary buffer current address.
13	Auxiliary buffer word count.
14	Control word (data field, interrupt enable, etc.)
15-17	Up to three optional output registers for programmable digital instruments.

40, 44, 50, 54: Analog channels assigned to first sample/hold amplifier,
41, 45, 51, 55: Analog channels assigned to second sample/hold amplifier,
42, 46, 52, 56: Analog channels assigned to third sample/hold amplifier,
43, 47, 53, 57: Analog channels assigned to fourth sample/hold amplifier.

AIP-12 Differential Preamplifiers.

Input voltage range (standard):	± 2 volts
Common mode input range:	± 8 volts
Common mode rejection ratio (see note 1):	60 dB min
Gain stability:	± 0.02% per month
Gain temperature coefficient:	±0.01% per °C max.
Zero stability:	± 200 uV per month
Zero temperature coefficient:	± 50 uV per °C max.
Noise (peak-to-peak, 99.9% confidence):	100 uV typical
Slow rate:	0.1V/usec typical
Bandwidth (3 lb down, small signal):	100 kH_z min.
Input impedance:	500 Megohms min.
Input current:	500 nA max.

NOTE

This applies for frequencies up to 100 H_z, source resistance unbalance up to 500 ohms, and a source resistance to ground of less than 1 megohm.

AIP Interface Connections.

Analog input connections to AIP are made by means of BNC connectors for each channel. Digital connections, both input and output, are made with 42 pin Burndy MS series connectors. One of the BNC connectors for each analog channel is for external synchronization of sampling. All of these connectors are mounted on a 10½" high rack panel which can be in the front or rear of the equipment cabinet.

FPP — FLOATING POINT PROCESSOR

The floating-point processor increases the capability of the PDP-12 in performing complicated calculations especially on large blocks of data. The FPP12 performs floating point calculations more than two orders of magnitude faster than a softwave floating point package. Double precision calculations are performed as a subset of floating-point arithmetic. The FPP12 addressing methods permit direct and indirect addressing of 32K of memory without fixed page or field boundaries. The hardware allows indexing over 4096 floating point or double precision numbers located sequentially beginning at any point in memory.

The floating-point processor attaches to any DEC12 bit computer that has a direct access to memory or data break facility. Similar to a disk, the FPP12 is activated via PDP-8 mode IOT's. Once activated, the FPP12 receives instructions and stores results in core via the data break facility, behaving very much as a parallel CPU. While operating, the FPP12 "steals" an average of 50 percent of the available memory cycles.

Floating Point Number System

The term, floating point, implies a moveable binary point in a similar manner to the moveable decimal point in scientific notation. An exponent is used to keep track of the number of spaces the binary or decimal point is moved.

Examples of scientific notation:

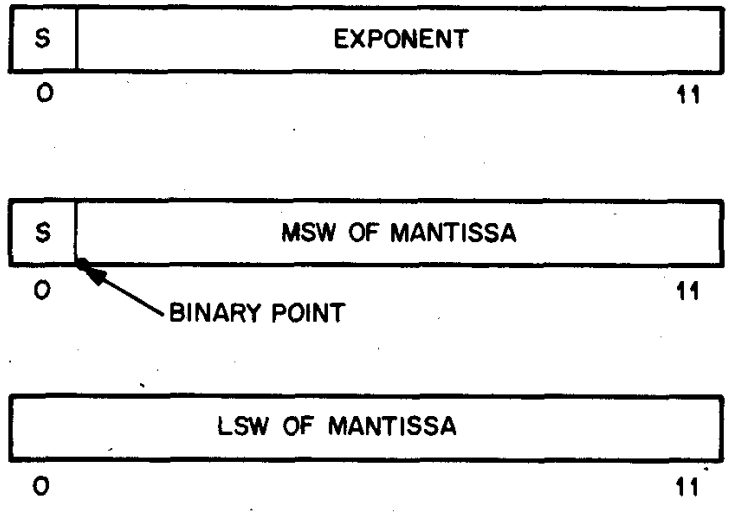
$$234 = 23.4 \times 10^1 = 2.34 \times 10^2$$

Examples of binary floating-point notation:

$$(1011) = (101.1) \times 2^1 = (10.11) \times 2^2 = (1.011) \times 2^3$$
$$(1.011) \times 2^3 = 0.1011 \times 2^4 = 0.01011 \times 2^5$$

Note that in all cases of binary floating-point notation given above, there are four significant bits. However, in the last example the mantissa which multiplies the exponent contains six bits. Given a fixed number of bits, it is desirable to adjust the exponent and the binary point to eliminate leading zeroes to retain the maximum significance for a given format length. The FPP12 normalizes or removes leading zeroes as the last step in every floating-point arithmetic operation.

The floating-point data format used by the FPP12 is identical to the format used by the PDP-8 floating point system (DEC-08-YQYB-D). As shown below, there is a 12-bit signed 2's complement exponent and a 24-bit signed 2's complement mantissa.



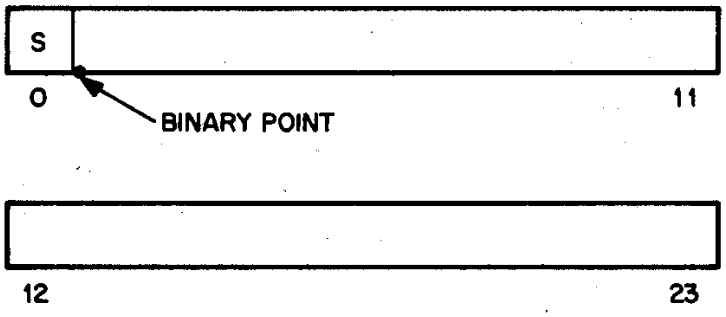
The FPP12 carries all calculations to 28 bits of precision then rounds to 24 bits after normalization. After rounding, the results are rechecked for proper normalization prior to completion of the instruction.

In fixed point arithmetic, a calculation which results in a number whose magnitude cannot be expressed in 12 or 24 bits is an error. With the FPP12, the number range is 2^{+2047} to 2^{-2048}. Exceeding the upper limit, 2^{+2047}, causes the FPP12 to interrupt the PDP-12 CPU and set its exponent overflow status bits. A calculation resulting in an exponent smaller than 2^{-2048} is an exponent underflow which normally causes a program interrupt. The programmer has the option at initialization to request that the underflow trap be ignored. In which case, the result of a calculation in which underflow occurred is set to 0.

Double Precision

For those calculations where full 24-bit precision isn't necessary and where core space is of a premium, the FPP12 is used in fixed point double

precision mode. Each operand consists of a 24-bit signed 2's complement fraction as shown below. As with the floating-point mode, each calculation is carried to 28 bits of precision and rounded to 24 bits. In this instance, normalization isn't performed allowing the occurrence of leading zeroes which reduces the precision of subsequent calculations. The largest numbers that may be represented in double precision format are $+ 2^{23} - 1$ and 2^{23}. Calculations producing numbers that exceed this range cause the floating point processor to initiate a program interrupt with the fraction overflow status bit set to a one.



Operation

The FPP12 is initialized and interrogated as to its status through PDP-8 mode IOT's. Once initialized, the FPP12 operates much like a central processor fetching instructions and operands and storing results in memory. Data breaks are generally requested as needed. However, the usual number of breaks requested by the FPP12 is two per instruction performed by the processor. This means that while the FPP12 is "stealing cycles,"programs can be run simultaneously at slightly reduced speed. Typically LINCtape, display, and other forms of I/O can be performed by the PDP-12 while the floating-point processor is performing calculations.

Active Parameter Table Format

Location

P	Field Bits of Operand Address	Field Bits of Base Reg.	Field Bits of Index Register Location	Field Bits of FPC
P+1	Lower 12 bits of FPC			
P+2	Lower 12 bits			
P+3	Lower 12 bits of Base Reg			
P+4	Lower 12 bits of operand address			
P+5	Exponent of FAC			
P+6	MSW of FAC			
P+7	LSW of FAC			
NOTE: APT address points to location P.				

TABLE I

It should be noted that once initialized the FPP12 will execute programmed instructions until

1. an error condition occurs,
2. an exit instruction is reached,
3. an exit IOT is issued,
4. an I/O preset is issued by the PDP-CPU*,
5. the PDP-CPU encounters any type of halt.

Initialization

In order to execute the first instruction of any program the FPP12 must have the following information:

1. The address of the first instruction (FPC)
2. The initial contents of the floating AC (FAC)
3. The core address of index register 0. (Index registers 1 through 8 are stored in the next 7 sequential 12 bit words.) (X0)
4. The base register which contains the core address of the first location in the data block. (The data block consists of 128 thirty-six bit words.)

*This operation while the FPP-12 is running might necessitate a program reload.

To simplify initialization, the four parameters listed above are placed in core in an active parameter table (APT), shown in TABLE 1, by the CPU. Two initializing IOT's are then issued to the FPP12. FPCOM (6553) loads a command register and the most significant 3 bits of the APT pointer. FPST (6555) loads the remaining 12 bits of the APT pointer and starts the floating-point processor. Whenever the floating-point processor performs an exit, the current values of the FPC, FAC, XO, base reg., and operand address are deposited in the APT to be used either for restarting the FPP12 or for debugging.

A complete list of IOT's, the command register, and the status register is shown in the TABLES 2, 3, and 4.

IOT List

FPINT	6551	Skip on FPP "interrupt request" flag.
FPHLT	6554	Halt the processor at the end of the current instruction. Store active registers in core, set a status register bit, and the "interrupt request" flag.
FPCOM	6553	If the FPP is not running and the FPP "interrupt request flag" has been reset, set the command register to the contents of the AC. The three least significant bits of the AC set the field bits of the "Active Parameter Table" address. If the FPP is running or the interrupt request flag is set, the instruction is ignored.
FPICL	6552	Clear the FPP "interrupt request" flag.
FPST	6555	If the FPP is not running and the FPP "interrupt request flag" is reset, set the location of the "Active Parameter

		Table" to the contents of the AC, initiate the FPP and skip the next instruction. If the FPP is not running or the FPP "interrupt request flag" has not been reset, the instruction is ignored.
FPRST	6556	Read the FPP status register into the AC.
FPIST	6557	Skip on FPP "interrupt request" flag. If the skip is granted, clear the flag and read the FPP status request into the AC.

TABLE II

CPU AC After Read Status Instruction

AC0	Double Precision Mode
AC1	Instruction Trap
AC2	C.P.U. Force Trap
AC3	Divide by Zero
AC4	Fraction Overflow (double precision mode only)
AC5	Exponent Overflow
AC6	Exponent Underflow
AC7, AC8, AC9, AC10	Unused
AC11	Run

TABLE III

The following data are transferred to the FPP by issuing the FPCOM (load command register instruction 6553):

AC0	Select double precision mode
AC1	Exit of exponent underflow error
AC2	Enable memory protection
AC3	Enable interrupt
AC4	Do not store op address on exits
AC5	Do not store address of index registers on exits
AC6	Do not store address of indirect pointer list on exits
AC7	Do not store FAC of exits
AC8	Unused
AC9, AC10, AC11	Data field of "Active Parameter Table"

TABLE IV

Instruction Set and Detailed Programming Spec

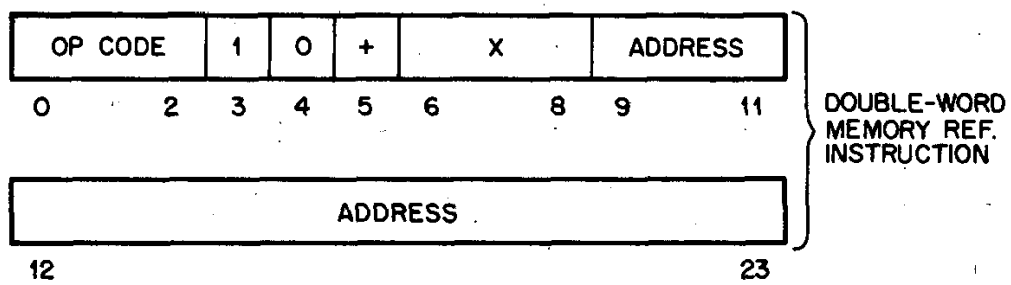
Methods for Memory Reference Instructions

The FPP-12 is capable of three modes of addressing for memory referencing instructions:

1. Double-word direct addressing
2. Single-word direct addressing
3. Single-word indirect addressing

A full indexing capability is available for both methods 1 and 3. The determined address for memory referencing instructions indicates the

exponent in floating-point mode and generally directs to the most significant word in double precision mode. The format for double-word addressing is shown below:



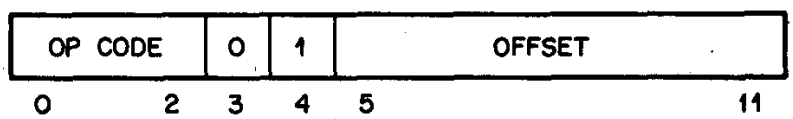
Example 1

If bit 4 is a 0, a double-word instruction is indicated. Setting bit 3 of double-word instruction to a 1 indicates a memory referencing instruction. A non-zero quantity in bits 6-8 causes the address contained in bits 9-23 to be modified by a specified index register. Setting bit 5 to a one causes the specified index register to be incremental prior to use in modification of the address. It should be noted that index register zero can be incremental and tested but is not used for address modification.

Single-Word Addressing Formats

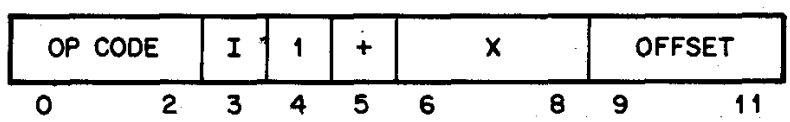
The two single-word address formats utilize a data block that is specified by a base address which is contained in the base register. The data block contains 128 3-word locations. In double-precision mode, the exponent of locations is ignored on the data block.

Single word formats are distinguished by bit 4 being a one. Bit 3 is the indirect indicator in a similar manner to PDP-8 code. The single-word direct address format example shown below the core address is equal to the sum of the 7-bit offset times 3 plus the contents of the base register.



Example 2

If bit 3 is a one, the following indirect format is specified:



Example 3

The effective address for Example 3 is given by the following equation:

$$\text{address} = C\,(\,(\text{offset} * 3) + \text{base address}) + \underbrace{[C\,(X+XO) + \text{bit } 5 * 1)]\,[2 \text{ or } 3]}_{\text{This term} = 0 \text{ if } X = 0}$$

Index Registers

Any core location may be used as an index register. Index register 0 is determined by the 15-bit X0 address. The X0 address is initially set from the active-parameter table, but may be altered by the MVX instruction. Index register X is in core location X0 + X where X = 0, ... 7.

Accessing successive data points in floating-point mode requires incrementing the operand address by $(3)_8$ for each new data point. In double-precision mode, the proper increment is $(2)_8$ for each new data point. To account for the difference between the two modes, the selected index register is multipled by 3 in floating-point mode or 2 in double-precision mode before it is used as an address modifier.

Instruction Set

OP CODE	MNEMONIC	MEMORY REFERENCE INSTRUCTIONS
0	FLDA	Load the FAC from the effective address.
1	FADD	Add the operand to the contents of FAC and store the result in the FAC.
5	FADDM	Add the operand to the contents of the FAC and store the results in the operand.
2	FSUB	Subtract the operand from the contents of the FAC and store the result in the FAC.
3	FDIV	Divide the operand into the contents of the FAC and store the results in the FAC.
4	FMUL	Multiply the contents of the FAC by the operand and store the result in the FAC.
7	FMULM	Multiply the contents of the FAC by the operand and store the results in memory.
6	FSTA	Replace the operand with the contents of the FAC.

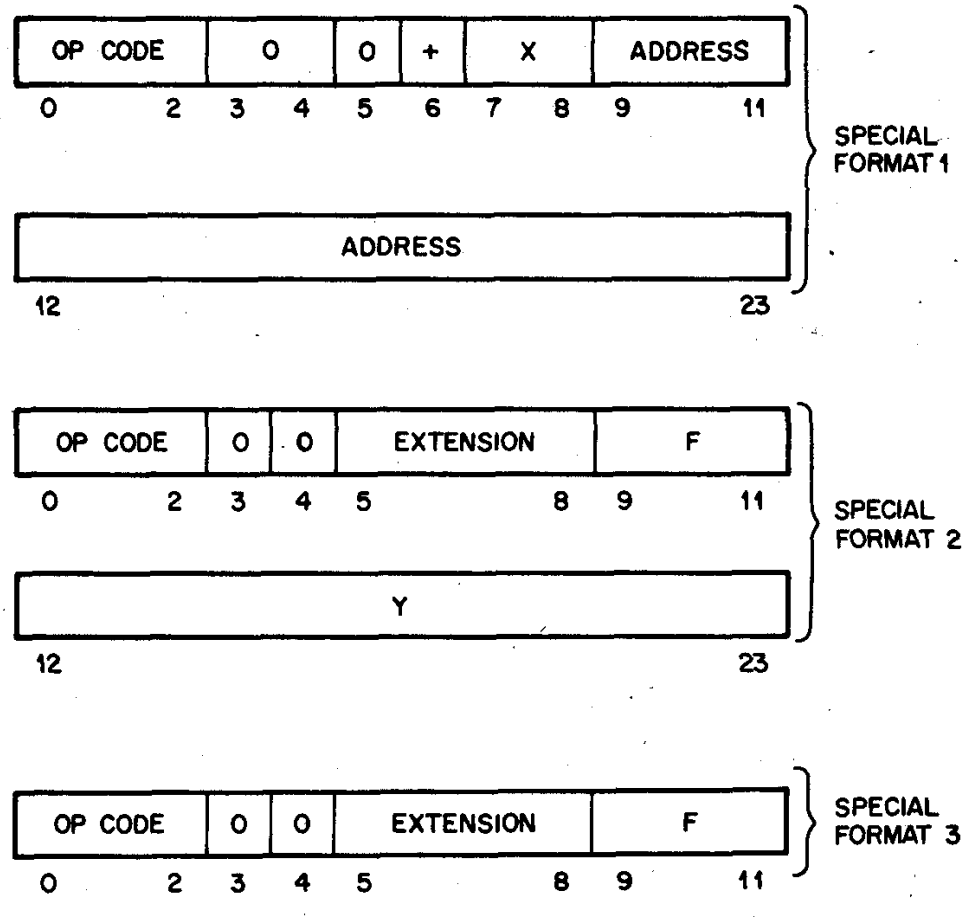
Program Examples

LOCATION	MNEMONIC	OCTAL CODE	
X	FSUB A	2401 5432	/subtract 1 from the /FAC
15432	A,	0001 2000 0000	
X	FSUB B	2205	/subtract 1 from the /FAC
Base Register + 5	B,	0001 2000 0000	
X	FSUB C, 2	2421 5432	/subtract 1 from the /FAC
15432 + 3 (Index Reg 2)	C',	0001 2000 0000	

LOCATION	MNEMONIC	OCTAL CODE	
X	FSUB I D	2603	/subtract 1 from the AC
Base Register + 3	D,	5412	
		0132	
		6724	
26724		0001	
		2000	
		0000	

Special Instructions

The FPP-12 special instructions are similar in nature to the nonmemory referencing instructions for the PDP-8. The set of special instructions includes conditional jumps, two types of subroutine calls, two types of unconditional jumps, several index register operations, a number of accumulator controls, two mode control instructions, and several operational instructions. Altogether, the special group has 26 defined instructions, five trapped instructions, and 14 undefined codes which don't perform any operation. Special instructions which may consist of 1 or 2 12-bit words are denoted by zeros in bits 3 and 4 as shown below:



SPECIAL FORMAT 1

OP CODE	MNEMONIC	
2	JXN	The index register X is incremented if bit 5 = 1 and a jump is executed to the address contained in bits 9-23 if index register X is nonzero.

The JXN instruction is similar to the following sequence of PDP-8 instructions.

```
        ISZ
        JMP  TAG
```

3 4 5 6 7	Trapped Instructions	The "instruction trap" status bit is set and the FPP-12 exits causing a PDP interrupt. The unindexed operand address is dumped into the active parameter table.

SPECIAL FORMAT 2

OP CODE	EXTENSION	MNEMONIC	
0	10	FSTAX	The contents of the index register specified by bits 9-11 are replaced by the contents of bits 12-23.
0	11	ADDX	The contents of bits 12-23 are added to the index register specified by bits 9-11.
0	12-17	NOP	These codes are undefined single-word instructions and perform no operation.

Conditional Jumps — Jumps, if performed, are to the location specified by bits 9-23 of the instruction.

1	0	JEQ	Jump if the FAC = 0
1	1	JGE	Jump if the FAC 0
1	2	JLE	Jump if the FAC $\leqslant$ 0
1	3	JA	Jump always
1	4	JNE	Jump if the FAC $\neq$ 0
1	5	JLT	Jump if the FAC $<$ 0
1	6	JGT	Jump if the FAC $>$ 0
1	7	JAL	Jump if impossible to fix the floating-point number contained in the FAC; i.e., if the exponent is greater than $(23)_{10}$.

POINTER MOVES

1	10	SETX	Set X0 the location of index register zero to the address contained in bits 9-23 of the instruction.
1	11	SETB	Set the base register to the address contained in bits 9-23.

SPECIAL FORMAT 2 — continued

SUBROUTINE CALLS

OP CODE	EXTENSION	MNEMONIC	
1	13	JSR	Jump and save return. The jump is to the location specified in bits 9-23 and the return is saved on the 1st location of the data block.

The JSR is used in writing re-entrant code as the return address is stored in the user's data block. A possible return from a re-entrant subroutine is via the two instruction sequences as follows:

```
        LDA 0       0200    /Load AL with contents
                            /of 1st location of the data
                            /block
        JAC         0007    /Jump to the location
                            /specified by the
                            /least significant 15 bits
                            /of the AC mantissa
                            /JAC is a special
                            /Format 3 instruction
```

1	12	JSA	An unconditional jump is deposited in the address and address + 1 where address is specified by bits 9-23. The FPC is set to address + 2.
1	14-17	NOP	These codes are single-word NOP's.

SPECIAL FORMAT 3 INSTRUCTIONS

0	1	ALN	The mantissa of the FAC is shifted until the FAC exponent equals the contents of the index register specified by bits 9-11. If bits 9-11 are zero, the FAC is aligned such that the exponent = 23_{10}.[1] In fixed-point mode an arithmetic shift is performed on the FAC fraction. The number of shifts is equal to the absolute value of the contents of the specified index register. If the contents of the index register is positive, shifting is towards the least significant bit; otherwise shifting is towards the most significant bit. In fixed-point mode the FAC exponent is not altered.

[1]Setting the exponent = $(23)_{10}$ intergerizes or fixes the floating point number. The JAL instruction tests to see if fixing is possible.

OP CODE	EXTENSION	MNEMONIC	
0	2	FLATX	The FAC is fixed and the least significant 12 bits of the mantissa are loaded into the index register specified by bits 9-11. In fixed-point mode the least significant 12 bits of the FAC is loaded into the specified index register. The FAC is not altered by the FLATX instruction.
0	3	FLDAX	The contents of the index register specified by bits 9-11 are loaded right justified into the FAC mantissa. The FAC exponent is loaded with $(23)_{10}$ and then the FAC is normalized. This operation is typically termed floating a 12-bit number. In fixed-point mode the FAC is not normalized.
0	4-7	NOP	These single-word instructions perform no operation.

OPERATE GROUP

OP CODE	EXTENSION	BITS 9-1	MNEMONIC	
0	0	0	FEXIT	Dump active registers into the active parameter table, reset the FPP-12 run flip flop to the 0 state, and interrupt the PDP processor.
0	0	1	FPAUSE	Wait for external synchronizing signal. This instruction is designed to cooperate with the AIP-12 option.
0	0	2	FCLA	Zero the FAC mantissa and exponent.
0	0	3	FNEG	Form the two's complement of the FAC mantissa.
0	0	4	FNORM	Normalize the FAC. In fixed-point mode FNORM is a NOP.

OP CODE	EXTENSION	BITS 9-1	MNEMONIC	
0	0	5	START F	Start floating-point mode.
0	0	6	START D	Start double-precision mode.
0	0	7	JAC	Jump to the location specified by the least significant 15 bits of the FAC mantissa.

SYSTEM EFFECTS OF AIP AND FPP

Direct Memory Access Port Allocation

PDP-12 systems are built with two direct memory access ports. One is permanently alloted to the LINCtape processor. For interfacing of more than one additional data break device, a memory multiplexer is required (DMO4). However, in this computer-based system, the DMO4 is not required because AIP-12 includes the multiplexer device for itself and FPP-12. Should one desire an additional data break device the DMO4 should be added to the system. Higher priority is placed on data acquisition than on calculation. Therefore, the AIP is given priority over the FPP-12.

Ancillary Hardware Considerations

Because of the great number of peripherals for PDP-8 and PDP-12 computers, the codes used to address devices sometimes conflict. Rarely will a single installation utilize every code. In cases of conflict in equipment purchased from Digital as part of the PDP-12KDP system, the device codes will be modified to alleviate conflicts.

SOFTWARE DESCRIPTION

Overall Philosophy

A computer-based laboratory instrument computer must be a unified hardware/software package in which the operating system ties all the hardware together and provides the user with a convenient, applications-oriented language. Analytical Instrument Package Operating System provides these capabilities.

Monitored and controlled by AIPOS are various categories of programs. All of these programs utilize AIPOS for their input/output requirements, thus unifying the components of the system. Data acquisition, data manipulation, and display-oriented analyses are the other integrated sections of software.

PDP-12/LDP is designed for ease of use by laboratory researchers, yet retains total flexibility for experimenting with various data handling techniques. Data acquisition, where data rates permit, places all raw data on tape for permanent storage. In this manner, it may be manipulated in countless ways while the original source remains untouched.

Long series of digits which usually characterize computers are psychologically difficult to relate to results usually perceived in line spectrum form. It is for this reason that the PDP-12/LDP is totally scope oriented. First-line output is fast and in familiar format. From this presentation, one easily selects his final reporting medium which may be teletypewriter, punched tape, chart recorder, line printer, plotter, or magnetic tape.

A further consideration is the real-time environment of the laboratory instrument computer. Languages such as FORTRAN, although convenient, require off-line compiling; which dedicates the system to program preparation. AIPOS and FOCAL-12 (which is used as a part of AIPOS) are real-time, high-level, interpretive languages. This means that as a command is given, it is executed immediately. Also modification is done in real time. Although many applications programs are supplied, almost every-

one wants to do something a little bit differently. Only a real-time, English-like language like FOCAL-12 can make this job easy.

Analytical Instrument Package Operating System

The operating system is a monitor, loader, interrupt handler, several I/O handlers, directory system, converter, system builder, job control, and system tester. AIPOS utilizes LINCtapes marked with 1600 blocks as opposed to the standard 1000-block DIAL tapes. Conversions are facilitated through AIPOS.

The operating system has a syntax which allows identical communication throughout all programs. Its complete format is:

SN FUNIT: FUNCTION OUNIT: OUTFIL. EXT. =
IUNIT: INFIL. EXT, PARAMETERS

SN	— Statement number
FUNIT:	— Specifies storage unit from which function will be loaded
FUNCTION	— Name of binary program to be loaded
OUNIT:	— Output storage unit
OUTFILE. EXT.	— Specifies file name and extension for output
IUNIT:	— Input storage unit
INFILE. EXT.	— Specifies file name and extension for input
Parameters:	— Used to define non-default conditions

This very general statement simplifies in many cases to one of the two following formats:

1) FUNCTION: UNIT:FILE.EXT=UNIT:FILE.EXT
2) FUNCTION; PARAMETERS

Typical commands would look like:

1.5 LT0:FFT LT1:NMR23F = LT1:NMR23R

which says, as statement 1.5 load the fast Fourier transform program from LINCtape unit 0 and operate on the raw NMR data on LINCtape unit 1 (called NMR23R) and store the result on LINCtape unit 1 calling it NMR23F.

The system builder, within AIPOS, will initialize the file structures on available peripherals and support the use of a specific configuration.

MIDAS — Multi-Instrument Data Acquisition System

This system, operating under AIPOS control, will independently acquire data from up to 8 instruments simultaneously with a total aggregate data rate of 1000 points per second.

Intermixing connected instruments such as continuous wave NMR, spectrophotometers, gas chromatographs is the forte of MIDAS. Instruments may be time dependent, or not. The three modes of operations accept y inputs as analog voltages or BCD inputs. Values of X are accepted as inputs from shaft encoders, clocks, or oscillators, and analog voltages.

Raw data is directly stored on tape, and file size (number of data points) is virtually unlimited. Instruments may be initialized, started, stopped, paused, and inputs observed; all independently of other instruments or users.

Data files are stored with experiment name, run number, date and time, and run parameters for future reference. Connections are made so that the computer-based system is totally enslaved to each of the connected instruments.

A simulation mode allows ease of initialization and optimization of instrument parameters.

DORA and DISPLAY — Display-Oriented Research Analysis

These are the common display routines which all AIPOS software utilize. DISPLAY is an abbreviated version of DORA when all of DORA's power is not required. This package displays multiple files with cursors and decimal x-y coordinates. Outputs are to TTY, x-y recorder, and bulk storage units.

Command structure, handled by AIPOS, allows dual or single, split screen or overlapped selectable display of named, stored spectra. Calibration of x-y readout is allowed. Included features are:

1. Horizontal and vertical displacements
2. Moving tape windows
3. Two cursors moving in x and y directions
4. Polarity change
5. x-y recorder output
6. Save cursor value

A typical use of DORA is to display an entire spectrum on the lower half of the scope at reduced resolution, while using the moving window feature to examine regions of interest at full resolution on the upper half.

DORA, used by all other programs, truly makes this computer-based laboratory instrument computer display oriented and easy to use. Using the front panel knobs to adjust data arrays for stripping or comparison, the PDP-12/LDP is analogous to a fast, powerful, dual-spectra slide rule.

Math Routines — Binary Programs for Data Manipulation

This class of programs, from a user point of view, is really the system's heart. These common functions, used in various combinations, are required for handling spectra of every description. The list put forth here is current as of October, 1970; it will be updated continually.

1. Calibrator

   It allows, through scope interaction, calibration of known and unknown spectra to absolute values through the use of up to 4 internal standard or reference peaks using the method of local slopes. Spectra need be calibrated only once and stored for future use.

2. Binary Commands — Dual Spectra Commands
   a. Add spectra
   b. Subtract spectra

3. Unary Commands — Single Spectra Commands
   (Performed on either displayed spectrum.)
   a. Scale
   b. Offset

c. Differentiate
d. Normalize
e. Smooth
f. Strip peak
g. Integrate entire spectrum
h. Integrate between cursors
i. Baseline strip/add to spectrum
j. Baseline strip between cursors
k. Save results

4. Smooth — Least Squares
   Real time, on-line smoothing of noisy data is accomplished with output to tape.

5. Averaging
   a. Multiscan, on line averaging.
   b. Moving window averaging.

6. Fourier Transform
   This program progresses from the specified data array to the Fourier Transform, inverse Fourier Transform, or real, and imaginary spectra. Output is to tape, scope, and recorder. This program will handle up to 16,000 data points.

7. Correlation
   Auto and cross correlation of named files is carried out automatically with output to scope, tapes, and recorder.

Specific Applications Software

Due to the unique requirements of certain analytical instruments, Digital Equipment Corporation supplies dedicated packages for CW and pulsed NMR, high-resolution M.S., low-resolution M.S., and gas chromatography. This software can run sequentially with AIPOS.

1. Gas Chromatography
   This is a research-oriented system which will handle eight G.C.'s automatically and progress from raw input to reporting of corrected concentration without operator intervention. Data is, of course, then available for further manipulation through the AIPOS system.

2. High Resolution Mass Spectroscopy
   This software includes scope-oriented initialization and baseline correction using software thresholding. Centroid times are calculated and mass numbers computed to 1 to 2 millimass numbers. Tabular output is to Teletype, and line plot to recorder.

3. Low-Resolution Mass Spectroscopy
   A unified program handles raw data from a G.C. — M.S. instrument and accepts data at 20 kHz while picking peaks in real time. A mass marker or Hall effect probe on the instrument is required. Output is to scope TTY, and recorder. Spectra are reported as normalized to most intense peak or total ion current.

4. Pulsed NMR

   This package, having been designed for ^{13}C work, handles ^{1}H, and ^{19}F with ease. Over sweep widths of 6000 Hz, resolution of 0.35 Hz, is obtained. Multisweep averaging, Fourier transformation, and reporting are included. Spin simulation and curve fitting routines are used in CW as well as PNMR.

Many other specific applications programs are available through both DECUS and the DEC Program Library.

High-Level Languages

BASIC, FOCAL, FOCAL-12, and FORTRAN are supplied with the system affording powerful computational facility. See the FOCAL-12 section for a detailed description of this powerful language.

CHAPTER 3
EDUCATIONAL SYSTEMS

A presidential commission recently reported that "undergraduate education, without adequate computer education is deficient education". As computers have found wider application in science, industry, government, and the professions; today's student must be given the opportunity to learn about the use of computers irregardless of his speciality.

The graduate engineer, scientist, or technician will encounter the computer in his professional life not simply as a computational aid, but as an integral part of an industrial system or scientific experiment. Small computers are performing functions such as computerized process control, systems, monitoring, data acquisition, and on-line data reduction. These applications will require a thorough understanding of the computer as a tool. Students must understand the techniques required to incorporate the computer into a total system.

The PDP-12 laboratory Computer System is uniquely configured to educate students in computer technology, programming and computer applications. The PDP-12 serves as a complete educational computer system. The many standard peripherals of the PDP-12 help make it exceedingly useful for teaching concepts of computer technology and systems design. In one compact package, the instructor and student have immediate access to a complete stand-alone computer system. The standard peripherals of the PDP-12 allow it to be used as a microcosm system representing all computers. These peripherals include magnetic tape, an analog-to-digital converter, a CRT display, programmable relay outputs, and external sense line inputs. These devices are required elements in a comprehensive computer technology curriculum.

The PDP-12 serves as a total educational computer lending itself to many areas of student instruction.

- Programming for simulation and problem solving
- Computer systems and hardware familiarization
- Logic design and interfacing techniques
- Laboratory applications in physics, chemistry, biology and psychology.

The PDP-12 is an approachable computer. Student "hands-on" interaction is encouraged by the easy-to-use nature of the system. The "hands-on" approach has the advantage of acquainting the student by experience with the operations of the computer. Practical computer concepts can be studied in detail including hardware/software efficiency, control timing and systems synchronization.

ENGINEERING CURRICULUM

Knowledge of the techniques required to integrate a computer into large industrial and scientific systems is crucial for today's chemical, nuclear and process engineers and scientists. Computers to be used as process controllers and monitors must be properly interfaced into the total system. This is accomplished through the use of uniquely-designed logic circuits.

Integrated circuit technology allows for state-of-the-art circuit fabrication by students who are not necessarily electrical engineers, but who have an understanding of basic logic.

A complete curriculum for teaching logic design and computer interfacing is available for use with the PDP-12. Developed at a leading technical university, a complete course textbook and laboratory guide is available. It features details of many student-proven logic design experiments using integrated circuitry. The text is entitled *Experiments in Logic Design and Computer Interfacing.*

Laboratory exercises include the design and fabrication of working computer interfaces. Experiments are built around the H309-A option which makes the input-output lines of the PDP-12 immediately available to the student via simple connectors.

Type H309 I/O Bus access option allows for quick and easy access to the input/output Bus lines of the PDP-12 computer. It was designed for researchers, students, and engineers who must design and quickly interface logic devices and peripherals. The I/O access option extends the entire I/O Bus and external sense lines to the front of the system cabinet via six thirty-six pin connector blocks. These terminals mate with standard DEC M903 or M904 cable connectors.

A standard feature of the option is a regulated +5 volt power supply delivering sufficient current to drive approximately 50 to 100 standard integrated circuits. Over-current and over-voltage protection is provided When maximum limits are exceeded, power shutdown occurs, the current level drops to zero, and an overload condition indicator is displayed. The regulator circuit can be reset by operating the power supply on/off/reset switch.

Power supply characteristics:

D.C. Voltage:	+5 volts ± 5%
Maximum current:	3 amperes
Shutdown time:	10 milliseconds (approx.)

Shutdown occurs when maximum electrical limits are exceeded. All interface connections to the PDP-12 can be made at the assigned module receptacle connectors on the I/O access panel. The module receptacles and assigned use for interface signal connectors are:

RECEPTACLE	USE
N13	Sense Lines
N14	AC, IOP timing outputs
N15	MB Outputs
N16	AC, Skip, interrupt request inputs
N17	Data break address inputs
N18	Data break data inputs

Laboratory kits (type H311 Logic Design Laboratory) are also available for the curriculum. They contain the W979 breadboarding module card, IC sockets, and wire-wrapping tools. These "carry along laboratories" permit the student to do his own designing and wiring independently.

Once he has fabricated his logic card, each student tests his circuit on a PDP-12 utilizing the H309 front panel I/O bus connector system. The student writes his checkout program in machine language and observes the operation of his circuit on an oscilloscope.

Experience has shown that non-EE students successfully learn how to design logic and interfacing systems in a matter of weeks.

COMPUTERS IN THE SCHOOL LABORATORY

As the previous sections implied, the computer is fast becoming a standard tool in the laboratory environment. Laboratory computer systems speed up the process of obtaining data from analytical instruments and reduce the number of errors. Computers perform a more thorough and accurate analysis than is possible when computations are done manually.

For the school, the PDP-12 is an ideal system on which to train students in laboratory computer applications. The PDP-12 helps eliminate the need for complex special interfaces to several types of analytical instruments. The analog-to-digital converter of the PDP-12 is used to input instrumentation signals. Its relay outputs are for instrument control or range switching. The cathode ray tube display, with its graphic and alphanumeric capability, allows for the effective presentation of experimental data and results. Simple operating instructions make the PDP-12 easy to use, even by the unskilled operator.

The PDP-12 is easily interfaced to analytical instruments typically found in school laboratories:

- Spectrophotometers
  - Infra-red
  - Ultra-violet
  - Colorimeter
  - Raman
- Mass Spectrometers
- Gas Chromatographs (high level)
- Nuclear Magnetic Resonance
- Polarographs
- Differential Thermal Analysis
- PH Meter
- Electron Spin Resonance
- Auto Analyzers

STUDENT PROGRAMMING

The PDP-12 has a complete selection of programming languages in addition to its powerful machine language instruction set. These languages include BASIC, FORTRAN, FOCAL, FOCAL-12 and DIBOL.

BASIC

The BASIC language is composed of easy-to-learn English statements and mathematical expressions. Digital's BASIC is a modified version of the very popular elementary algebraic language developed at Dartmount College.

BASIC allows even beginners to use the computer in a few hours. Commands are simple English words and mathematical expressions.

```
10 print "enter the sides of the right triangle"
20 input A,B
30 let C=SQR (A + 2 + B + 2)
40 print "the hypotenuse is" C
50 end
```

PDP-12 BASIC supports a mark sense card reader for Batch processing and/or the Teletype for on-line problem solving. Program file storage is handled by LINCtape magnetic tape and/or magnetic disk.

FORTRAN Compiler (4K)

The 4K FORTRAN Compiler lets the user express problems in a potpourri of English words and mathmetical statements. It reduces the time needed for program preparation and enables the user with little knowledge of the computer's organization and operating language to write effective programs. FORTRAN language consists of four general types of statements: arithmetic logic, control, and input/output. FORTRAN functions include addition, subtraction, multiplication, division, sine, cosne, arctangent, square root, natural logarithm, and exponentiation.

FORTRAN Compiler (8K)

The FORTRAN 8K system features USA Standard FORTRAN syntax; subroutines; two levels of subscripting; function subprogram; input/output supervisor; relocatable output loaded by the Linking Loader; COMMON statements; I, F, E, A, X and H format specifications; and arithmetic and trigonometric library subroutines. It consists of the two-pass FORTRAN Compiler, Linking Loader, Run-Time Monitor, and a library of sub-programs.

The FORTRAN 8K Compiler translates a source program into a symbolic language and then the symbolic version of the program is translated into relocatable binary code, the language of the computer. The binary code is then reloaded into the computer for running of the program.

Business Programming-DIBOL

DIBOL (Digital Equipment Corporation Business-Oriented Language) is a complete business-oriented software system. It allows the programmer to produce complete business applications on the PDP-12 computer. Schools and university departments can use the DIBOL software system as the means for writing their own EDP management/accounting procedures. The small school can use the power of the PDP-12 computer efficiently and economically to teach and construct programs for billing, accounts receivable, inventory control, payroll, and general ledger.

The DIBOL software system contains:

1. DIBOL language
2. Data management system to provide automatic input, sorting, and file maintenance.
3. Monitor to tie the various subsystems together

FOCAL

FOCAL is a conversational language developed by Digital for its family of small computers. If a problem can be stated in the English language, it can just as easily be programmed in FOCAL.

Sitting at the teletype simply type

FOR ADDITION SUBTRACTION
MULTIPLICATION AND DIVISION

USER: TYPE 25.38 + 12.479 − 4.629*
4.747/1.558
FOCAL: = 23.7551*

FOR SINE AND COSINE IN RADIANS
USER: TYPE FSIN (1.57) + FCOS (.1* 1.47)
FOCAL: = 1.9892*

FOR SQUARE ROOT
USER: TYPE FSQT (21.56)
FOCAL: = 4.6433*

FOR EXPONENTIATION
USER: TYPE 25 2
FOCAL = 625.0000*

TO COMPUTE 300 FACTORIAL
USER: SET A=1
FOR 1 = 1.000; SET A = A*1
TYPE %, A
FOCAL: = 0.306051E + 615*

Unlike any other language FOCAL features 14 functions automatic error tracing and character editing.

Multi-User Segments

FOCAL can be shared simultaneously by more than one user by parcelling computer time among the various users. Such a system, referred to as minitime-sharing, permits one computer to serve several persons, allowing each user to feel he has the system all to himself. No detectable delays occur under normal operating conditions. With a very heavy workload, some users may detect only a slight delay, less than a second, in response to their commands to FOCAL. The two multi-user systems associated with FOCAL are detailed below.

QUAD (FOUR-USER FOCAL)

QUAD permits from one to four persons to use FOCAL simultaneously on an 8K PDP-12 computer. Up to four Teletype consoles and appropriate communicating units are required.

LIBRA (SEVEN-USER FOCAL)

LIBRA allows up to seven persons to use FOCAL efficiently on one 8K PDP-12 computer. LIBRA requires, in addition to from one to seven Teletype consoles, appropriate PT08's or DC02's, and at least one disk (RF08 or DF32). There are two versions of LIBRA available, depending on the user's disk system, i.e. RF08 or DF32 version. A disk initialization routine, DISKIN, prepares the disk for use by

LIBRA. With LIBRA, user programs can be saved, retrieved, or deleted from the disk by library capabilities, i.e., each program is assigned a name by the user, and a three-word command tells LIBRA what to do with that program. In all cases, the name of a program must be one to four characters. A directory of saved (stored) user programs can also be listed by LIBRA.

FOCAL 12

FOCAL-12 is a version of the FOCAL language designed to take advantage of the features of the PDP-12 computer. FOCAL-12 allows the user to quickly program simple data acquisition and reduction tasks, without using machine language, and in addition, analyze previously generated data stored on the LINCtape. FOCAL-12 will utilize all standard PDP-12 peripherals, including LINCtape, CRT display, and A-D converter. Refer to the FOCAL-12 section of this book for details.

Time-Sharing

The unparalleled Educational Application of the PDP-12 is further demonstrated by its time-sharing capability. TSS/12 is complete time-sharing hardware/software package allowing up to sixteen simultaneous users in multi-language operation. Built around a powerful field-proven time-sharing monitor, TSS/12 includes three higher-level languages, a full PDP-8 assembly language package, and several important utility programs.

Individual users may select the language best suited to his problem. Different users with different language preferences may utilize the system simultaneously. Available languages include BASIC, FOCAL®, FORTRAN-D, PAL-D, and the utility program.

The assembly language capability is especially important. Advanced or specialized users are not bound by the constraints of higher-level languages. Science and engineering students (who will encounter minicomputers on the job) can be exposed to actual machine programming. Beginning computer students get a quick look at how a computer really works. Systems programmers may add library programs, even whole new language processors. In addition to the standard PDP-8 assembly instructions, a number of input-output instructions have been added. These instructions allow the user's program to control the PDP-12's analog to Digital converter, relay register, and display scope. The list of the additional IOT's is given below.

MNEMONIC CODE	OCTAL CODE	FUNCTION
SRLY	6144	Set and read relay register. If the AC is positive, the relay register is set from AC bits 6-11. If the AC is negative, the current relay register is read into the AC. For both cases (positive or negative AC) the AC has the state of relays on return from this IOT.
SADC	6145	Sample the specified A-D channel. The AC contains the channel you want to

		sample. On return, the AC contains the value of the channel.
DISP	6146	Display points. This IOT enables the user to display a string of points on the scope. The AC contains a pointer to a core list. The first word of this list contains the number of *pairs* of X-Y points you want to display. Bits 2-11 of the AC are used. Up to 1777_8 points may be displayed. The remainder of the list contains the actual X-Y points to be displayed. The AC is not changed upon return from this IOT.
DISC	6147	Display characters. This IOT is similar to DISP except that it displays characters instead of points. The AC points to a 5 word list. The first word on this list is the number of characters to be displayed. Only bits 6-11 determine this so up to 63 characters may be displayed. The second word contains the character size for the extended functions register. If bit 4 is a zero, half size is used, if bit 4 is a one, full size characters are used. The remainder of the word is ignored. Words 3 and 4 contain the starting X and Y values of the characters to be displayed. Word 5 contains a pointer to the character string. The character string is 6 bit ASCII, packed two words to a character. All 64 characters are printing characters. There are no control characters. Thus Code 43 is a # and Code 00 is a @. No check is made for wrap around on the scope.

These peripheral controlling input-output instructions are designed to allow the programmer to code and debug his program in the time-shared environment. When he is ready to run the program "stand alone" for his application, the input-output instructions may be replaced with predefined subroutine calls to peripheral routines.

FOCAL 12

FOCAL-12 is a highly-interactive, interpreter-level language designed for rapid communication between man and machine. It gives students, engineers, and scientists high-level language control of the AD-12 analog-to-digital converter, KW12-A real-time clock, VR12 display LINCtape, and disk files. Simple data acquisition and reduction tasks may be quickly programmed using FOCAL12. Data may be saved on magnetic tape or disk and later retrieved for processing.

FOCAL-12 operates in an on-line, conversational mode and is programmed through the use of short, easy-to-learn, imperative English statements.

FOCAL-12 makes fast and efficient use of the total system capability of a PDP-12 computer equipped with 8K of core memory.

Getting Started in Programming

This is the starting point for learning how to solve numerical problems using FOCAL-12 (Formulating On-line Caluculations in Algebraic Language on the PDP-12 computer).

Loading FOCAL-12

The FOCAL-12 system is provided on a standard LINCtape and is stored under the aegis of the LAP6-DIAL Operating System. FOCAL-12 may be started by loading the LAP6-DIAL System, typing a linefeed key followed by: LO FOCAL-12, unit.

FOCAL-12 begins by typing an asterisk on the Teletype which indicates that it is ready to accept commands.

FOCAL-12 is called a conversational language because the system reacts immediately to the things that you do.

Communicating with the PDP-12

If you press down on the Control (marked CTRL) key and at the same time press on the C key, FOCAL-12 will respond with "?01.00*". The "?01.00" is a coded message from FOCAL-12 meaning the FOCAL-12 program is loaded into the computer. To help you decipher other coded messages, we have included a list of codes, and the meaning of each, in the back of this chapter. Generally speaking, if you write a command which FOCAL-12 cannot interpret, or if you break any of programming rules for writing FOCAL-12 statements, you will get a coded error message.

FOCAL-12 programs can be created on the VR12 Display Scope or on the Teletype. The Teletype is automatically selected first when FOCAL-12 starts. To utilize the scope, type the following command: *OUTPUT SCOPE or *O S

The text on the scope can always be cleared by the command *OUTPUT ERASE or *O E

Return to the Teletype is accomplished by use of the command *OUTPUT TELETYPE or *O T

High-Speed Calculations Using the Type Command

You only need to learn one FOCAL-12 command, TYPE (abbreviated T), calculations such as the following:

$$10^3 \times \frac{3}{10} + 21\text{-}2$$

To do this in FOCAL, you type:

```
* O S
*T 10↑3 *3/10 + 21-2
= 319.0000
* O C
```

(and after you press the RETURN key, FOCAL computes this result. Every line must end with a RETURN.)

This example shows the arithmetic operations performed by FOCAL. These are done from left to right except that exponentiation ( ) is done first, then multiplication (*), then division (/), and then addition (+) or subtraction (−).

This means that:

```
6 + 6*2
```
(which is 18 because multiplication is done before addition)

```
IS NOT (6 + 6) *2
```
(which is 24)

Enclosures

To make sure that the computer performs these operations in the order you want, you can place them inside parenthesis marks. When the computer sees an expression enclosed in parentheses, it does that first. If the statement includes parentheses with parentheses (nesting) it computes the innermost first.

```
7+(6/3) - (5↑2) *3
```

In this example, the computer first computes the values of the expressions enclosed in parentheses: (6/3) is 2, and (5↑2) is 25. Then 7 + 2 − 25*3=−66.0000.

You can also use square brackets, [and], and angle brackets, < and >, to enclose expressions. All of these enclosure symbols are evaluated equally, but the innermost will always be done first. They must always be used in pairs. The [and] enclosures are typed using SHIFT/K and SHIFT/M, respectively.

Another Command: Set

This useful command tells FOCAL-12 "store this symbol and its numerical value. When I use this symbol in an expresion, insert the numerical value."

```
*SET PI=3.14159; SET E=2.71828; SET R=9.12739
```

Symbols may consist of one or two alphanumeric characters. The first character must be a letter, *but must not be the letter F.*

Just for practice, let's use FOCAL-12 to calculate the volume of a sphere which has a radius of 9.12739. (We're going to use two of the symbols we have just defined in the SET commands above.)

The formula is $V = \frac{4}{3}\pi r^3$

which we can type like this:

```
R↑3*PI*4/3
```

You might be interested in running a timing test to show how long it takes to do such calculations by hand, with a calculator, and with FOCAL-12.

The Talking Computer

To make the output of your program absolutely clear to other people, it

is sometimes useful to give FOCAL-12 certain messages or column headings. We call these character strings. These messages are enclosed in quotation marks.

```
*SET E=2.71828
*SET PI=3.14159
*TYPE "PI TIMES E" PI*E
```

and FOCAL-12 types out

```
PI TIMES E=     8.5397*
```

You are not allowed to use the carriage return, line feed or leader-trailer characters in these character strings. But you can tell FOCAL-12 to do a carriage return/line feed by inserting an exclamation mark (!). You can get a carriage return by inserting a number sign (#).

Spaces can be used in character strings as needed.

Command Strings

More than one command can be combined on a single line with up to 50 characters per line on the display, and 72 on the Teletype. Each command must be separated by a semicolon.

```
*S A=123.12;S B=456.45;T A+B
579.5700*
```

FOR Command

This command is used for convenience in setting up program loops and iterations. The general format is

```
*FOR A=B,C,D;(COMMAND)
```

The identifier A is initialized to the value B, then the command following the semicolon is executed. When the command has been executed, the value of A is incremented by C and compared to the value of D. If A is less than or equal to D, the command after the semicolon is executed again. This process is repeated until A is greater than D, at which time FOCAL-12 goes to the next sequential line.

The identifier A must be a single variable. B, C, and D may be either expressions, variables, or numbers. If the coma and value C are omitted, it is asumed that the increment is one. If C, D is omitted, it is handled like a SET statement and no iteration is performed.

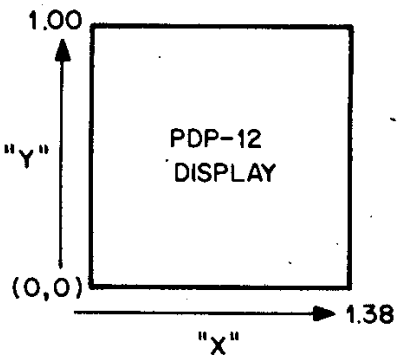
The FOR command is used for performing repetitive calculations as is shown below:

```
*FOR I=0,1,5;  TYPE "I=",I,"    ","I SQUARED=",I↑2,! !
I=     0.0000        I SQUARED=    0.0000
I=     1.0000        I SQUARED=    1.0000
I=     2.0000        I SQUARED=    4.0000
I=     3.0000        I SQUARED=    9.0000
I=     4.0000        I SQUARED=   16.0000
I=     5.0000        I SQUARED=   25.0000
```

Display Graphics

Points may be plotted and maintained on the PDP-12 display by utilizing the function "FDIS (X, Y)." "X" is the horizontal coordinate and "Y" is the vertical coordinate of the displayed point.

The range for "X" is from 0 to 1.38. The range for "Y" is from 0 to 1.00. Point (0, 0) is at the lower right-hand corner.



Points displayed remain until cleared by the "OUTPUT CLEAR" (O C) command.

To plot a horizontal line (multiple points) in the middle of the scope:

```
*FOR I = 0,.01,1.38; SET A = FDIS(I,.5)
```

To plot a vertical line down the center of the display:

```
*FOR I = 0,.01,1; SET A = FDIS(.69,I)
```

A diagonal line is generated by:

```
*FOR I = 0,.01,1; SET A = FDIS(I,I)
```

Slow Plotting

The command "OUTPUT DELAY" (O D) causes the scope to be refreshed once. Although this costs a delay in computation, it allows the user to see his graphics program "grow" on the display. Added to the above programs, the "OUTPUT DELAY" command will cause the partial lines to be displayed as new points are being generated one at a time.

```
*FOR I = 0,.01,1; SET A = FDIS(I,I);OUTPUT DELAY
```

Keeping Track of the Decimal Point

FOCAL-12 results are accurate to six significant digits. As we have shown in the examples so far, FOCAL-12 assumes at the start that you want to see your results with 4 digits (or spaces) to the left of the decimal point and 4 digits to the right of the decimal point. This is called fixed-point notation.

You can change the output format within a TYPE statement by typing "%x.y" where x is the total number of digits to be output, and y is the number of digits to the right of the decimal point. Both x and y are positive integers equal to or less than 31. If y is a single digit, it must be preceded by 0. For example, %6.02 indicates four digits to the left and two to the right of the decimal point.

If your results exceed the format you have specified, FOCAL-12 gives you results in floating-point format, like this:

```
= ±Ø•XXXXXXE ± Z
```

where Z is an exponent of 10.

To switch to floating-point, you include a percent sign (%) followed by a comma, in a TYPE command.

```
*TYPE %, 11
=Ø• 11ØØØØE+ØZ*
```

which is 0.11 times 10^2, or 11. The largest number that FOCAL-12 can handle is + 0.999999 + 615, and thhe smallest is −0.999999 E + 615.

Correcting Mistakes

If you should strike the wrong key, you can delete it by striking the RUBOUT key. Each time you strike RUBOUT, another previously typed character will be deleted. When you strike RUBOUT, FOCAL-12 echoes back a backslash (/) to tell you how many characters you deleted.

Summary

Previously you have learned how one FOCAL-12 command TYPE, is used to evaluate expressions, to type out character strings enclosed in quotes, and to use symbols (defined in SET commands) in expressions.

In the following paragraphs you will learn the other commands, and the use of line numbers to write a sequence of FOCAL-12 statements. As you learn these techniques, you will be advancing rapidly in the art of computer programming.

SEQUENTIAL COMMANDS

Indirect Commands

Up to this point, only commands which are executed *immediately* by FOCAL have been discussed. If a Teletype line is prefixed by a line number, that line is not executed immediately; instead, it is stored in the computer's memory for later execution, usually as part of a sequence of commands.

Line numbers must be in the range from 1.01 to 31.99. The numbers 1.00, 2.00, etc., are illegal line numbers; they are used to identify the entire group. The number to the left of the point is called the group number; the number to the right is called the step number.

```
*1•1 SET A=1
*1•3 SET B=2
*1•5 TYPE %1, A+B
```

Indirect commands are executed by typing GO, GOTO, or DO commands which may be direct or indirect.

GO Command

The GO command causes FOCAL-12 to go to the lowest numbered line to begin executing the program. If the user types a direct GO command

after the indirect commands in the example above, FOCAL-12 will carry out the command at line 1.1, and then the others, sequentially.

```
*GO
=3*
```

The GOTO command causes FOCAL-12 to transfer control to a specific line in the indirect program. It must be followed by a specific line number. After executing the command at the specified line, FOCAL-12 continues to the next higher line. The GOTO causes a program *branch;* we have *jumped* from one sequence of lines to another. Sometimes we merely jump back and repeat a sequence of commands. This technique of repeating sequences is called *iteration,* and it is often used by experienced computer programmers.

```
GOTO 1.3
= 2*
```

DO Command

The DO command is used to transfer control to a specified step, or group of steps, and then return automatically to the command following the DO command.

```
*1.1 SET A=1; SET B=2
*1.2 TYPE "STARTING"
*1.3 DO 3.2
*2.1 TYPE " FINISHED"
*3.1 SET A=3; SET B=4
*3.2 TYPE %1, A+B
*GO
STARTING= 3 FINISHED= 7*
```

If a DO command is written without an argument, FOCAL-12 executes the entire indirect program.

```
*1.1 SET A=1
*1.3 SET B=2
*1.5 TYPE %1, A+B
*DO
```

The following example causes a programming loop, which could be terminated by inserting line 1.5 QUIT, see below.

```
*1.1 SET A=1
*1.2 TYPE A
*1.3 DO 2.0
*1.4  TYPE "FINISHED"

*2.1 SET A=A-1
*2.2 TYPE A
```

DO commands cause specified portions of the indirect program to be executed as closed subroutines. These subroutines may also be terminated by a RETURN command, explained below.

RETURN Command

The RETURN command is used to exit from a DO subroutine. When a RETURN command is encountered during execution of a DO subroutine, the program exits from its subroutine status and returns to the command following the DO command that initiated the subroutine status.

QUIT Command

A QUIT command causes the program to halt and return control to the user. FOCAL-12 types an asterisk and the user may type another command.

COMMENT Command

Beginning a command string with the letter C will cause the remainder of that line to be ignored so that comments may be inserted into the program. Such lines will be skipped over when the program is executed, but will be typed out by a WRITE command.

WRITE Command

The WRITE command without an argument can be used to cause FOCAL-12 to print out the entire indirect program, allowing the user to visually check it for errors.

A group of line numbers, or specific line, may be typed out with the WRITE command using arguments, as shown below.

```
*7.97 WRITE 2.0          (FOCAL-12 types all group 2 lines)
*7.98 WRITE 2.1          (FOCAL-12 types line 2.1)
*7.99 WRITE              (FOCAL-12 types all numbered lines)
*
```

More about Symbols

The value of a symbolic name or identifier is not changed until the expression to the right of the equal sign is evaluated by FOCAL-12. Therefore, before it is evaluated, the value of a symbolic name or identifier can be changed by retyping the identifier and giving it a new value.

```
*SET A1=3↑2
*SET A1=A1+1
*TYPE %2, A1
=      10*
```

NOTE

Symbolic names or identifiers must *not* begin with the letter F (see section on mathematical functions).

The user may request FOCAL-12 to type out all of the user defined identifiers, in the order of definition, by typing a dollar sign ($) after a TYPE command.

```
*TYPE %6.05,$
```

The user's symbol table is typed out like this

```
A@(00)=   0.306051E+615
B@(00)=   1111.11
C@(00)=   39.0000
I@(00)=   301.000
A1(00)=   10.0000
D@(00)=   0.00000
E@(00)=   0.00000
G@(00)=   0.00000
*
```

If an identifier consists of only one letter, an @ is inserted as a second character in the symbol table printout, as shown in the example above. An identifier may be longer than two characters, but only the first two will be recognized by FOCAL-12 and thus stored in the symbol table. Notice that for numbers with more than one integer part, the output format operator %6.05 is ignored so that the whole number can be printed.

Subscripted Variables

FOCAL-12 always allows identifiers, or variables symbols, to be further identified by subscripts (range ±2047) which are enclosed in parentheses immediately following the identifier. A subscript may also be an expression:

```
*SET A1 (I+3*5)=2.71; SET X1(K+3*J)=2.79
```

The ability of FOCAL-12 to compute subscripts is especially useful in generating arrays for complex programming problems.

When FOCAL-12 types out symbol subscripts, only two digits are shown in the range 00-99. Despite this, subscripts up to ±2047 may be used in calculations.

ERROR DETECTION IN INDIRECT STATEMENTS

When an error occurs in an indirect statement, the error message is typed out when the statement is encountered during program execution. In addition to the error code, FOCAL-12 types the line number containing the error, as shown in the following example.

```
*1.10 SET A=2; TYPE "A",A,!
*1.20 SET B=4; TYPE "B",B,!
*1.30 GOTO 1.01
*1.40 TYPE "A+B",A+B
*GO
A=      2.0000
B=      4.0000
?03.05 @ 01.30
*
```

FOCAL-12 executes lines 1.1 and 1.2 and then recognizes that line 1.3 is an illegal command. Therefore it issued the error message to show you that an illegal command was used.

To pinpoint an error in line 3.3, for example, type "DO 3.3?" and the program will be traced until the error is found.

CORRECTIONS

If the user types the wrong character, or several wrong characters, he can used the RUBOUT key as we have explained previously, which echoes a backslash (/) for each RUBOUT typed, to erase one character to the left each time the RUBOUT key is depressed. For example,

```
*ERASE ALL
*1.1 P/TYPE X-Y
*1.2 SET $=13////X=13
*WRITE
C-FOCAL,1969

01.10 TYPE X-Y
01.20 SET X=13
```

The left arrow (←) erases everything which appears to its left on the same line.

```
*WRITE
C-FOCAL,1969

01.10 TYPE X-Y
01.20 SET X=13
*
```

A line can be corrected by retyping the line number and typing the new command.

```
*14.99 SET C9(N+3) = 15
*
```

is replaced by typing

```
*14.99 TYPE C9/Z5-2
*WRITE 14.99
14.99 TYPE C9/Z5-2
*
```

ERASE Command

A line or group of lines may be deleted by using the ERASE command with an argument. For example, to delete line 2.21, the user types

```
*ERASE 2.21
*
```

To delete all of the lines in group 2, the user types

```
*ERASE 2.0
*
```

Used alone, without an argument, the ERASE command causes FOCAL-12 to erase the user's symbols. Since FOCAL-12 may not zero memory when loaded, it is good practice to ERASE ALL before starting a new program.

Typing WRITE after making corrections causes FOCAL-12 to print the indirect program as altered. This is useful for checking commands and for obtaining a "clean" program printout.

ASK Command

The ASK command is normally used in indirect commands to allow the user to input data at specific points during the execution of his program. The ASK command is written in the form

```
*11.99 ASK X,Y,Z
*
```

When line 11.99 is encountered by FOCAL-12, it types a colon (:). The user then types a value in any format for the first identifier, followed by a terminator.[2] FOCAL-12 then types another colon and the user types a value for the second identifier. This continues until all the identifiers or variables in the ASK statment have been given values,

```
*11.99 ASK X,Y,Z
*DO 11.99
:5:4:3*
```

where the user typed 5, 4, and 3 as the values, respectively, for X, Y, and Z.

FOCAL-12 recognizes the value when its terminator is typed. Therefore, a value can be changed but only before typing its terminator. This is done by typing a left arrow (←) immediately after the value, and then typing the correct value followed by its terminator. This is the exception to the use of the left arrow, as explained in the previous section on corrections.

The ALT MODE key, when used as a terminator, is nonspacing and leaves the previously defined variable unchanged, as shown below.

```
*SET A=5
*ASK A
:123*                          (user depressed the ALT MODE key
*TYPE A                         after typing 123)
=   5.0*
```

ALT MODE is frequently used when the user does not wish to change the value of one or more identifiers in an ASK command.

```
*11.99 ASK X,Y,Z
*DO 11.99
:5,:4,:3,*                     (User did not wish to enter new value
*DO 11.99                       for Y, so he typed ALT MODE in
:8,::10,*                       response to second colon.)
*TYPE X,Y,Z
=        8=      4=      10*
```

[2]Terminators are space, comma, ALT MODE, and RETURN keys.

A text string may be included in an ASK statement by enclosing the string in quotation marks, just as in the TYPE command.

```
*1.10 ASK "HOW MANY APPLES DO YOU HAVE?" APPLES
*DO 1.10
HOW MANY APPLES DO YOU HAVE?:25
*
```

The identifier AP (FOCAL-12 recognizes the first two characters only) now has the value 25.

IF Command

In order to transfer control after a comparison, FOCAL-12 contains a conditional IF statement. The normal form of the IF statement consists of the word IF, a space, a parenthesized expression or variable, and the three line numbers separated by commas. The expression is evaluated, and the program transfers control to the first line number if the expression is less than zero, to the second line number if the expression has a value of zero, or to the third line number if the value of the expression is greater than zero. The IF expression or variable must be enclosed in parentheses.

The program below transfers control to line number 2.10, 2.30, or 2.50, according to the value of the expression in the IF statement.

```
*2.1 TYPE "LESS THAN ZERO"; QUIT
*2.3 TYPE "EQUAL TO ZERO"; QUIT
*2.5 TYPE "GREATER THAN ZERO"; QUIT
+IF (25-25)2.1,2.3,2.5
EQUAL TO ZERO*
```

The IF statement may be shortened by terminating it with a semicolon or carriage return after the first or second line number. If a semicolon follows the first line number, the expression is tested and control is transferred to that line if the expression is less than zero. If the expression is not less than zero, the program continues with the next statement.

```
*2.20 IF (X) 1.8; TYPE "Q"
*
```

In the above example, when line 2.20 is executed, if X is less than zero, control is transferred to line 1.8. If not, Q is typed out.

```
*3.19 IF (B)1.8,1.9
*3.20 TYPE B
*
```

In this example, if B is less than zero, control goes to line 1.8, if B is equal to zero, control goes to line 1.9. If B is greater than zero, control goes to the next statement, which in this case is line 3.20, and the value of B is typed out.

If a GOTO or an IF command is executed within a DO subroutine, two actions are possible:

1. If a GOTO or IF command transfers to a line *inside* the DO group, the remaining command in that group will be executed as in any subroutine before returning to the command following the DO.

2. If transfer is to a line *outside* the DO group, that line is executed and control is returned to the command following the DO; unless that line contains another GOTO or IF.

```
*ERASE ALL
*1.1 TYPE "A"; SET X=-1; DO 3.1; TYPE "I"; DO 2
*1.2 DO 2
*
*2.1 TYPE "G"
*2.2 IF (X)2.5,2.6,2.7
*2.5 TYPE "H"
*2.6 TYPE "I"
*2.7 TYPE "J"
*2.8 TYPE "K"
*2.9 TYPE %2.01, X; TYPE " "; SET X=X+1
*
*3.1 TYPE "B"; GOTO 5.1; TYPE "F"
*
*5.1 TYPE "C"
*5.2 TYPE "E"
*5.3 TYPE "L"
*GO
```

(FOCAL-12 types the answer)

```
ABCDGHIJK=-1.0 GIJK= 0.0 GJK= 1.0 BCEL*
```

MODIFY Command

Frequently, only a few characters in a particular line require changing. To facilitate this job, and to eliminate the need to retype the entire line, the FOCAL-12 programmer may use the MODIFY command. Thus, in order to modify the characters in line 5.41, the user types MODIFY 5.41. This command is terminated by a carriage return, whereupon the program waits for the user to type that character in the position in which he wishes to make changes or additions. This character is not printed. After he has typed the search character, the program types out the contents of that line until the search character is typed.

At this point, the user has seven options:

1. Type in new characters in addition to the ones that have already been typed out.

2. Type a form feed (CTRL/L); this will cause the search to proceed to the next occurrence, if any, of the search character.

3. Type a CTRL/BELL; this allows the user to change the search character just as he did when first beginning to use the MODIFY command.

4. Use the RUBOUT key to delete one character to the left each time RUBOUT is depressed.

5. Type a left arrow (←) to delete the line over to the left margin.

6. Type a carriage return to terminate the line at that point, removing the text to the right.

7. Type a LINE FEED to save the remainder of the line.

The ERASE ALL and MODIFY commands are generally used only in immediate mode because they return to command mode upon completion.

During command input, the left arrow will delete the line numbers as well as the text if the left arrow is the rightmost character on the line.

Notice the errors in line 7.01 below.

```
*7.01 JACK AND BILL W$NT UP THE HALL
*MODIFY 7.01
JACK AND B/JILL W$/ENT UP THE HA/ILL
*WRITE 7.01
07.01 JACK AND JILL WENT UP THE HILL
*
```

To modify line 7.01, a B was typed by the user to indicate the character to be changed. FOCAL-12 stopped typing when it encountered the search character, B. The user typed the RUBOUT key to delete the B, and then typed the correct letter J. He then typed the CTRL/BELL keys followed by the $, the next character to be changed. The RUBOUT deleted the $ character, and the user typed an E. Again a search was made for an A character. This was changed to I. A LINE FEED was typed to save the remainder of the line.

Caution

When any text editing is finished, the values in the user's symbol table are reset to zero. Therefore, if the user defines his symbols in direct statements and then uses a MODIFY command, the values of his symbols are erased and must be redefined.

However, if the user defines his symbols by means of indirect statements prior to using a MODIFY command, the values will not be erased because these symbols are not entered in the symbol table until the statements defining them are executed.

Notice in the example below that the values of A and B were set using direct statements. The use of the MODIFY command reset their values to zero and listed them after the defined symbols.

```
*ERASE ALL
*SET A=1
*SET B=2
*1.1 SET C=3
*1.2 SET D=4
*1.3 TYPE A+B+C+D; TYPE !; TYPE $
*MODIFY 1.1
SET      C=3/5
*GO
=  9.00
C@(00)=   5.00
D@(00)=   4.00
A@(00)=   0.00
B@(00)=   0.00
*
```

USING THE TRACE FEATURE

The trace feature is useful in checking an operating program; those parts of the program which the user has enclosed in question marks will be printed out as they are executed.

```
*ERASE ALL
*1.1 SET A=1
*1.2 SET B=5
*1.3 SET C=3
*1.4 TYPE %2, ?A+B-C?,!
*1.5 TYPE ?B+A/C?,!
*1.6 TYPE ?B-C/A?
*GO
A+B-C=    3
B+A/C=    5
B-C/A=    2*
```

In the following example, parts of 3 lines are printed.

When only one ? is inserted, the trace feature becomes operative as FOCAL-12 encounters the ? during execution, and the program is printed out from that point until another ? is encountered. The program may loop through the same ? until an error is encountered (execution stops and an error message is typed), or until program completion.

```
*ERASE ALL
*1.1 ?SET A=0B; TYPE %3,A!
*1.2 FOR B=1,1,4; TYPE B+A!
*GO
SET A=0B; TYPE %3,A!
=    0FOR B=1,1,4; TYPE B+A!
=    1 TYPE B+A!
=    2 TYPE B+A!
=    3 TYPE B+A!
=    4*
```

In this example, FOCAL-12 encountered the ? as it entered line 1.1 and traced the entire program.

MATHEMATICAL FUNCTIONS

The functions are provided to improve and simplify arithmetic capabilities and to give potential for expansion to additional input/output devices. A standard function call consists of four (or fewer) letters beginning with the letter F and followed by a parenthetical expression.

```
FSGN(A-B*2)
```

There are three basic types of functions: simple, extended, and I/O. The first type consists of integer part, sign part, absolute value and square root functions.

In the second type, the extended arithmetic functions, are loaded at the option of the user. They compute logarithms, exponentials, arctangents, sines, and cosines.

The input/output functions are the third type. These include a nonstatistical random number generator (FRAN) whose values range from .5 to .9. There are also functions available to control scopes and analog-to-digital converters.

Plotting Mathematical Functions

The following program uses the FSIN and FCOS functions to plot a circle on the scope.

```
*1.10 SET A = .5; SET B = .5

*1.20 SET C = .3; SET D  = .3

*1.30 FOR I = 0,.04,6.25;SET P = FDIS(A+C*FSIN(I),B+D*FCOS(I))
```

Changing the value A will cause the circle to be moved to the right or left. The value B locates the vertical position of the circle.

Modifying the values C and D changes the horizontal and vertical displacement of the points. If only one is changed, an ellipse will be plotted. Changing both equally will cause the overall size of the circle to be different.

Analog-to-Digital Conversion

The analog-to-digital converter is programmed through use of the FADC(X) function where X is the input channel selected.

Thus, the program

```
*SET B = FADC (1); TYPE B
```

will print the digitized value of the signal on analog channel 1.

The short program below will take 100 data points from analog channel four and display the points on the scope.

```
*F X = 0,.01,1; SET A = FDIS (X,FADC(4)); O D
```

CALCULATING TRIGONOMETRIC FUNCTIONS IN FOCAL-12

FUNCTION	FOCAL12 REPRESENTATION	ARGUMENT RANGE	FUNCTION RANGE
Sine	FSIN(A)	$0 \leq \|A\| < 10\uparrow 4$	$0 \leq \|F\| \leq 1$
Cosine	FCOS(A)	$0 \leq \|A\| < 10\uparrow 4$	$0 \leq \|F\| \leq 1$
Tangent	FSIN(A)/FCOS(A)	$0 \leq \|A\| < 10\uparrow 4$ $\|A\| \neq (2N+1)\pi/2$	$0 \leq \|F\| < 10\uparrow 6$
Secant	1/FCOS(A)	$0 \leq \|A\| < 10\uparrow 4$ $\|A\| \neq (2N+1)\pi/2$	$1 \leq \|F\| < 10\uparrow 6$
Cosecant	1/FSIN(A)	$0 \leq \|A\| < 10\uparrow 4$ $\|A\| \neq 2N\pi$	$1 \leq \|F\| < 10\uparrow 6$
Cotangent	FCOS(A)/FSIN(A)	$0 \leq \|A\| < 10\uparrow 4$ $\|A\| \neq 2N\pi$	$0 \leq \|F\| < 10\uparrow 440$
Arc sine	FATN(A/FSQT(1−A↑2)	$0 \leq \|A\| < 1$	$0 \leq \|F\| \leq \pi/2$
Arc cosine	FATN(FSQT(1−A↑2)/A)	$0 < \|A\| \leq 1$	$0 \leq \|F\| \leq \pi/2$
Arc tangent	FATN(A)	$0 \leq A \leq 10\uparrow 6$	$0 \leq F < \pi/2$
Arc secant	FATN(FSQT(A↑2−1))	$1 \leq A \leq 10\uparrow 6$	$0 \leq F < \pi/2$
Arc cosecant	FATN(1/FSQT(A↑2−1))	$1 < A < 10\uparrow 300$	$0 < F < \pi/2$
Arc cotangent	FATN(1/A)	$0 < A < 10\uparrow 615$	$0 < F < \pi/2$
Hyperbolic sine	(FEXP(A)−FEXP(−A))/2	$0 \leq \|A\| < 700$	$0 \leq \|F\| \leq 5*10\uparrow 300$
Hyperbolic cosine	(FEXP(A)+FEXP(−A))/2	$0 \leq \|A\| < 700$	$1 \leq F < 5*10\uparrow 300$
Hyperbolic tangent	(FEXP(A)−FEXP(−A))/ (FEXP(A)+FEXP(−A))	$0 \leq \|A\| < 700$	$0 < \|A\| \leq 1$
Hyperbolic secant	2/(FEXP(A)+FEXP(−A))	$0 \leq \|A\| < 700$	$0 < F \leq 1$
Hyperbolic cosecant	2/(FEXP(A)−FEXP(−A))	$0 \leq \|A\| < 700$	$0 < \|F\| < 10\uparrow 7$
Hyperbolic cotangent	(FEXP(A)+FEXP(−A))/ (FEXP(A)−FEXP(−A))	$0 \leq \|A\| < 700$	$1 \leq \|F\| < 10\uparrow 7$
Arc hyperbolic sine	FLOG(A+FSQT(A↑2+1))	$-10\uparrow 5 < A < 10\uparrow 600$	$-12 < F < 1300$
Arc hyperbolic cosine	FLOG(A+FSQT(A↑2−1))	$1 \leq A < 10\uparrow 300$	$0 \leq F < 700$
Arc hyperbolic tangent	(FLOG(1+A)−FLOG(1−A))/2	$0 \leq \|A\| < 1$	$0 \leq \|F\| < 8.31777$
Arc hyperbolic secant	FLOG((1/A)+FSQT((1/A↑2)−1))	$0 < \|A\| S1$	$0 \leq F < 700$
Arc hyperbolic cosecant	FLOG((1/A)+FSQT((1/A↑2)+1))	$0 < \|A\| < 10\uparrow 300$	$0 \leq \|F\| < 1400$
Arc hyperbolic cotangent	(FLOG(X+1)−FLOG(X−1))/2	$1 < A < 10\uparrow 616$	$0 \leq F < 8$

Data Storage and Retrieval

As part of the data collection and reduction task, the typical user will collect data and store it on LINCtape or disk. FOCAL-12 will access such data either as named binary files* under the DIAL system; or by absolute block number addressing, where the location of the data is the users responsibility. The data set may be in any of these data formats — unsigned 1 word integers, signed 2 word fractions, has been "opened" (see section 3.2) any element of the data array may be addressed as a subscripted variable and the actual tape/disk operations are invisible to the program. Each block in the file will contain 256 integers or 128 fractions or 85 floating point numbers (the last word of the block is unused).

Library Make

When a set of data is to be saved in a file, the file must already be defined in the DIAL filing system. Such a file may be created by using the LIBRARY MAKE command

L M, length, name, unit

where "length" is the number of blocks required to hold the data**; "name" is the name to be assigned to the file for the DIAL index; and "unit" is the appropriate device unit number, as defined in DIAL-MS:

Device	DEVICE UNIT NUMBER
8 LINCtapes	0-7
4 RS08 disks	10-17
1st RK08 disk	10-15
2nd RK8 disk	20-25
3rd RK8 disk	30-35
4th RK8 disk	40-45

Thus L M, 19, DATA, 0

would create a 19 block file on tape 0, naming it, DATA.

Library Open

Before an array on tape or disk can be utilized by a FOCAL-12 program, it must be "opened" via the LIBRARY OPEN command

L O, file number, format, name, unit

where "file number" is specified as Fn, where $0 \geq n \geq 7$; "format" is F for floating point, S for signed 2 word fraction, of U for unsigned 1 word integer; "name" is either the DIAL file name; "block number" typed as #number, is the starting block number; and "unit" is the device unit number.

The function of the OPEN command is then to associate a file number, Fn, with a data array on tape or disk and to define the type of data. For example,

L O, F1, F, DATA, 1

declares an array of floating point numbers in a file named DATA, on tape unit 1, is to be referred to as file number F1. Any piece of data in

*To those familiar with DIAL formats, there is no "header" block

**The number of blocks can be estimated using the constant 256 words/

the array may now be accessed using the standard FOCAL subscripting procedure. For example,

```
1.01  L M, 1, DATA, 0
1.02  L O, F2, U, DATA, 0
1.03  F I = 0, 255;S F2 (I) = 0
1.04  Q
```

would create a one block file named DATA on tape 0; clear the array of unsigned integers to 0's; and quit.

Similarly

```
1.01  L O, F2, U, #100, 0
1.02  F I = 0, 511;S F2 (I) = 0
1.03  Q
```

would set blocks 100 and 101 of tape 0 to all 0's and quit.

Library Close

Open data files are "closed" with the LIBRARY CLOSE command in the form

L C, file number

For example,

```
1.01  L M, 19, COPY, 1
1.02  L O, F1, F, COPY, 1
1.03  L O, F2, F, ORIG, 0
1.04  F I = 1, 1200;S F1 (I) = F2 (I)
1.05  L C, F1; L C, F2
1.06  Q
```

would
— create a file on unit 1 that is 19 blocks long called "COPY".
— copy 1200 floating point numbers from file ORIG of LINCtape 0 to LINCtape 1 file copy.
— QUIT after closing both files.

LIBRARY SAVE

FOCAL-12 programs may be saved on LINCtape or disk for later use. The LIBRARY SAVE command is used to store the current (just edited) FOCAL-12 program:

L S, name, unit

where "name" is the binary file name to be inserted in the DIAL index and "unit" is device unit number.
For example,

L S, $NEWPRGM, 7

would save the program (text and variables) just typed in as a binary file, named $NEWPRGM, on tape unit 7. Since the program is saved just as a binary DIAL program, and listed as such in the DIAL index, it is sug-

gested that FOCAL-12 programs be filed under some standard notation, such as dollar sign for the first character. Note that a copy of the program just saved remains in core after an L S command and may be executed using the GO command.

LIBRARY LOAD

A FOCAL-12 program that has been saved by a LIBRARY SAVE command can be retrieved from the tape or disk by a LIBRARY LOAD command in the format:

```
L L, name, unit
```

where "name" and "unit" are as previously described in 4.1. For example,

```
L L, $NEWPRGM, 0
```

would retrieve the program saved in the example in section 4.1 (assuming the tape had been put on transport 0). Once loading is complete, FOCAL-12 prints an asterisk to indicate editing may continue or the program may be started.

LIBRARY GO

A FOCAL-12 program that has been saved by a LIBRARY SAVE command can be retrieved from a DIAL binary file on tape or disk and started automatically by a LIBRARY (load and) GO command in the format

```
L G, name, unit
```

where "name" and "unit" are as specified for LIBRARY LOAD. For example,

```
L G, $NEWPRGM, 0
```

would not only load the program, $NEWPRGM, into memory, but would start it automatically.

Note that this feature can be used in conjunction with FOCAL-12's data file handling to enable the operation of large programs by "segmenting" or "chaining". For example, one segment could set up an experiment, acquire data, store it into a file, and load and start a second segment using the LIBRARY GO command:

```
1.05  L M, 10, DATA,0
1.10  L O, F0, U, DATA, 0
        .       .
        .       .
        .       .
2.75  L G, $2NDSGMT, 0
```

The second segment could than process the data, put up a display of results, etc.

```
1.05  L O, F0, U, DATA, 0
        .
        .
        .
        .
        .
```

The processing of the "L G" command requires the use of a portion of the display buffer; however, up to approximately 450 points may be retained. If the display is to be cleared, of course, the second segment should merely start with an "O C" command.

OUTPUT INTERVAL

The KW12A Clock can be used by FOCAL-12 for user specified interval timing, permitting a delay of known duration to elapse between events. This interval is established by the O I command

O I, N

where n, which may be an expression, is the length of the interval in seconds, with .01 n 40.95. Thus an interval of 2 seconds is specified by

O I, 2

In this case, FOCAL-12 would start the clock such that it would "tick" at 2 second intervals. If an O I command is now issued without an argument, FOCAL-12 will delay the program until the next tic (up to 2 seconds), thereby synchronizing the program with the real time clock. The following program averages A/D channel 1 samples taken once per second for 20 seconds.

```
1.01  O I, 1; S A = 0
1.02  F I = 1, 20; S A = A + FADC (1); O I
1.03  T A/20,!
1.04  Q
```

ADDITIONAL FUNCTIONS

An overlay to FOCAL-12 allows for the programmed control of the PDP-12 console switches, relay register, and external sense lines. The generalized format is FX(n) where n has the following values:

0-5	Sense Switches 0-5 = 0 if off = 1 if down
6	Left Console Switches
7	Right Console Switches
10-15	Status of Relays 0-5 0 = Open 1 = Closed
20-25	Open Relay 0-5
30-35	Close Relay 0-5
40-55	Status of External Sense Lines 0 = Low 1 = High

Programming Techniques

To decrease program length, maximize available core area, and assist in preparing complex routines, the experienced programmer can implement the following suggestions:

1. All commands can be abbreviated to their first letter.
2. A string of commands, except WRITE, RETURN, MODIFY, QUIT, T $, and ERASE, can be combined on one line (up to 72 characters), with each command separated by a semicolon.

3. When creating a lengthy program, it is a good programming practice to leave free line numbers scattered throughout the body of the program. This will permit insertion of additional commands without complicated referencing routines. Remember that programs are executed sequentially by line number; consequently, an addition to the program placed physically at the end will be executed in turn. Line numbers must be in the range 1.01 to 31.99.

4. Some programs may require a keyboard response of YES or NO to a question asked during program execution. The answer typed to the question determines the next command to be executed (for example, in the initial dialogue). For this purpose, alphanumeric numbers are used in an IF statement to direct the execution.

```
*1.1 TYPE "DO YOU WANT A LINE?",!
*1.2 ASK "TYPE YES OR NO",ANS,!
*1.3 IF (ANS-ØYES)2.1,2.2,2.1
*
*2.1 QUIT
*2.2 TYPE "--------------",!
*2.3 GOTO 1.1
*GO
DO YOU WANT A LINE?
TYPE YES OR NO:YES

--------------
DO YOU WANT A LINE?
TYPE YES OR NO:NO

*
```

If the user types the answer YES, the identifier ANS is given the alphanumeric value of YES. When the IF statement is executed, the parenthetical expression ANS-OYES equals zero, and the command at line 2.2 is executed. If the user types YES in answer to the ASK question, then when its alphanumeric value is substituted in the parenthetical expression, the expression will not equal zero and line 2.1 will be executed. Note that for YES/NO responses, the sign of the parenthetical expression is irrelevant; only its zero or non-zero value is of interest.

5. To avoid filling storage with the push-down list during long routines, it is helpful to limit the number of levels of nested expressions in a command. Use of abbreviations and limited number of variable names will maximize storage space. An FCOM function to increase variable storage is explained in DEC-08-xJyA-D, FOCAL-71 System User's Guide.

SUMMARY OF COMMANDS

FOCAL-12 Command Summary

COMMAND	ABBREVIATION	EXAMPLE OF FORM	EXPLANATION
ASK	A	ASK X, Y, Z	FOCAL types a colon for each variable; the user types a value to define each variable.
COMMENT	C	COMMENT	If a line begins with the Letter C, the remainder of the line will be ignored.
CONTINUE	C	C	Dummy lines
DO	D	DO 4.1	Execute line 4.1; return to command following DO command
		DO 4.0	Execute all group 4 lines.
		DO ALL	Return to command following DO command, or when a RETURN is encountered.
ERASE	E	ERASE	Erases the symbol table.
		ERASE 2.0	Erases all group 2 lines.
		ERASE 2.1	Deletes line 2.1.
		ERASE ALL	Deletes all user input.
FOR	F	For i x,y,z; (commands)	Where the command following is executed at each new value.
		FOR i x,z; (commands)	x initial value of i y value added to i until i is greater than z.
GO	G	GO	Starts indirect program at lowest numbered line number.
GO ?	G ?	GO ?	Starts at lowest numbered line number and traces entire indirect program until another ? is encountered, until an error is encountered, or until completion of program.
GOTO	G	GOTO 3.4	Starts indirect program (transfers control to line 3.4). Must have argument.
IF	I	IF (X) Ln, Ln, Ln	Where X is a defined identifier,
		IF (X) Ln, Ln; (commands)	a value, or an expression, followed by one to three line numbers.
		IF (X) Ln; (commands)	If X is less than zero, control is transferred to the first line number, if X is equal to zero,

			control is to the second line number. If X is greater than zero, control is to the third line number.
MODIFY	M	MODIFY 1.15	Enables editing of any character on line 1.15 (see below).
QUIT	Q	QUIT	Returns control to the user.
RETURN	R	RETURN	Terminates DO subroutines, returning to the original sequence.
SET	S	SET A = 5/B*C;	Defines identifiers in the symbol table.
TYPE	T	TYPE A + B − C;	Evaluates expression and types out = and result in current output format.
		TYPE A − B, C/E;	Computes and types each expression separated by commas.
		TYPE "TEXT STRING"	Types text. May be followed by ! to generate carriage return-line feed, or # to generate carriage return.
WRITE	W	WRITE	FOCAL-12 types out the entire indirect program.
		WRITE ALL	
		WRITE 1.0	FOCAL-12 types out all group 1 lines.
		WRITE 1.1	FOCAL-12 types out line 1.1.
OUTPUT SCOPE		O S O S	All "Teletype" output is diverted to the display scope. Includes Teletype echo, output from TYPE and WRITE commands, error diagnostics, etc.
OUTPUT TELETYPE		O T O T	Negates output scope command.
OUTPUT DELAY		O D O D	Interrupt computation to refresh the display scope.
OUTPUT CLEAR		O C O C	The display scope is cleared of all points and characters.
OUTPUT ERASE		O E O E	Clears the display scope of all alphanumeric characters.
LIBRARY MAKE		L M L M, length, name, unit	Create a data file: length = Number of blocks (256 words/block)

name = File name for LAP6-DIAL index (8 characters maximum)

unit = Device selection: LINCtapes = 0–7 Disk = 10–17

LIBRARY OPEN	L O	L O, file number, format, name, unit Activate a data file created by the LIBRARY MAKE command. file Number = F0–F7 format F = Floating Point S = Signed 2-word fraction U = Unsigned 1-word integer name = File name as as used in LIBRARY MAKE command unit = Device selection
LIBRARY CLOSE	L C	L C, file number Deactive a file previously opened by the LIBRARY OPEN command.
LIBRARY SAVE	L S	L S, name, unit Save the current FOCAL12 program under the specified file name on the uniit selected.
LIBRARY LOAD	L L	L L, name, unit Load the saved FOCAL12 program from the unit selected. New program replaces old.
LIBRARY GO	L G	L G, name, unit Load and automatically start the selected FOCAL12 program.

FOCAL12 Operations and Their Symbols

Mathematical operations:

↑	Exponentiation
*	Multiplication
/	Division
+	Addition
–	Subtraction

Control Characters:

%	Output format delimiter	
!	Carriage return and line feed	
#	Carriage return	
$	Type symbol table contents	
( )	Parentheses	(mathematics)
| |	Square brackets	(mathematics)
< >	Angle brackets	(mathematics)
" "	Quotation marks	(text string)
? ?	Question marks	(trace feature)
*	Asterisk	(high-speed reader input)

FOCAL-12's Functions

FSQT( )	Square Root
FABS( )	Absolute Value
FSGN( )	Sign Part of the Expression
FITR( )	Integer Part of the Expression
FRAN( )	A Random number Generator
FEXP( )	Natural Base to the Power
FSIN( )	Sine
FCOS( )	Cosine
FATN( )	Arctangent
FLOG( )	Naperian Log
FDIS( )	Scope Functions
FADC( )	Analog to Digital Input Function
FNEW( )	User Function
FCOM( )	Storage Function

CHAPTER 4
INDUSTRIAL SYSTEMS

Industrial laboratories have many missions. In some, there is a great deal of basic research being done. In others applied research and product development takes place. Still other laboratories are engaged in product evaluation, product test and quality control. Metrology laboratories and Standards Laboratories are also found in the industrial environment.

Such laboratories have many things in common despite the diversity of the missions being accomplished. First of all, the laboratory personnel are in general highly intelligent, well qualified people, usually quite conversant with a number of skills. They have typically used computers in their problem solving, and in many environments the computer forms a part of the laboratory itself. The different levels of motivation inherent in such laboratory workers means that they are typically quite willing to innovate.

Industrial laboratory researchers have a common mission and a common goal — the achievement of meaningful research. The name or label that this research bears is not as important as its accomplishments. It is not surprising then that in all such laboratories the general purpose digital computer has been recognized as a means of assisting and achieving these goals.

The tools of the trade of the industrial researcher range from quite sophisticated chemical or mechanical instruments through strain gages, thermocouples, accelerometers, proximity detectors down to the simplest of limit switches, lights, etc. In many cases these transducers have been tied together in a systematic approach to data acquisition using analog or digital logic. It is also quite common to find the integrating digital voltmeter as a tool of the industrial researcher, converting analog signals to digital information for further processing.

In general, the computer in the laboratory can perform many tasks. It can be used for data acquisition and data logging. It can perform process monitoring and control. The computer is commonly used as a means of displaying data after acquisition. It can also store and retrieve the data. Data reduction is often required and this can take place in one of two ways: either the data can be pre-processed, then transmitted to a central computer or the data can be fully processed in the laboratory, eliminating the requirement for further processing outside the laboratory environment. The computer can also be used to format, file, and catalog data for subsequent use at some later point in time. The data which has been acquired, stored and processed is frequently required for outputting in some report format or in graphic form.

There are many questions which must be answered before a specific computer system can be installed in a laboratory. The first item of interest is the number of variables or signals in which the researcher is interested. Aside from the number of signals that he is interested in acquiring, he must also specify the rate at which these signals are to be sampled, the duration of time over which he wishes to sample this information, and the source of the information as well as the environment.

If he is also interested in controlling an experiment or transducers, the user must specify the type of control that he wants to achieve; simple on-off control, proportional control or some other means of control. This will determine the kinds of interfacing devices required by the computer.

When data is being acquired, the amount of data which must be stored will determine the amount of bulk storage necessary. If data is to be acquired over long periods of time at other than extremely low data rates, the amount of bulk storage required can be considerable. If on the other hand most of the data is processed and examined by the researcher, the amount of bulk storage can be considerably reduced.

In keeping with the philosophy of reducing the amount of unnecessary information resulting from the experiment, the computer can successfully cut-down on the amount of information that the researcher must examine to determine the result of his work. By doing extensive processing and reduction of data, the computer can cut to a bare minimum the number of variables which must be examined in order to arrive at a conclusion about one's research. This serves to keep the researcher intimately involved with what he is doing, and should minimize the possibility of useless data being recorded or taking up valuable storage space in the system.

In addition to the data acquired from an experimental set up, it is often necessary to store textual information explaining the data. Aside from the title of experiment or test, it states the experimental conditions and the significant parameters not being directly monitored. Many times the user would like to insert a considerable number of comments regarding what he feels to be the significance of a particular experimental run. These comments can also be categorized as parameters for cataloged file retrieval — hence the importance of being able to store many different kinds of information, a capability which the digital computer lends itself to easily.

The computer has become a useful tool in laboratories throughout the world because it can reduce tedium, not create it. Reports, previously drudgery, can easily be done using conversational languages and higher level languages such as FOCAL-12. Graphic output is possible using programs created for such purposes. The PDP-12 has two programs available for such output — CATACAL (with an analog plotter), or GRAPHA which uses a digital incremental plotter, (available from Digital Equipment Corporation). Both of these programs, when used on the PDP-12 make use of the VR12 display, permitting a preview of the graphic data before it is plotted, eliminating unwanted outputs.

The difficulty involved in using general purpose computers in the laboratory generally arises from the fact that the "real-world" is analog in nature while the general purpose computer is a digital device. The interrelationship between the "real-world" and the computer is not then a trivial matter.

Most instruments, for example, have analog voltages as outputs. Only recently have instrument manufacturers begun the practice of imple-

menting various coded binary outputs to improve the facility of interfacing their instruments with computers. Transducers are generally analog devices, although some piezoelectric devices with digital outputs are presently available, particularly pressure-sensing transducers.

By way of contrast, monitoring of events implies utilizing limit switches or other similar devices which can be treated in a digital fashion and consequently are not too difficult to interface to the computer. The same thing applies to the integrating digital voltmeter, a voltage to frequency converter producing a digital output corresponding to an analog input.

As far as the digital computer is concerned it normally accepts binary, binary coded decimal, octal or other alphanumeric inputs. In all but the binary case, the input requires some kind of code conversion via a computer program before it can be easily used.

In addition to the direct binary and coded binary inputs which the computer will accept, it is possible to use various analog-to-digital techniques to produce a binary number which is proportional to the amplitude of an analog voltage. A great deal of work has been done in this area over the last several years and many A/D devices are now available at reasonable prices.

The PDP-12A, recognizing its role as a laboratory computer, includes as one of its standard input devices the AD12 a 16 channel multiplexed analog-to-digital converter. This device is unique in that it is controlled by a one computer word instruction. This sample instruction (SAM) makes it very easy to use the A to D converter on the PDP-12. In addition to the AD12, many other standard A to D converters are available for use on the PDP-12. This permits the selection of the conversion devices most appropriate for the problem being undertaken.

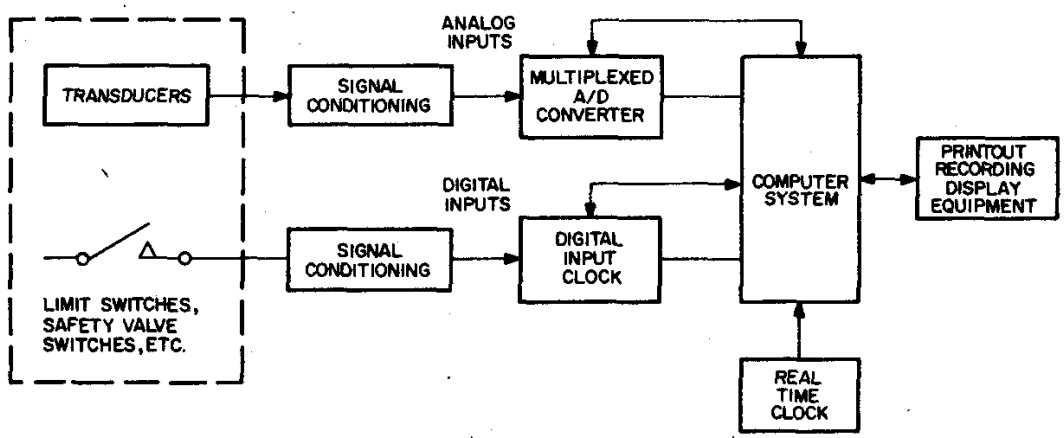
The laboratory often requires computer outputs for device control, and these must be compatible with the lab environment. This means that the computer must provide both analog and digital outputs. Digital outputs, in the form of binary or coded binary words or relay closures, are easily provided on the PDP-12. The analog outputs from a computer come from a digital-to-analog converter which is similar in concept but opposite in function to the A to D converter. Such a device takes a binary number and produces an output voltage proportional to that number. Both forms of output have their uses and computers in the laboratory environment utilize both of them.

The marriage of the Laboratory and the Computer poses a problem within the industrial environment. This is the same problem that arises whenever one interfaces a computer to the analog world. These problems have been overcome in our many installations throughout the world, and the experience that DEC has gained in solving them can be brought to bear on new installations in the future. Of special interest in the laboratory is the ability of the computer to be used for data acquisition.

The high speed, accuracy and data handling flexibility of digital computer systems can lead to data gathering techniqes beyond the ability of simple displays and recorders. Once process variables are converted to digital

form, all the resources of automatic digital data processing may be applied to the task, from a simple strip printout to sophisticated data reduction and correlation resulting in a labeled and tabulated printout. Data acquisition applications range from a single-channel A/D converter measuring slow-changing data once per minute to an automatically sequenced multi-channel system that scales inputs, performs limit checks and logical decisions, and formats data for recording on magnetic or punched tape.

The basic elements of a computerized data acquisition system are illustrated.



Data Acquisition System

ANALOG INPUTS

Process sensors such as thermocouples, potentiometers, strain gauges and flow meters convert physical events or quantities to proportionally-varying electrical signals. These signals must be converted to digital numbers acceptable to a data processor.

Signal conditioning may be required for impedance matching, to supply excitation voltage for passive transducers, to provide amplification for low-level signals, or to transmit signals over long line.

Multiplexing is required to time-share the A/D converter when many inputs must be scanned. The type of analog-digital converter and multiplexer combination can determine whether signal conditioning is needed. Multiplexer-converters that use solid state switching operate at the highest scanning rates but require high level input signals. A converter system that uses guarded three-wire multiplexing and differential amplification, or one using a voltage-to-frequency conversion method, will often accept low-level signals from strain gauges or thermocouples without conditioning.

DIGITAL INPUTS

In addition to converted analog inputs, the computer can accept digital or contact-closure information. Process-related on-off signals such as alarms, limit indications, and selector switch settings can then be correlated with the analog data. Direct digital transducers such as turbine

flow meters and digital shaft position encoders interface to digital input channels without the intermediate A/D conversion.

Digital inputs may need signal conditioning to match the computer's logic levels. Excitation voltage may be needed for contacts. Pulses must be retained long enough for the computer to sample them.

A few digital inputs are simply gated into the computer at programmed intervals. Computing time is saved if the digital input system interrupts the computer and requests read-in only when the data changes, or when an external "read" request occurs.

When the number of input lines exceeds the computer word length, a scanning technique must be used. The digital input channel connects a group (12, 18, or 36 lines) of digital signals to the computer input lines, then connects the next group of signals to the same input lines, and so on. Systems of this type can be expanded to handle thousands of inputs. Input groups can be selected by the computer program, or the scanner can sequence from group to group automatically, interrupting the computer when each group is ready to be sampled.

REAL-TIME CLOCKS

A special variety of digital input is the real-time clock. The simplest type generates timing pulses than can cause a computer interrupt; an internal computer memory location counts elapsed time. The clock source can be either a multivibrator or the 60 Hz power line. Another type, a preset interval timer, is set up by the computer and enabled to count. When the count equals a preset value, an interrupt is generated. The third type is a true elapsed-time counter that can be set to zero, then read at any time by the computer.

ROLE OF THE COMPUTER

Data Handling:

Once the inputs are brought into computer memory, the data can be verified, manipulated, formatted, printed out, and displayed in a variety of ways. The arbitrary voltage or frequency readings from the A/D converter can be compared against limits and converted to meaningful engineering units (3.26 millivolts from one sensor may correspond to + 132° F; from another sensor, the same voltage might signify "84 gallons per minute"). Standard data handling programming can be used to sort the data, arrange related data in labeled columns, add the time of reading, and initate print-out by high-speed printer or CRT display. Formatted data can be accumulated by drum or disc mass-storage, then transferred to magnetic tape for later analysis at an off-site data processing center. Through communication links, data can be transferred to remote processors as it is collected. Operators can select blocks of data for display, or adjust conversion factors, by simple keyboard input requests.

Functions that can be performed by the computer in a data acquisition system are summarized below.

Input Data Scanning:

Control the A/D converter, multiplexer, and digital line scanners
Control variable-gain scaling amplifiers preceding the A/D converter
Modify the scanning sequence based upon either the input data, or operator command or both
Maintain time of day without the need for an elaborate clock
Count pulses, measure periods and frequencies of recurrent digital inputs

Data Formatting:

Linearize transducer outputs
Correct transducer outputs for zero offset effects
Convert raw data to physical units
Check converted data against upper and lower limits for out-of-tolerance conditions; alarm monitoring, faulty transducer detection
Perform signal averaging, integration, and filtering
Perform logical checks on the data

Calibrating, Failure Detection:

Calibrate analog-to digital converters by measuring voltage standards
Detect equipment failures in the operating system

Storage, Display, Printout:

Control recording, printout, display devices
Communicate with an operator and present data via printers, X-Y plotters, meters, lights, etc. for best operator comprehension
Maintain data files for an on-demand inspection by an operator
Assemble pre-processed data for visual display, recording, or off-line data reduction
Provide "quick-look" results for processes which are of finite time duration after the run is done
Retransmit data over long lines to a remote computer

Operator Aids:

Even without any hardware control links back to the process, a computer-based data acquisition system provides information that will contribute to increased production or process efficiency.

Overall process monitoring can be assisted by a cathode ray tube display system controlled by the computer. CRT displays can be programmed to select stored data and display it in the best form for quick operator comprehension. The equivalent of many process-flow chart graphic display panels can be stored in computer mass memory, updated as new information is acquired, and displayed when the operator requests.

Oscilloscope-type traces can be presented, prepared from information collected over time scales of seconds, minutes or days. Simultaneous values of many variables can be compared. Points on curves can be selected by light pen and numerical values printed out on demand. Used in this mode, a cathode ray tube display functions like a strip-chart recorder, except that it is not a permanent record of the variables being examined. Magnetic core and tape or drum memory form the more per-

manent record, while the display selects only what the operator needs at the moment. Guided by such displays, a central operator can provide essential supervisory control during critical periods such as startup.

Computation:

The ability of the computer to compute, as well as sort and display information, contributes to effective control. Even when the task is basically data gathering, collected data can be analyzed to relate performance to variations in raw materials, equipment design, and ambient conditions. Mathematical models of the process and the applied control techniques can be verified by experiment, and refined to predict the effect of important variables.

Management Aids:

Management aids are another important by-product of computer-based data acquisition. Used off-line, the computer can process production statistics and perform other relatively routine operations. Management can improve decision making and operational control by establishing a detailed process information system. The computer can provide information for: financial planning, production planning, production scheduling and operational control, and accounting information.

The PDP-12 provides a unique solution to the general problem of using computers in the laboratory. It is unique because it is not a "Bare-bones processor" with various devices tacked on to it, but is rather a complete, integrated computer system. This system consists of a powerful central processor capable of executing two complete order codes (i.e. instruction sets) and:

4096 12-bit words of 1.6 μs Core Memory, expandable to 32,768 words
Direct Memory Access Channel
Hardware Signed Multiply Instruction (9μs)
15 Auto-index Registers
6 Sense Switches
TC12 LINCtape Automatic Control fully buffered, DMA Transfer
1TU56 Dual Magnetic Tape Transport, expandable to 4 Dual Transports
VC12 LINCscope Control and Character Buffer, 2 Intensification Channels, 2 sized characters
VR12 7" x 9" CRT Display
AD12 Analog-to Digital converter and multiplexer, 16 Channels
(8 knobs, 8 phone jacks), expandable to 32 Channels, 10-bit accuracy, Sample and Hold, Differential
Preamplifiers, 50kHz conversion rate
$\pm$ 1 volt input
DR12 6 Programmable SPDT Relays
Data Terminal Panel
ASR33 Teletypewriter, 10 Char/Sec Paper Tape Reader and Punch
12 Sense Line Inputs
30" Freestanding Cabinet
Console Table

Most important for the researcher or laboratory user is the software available to make the computer system useful. Whether this is written by the user or by the computer manufacturer, it should incorporate the same systems approach that is encountered in the hardware system. In the case of the PDP-12 the hardware system supporting the software is a more complete system and consequently the software can have more powerful modules.

For example, one module might be written to sample data, digitize it, store it in core memory, display the stored data, perform some simple computation, transfer the data to tape or some other form of bulk storage and retrieve it at will during subsequent operations. Each of these functions can be treated via an individual "module" in the software and consequently at any point one of the functions may be omitted and the subsequent one substituted for it. For example, if the data coming in is of a known format it may only be necessary to sample, digitize and write the data on magnetic tape. Only after a known quantity of information is present would it be necessary to retrieve the data for examination and comparison, let us say with previously acquired data, for example.

Another example of a modular program might be to sample data, compare it with a previously acquired file, check for acceptability of the acquired data which is subsequently either stored or discarded. The program should provide for transfer to some bulk storage medium, averaging of successive acquired files and similar functions. Again, all these functions would be modular to the extent that after a specific function is performed the user could jump to another, "non-sequential" function if he desired.

In addition to data acquisition and file comparison, programs have been written and are available to carry out extensive processing of acquired signals, such as frequency analysis correlation and convolutions. Both Fast Fourier Transform and a more conventional Fourier Analysis program are available for use on the PDP-12. Auto- and Cross-correlation programs and convolution programs have been written and used extensively.

In most cases the results of such signal processing would be available via the VR12 CRT display or the teletype. Hard copy output might also be achieved using either a digital incremental plotter or an analog recorder.

In the case of the PDP-12, software can easily be written by the end user with the powerful DIAL operating system. This operating system provides a highly interactive and easy method for program preparation. It enables the user to take advantage of the presence of all the PDP-12 system devices, supported at the level of the basic instruction set. These system devices, all available on the PDP-12A systems, encompass all the commonly required "options" required in the industrial laboratory environment.

The analog-to-digital converter, AD12, provided with the PDP-12 is a 16 channel multiplexed A to D converter with sample and hold, overload protection and differential input. The normal input voltage is $\pm$ 1 volt full scale. It is possible to specify other input voltages, provided at

the request of the user. The addition of the AM12/AG12 expands the converter to 32 channels, of which 24 are available for external inputs. It is also possible to further expand the AD12 to a full complement of 128 channels. This is a special feature available by directly contacting PDP-12 engineering.

To accept signals from low level transducers such as thermocouples or strain gauges, it may be necessary for the laboratory user to introduce some means of signal conditioning. This signal conditioning equipment is best provided by the user himself, who is most aware of the characteristics of the equipment which he has in his laboratory.

The relay output on the PDP-12 provides six discrete relay closures capable of carrying 1 amp at 110 volts or 3 amps at 28 volts DC. These relays are supported in software and can be used to turn instruments on and off, raise and lower recorder pins, or set alarm signals in the laboratory. This relay output may also be expanded to provide more closures if required.

There are also 12 sense line inputs available on the PDP-12. These sense line inputs accept logic level signals and can be sensed under program control as a means of determining status of external events.

The digital-to-analog converters which drive the VR12 display are brought out through a connector on the front panel of the PDP-12, providing two analog signals which can be used to drive another display, energize an analog plotter or to perform selected control functions. These voltages are controllable via a single instruction. The DIS (Display a point) instruction.

The PDP-12A system provides a quarter of a million words of bulk storage capacity. This bulk storage, block addressable LINCtape, acts as a linear extension of core memory — a slow disc file, so to speak. The TC12, a fully buffered controller, permits the operation of the tape units in an overlapped mode. It is possible to add an additional 6 tapes units, with a minimum capacity of over 1.6 million words of storage when one uses extended tape addressing.

The VR12, an interactive point plotting display is also a part of the standard PDP-12A system. This display enables the viewing of acquired data and also permits text display. The DIAL operating system takes advantage of this display, for example, to permit text editing and text preparation in a very unique way.

The VR12 is a 7" x 9" scope with a 9-bit resolution in both the horizontal and vertical directions. This display, an integral part of the system, enhances the interactive natural of the PDP-12 and enhances user familiarity with the machine.

In addition to the system devices normally supplied with the PDP-12, many additional peripheral devices are available. Over and above the variety of the analog-to-digital and digital-to-analog converters previously mentioned, it is also possible to add disc files, additional LINCtapes units, industry compatible tape units, line printers, alphanumeric terminals

and various other devices. This provides the flexibility and variety necessary to solve a specific problem.

Of special interest to the laboratory users are two unique new devices specifically developed for this environment. The first is a data break controller for enhancing and simplifying data acquisition. The second, a floating point processor, provides a fully buffered device for floating point computations. This peripheral processor reduces computation time considerably and enables total data reduction to take place within the laboratory itself, rather than relying on the computer center.

The data break controller, described in much more detail elsewhere in this handbook provides the capability of using 12-bit + sign, 14 bit + sign, A to D converters, via four independent sample and hold circuits. It is a data break device and runs in parallel with center processor operation. The utility of this device is greatly enhanced by the availability of applications software, provided at no charge, written specifically for this device. The availability of applications software permits immediate use of the computer within the laboratory environment.

The floating point processor, another new peripheral device for the PDP-12, is fully buffered and shares the PDP-12 memory with the CPU, running in an overlapped mode using cycle stealing. The power of its instruction set is enhanced by the availability of an Assembler to permit user programs to be written quite simply. In addition, several applications programs described elsewhere in this handbook are available for end user convenience.

The PDP-12 with the data break controller and peripheral processor provides the laboratory user the total computer capability that he requires. Analog and digital information is easily acquired from or transmitted to the outside world. Data is easily stored on the LINCtape units provided with this system. Computation and data reduction is facilitated using the extremely powerful floating point processor, Information is easily reviewed using the PDP-12 CRT display. And all these peripheral devices are supported with an extremely flexible and powerful instruction set and operating system.

The Dial programming development system, an outstanding "tool" of the PDP-12, is described in much greater detail elsewhere in this handbook. The human engineering inherent in this system is the result of Dial being polished and perfected over many years.

Loading the operating system is done easily by reading information from the tape blocks through a command set into the console switches. Much use is made of the CRT for displaying text and other alphanumeric information.

In addition to the DIAL operating system. FORTRAN, BASIC and FOCAL may also be used in the PDP-12. Of particular interest is the availability of PS/8, an operating system developed on the PDP-8 but which can be run on the PDP-12. This operating system features device independent operation, and LINCtape may be used as the system device. More information

regarding this system is available in the DEC handbook "Programming Languages".

An extremely valuable tool to the laboratory use is FOCAL-12. The interpretive compiler, described in great detail elsewhere in this handbook, offers ease of programming without sacrificing computer power. The fact that FOCAL-12 can execute LINC instructions as well as the PDP-8 subset means that the LINC devices, such as the A to D converter and display, may be used in FOCAL programming. Thus the interactive characteristics of the PDP-12 are retained and the ease of using FOCAL is enhanced.

If a specific problem requires more specialized and sophisticated software than what can be achieved by modifying applications software provided with the computer or by the user writing his own programs, it is possible to contract with DEC's Software Group and have them write specific, well defined programs for particular needs. If such an approach is thought desireable, DEC is more than willing to assist the end user in writing specifications for such programs.

An example of a simple program to acquire data from an A to D channel, store the data on LINCtape and retrieve the data for subsequent display on the VR12 will illustrate how simply the PDP-12 can be programmed for data acquisition, storage and display.

```
        * 20
        LDA I
        100         /SET AC5 = 1
        ESF         /SET FAST SAM
        SET I 6     /SET NO. OF PTS TO SAM
        -400
        SET I 7     /TABLE LOC. OF DATA
        777
        SAM 10      /SET UP MPX CHAN 10
        SAM 10      /GET DATA PT
        STA I 7     /SAVE DATA PT
        XSK I 6     /400(8) PTS YET?
        JMP. -3     /NO
        WRC
        2750        /WRITE DATA TO BLK 750
DISST,  SET I 7     /RESET TABLE PTR
        777
        SET I 6     /NO OF PTS TO DISP
        -400
        SET I 5     /HOR 12 POS OF FIRST DATA
        177         /PT TO DISPLAY
        LDA I 7     /GET DATA PT
        DIS I 5     /DISPLAY, IT
        XSK I 6     /DONE?
        JMP. -3     /NO
        JMP DISST   /YES DISPLAY AGAIN.
```

All of these functions are initiated via simple Teletype commands and/or responses to Teletype inquiries.

The above two examples show how simple it is to acquire, store and display data. The second example illustrates the ease of using the applications programs available with the PDP-12. The first illustrates the inherent power of the instruction set when the computer is being used for such tasks. In either case, the end results is the same and the objectives of the user are achieved.

If available applications programs do not meet user needs, those which are provided are easily modified using the DIAL operating system. Such modifications provide the user with an extremely powerful and flexible means of programming his computer without having to start from scratch in every case.

Beside data acquisition, storage and display, the ability of the PDP-12 to do extensive signal processing with the addition of the FPP floating point processor makes this truly the complete laboratory system. No longer is it necessary to take pre-processed data to the computer center for further computation. In the event, however, that such further computation is required it is possible to interface the PDP-12 to other DEC machines such as PDP-15's or PDP-10's which are powerful and easy to use computer systems for data processing. Such hierarchies of computers provide the best of both worlds — the total independence of the individual user and the extreme power and flexibility of a large central processor. Whatever your needs are in the terms of computers, DEC has a solution to your problem.

CHAPTER 5
CLINICAL LABORATORY SYSTEMS

INTRODUCTION

Working closely with leading universities and hospitals, DEC recognized the need to develop a computer-based clinical laboratory system for physicians and technologists, already overburdened with an overwhelming amount of data being generated by ever increasing work loads. DEC responded to this need through a corporate commitment to develop a total system for the clinical laboratory based on considerations and suggestions from the users themselves.

The Clinical Laboratory System (Clinical Lab-12) has been developed and proved in phases. The initial work on the data acquisition portion of the system was performed by the University of Wisconsin Clinical Laboratory. A Monitor Control System and a Patient File System have been added to the LABCOM (LABoratory aided by COMputer) System, developed by the University of Wisconsin.

The development of the Clinical Lab-12 System has been a cooperative project between Digital Equipment Corporation, the University of Wisconsin, and several other institutions and hospitals.

FUNCTIONAL DECRIPTION OF THE CLINICAL LAB-12 SYSTEM

Operational Description

Clinical Lab-12 is a real-time, on-line, multi-terminal computer system that provides the clinical laboratory with an economical means of dealing with the problems of data collection, reduction, and analysis. Clinical Lab-12 monitors laboratory instruments, processes and analyzes data from these instruments, provides a summary patient file, and produces easy-to-read reports. The system is run by the existing staff of laboratory technicians using a conversational language (English) while reducing the errors inherent from manually logging data, producing AutoAnalyzer® "spin" sheets, and reporting test results.

All six interactive Teletypes® in the Clinical Laboratory System provide concurrent access (in real-time) to the computer with a special-purpose time-sharing monitor program. Under control of the monitor, the system programs share the same data files while concurrently carrying out the functions of the system. For example, while a technician is entering test requisitions at a given Teletype, another technician at a different Teletype can be simultaneously requesting work sheets; at the same time, the remaining Teletypes can be used for other operations in the system. This time-sharing principle on which Clinical Lab-12 operates enables each remotely located Teletype to function much like a single-user system.

The functions of the clinical laboratory are aided by several Clinical Lab-12 programs. These programs include: Requisition Entry; Work Sheet Genera-

®AutoAnalyzer is a registered trademark of Technicon Corporation.

®Teletype is a registered trademark of Teletype Corporation.

tion; Set-Up Analysis; Accession Number Entry; Summary Print, containing a number of options; Administrative Update; Test Update; and Delete Data.

COMPUTER-AUTOANALYZER INTERACTION

Operating procedures for laboratory equipment at a particular hospital will not have to be radically changed when Clinical Lab-12 is installed. The system was designed within the clinical laboratory environment; therefore, its introduction into a hospital does not result in a traumatic change. On the contrary, Clinical Lab-12 ensures a smooth transition to a computer-assisted system.

Typical Test Processing Sequence

ORDERING (Requisitions for) TESTS

As in most medical institutions, laboratory analyses on a given patient will be ordered at the request of the attending physician. The body fluid sample (usually blood) will be drawn by an authorized hospital staff member (intern, resident, blood bank technician, etc.) and then forwarded to the laboratory with a work requisition indicating the tests to be performed.

REQUISITION ENTRY INTO COMPUTER

At the laboratory test requisition station, several procedures are performed on the sample. Accession (test) numbers are assigned and attached to the test requisition form and the sample. The numbered sample tubes are then ready for centrifuging.

The patient's hospital number, the sample test number, and normal laboratory designations (i.e., BS, BUN, NA, K, CL, CO_2, BILI [RUBIN], AMY [LASE], SGOT, SGPT, LDH, etc.) for desired tests are entered into the computer by means of a Teletype. A card reader can also be used as a means of requisition input to the computer.

Data Collection and Processing

In actual operation the computer monitors the operation of each automated test instrument to collect information from all known (standards and controls) and unknown (patient) samples. On completion and verification of the accuracy of the test run, this information is placed in the patient's file on the disk. If this is the only test result, or the last one needed to complete the patient's file that day, an automatic summary printout will be generated on the line printer for the last *n* number of days (where *n* is the number of days tests were requested on the patient). If automatic printouts are not immediately desired, they can be requested at the end of the day. From the information on the disk, the computer can generate Patient Summary, Billing, and Ward Reports.

Entering Patients to the File

a. The Patient Summary Report is generated by the computer from information stored on the disk and provides the hospital medical staff with a survey of the majority of laboratory investigations on a given patient. Other summary report formats are available, which print the various tests across the page and vertically list

the time and date. This type of summary format can be tailored to many different user demands.

b. The Billing Report enables the laboratory to present the administrative staff with a list of the tests performed and a cost for each with a total on a daily basis. An eight-character code can be substituted for the cost if desired. This is another of the system-build features that create a powerful and flexible Clinical Lab-12 System.

c. The Ward (Interim) Report can be called at any time for the latest status report on all patients having laboratory work performed that day. The report is printed by ward and room number and lists all tests requested that day, their results if complete or comments if incomplete.

Laboratory Programs

The Clinical Lab-12 System uses a number of DEC developed and maintained programs for the clinical laboratory. Each program provides the user with a computerized function corresponding to a work function currently performed by a technician in the noncomputerized laboratory.

The following, easy-to-use programs correspond to laboratory functions and may be called into the computer from any free Teletype. All programs use a conversational computer language (English) designed to assist technologists in communicating with the computer programs.

REQUISITION ENTRY

Information from the doctor's test request form is entered into the computer using the requisition entry routine from any free Teletype or remote display scope.

PATIENT SUMMARY REPORT SAMPLE • DIGITAL EQUIPMENT CORPORATION

PATIENT NAME: BERGMAN, RALPH | PATIENT NO: 344566 | NSG STA: 711 | RM NO: 717-C | PHYS CODE: 67 | DATE: 07/31/69 | TIME: 17184

TEST NAME	TIME COLL	TEST UNITS	07/24/69	07/25/69	07/26/69	07/27/69	07/30/69	07/31/69
ELECTOLYTES SR								
SODIUM		MEQ/LITER				139.		140.
POTASSIUM		MEQ/LITER				4.2		4.8
CHLORIDE		MEQ/LITER				107.		108.
BICARBONATE,SR		MEQ/LITER				22.1		23.5
GLUCOSE	07:00A	MG/100 ML	120.	121.	118.	118.	119.	110.
UREA NITRO		MG/100 ML		47.	46.	45.	42.	33.
BILIRUBIN SR								
TOT BILIRUBIN		MG/100 ML				.45		
DIR BILIRUBIN		MG/100 ML				.45		

		MYEL %	META %	STAB %	SEGS %	LYMP %	MONO %	EO %	BASO %	HGB GMS	HCT %	RBC M/MM	MCV CU.U	MCH UU G	MCHC %
07/27/69	02:14P	0.	1.	2.	67.	25.	2.	3.	0.						
07/26/69	03:00P									10.5	31.	5.3	94.	32.	33.
07/27/69	03:00P									0.3	26.	5.3	96.	31.	37.
07/24/69	08:14A	0.	0.	1.	69.	25.	0.	5.	0.						

Patient Summary Report

WAYNE SALLY G 2862103 3E/341 DATE 03/02

ELECTROLYTE UNIT	15.00
DIFFERENTIAL CNT	7.00
VITAMIN A	13.00
ELECTROLYTE UNIT	15.00
CHEMISTRY SURVEY	17.00
CBC	7.50
DIFFERENTIAL CNT	7.00
TOTAL	81.50

Billing Report

WARD REPORT 03/02/70 12:18 PM

NAME WAYNE SALLY G PAT. # 2682103 N.S./ROOM # 3E/341

ACC #	TECH	TEST	RESULT	TIME
7229	12	GLUCOSE (FASTING)	92 MG/100ML	7:20A
	12	UREA NITROGEN	23 MG/100ML	7:20A
7230		ELECTROLYTE UNIT		7:20A
	12	SODIUM	138 MEQ/L	
	12	POTASSIUM	4.1 MEQ/L	
	12	CHLORIDE	110 MEQ/L	
	12	CO2 CONTENT	22 MEQ/L	
	12	PH	7.1	
	12	PC02	46 MM HG	
	12	OSMOLALITY	24.7 MOSM	
7230		PROTEIN UNIT		9:00A
	12	TOTAL PROTEIN	INCOMPL.	
	12	ALBUMIN	1.9 GM%	
	12	GLOBULIN	INCOMPL.	
	12	A/G	INCOMPL.	
105		CBC		9:00A
	4	HEMOGLOBIN	13.4 MG%	
	4	HEMOCRIT	42 %	
	4	WBC	10.4 M/CU.MM.	
105		DIFFERENTIAL CNT		9:00A
	4	BANDS	6 %	
	4	SEGS	10 %	
	4	LYMPS	26 %	
	4	MONO	6 %	
	4	EOS	2 %	
	4	BASO	3 %	
	4	DESCRIPTION	0 %	
7340	14	GLUCOSE	142 MG/100ML	11:10A
7340	14	CALCIUM	INCOMPL.	11:10A
7340	14	PHOSPHORUS	INCOMPL.	11:10A
7340	14	URIC ACID	INCOMPL.	11:10A

REPORTING FINISHED

TTY IS FREE

Ward (Interim) Report

WORK SHEET GENERATOR

The technician selects this program and requests that work sheets be generated and printed. The work sheets are generated from the test requisition data and are typed on the requesting Teletype or on the line printer. The work sheets list the work to be done during the day, categorized by test type.

SET-UP ANALYSIS

This program is called to inform the computer of the type of tests, number of cups to be run, and the Teletype to be used for each printout. After the computer has been informed of the above parameters, the technologist is free to start the data acquisition process when she is ready. The computer will be waiting to sense any channels that have been set up previous to the start of any data acquisition.

ACCESSION NUMBER ENTRY

With conversational communication through a Teletype, this program enables positive sample identification by cross-checking the accession number of the test sample against the patient's name. At this time, the technologist can accept, reject, or modify the test results, in accordance with the prescribed laboratory procedures. The test results must be verified and accepted by the technologist before this program stores the test results in the patient's file. This program validates the test results collected during the data acquisition process.

SUMMARY PRINT

This program can be called from any Teletype to carry out any of the following functions:

a. Print a *Patient Summary Report* on the line printer.

b. Print a *Summary Report* for every patient in the file on the line printer. (Reports are sequenced by patient number.)

c. Print an *End-of-Day Report* — a line printer listing, containing a summary report of all patients with incomplete tests. (Reports are sequenced by patient number.)

d. Print an *Inquiry Report* — a Teletype listing, containing the requested tests for a particular patient for a specified day.

e. Print a *Ward Interim Report* — a line printer listing, containing tests for the current day for all patients in the file. (Reports are sequenced by nursing station and room number.)

f. Print a *Billing Report* — a line printer listing containing all the requested tests with the charge for each test. (Billing reports are sequenced by patient number.)

In addition to generating the above reports on request, the Summary Print Program automatically generates a Summary Report (the same as a. above) when, on a given day, all the requested tests for a patient have been completed. If this automatic summary print method is not desired,

it can be suppressed and the summary reports will be printed at any time during the day when requested by the laboratory personnel.

ADMINISTRATIVE UPDATE

As patients are admitted to the hospital each day, this program is called to add patients to the file. This program is also used to modify administrative data (e.g., room number change, etc.) in a patient's file. These functions are accomplished from a Teletype or Cathode Ray Tube (CRT) using conversational language to communicate to the program the necessary changes to update the patient file.

For normal test requisitions, the system must have the patient's name, room number, and other pertinent information on file. For emergency cases, provisions are made within the program to request laboratory tests without this information.

TEST UPDATE

Test results from instruments not interfaced to the computer are entered manually. This is accomplished by calling the Test Update Program and manually typing the test results into the computer via the Teletype or display scope. This program is also used to examine and modify test results already stored in the patient file.

DELETE DATA

The Delete Data Program is used to remove patient data from the active patient file on the disk when the patient is released from the hospital. This program is also used to delete test data from a patient's file without deleting his administrative data.

CLINICAL LAB-12 SOFTWARE

System Software

Clinical Lab-12 software consists of three major segments:

a. Monitor System

b. On-Line Data Acquisition System

c. Patient File System

Extreme care has been taken to build an error-detection/recovery capability into the programs. This dual capability is accomplished primarily by editing features that give the user an easy-to-use format to correct errors by responding YES when the program requests changes via the Teletype. In addition, various routines in the programs conduct certain error detection tests. If the program determines that the information entered is improperly formatted, the computer notifies the user by an appropriate error message.

MONITOR SYSTEM

Monitor System software coordinates the operation of the On-Line Data Acquisition System and the Patient File System. Events in the Monitor System are controlled by monitor system software, a program residing in core. Monitor software controls all interrupts in the system, processes all input/output requests from peripheral devices, and allows multiple-user access to the system almost simultaneously by means of a simple round-robin queue schedule.

ON-LINE DATA ACQUISITION SYSTEM

The On-Line Data Acquisition System gathers data from automated laboratory instruments and sets up and maintains a file on the disk for each instrument channel (up to 24) and all the test information in that channel. Digital inputs are handled in the same manner.

LABMON (LABoratory MONitor), the primary program for this system, scans the test instrument channels and determines the status of each channel at 1 second intervals.

When the core buffers are filled with data from the laboratory test instrumentation, LABMON requests the Auto-Sort Program to perform the necessary calculations and to transfer the results to the Disk Memory System. After verification of each test result in the channel storage area, results are placed in the patient file and are available for report generation, updating, deletion, etc., and can be retrieved on demand at any laboratory terminal.

PATIENT FILE SYSTEM

The Patient File System establishes and maintains a central patient file. A variety of reports are generated by the system, either automatically or when requested by the user.

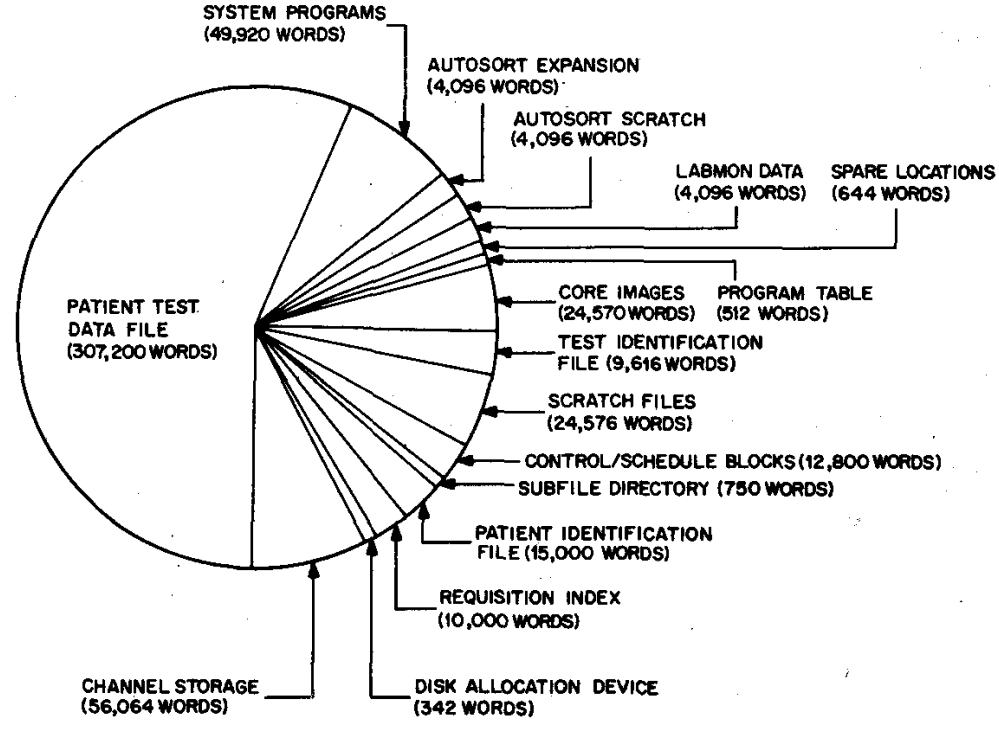
The programs in the Patient File System generally are operated under the user's direction at one of the six Teletypes/CRT terminals that can be used in the system; however, the Summary Print Program is activated by the Monitor System when all of the tests requested for a particular patient, on a given day, have been completed. The capability for 12 independent user Teletypes/CRT terminals will be available by November 1970.

Additional Systems

Two other unique components of the Clinical Lab-12 software are the Disk Storage System and the System Build Facility.

DISK STORAGE SYSTEM

The Disk Storage System provides random access storage for data and programs. Programs stored on the disk are immediately available when needed to perform system and user-requested functions. Minimum disk storage capacity is 524,288 12-bit words.



Disk Allocation

SYSTEM BUILD FACILITY

The System Build Facility of Clinical Lab-12 enables non-programmer personnel to tailor the system to the needs of a particular clinical laboratory. Full (English) test names, abbreviated test names, and costs can be defined for an individual test or groups of tests. In addition, test units and various English comments can be defined to identify individual test results. When changes are required, the data for both the individual tests and groups of tests can be updated, off-line, by non-programmer personnel using modify, add, or delete procedures provided by the System Build Facility. Another feature of this facility is the amount of added control that each laboratory has over test order on summary printouts.

Clinical Lab-12 Programs

I. MONITOR SUBSYSTEM
   - A. Line Printer I/O Handler
   - B. Magnetic Tape I/O Handler
   - C. Disk I/O Handler (File Oriented)
   - D. Teletype I/O Handler
   - E. Interrupt Processing
   - F. Job and Queue Control
   - G. Program Loading

II. STARTUP

III. SYSTEM BUILDER

IV. PATIENT FILE SYSTEM

A. Administrative Update (AD)
B. Requisition Entry (RE)
C. Work Sheet Generator (WO)
D. Master Work Sheet (MA)
E. Accession Number Entry (AC)
F. Cleanup Routine (CL)
G. Update Test Results (TE)
H. Patient File System Update — Patient Summary Printouts (SU)
I. System Inquiry Program (IN)
J. Delete Data (DE)
K. Billing Routine (SU,B)
L. Control/Schedule Block Generator (CS)
M. Daily Test Census (DA)

V. ON-LINE SUBSYSTEM

A. Set-up Analysis (SE)
B. Laboratory Monitor (LABMON)
C. Real-Time Result Calculations
D. Diagnostic Printouts
E. Calculations (Manual)
F. Display Channels (DI)

Central Processing Unit (CPU)

The CPU, a PDP-12 with an 8192-word core memory, performs the mathematical calculations, data formatting, and other processing functions of the Clinical Lab-12. It also controls the operation of the peripheral equipment in the system.

The PDP-12 has two distinct operating modes within its single processor. Each of these modes has its own instruction set. In addition, the system has flexible I/O capability that enables numerous peripheral devices to be easily interfaced with it.

Clinical Lab-12 CPU features:

- 8192-word, 12-bit core memory with 1.6-μs cycle time
- Full power processor prewired for the addition of a large number of options and peripherals
- Low-cost core memory expansion to 32,768 words, low-cost mass storage with DECdisks, and IBM — compatible synchronous and incremental magnetic tape
- Single- and three-cycle direct memory data break facilities, standard
- All active registers continually displayed
- Signed multiple instruction, standard
- Fifteen auto-index registers, standard
- LINC feature to facilitate multiple-precision arithmetic
- Two's and one's complement arithmetic
- 24-bit console switch register
- Six sense switches
- Complete with 30-inch free-standing cabinet, console table, Model ASR-35 teleprinter including paper-tape reader and punch

- Twelve digital-sense line inputs, standard
- Six single-pole, double-throw relay outputs
- A/D converter
- Two LINC tapes, standard
- Oscilloscope for conversational language

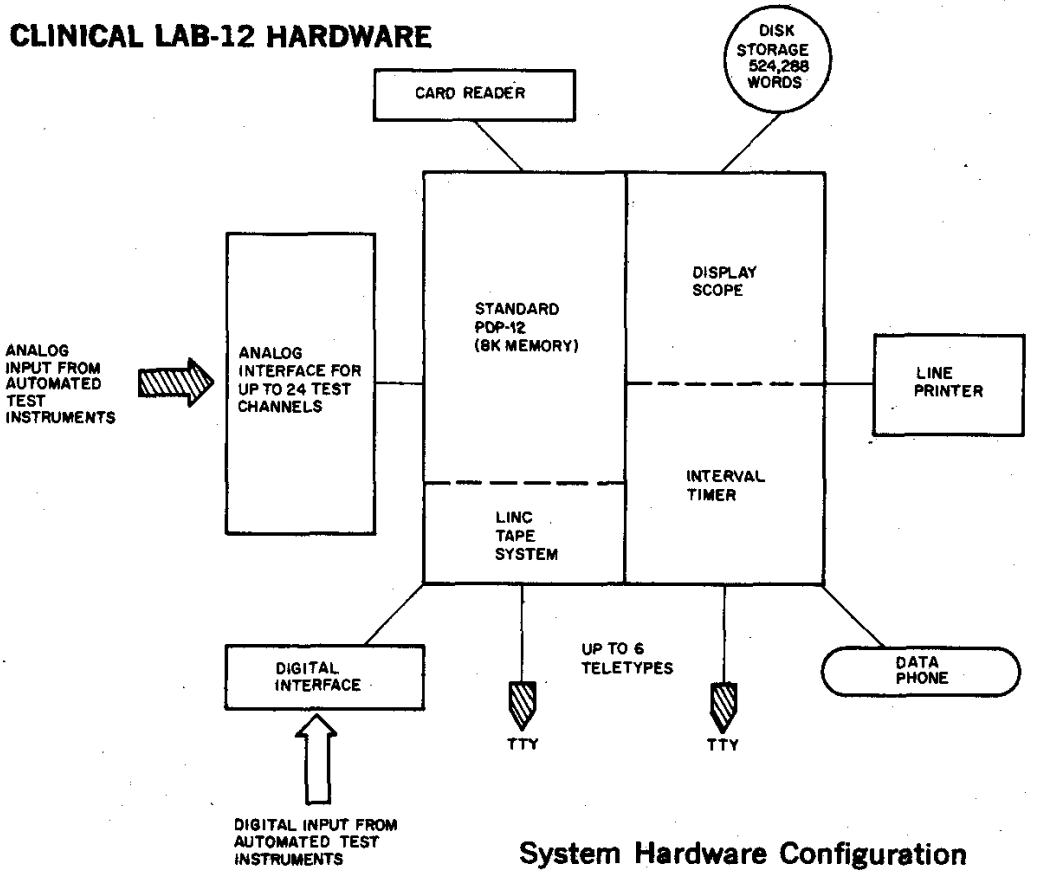
Disk Memory System

The Clinical Lab-12 Disk Memory System provides rapid, economical, random-access bulk storage for the system files and programs. Random-access bulk storage enables the Clinical Lab-12 to react rapidly to the needs of data acquisition and retrieval. Proper use of a disk allows the user to collect data, enter manual input, and obtain printouts simultaneously, thereby avoiding the long delays that are encountered in a tape-based system. In operation, the Disk Memory System serves as an extension of the computer's core memory.

Basically, the Disk Memory System consists of one RF08 Control Unit and two RS08 Disks. This combination provides 524,288 words of storage. Storage capacity of the system can be expanded by adding disks to the control unit. The maximum capacity of the system is 1,048,576 (12-bit) words. The capacity is adequate to handle 1750 patients, assuming an average of 80 laboratory tests per patient stay.

The RF08 Disk System is used for "swapping" when information must be accessed in a very short time. In the event of additional requirements for mass storage, DEC can provide disk packs with a storage capacity in excess of 10 million words.

CLINICAL LAB-12 HARDWARE



System Hardware Configuration

Cathode Ray Tube (CRT) Display System

The Cathode Ray Tube (CRT) Display System allows the user to interact with the system by responding to queries displayed on the screen. Use of a conversational language eliminates the need for the technologist to acquire programming experience to operate the system. A very bright phosphor is used to allow viewing under normal ambient lighting conditions.

Linc Tape System

The two main functions of the LINC Tape System are off-line storage and initial loading of the programs into the system.
The basic LINC Tape System consists of two Transports, controlled by a fully buffered tape control. A single ten-channel tape head serves for reading and writing. Information is redundantly recorded in two nonadjacent channels. The redundant recording feature of the LINC Tape System virtually eliminates the possibility of information being lost.

Teletype Units

Up to six Teletype units (generally, one 35 ASR and five KSR), placed in strategic locations throughout the laboratory for concurrent communication with the central processor, can be used in Clinical Lab-12. (Expansion of this capability is underway.) Remote CRTs can be used in place of Teletypes.

CRT Display Terminal

This remote display terminal has its own memory buffer and is Teletype compatible. It can be used for many functions in the laboratory, especially for inquiring about the status of a particular patient, admission, or requisition entry.

The speed of this terminal is variable from 10 characters per second up to 120 characters per second in the Clinical Lab-12 System. The CRT Display Terminal can also be easily interfaced with the normal communication circuits for long distance transmission.

Line Printer

A line printer is the output device for the ward reports, billing reports, summary reports, and others generated by Clinical Lab-12. The increased speed of the line printer (up to 300 lines per minute, up to 132 characters per line) ensures the timely generation of reports by nursing, medical, and administrative personnel.

Another helpful feature of the line printer is that the 132 column line width of the printer allows for up to seven days of data to be printed on a single summary report sheet. This provides the physician with a convenient record for easy review of information.

Card Reader and Control

A card reader is optional in the system and can be used as an input device to request tests; read cards with punched hole or mark sense information from nonautomated test stations; and perform a variety of functions normally performed by a Teletype (TTY) or oscilloscope.

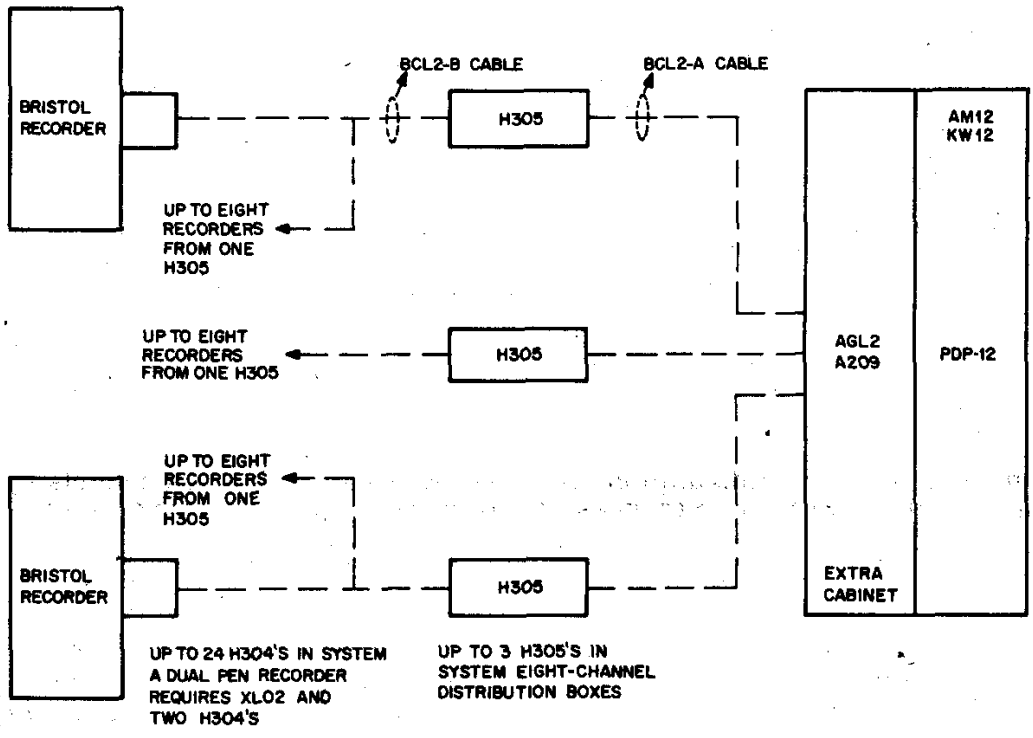
The CR12 Card Reader and Control Unit is presently offered. This unit reads 12-row, 80-column punched cards at a nominal rate of 200 cards per minute. Cards are read by column, beginning with column 1. Data in a card column is photo-electrically sensed.

In addition to the CR12, DEC is planning to incorporate a Mark/Sense Card Reader in the system. This unit will be capable of reading pencil check marks on preprinted request forms filled out by hospital personnel. This capability eliminates the intermediate step of copying or punching and, thereby reduces the possibility of errors and enhances input efficiency.

Laboratory Instrument Analog Interface

The PDP-12 Computer has provisions for eight active analog channels in the minimum configuration. This can be expanded to 24 channels by adding the AM12 and A12 Expander Unit. Typically, an SMA12 requires one channel, a single-channel analyzer (one) and a dual-channel analyzer (two), etc. Clinical Laboratory interface information consists of the following:

- AGL2 — This is the special 1943 Panel that has power supply cable connectors and is prewired to accommodate up to 24 preamplifiers that handle the 22-wire cable from an H305.
- H304 — Terminal Box — This is the small module that attaches with nylon fasteners to the side of the Bristol Recorder. The H304 contains switches, potentiometers, cable connectors, etc. One side attaches directly to the potentiometer on the shaft of the recorder, the other end goes to the eight-channel distribution box.
- H305 — Eight-Channel Distribution Box — This box can be located strategically in various sections of the laboratory to cut down on cabling costs and to give more flexibility to the system.
- BCL2-A — Eight-Channel Trunk Cable — This 22-wire cable is used to connect from AGL2 (in computer) to the H305 (8-channel distribution box).
- BCL2-B — Single-Channel Data Cable — This 3-wire cable connects H304 (on side of Bristol Recorder) to H305 (8-channel distribution box).
- XL01 — Bristol Recorder Interface Kit — This single-channel kit contains parts, wires, brackets, etc., to connect to shaft of the Bristol Recorder and electrical connection to the H304. (This is part of the H304 and is not ordered separately.)
- H307 — Pulsed Output Converter — This converter is required for SMA12 series, in addition to H304.
- A209 — Analog Pre-amp (A202 with 0 + 2V input) — this is a modification of the A202 to allow input signals between 0 and +2V.



Clinical Lab Interface

Laboratory Instrument Digital Interface

DEC will make available a general digital interface for many of the existing laboratory instruments that output digital signals instead of analog signals. At present there is an interface for the Robot Chemist and the Coulter "S" Counter. Depending on the methods employed in the laboratory, the existing programs can be used with the Coulter "S". Other digital input routines are in the process of being developed.

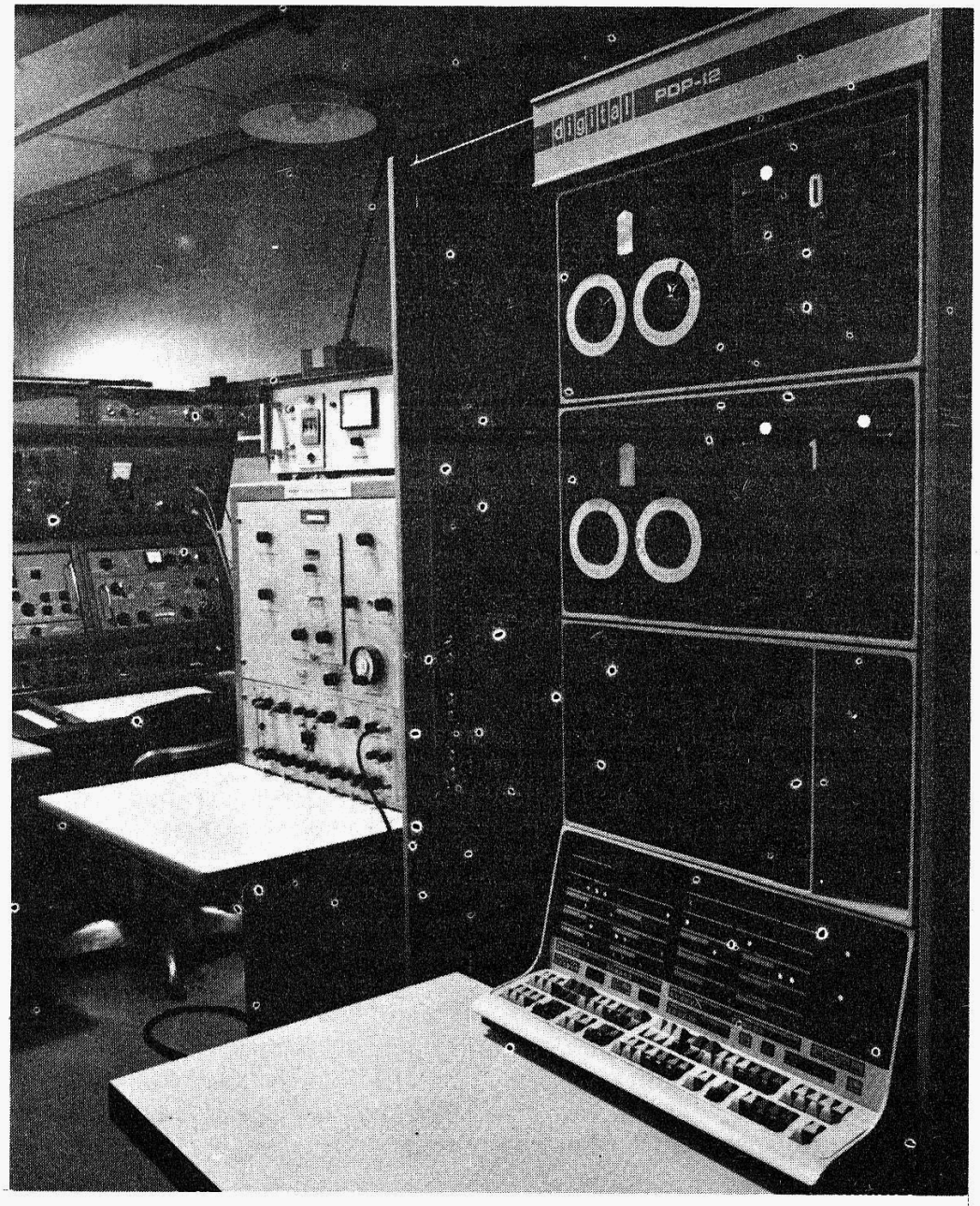
Manual Entry Console

DEC will make available a manual entry console with two-way communication to the computer for laboratory use. The objectives are compactness, flexibility, modularity, and low cost. The manual entry console will be under program control from the computer.

Communication Systems

Communications between two computers, or between a computer and a remote terminal, are normally handled by a Dataphone® in Clinical Lab-12. This Dataphone provides a serial data transmission over standard voicegrade telephone lines. The Dataphone interface is provided by DEC in the form of the DP12, while the modem itself is supplied by the telephone company.

®Dataphone is a registered trademark of the A. T. & T. Company.

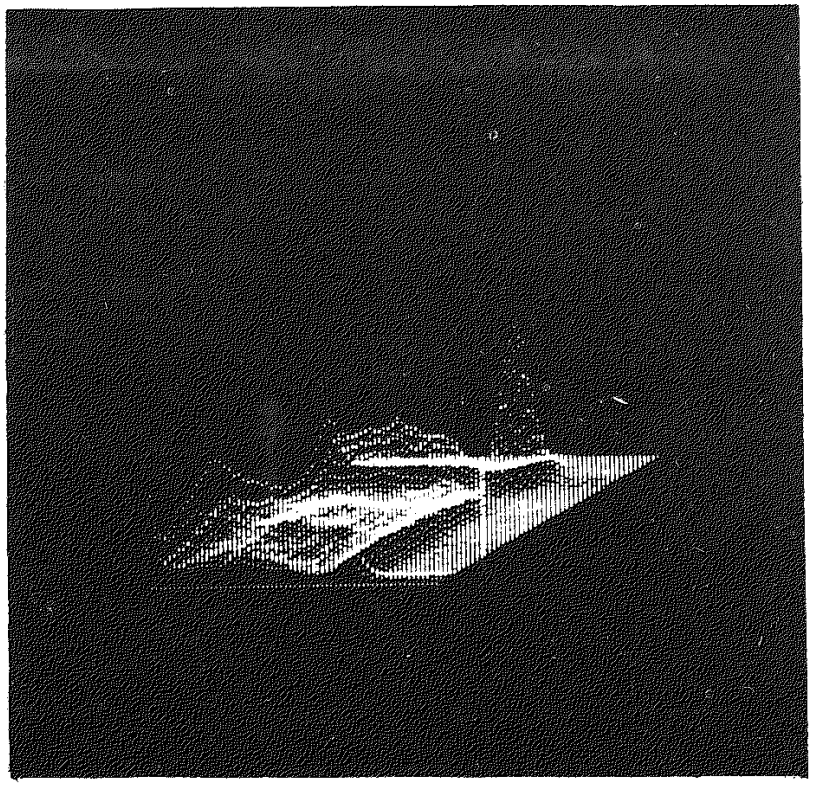


Major applications of the PDP-12 are in biomedicine, psychology, chemistry, and physics research. In institutional and industrial laboratories, the PDP-12 performs such functions as experiment control, data acquisition, and data manipulation.

CHAPTER 6
PHYSICS APPLICATIONS

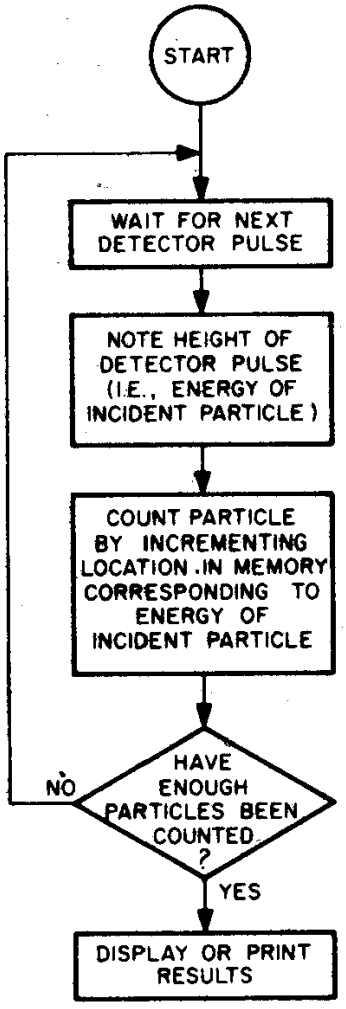
A BASIC PROGRAM FOR PULSE HEIGHT ANALYSIS

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.



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. 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.



Basic Algorithm for Pulse Height Analysis Experiment.

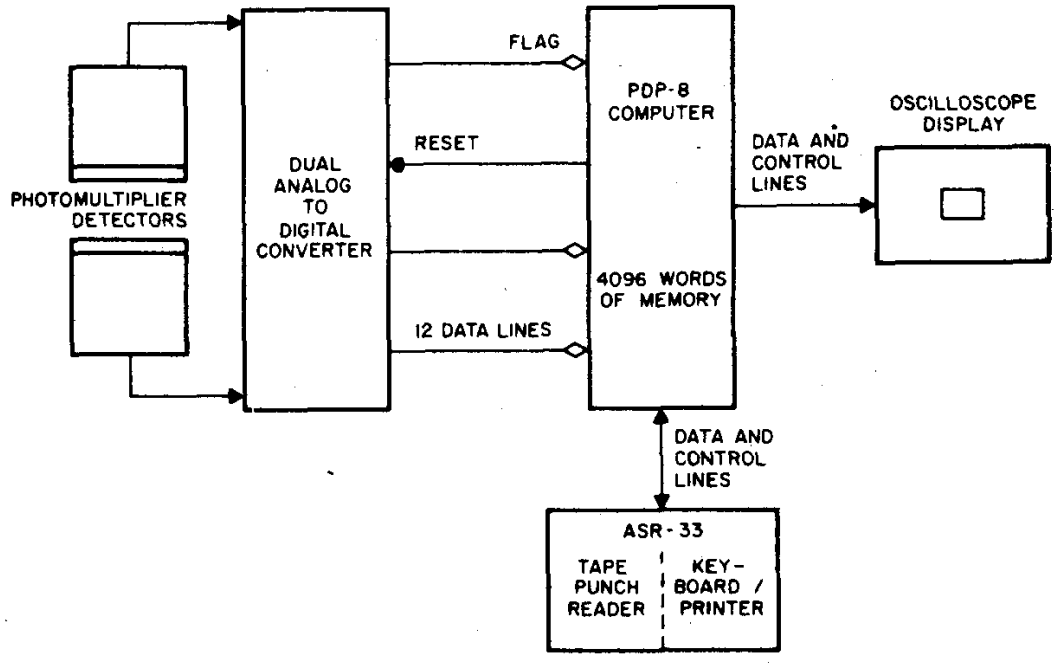
The algorithm shown above becomes somewhat unsatisfactory when we consider that we are requiring computers capable of some 30,000 to 300,000 instructions a minute to wait for a detector that outputs pulses at irregular intervals. Could not this wasted time be put to better use? Say, to produce a continuous, 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.

The figure below 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.

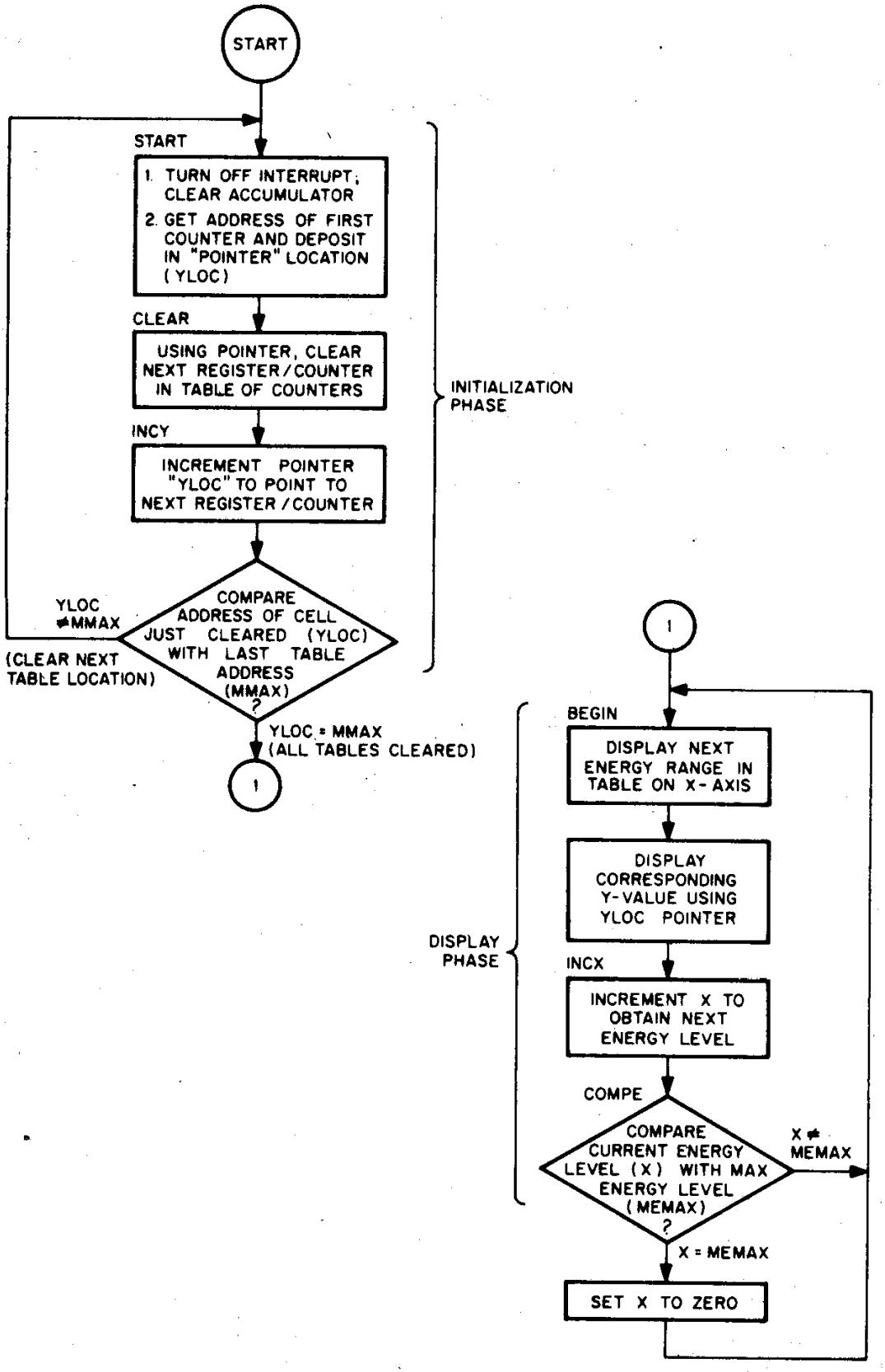


Block Diagram of Pulse Height Analysis Experiment.

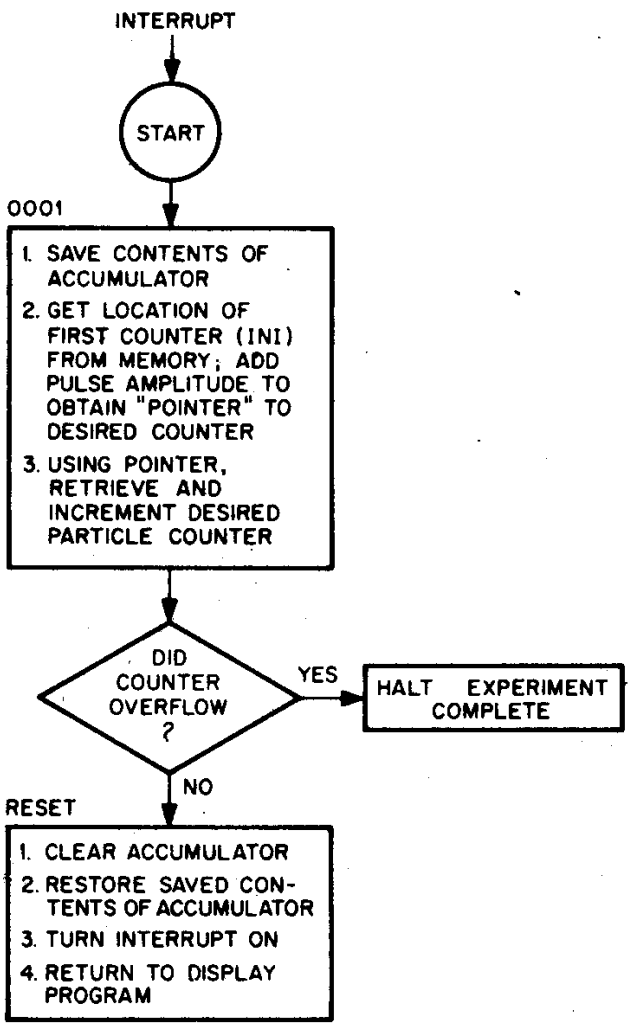
CONSTRUCTING THE DETAILED ALGORITHM

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 below.

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.



Display Routine.



Interrupt Routine.

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, procéeding from the lowest selected energy level to the highest in a continuous, interruptible loop.

The task of the interrupt routine 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 below. Labels appended to boxes on the flow charts correspond to symbolic location names within the program to simplify the task of keying the flow of charts to the program.

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.
          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 O        /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
         TAD INI       /COMPUTE Y LOCATION FOR INDIRECT
                        ADDRESSING.
         DCA YLOC
         TAD I YLOC    /READ # OF PARTICLES.
         LINC          /GO TO LINC MODE.
         L MODE
         DIS I X       /DISPLAY AND INCREMENT X.
         PDP           /GO TO 8-MODE.
         P MODE

/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)
```

PHA DISPLAY ROUTINE

```
BEGIN,   CLA
         TAD INI       /Compute Y location for indirect addressing
         DCA YLOC
         TAD I YLOC    /Read # of particles
         L MODE
         LINC          /Go to LINC mode
         DIS I X       /Display and increment X
         PDP           /Go to 8-mode
         P MODE
```

PHA-12

Digital Equipment Corporation presently offers prepackaged PDP-12 computer systems with standard applications software to perform the task

of Pulse Height Analysis. These systems are designated PHA-12 (4K) and PHA-12 (8K). They combine a Pulse Height Analyzer with the balance and flexibility of an interactive, general purpose computer system. With the sophisticated physics software routines supplied by DEC, PHA-12 systems will accumulate, store, display, and analyze energy spectra and record the results of a variety of applicable devices.

PHA-12 (8K) ANALYZER CHARACTERISTICS:

In the PHA-12 (8K) system, data collection is via AC mode through the NKO4-A interface in either single or dual parameter. The PDP-12 (8K) used in this system has 8K of core memory and a 1.6 microsecond memory cycle time.

Four PHA-12 (8K) programs are presently available:

1. A single parameter program which stores data in one 4096-channel data region. Up to 262,144 counts per channel are allowed via a one and one half precision storage routine. The program will display successive 1024-channel subsets of the entire data region.
2. A dual parameter program which stores spectra in a 64 x 64 channel data region with up to 262,144 counts per channel.
3. An off-line peak location and listing program.
4. An off-line program to output binary data onto industry-compatible incremental magnetic tape.

Programs 1, 2, and 3 will write out spectra on paper tape or LINCtape. In addition, programs 1 and 2 outlined for PHA-12 (4K) are fully compatible with a PHA-12 (8K) system and is furnished as standard applications software.

PHA-12 (4K) ANALYZER CHARACTERISTICS

With the PHA-12 (4K), data may be accumulated in single or dual parameter through the NK04-A nuclear interface. Data collection proceeds in AC mode directly from the ADC's to the PDP-12's accumulator. The PHA software then stores the data directly in core memory. The PDP-12 used in this system has 4K of memory, a fast memory cycle time of 1.6 microseconds, and also includes extended arithmetic element (EAE) for fast multiply and divide operations.

The PHA software will write out spectra on either papertape or LINCtape. The following six programs are provided:

1. A single parameter program which stores data in single precision in one of two 1024-channel data regions. Either of the two regions may be displayed, and one region's spectrum may be subtracted from the other spectrum. Up to 4096 counts per channel are accepted.
2. A single parameter program which stores data in a single 1024-channel data region. The other 1024 word memory space is

used for storing counts in double precision, permitting up to 16,777,216 counts per channel.

3. A single parameter program which stores data in one of two 1024-channel data regions. Either of the two regions may be displayed, and one region's spectrum may be subtracted from the other spectrum. Up to 1,098,578 counts per channel are accepted in square root storage mode. The square root routine is a statistical storage technique which can reflect a very large number of counts which avoiding the core memory space requirements of double precision.

4. A dual parameter program which stores spectra in a 64 x 64 channel data region with up to 4096 counts per channel.

5. A dual parameter program which stores spectra in a 64 x 44 channel data region with up to 66,536 counts per channel in square root storage mode.

6. An off-line peak location and listing program.

Although the extended arithmetic element is standard with the PHA-12 systems, you may purchase a PHA-12 configuration without EAE, in which case Programs 3 and 5 utilizing the square root storage mode would be inoperable.

PHA-12 (8K) & (4K) PULSE HEIGHT ANALYSIS PROGRAMS

Single Parameter:

DATA TAKING

1. Enable data taking.
2. Disable data taking.
3. Set live time clock at console.

DISPLAY

1. Display (a given) data region with specified lower and upper markers.
2. Expand the display between the markers.
3. Set full scale of display with console switches.

MANIPULATION

1. Zero (a given) data region.
2. Integrate data between markers.
3. Subtract one data region from the other and store in another specified region.

DATA INPUT/OUTPUT

1. Write out channels between the markers on TTY.
2. Punch out channels between the markers in binary.
3. Utilize DECtape as a data storage medium with ability to write files.

Dual Parameter:

DATA TAKING

1. Enable data taking from both ADC's.
2. Disable data taking from both ADC's.
3. Set live time clock at console.

DISPLAY

1. Display twinkle box.
2. Display isometric matrix.
3. Display differential contours.
4. Modify contour levels.
5. Set and modify horizontal and vertical markers.
6. Display horizontal and vertical slices.
7. Reverse axis.
8. Set full scale of display.

MANIPULATION

1. Zero data region.

DATA INPUT/OUTPUT

1. Punch data area in binary.
2. Write out slices of data area on TTY.
3. Utilize DECtape as a data storage medium with ability to write files.

ANALYZER CHARACTERISTICS				MINIMUM CONFIGURATION			
PHA System	ADC Inputs	Data Channels	Counts per Channel	PDP Computer	Computer Memory Size (words)	PHA Interface (model)	Oscilloscope Display (model)
PHA-12 (4K)	Single	2048	1,098,576	PDP-12A	4096	NK04-A	VR-12
	Dual	50 x 64	65,536				
PHA-12 (8K)	Single	4096	1,098,576	PDP-12A	8192	NK04-A	VR-12
	Dual	64 x 64	65,536				

CHAPTER 7
REAL TIME CLOCK

The family of KW12 real time clocks consists of the Real Time Interface, (type KW12-A, and two Fixed Interval Clocks, (KW12-B and KW12-C. These clocks are designed to cover a wide range of programming and research requirements by filling a need for differing degrees of flexibility.

The KW12-A, for instance, can be used to:

1. Synchronize actions taken within a computer program with external events;

2. Measure time intervals between events;

3. Measure intervals between an initial event and responses to it;

4. Count external events and/or

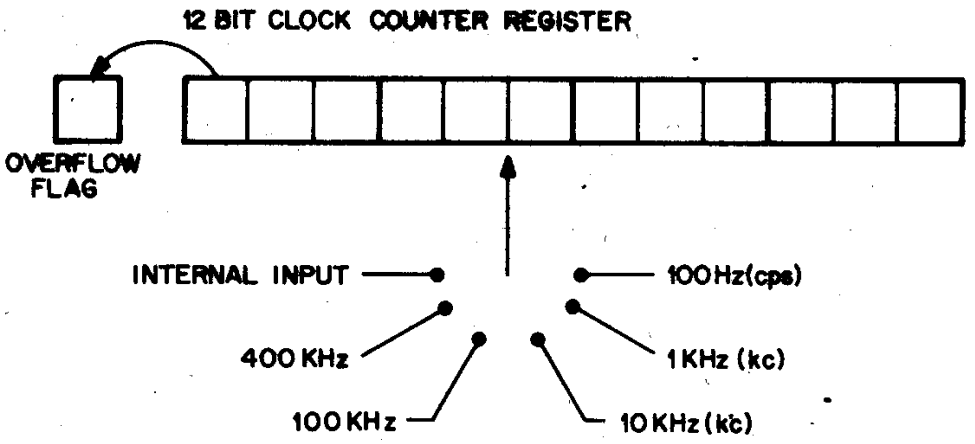
5. Provide program interrupts at program selectable intervals from 2.5 μsec to over 40 sec by using count pulses provided by a 400kHz crystal clock at any of five frequencies from 100Hz to 400kHz.

The KW12-B also provides a means of having the clock cause program interrupts. The intervals at which it interrupts, rather than being program selectable, are determined by an RC oscillator whose frequency may be varied within the range from 175Hz to 50kHz by physically resetting the desired frequency using an oscilloscope and changing one inter-connection if necessary.

The KW12-C, like the KW12-B, also provides program interrupt capability but uses a single fixed frequency crystal oscillator in place of the KW12-B's variable RC oscillator to provide the time base for determining the intervals at which it can interrupt. The frequencies available are in the range of 5kHz to 50kHz and must be specified in advance by the customer.

KW12-A REAL TIME INTERFACE

The KW12-A is a highly flexible pre-wired PDP-12 option, featuring an Input Control Panel, a 400kHz crystal clock and five 12-bit registers for controlling/monitoring the operation of the clock. The five program selectable count pulse rates (i.e. time bases) derived from the 400kHz crystal clock are: 400kHz, 100kHz, 10kHz, 1kHz, and 100Hz. Each of the count pulses increment one of the KW12-A's 12-bit registers, the Clock Counter Register. This same counter register can also be incremented by an external signal. (See discussion of Input Control Panel)

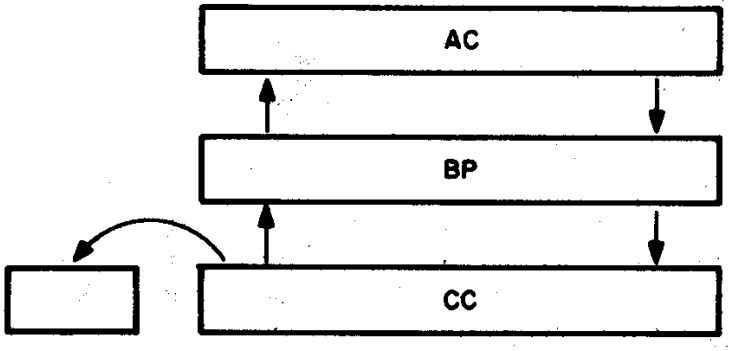


CLOCK COUNTER REGISTER

The Clock Counter Register is one of the five 12-bit registers in the KW12-A which are accessible to the program. Each generated pulse causes the Clock Counter to be incremented by 1. The counter increments up to 7777_8 and then "overflows" on the next pulse, causing the "overflow flip flop" or flag to be set to 1. The overflow flag being set is program detectable and can be sensed by a program interrupt, a skip instruction and/or a read status instruction. The contents of the clock counter at any given point may be determined by an 8-mode IOT (CLCA) which puts the contents of the Clock Counter into the AC via the Buffer-Preset Register. The contents of the Clock Counter Register may also be set by the user via the Buffer-Preset Register and the Clock Control Register.

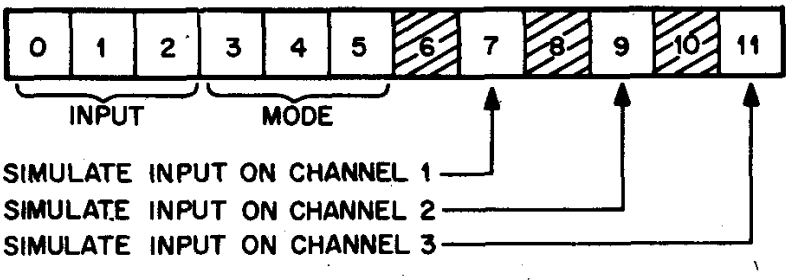
BUFFER-PRESET REGISTER

The 12-bit Buffer-Preset is the link between the processor accumulator, AC, and the clock counter. It is used, in executing the CLCA IOT, to read the contents of the Clock Counter Register into the AC. It is also used to buffer the current count in the clock counter at the occurrence of each event when the clock is running in mode 2 or 6. This count is then made available to the program by the 8-mode IOT CLBA which reads the Buffer-Preset Register into the AC. Another use is made of this register in holding the number to be transferred into the Clock Counter each time the counter overflows. The number is placed in the Buffer-Preset Register from the AC by the 8-mode IOT CLAB. This allows the counter to be set to some desired starting value from which it can count up to overflow thus giving the program greater control over "how long" it will take to cause overflow, i.e., how many counts or pulses are needed.



CLOCK CONTROL REGISTER

The Clock Control Register is the 12-bit register used to determine the "what, when and how" of the clock.



Clock Control Register

The counting rate and mode of operation are specified by settings of the Clock Control Register. Simulated inputs (generally used for diagnostic programming) can also be indicated with this Register. The Clock Control Register is loaded from the AC by issuing the 8-mode IOT CLLR.

Rate Selection

Bits 0-2 of the clock control register can be considered the "count rate register" which is used to indicate one of the various counting rates.

CONTENTS OF BITS 0-2	FREQUENCY OF COUNT PULSES
000	Stop
001	400kHz
010	100kHz
011	10kHz
100	1kHz
010	100HZ
110	External event: (i.e. Use pulses generated on Input Channel 1 to increment Clock Counter)
111	Stop (i.e. no pulses)

NOTE

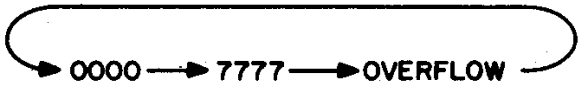
When the events occurring on Input Channel 1 (see Input Control Panel) are used as the time base for the counter, the Channel 1 Event flag (see discussion of Clock Status Register) is automatically cleared and Channel 1 Interupt Enable would generally be left off (i.e., bit 6 of Clock Enable Register set to 0).

Also note, the clock counter is incremented one count each time an I/O Preset is performed whether manually or under program control.

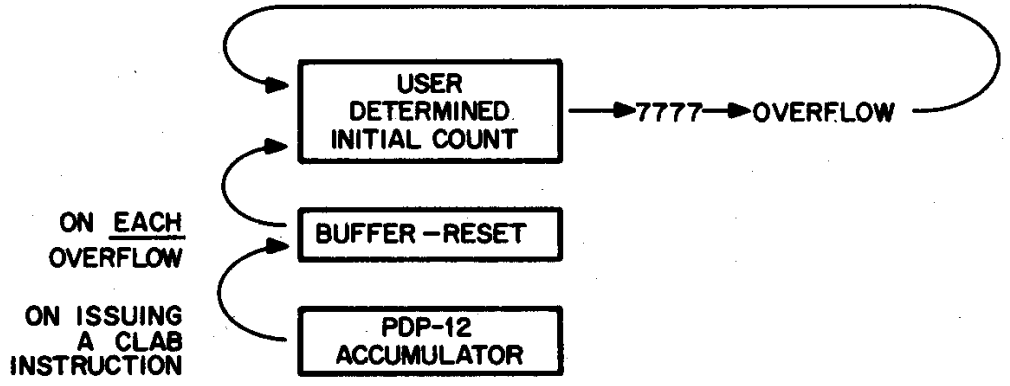
Mode Selection

Bits 3-5 of the Clock Control Register can be considered the "Mode Control Register" which is used to determine the method by which the clock operates.

CONTENTS OF BITS 3-5	MODE OF CLOCK OPERATION
000	Free Run — the counter is incremented at the rate indicated in bits 0-2 of the Control Register. Counting goes from 0 to 7777_8 and then overflows and starts counting from 0 again. Overflow, therefore, occurs every 4096_{10} counts (or every '4096 x counting rate' cycles.) The overflow flag remains set until the 8-mode IOT CLSA (6135) is issued causing it to be cleared.



001	Present Time — as in mode 0, the counter is incremented at the rate indicated in bits 0-2. Each time overflow occurs, however, the contents of the Buffer-Preset Register are transferred automatically to the Counter which then continues counting up from that value. The Buffer-Preset Register is usually set to the negative (2's complement) value of the number of counts desired before overflow. In this mode, the user has not only determined the rate of counting but also the number of counts before overflow, thus allowing him two dimensions in selecting the time intervals between overflow. In this mode, as in mode 0, the overflow remains set until a CLSA is issued. When the mode is changed from 000 to 001, (or 100 to 101, see below), the clock counter is zeroed.



NOTE

In order to avoid having the first overflow occur at an undertermined time, the following procedure should be used to start things off:

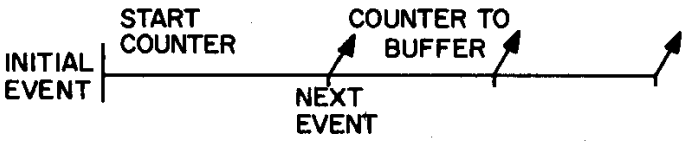
```
CLA          /Clear the accumulator.
CLAB         /Load the Buffet-Preset Register with 0 (i.e., clear it)
CLLR         /Load the Clock Control Register with Rate=Stop (000)
               and Mode=0
TAD K0100    /Pick up constant=0100
CLLR         /Load Clock Control Register with Rate=Stop and Mode=
               1. This will cause the clock counter to be cleared (i.e.,
               set to 0000). If the counter happened to be 7777 when
               the mode was changed from 0 to 1, clearing it would
               cause overflow. The Buffer-Preset Register was cleared
               above since in mode 1 overflow causes the Buffer-Preset
               Register to go to the clock counter.
CLSA         /Clear Clock Status (of which the overflow flag is a part)
               since overflow may have been caused by clearing the
               counter.
CLA          /Clear the AC and
TAD NUM      /Pick up the negative 2's complement of the number of
               counts desired before overflow.
CLAB         /Load the Buffer-Preset Register with that number.
CLA          /Clear the AC and
TAD K0200    /Pick up constant=0200
CLLR         /Load Clock Control with Rate=100Hz (101) and Mode=
.              1. The rate (bit 0-2) can obviously be any other available
.              rate, 100Hz is merely an example.
.
```

The clock is now running at 100Hz. After the predetermined number of counts will overflow and "NUM" will automatically be reset into the counter to repeat the cycle.

The overflow flag should be cleared with CLSA each time overflow occurs in order for every overflow to be program detectable.

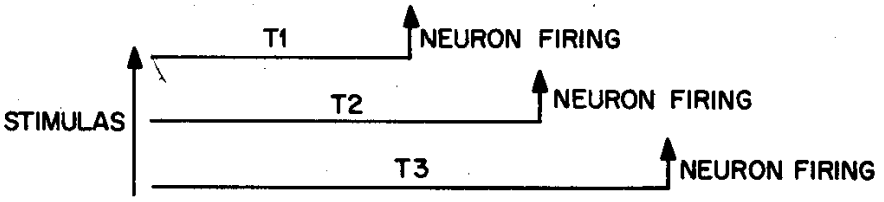
010 Time Base from initial event — as in modes 0 and 1, the clock counter is incremented at the rate indicated by bits 0-2.

On each occurrence of an Input event on a selected input channel (see Clock Enable Register), the contents of the clock counter are automatically transferred to the Buffer-Preset Register and the counter continues to count.



This mode is useful for determining the total elapsed time between some initial event (a stimulus possibly) and subsequent events which might be caused by it (muscle reactions). In using Mode 2, the Clock Status Register is used in conjunction with the Clock Enable Register to detect the occurrence of the input events. Each time such an event is recognized, the contents of the Buffer-Preset Register can be picked up by the program and stored away or processed as the user wishes.

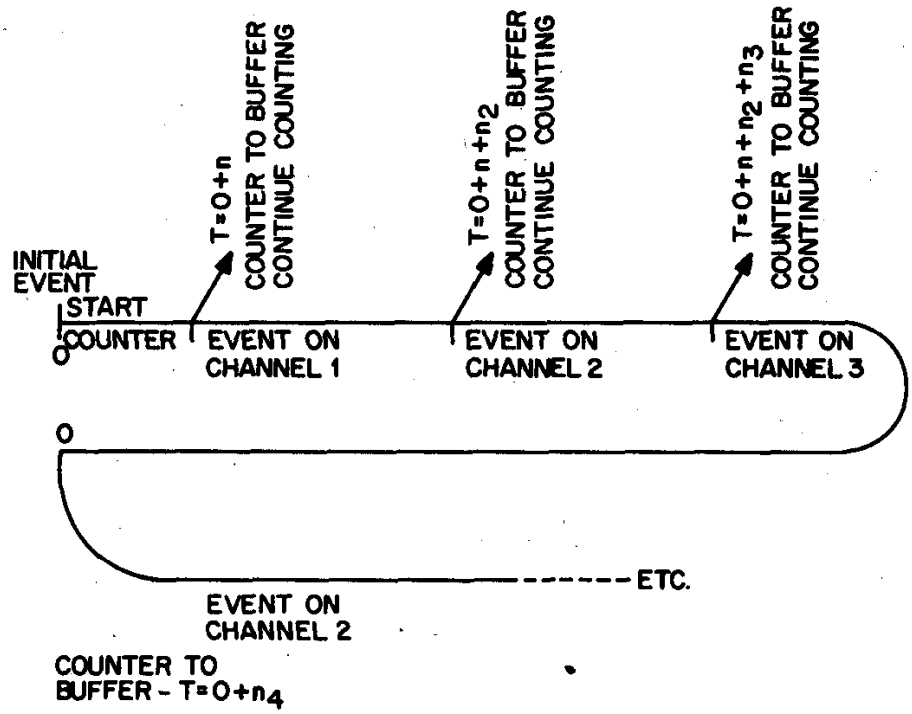
A poststimulus histogram (PST) could be generated in this fashion. A stimulus could be issued to a subject and the clock started. Then the elapsed time to succeeding neuron firings (i.e., input events) could be determined by saving the time counts which were transferred from the counter to the Buffer-Preset on each firing.



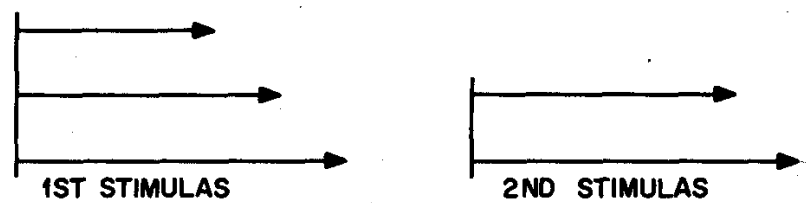
011 Time Base between two events — as in modes 0, 1 and 2, the clock counter is incremented at the rate indicated by bits 0-2.

On each occurrence of an Input event on a selected input channel (see Clock Enable Register), the contents of the clock counter are automatically transferred to the Buffer-Preset Register. This is identical to mode 2 so far.

Here is where mode 2 and mode 3 differ. In mode 2(010) the clock counter continues to count after the transfer no matter on which of the three input channels the event occurred. In mode 3(011) the clock counter continues to count after the transfer only if the event occurred on input channel 1 or input channel 2. If the event occurred on input channel 3, the clock counter is cleared after being transferred to the Buffer-Preset Register; counting then continues from 0 at the specified rate.



This mode might be used to measure elapsed time from event to event by resetting the counter after each event. It is also useful for restarting the counter after one kind of external event but not after other. For instance, look back to the PST example under mode 2. If the stimulus signal was connected to external input channel 3 (see discussion of Input Control Panel, and the neuron firings to channels 1 & 2, each new stimulus would clear the clock counter while each neuron firing would give total elapsed time from its own stimulus.



100
101
110
111

Modes 4(100), 5(101), 6(110), and 7(111) are identical to modes 0(000), 1(001), 2(010), and 3(011) respectively with this exception: when Clock Control Register bit 3 (i.e., mode control bit 0) is set to a 1 and the A/D control also has FAST SAMPLE enabled (see Special Function Register), the occurrence of overflow in the clock counter or the occurrence of an external event on a selected Input channel (see Clock Enable Register and Input Control Panel) causes the A/D converter to initiate a con-

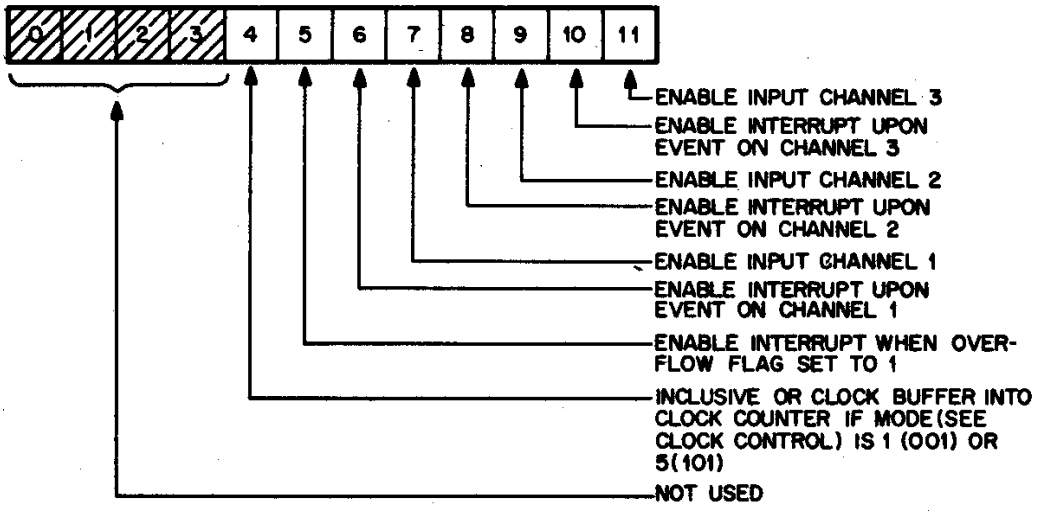
version on whatever A/D channel was last sampled by a SAM instruction (see A/D Converter). In these four modes, the analog to digital conversions take place under the automatic timing control of the clock. They are triggered by (and only by) the clock control overflow or an external event on a selected Input Channel; the SAMn instruction, which would then be given on program detection of this overflow or external event, reads the result of the previous conversion (that caused by the clock overflow or external event) and sets the channel number for the next conversion (the one which will occur on the next clock overflow or external event). The SAMn itself does not under these conditions (mode 4-7 and FAST SAMPLE function enabled) cause an A/D conversion.

Simulated Inputs

Generally, bits (0-5) of the Clock Control Register are the only bits with which the average programmer is concerned. Bits 6, 8, and 10 are not used at all and bits 7, 9, and 11 are generally used only for diagnostic purposes. If Bit 7 is set to a 1 and the counting rate is set to 110, (external event), then each time a CLLR is issued, an event on channel 1 is simulated (i.e., the event bit in the clock status register is set, etc.). The same happens for bit 9 or 11.

CLOCK ENABLE

The Clock Enable Register is the 12 bit register that determines the "who" of the clock, just as the clock control determines the "what, when and how".



The Clock Enable Register determines basically four things:

1. Which, if any, of the three External Input Channels (see Input Control Panel) will be considered active (i.e., selected) at any particular point.

2. Whether the selected input channels will be able to cause a program interrupt each time an external event occurs or merely set the appropriate bit in the Status Register (see Clock Status Register).

3. Whether the occurrence of overflow in the clock counter will be able to cause a program interrupt each time the overflow flag is set or merely set the appropriate bit in the Status Register.

4. If the Buffer-Preset should be transferred to the clock counter when the CLEN IOT is given. This is generally used in conjunction with Mode 1 or 5 (Preset Time) to set up the counter initially before starting the clock running (see note under Clock Control Register Mode 1).

The Clock Enable Register is loaded from the AC when the 8 mode IOT CLEN is issued. The conditions enabled for any channel determine what action must be taken to detect an event.

For example, if bit 7=1 and bit 6=0, any events which occur on Input Channel 1 can be detected only by continually checking for the appropriate bit in the Clock Status Register to be set to 1. If, however, bit 7=1 (Enable Channel 1) and bit 6=1 also (Enable Interrupt on Channel 1), then an event occurring on Input Channel 1 will cause a Program Interrupt as well as setting the appropriate Status Register bit.

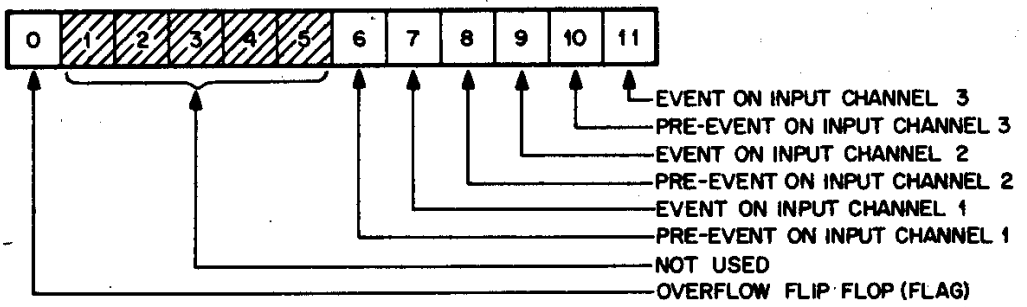
NOTE

The Program Interrupt facility must have been enabled by issuing an ION (6001) before an actual Program Interrupt can occur. If an ION has not been issued, Enabling Interrupt on an Input Channel or on Overflow will still cause the clock flag to be set to 1, but the program will not trap to 0 (or 140 if Linc mode). The clock flag can be checked, like any other device flag, with a skip on flag=1 instruction CLSK.

Bit 4 of the Clock Enable Register is a little bit different in that setting it to 1 doesn't enable the possibility of something happening later on, but rather enables the user to put into the clock counter a predetermined value which he placed in the Buffer Preset Register with IOT CLAB. With Bit 4=1, the Buffer Preset Register is inclusively OR'ed into the clock counter when the CLEN IOT is given.

CLOCK STATUS REGISTER

The Clock Status Register is the last of five 12 bit program accessable registers associated with the KW12-A. It is essentially the "who done it" register of the clock.



The Clock Status Register records "what has happened and where" upon its occurrence. The program can check this register by giving a CLSA IOT and then take the appropriate action. Each time a CLSA is issued, reading the clock status into the AC, the bits of the clock status which had been set are cleared. This simplifies the programming somewhat by ensuring that only one occurrence of any event will be transferred to the program.

Bit 0 is set to a 1 whenever overflow occurs in the clock counter (i.e., whenever the clock counter goes from 7777 to 0). If Clock Enable Register bit 5 is set to 1 when overflow occurs, a program interrupt will occur, but this bit will still be set to 1 in order to identify what caused the interrupt.

Bits 1-5 are not used.

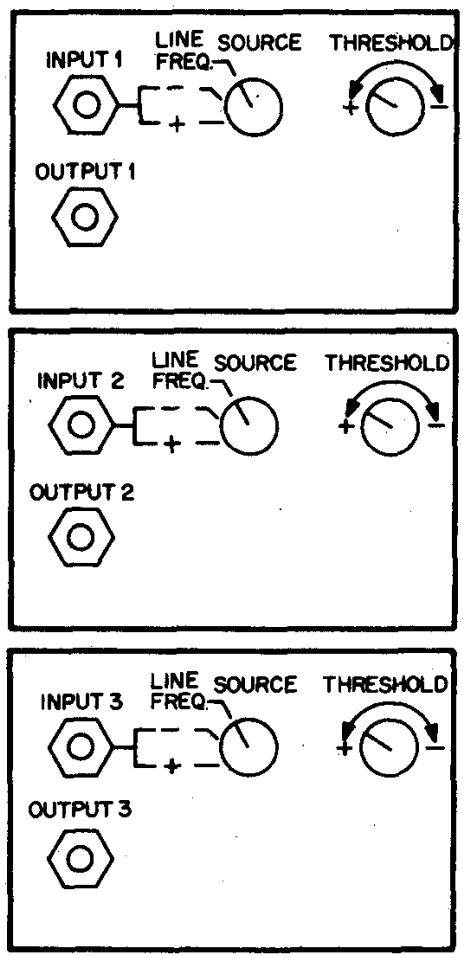
Bits 6-11 record the occurrence of an event on a specific selected Input Channel. If more than one event occurs before this register is read, another bit is set on its channel, indicating an event overlap condition. If the second event occurred on the same Input channel as the first event then the "pre-event" bit for that channel will be set as well as the event bit which will remain set. All bits set in the Clock Status Register stay set until a CLSA instruction is issued. The only exception to this is if Input Channel 1 is being used for clock timing rate (i.e., rate 110). If so, Channel 1 Event flag (bit 7) is automatically cleared.

INPUT CONTROL PANEL

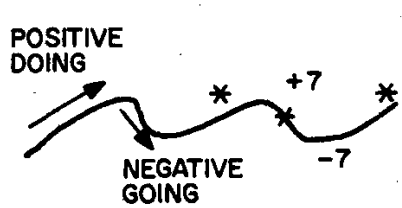
We have done a lot of talking about external events and External Input Channels without explaining too much about the Input Channels themselves or about what determines an event. Let us now turn to the Input Control Panel which is mounted in the main frame of the PDP-12 behind the vertical green door to the left of the console. It is labeled "KW12 Clock Control" in white printing and has three white outlined boxes delineated on it, each with a similar set of controls. Each box represents one input channel to which an external signal may be connected. Within each box there are two phone jacks, marked "input #" and "output #" and two rotating control knobs marked "source" and "threshold." The input jack and the output jack for each channel are jumpered together to provide "daisy chaining" from one input to another or to an A/D converter, etc. By connecting an external signal from a laboratory instrument via the phone jack to one of these input channels, certain occurrences in

that signal can be used to drive the clock. Each input channel consists of an input Schmitt trigger with a pulse generator five flip-flops, and associated control gating. The Schmitt trigger is governed by the settings of the

KW12 CLOCK CONTROL



"source" and "threshold" controls. A voltage between +5 and −5 is coarsely selected by setting the threshold knob to the far right for +5, far left for −5, or at some point in between. The slope, either positive going or negative going, is then chosen by setting the source knob to either the + or − stop on the dial. At this point, each time the external signal crosses the preset voltage in the indicated direction, the Schmitt trigger will fire, causing a pulse to be generated. This is the pulse which we have been speaking of as an event. Events occurring on Channel 1 can be used to supply the timing pulses if clock rate 6(110) is chosen.



* IF THRESHOLD WAS SET TO +4 VOLTS, AND SOURCE TO +, THEN THE *'S INDICATES A PULSE GENERATED WHILE THE X DOES NOT. EVEN THOUGH THE THRESHOLD WAS CROSSED, THE SECOND CONDITION OF SLOPE WAS NOT SATISIFIED AND SO A PULSE WAS NOT REQUIRED.

You will note there is one more stop on the source knob which says LINE FREQ. Setting the source knob to this stop causes the power line wave form to act as the external signal rather than supplying an actual signal from a laboratory instrument. To use line frequency to drive the clock, an input jack is obviously not needed.

Note that external input channels, while physically located on the Input Control Panel, are selected under program control by setting the appropriate bits in the Clock Enable Register. The voltage threshold and signal slope are, however, selected via the controls of the panel.

Summary of KW12-A Real Time Interface Instructions

The KW12-A is controlled by PDP-12 IOT instructions. These instructions can be used from either 8 or LINC mode. Execution time for the IOT's is 4.25μs when in 8 mode and 5.9μs when in LINC mode (including IOB).

CLSK Skip on Clock Interrupt

Octal Code: 6131
Event Time: 1
Execution Time: 4.25μs
Operation: Skip if clock interrupt condition exists (i.e., clock flag=1).

The interrupt conditions are as follows:

a. Enable Event 1 Interrupt (1) and Event 1 (1).
b. Enable Event 2 Interrupt (1) and Event 2 (1).
c. Enable Event 3 Interrupt (1) and Event 3 (1).
d. Enable Overflow Interrupt (1) and Overflow (1).

CLCA Counter to AC

Octal Code: 6137
Event Time: 1, 2, 3
Execution Time: 4.25μs
Operation: The AC is cleared and the contents of the Clock Counter are transferred via the Buffer-Preset Register into the AC.

CLBA Buffer Preset Register to AC

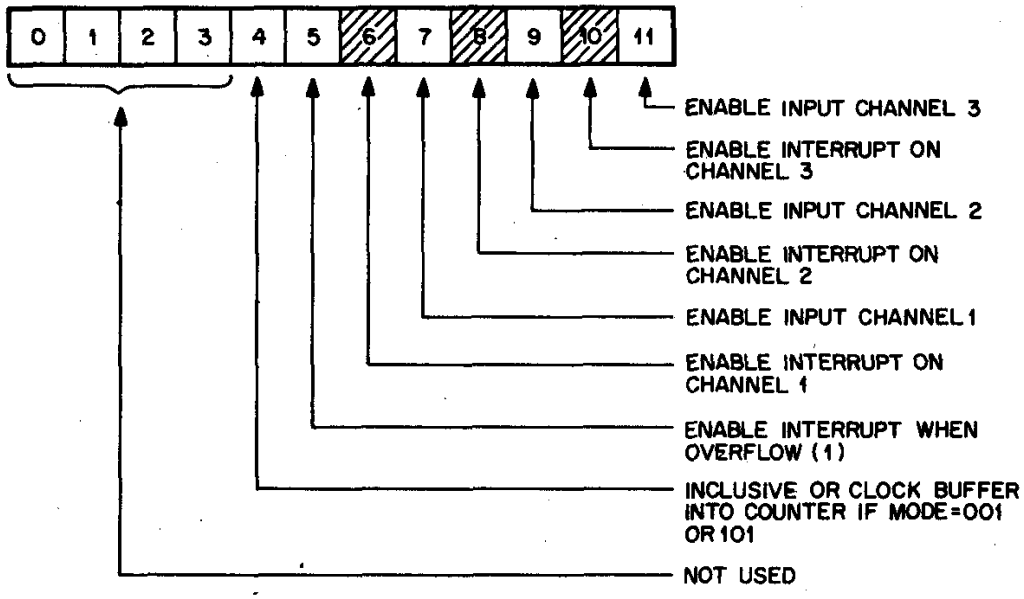
Octal Code: 6136
Event Time: 2, 3
Execution Time: 4.25μs
Operation: The AC is cleared and the contents of the Clock Buffer Preset Register are transferred into the AC.

CLAB AC to Buffer Preset Register

Octal Code: 6133
Event Time: 2
Execution Time: 4.25μs
Operation: Transfer AC to Buffer Preset Register. The previous contents of the Buffer Preset Register are lost and the AC is unchanged.

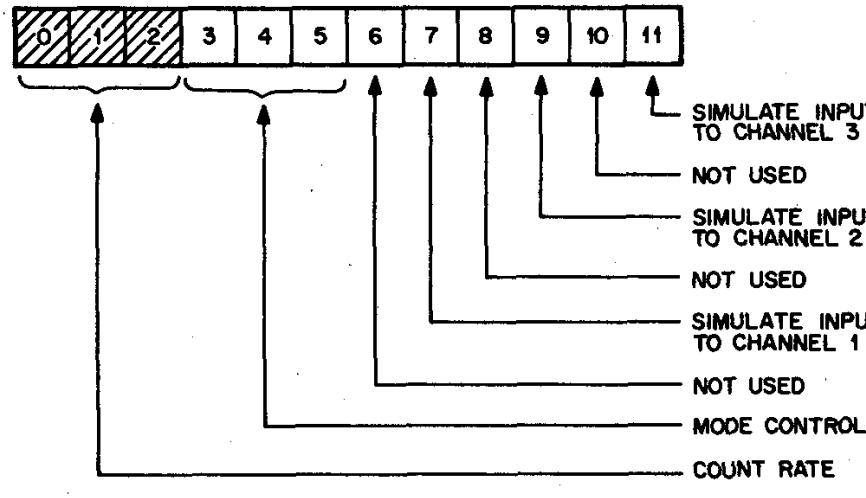
CLEN Load Clock Enable Register

Octal Code: 6134
Event Time: 3
Execution Time: 4.25μs
Operation: The contents of the AC are transferred to the Clock Enable Register. The function of each bit is given below:



CLLR Load Clock Control Register

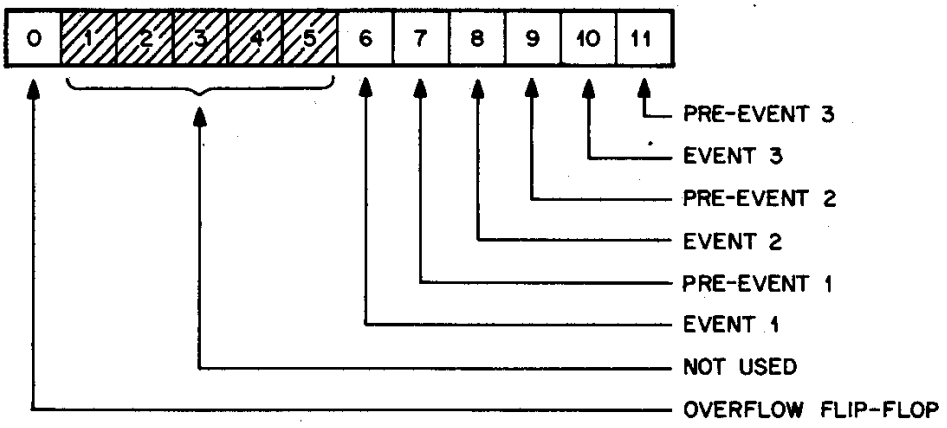
Octal Code: 6132
Event Time: 2
Execution Time: 4.25μs
Operation: The contents of the AC are transferred to the clock control register. Three bits are used to provide simulated data input to each of the three Event input channels. The AC is unchanged.



CLSA Clock Status to AC

Octal Code: 6135

Event Time: 1, 3
Execution Time: 4.25μs
Operation: This instruction interrogates the Clock Input and Overflow Status flip-flops. The clock status information is inclusive ORed into the AC. Then the clock status and overflow bits which are set are cleared. This ensures that only one occurrence of an Event will be transferred to the program.



KW2-B AND KW12-C FIXED-INTERVAL CLOCKS

The KW12-B provides a means of interrupting the Central Processor at intervals determined by a variable RC Oscillator. The KW12-C is similar to the KW12-B except that the RC oscillator is replaced by a crystal oscillator with a single fixed frequency. The KW12-B or KW12-C may be turned off under program control. However, variations in frequency require physically altering or changing the oscillator.

Summary of Instruction Set

The KW12-B and KW12-C are controlled by IOT instructions. These instructions can be used from either mode. Execution time for the Simple Clock IOTs is 4.25μs when in 8 mode and 5.9μs when in LINC mode (including IOB).

CSOF Skip on Clock Flag
Octal Code: 6131
Execution Time: 4.25μs
Operation: Skip if clock flag is set.

CTOC Turn Off Clock
Octal Code: 6132
Execution Time: 4.25μs
Operation: Turn off the clock, clear the clock, flag, and disable the clock interrupt. I/O preset issued either manually or under program control also performs these operations.

CTON Turn On Clock

Octal Code: 6134
Execution Time: 4.25μs
Operation: Turn on the clock and clear the flag.

CRUN Clock Running

Octal Code: 6135
Execution Time: 4.25μs

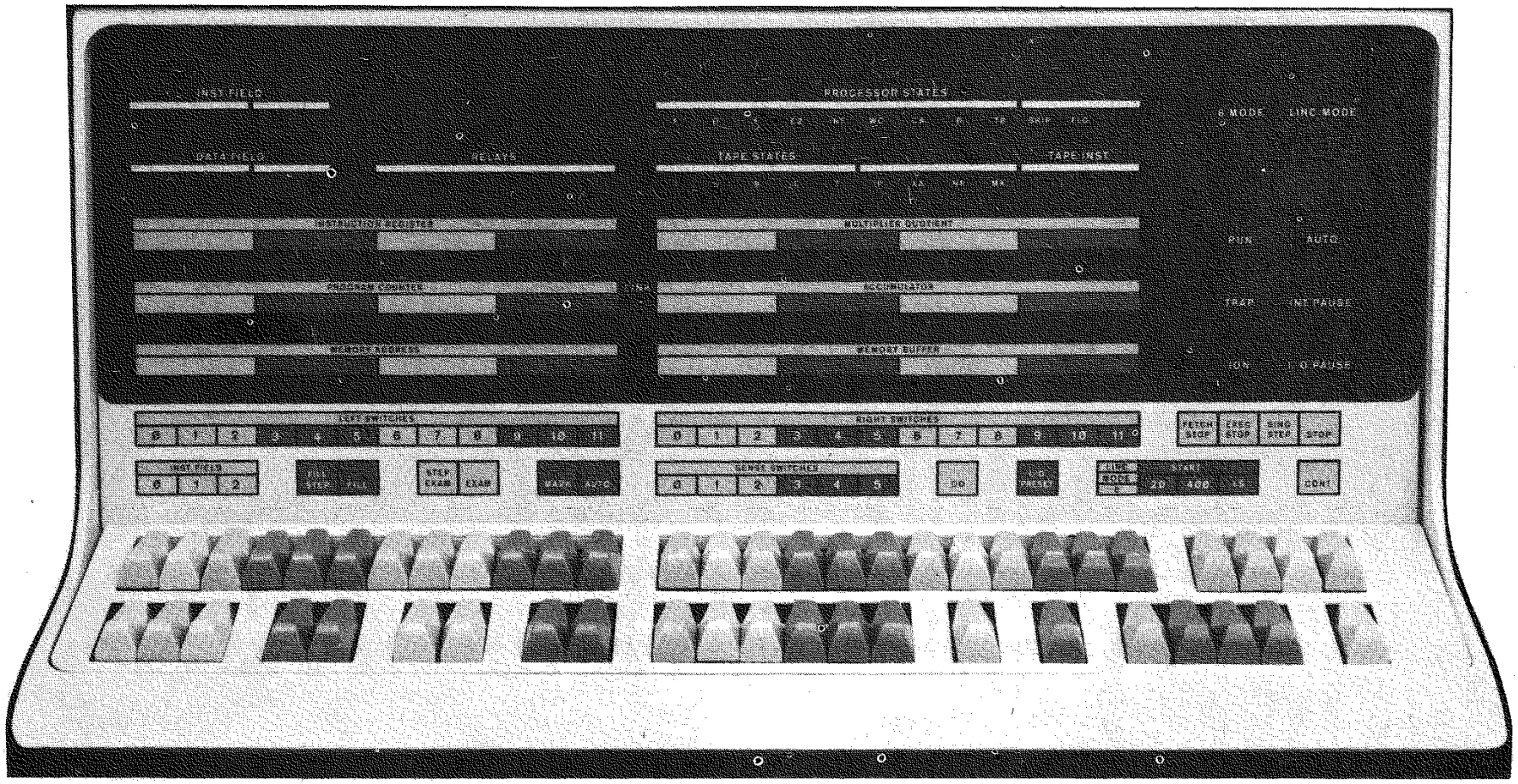
Operation: Turn on the clock, enable the clock interrupt, clear the clock flag, skip if the clock flag was set when the instruction was issued.

Frequency Range:

KW12-B The KW12-B uses the M401 variable clock in slot F18 for a time base. The frequency may be varied within a range by adjusting a potentiometer on the module. Five frequency ranges are available. The KW12-B is nominally set to the 1.75 kHz-to-17.5 kHz frequency range by jumpering F18N2 to F18T2. Other frequency ranges may be achieved by removing the jumper from F18T2 and attaching as shown below. The exact frequency may be checked by observing the signal on F18D2 with an oscilloscope.

FREQUENCY RANGE	INTERCONNECTIONS REQUIRED
17.5kHz to 50kHz	F18N2 — F18S2
1.75kHz to 17.5kHz	F18N2 — F18T2 (nominal setting)
175Hz to 1.75kHz	F18N2 — F18P2

KW12-C The KW12-C uses the M405 crystal clock in slot F18 for a time base. The frequencies are fixed by a series resonant crystal oscillator to obtain a frequency stability of .01 percent of the specified value between 0°C and +55°C. The frequencies available are in the range of 5kHz and must be specified in advance by the customer.



PDP-12 Console.

CHAPTER 8
PDP-12 PROGRAMMING

INTRODUCTION

The PDP-12 computer operates in either of two modes: "LINC mode" or "PDP-8 mode". Because of the BiModal Nature of the PDP-12, it uses the software available for both the family-of-eight and the LINC computers as well as having two powerful instruction sets to work from when writing original programs.

Use of Existing Software

Software packages developed for family-of-eight computer are run without modification on the PDP-12. This provides the user with powerful programs such as the many mathematical routines that would otherwise have to be written from the beginning. Most of the programs written on the LINC-8 can be run directly on the PDP-12 using the I/O handler (LINC-8 simulator) provided with the PDP-12 basic software.

Control Console

Understanding and using the Control Console in a small or medium scale computer is the first step in learning to use the system.

The preceding page shows the PDP-12 Console. Note the location of all switches and indicated register.

Main Registers

Accumulator (AC) 12 Bits	This register contains data being operated upon: Its contents may be shifted or rotated right or left; incremented, cleared, or complemented; stored in memory or added to the contents of a memory register; and logically or arithmetically compared with the contents of any memory register. The AC holds the sum after an addition, and part of the product after a multiplication. The AC is also involved in the transfer of data to and from various other registers outside the central processor.
Link (L) 1 Bit	The Link is an extension of the AC. When a carry occurs out of bit 0 of the AC during a 2's complement addition, the Link is complemented. It may be set or cleared independently of the AC, and may be included (or not) in shifting and rotating operations performed on the contents of the AC.

Multiplier Quotient (MQ) 12 Bits	This register is used as a second arithmetic register for multiply and some rotate instructions. It is also used for extended Arithmetic Option (KE12) functions.
Program Counter (PC) 12 Bits	This register contains the address of the Next instruction to be executed within the memory field selected by the Instruction Field Register (see below). In PDP-8 mode, the PC acts as a 12-bit counter; in LINC mode, it acts as a 10-bit counter.
Memory Address Register (MA) 12 Bits	This register contains the address for memory references. Whenever a core memory location is being accessed, either for reading or for writing, the MA contains the address of that location.
Instruction Register (IR) 12 Bits	This register contains the complete binary code of the instruction being executed.
Memory Buffer (MB) 12 Bits	All information passing between memory and any other register in the PDP-12 must go through the Memory Buffer Register, whether the transfer involves the central processor, an external device, or another memory register.

Major State Generator

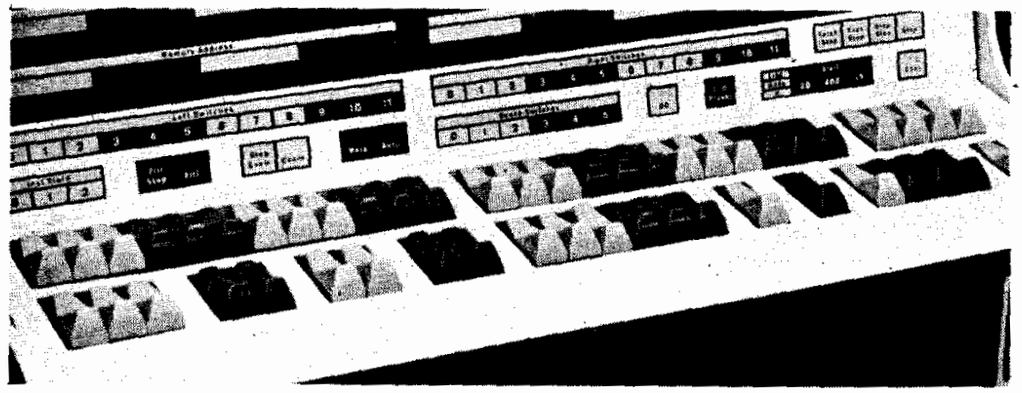
The Major State Generator establishes the proper states in sequence for the instruction currently being executed. One or more major states are entered serially to execute each programmed instruction. During a *Fetch* State, an *instruction* is loaded from core memory at the address specified by the Memory Address Register into the memory buffer and the instruction register. The Defer State is used in conjunction with indirect addressing as discussed in detail under "Indirect Addressing" later in the chapter.

During the *Execute* State, the instruction in the Instruction Register is performed on the assigned data.

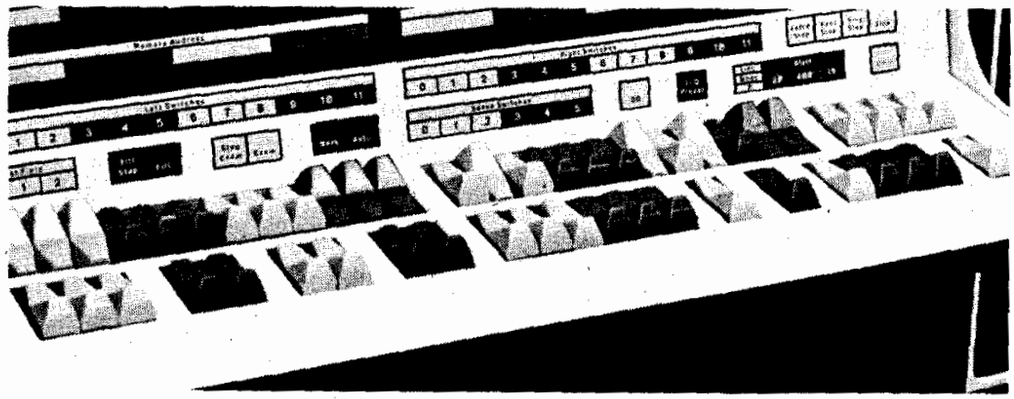
Instruction Field Register (IF) 5 Bits	This register selects the memory field containing the executable program. In LINC mode, it is used to designate one of up to thirty-two 1024-word segments In PDP-8 mode, the three high-order bits of the IF are used to designate one of up to eight 4096-word fields.
Data Field Register (DF) 5 Bits	This register selects memory field containing data to be indirectly accessed by the memory reference instructions of a program. The fields are specified in each mode in the same way that the IF specifies the Instruction Field.

USING KEYS AND SWITCHES

The following two illustrations show the console keys on the PDP-12.



Console with all keys in zero position.



Left Switches set to $7007_{(8)}$ right switches set to $2053_{(8)}$

Left Switch Register (LSW)

The LSW is a 12 bit register located on the left side of the Control Console. The LSW can be accessed by the central processor while in the *LINC* Mode. This register is also used by the *Fill* and *Exam* keys as address switches and by the *DO* key as the first word of an instruction.

Right Switch Register (RSW)

The RSW is a 12 bit register located on the right side of the Control Console. The RSW can be used to load information for transfer into the Memory Buffer as data to be stored in core memory by means of the fill step switch. The RSW can also be accessed by the central process when in either Linc or 8-mode. Twelve bit words can be entered into memory by the used of the fill and fill step keys.

Fill

The fill key allows the entering of the 12 bit words into specific memory locations. The *address of the memory location* to be filled is placed in the *Left Switch Register.* The data is placed in the *Right Switch Register.* Depressing the *Fill Key* executes the fill operation. Upon completion of the fill function, the *MA* register will display the Address of the memory location, just filled and the *MB* register will display the data just entered in that location.

Fill Step

The Fill Step Operation is in two steps:
Step 1. — Enter a data word into the current core memory location shown in the *M.A.* register by putting the word in the *Right Switch Register* and depressing the *Fill Step* key. The M.A. will show no change. The *M.B.* will display the word just entered into the memory location specified by the *M.A.*

Step 2. — Releasing the Fill Step Key will add *one* to the *M.A.* register changing the memory location addressed to MA + 1. The *M.B.* will show the *contents* of the new, incremented memory location.

NOTE

The *Fill Step* Key uses the current M.A. address to signify the location to be filled. To fill into a memory location other than the one currently addressed, use the *Fill Key.*

Once words have been stored in memory it is usually the practice to go back and make sure that everything has been entered properly before starting the program. The keys that allow an examination of memory are the exam and step exam keys.

Exam

The *Exam* key allows the examining of a memory location whose address is specified by the *Left Switch Register.* Depressing the Exam key executes the examine operation, the address examined will be displayed in the *M.A.* Register and the contents of that location will be displayed in the *M.B.* Register.

Step Exam

The *Step Exam* Key allows the examination of successive memory location. When the *Step Exam* Key is depressed, the *M.A.* is indexed by *one* and the *contents* of the *New M.A.* is displayed in the *M.B.* Register.

NOTE

The *Step Exam* Key uses the *Current M.A.* + *1* for addresses. To examine other addresses, use the *Exam Key.*

Once a program has been loaded and examined, it is ready to be run.

Mode Key

The *Mode Key* operates in conjunction with the *I/O Preset* Key. It allows the PDP-12 to initialize as either a *PDP-8/I* or *LINC.*

I/O Preset

I/O Preset clears the link, AC, and MQ, performs a general initialization of the PDP-12 central processor and all I/O connected to the system.

Start 20

Depressing *Start 20* will start the computer in automatic run at 8-mode absolute location 20 or LINC Mode location 20 *relative* to current *LINC Instruction Field.*

Start 400

The PDP-12 will enter automatic run an 8-mode absolute location 400 or LINC Mode location 400 *relative* to current *LINC Instruction Field.*

Start LSW

The PDP-12 will enter automatic run at the absolute *15 bit* location specified by the *Left Switch Register* and the three bit *Instruction Field Register.*

Single Step Key

When the Single Step is enabled the PDP-12 wil halt at the end of the current major state of the current instruction. Depressing the *Continue Key* will allow the computer to execute one more major state before coming to a halt. Thus, a program can be followed, one major state at a time to completion. This is a tremendous aid, since it allows inspection of all main registers as the program progresses to completion and errors in programming can be easily pinpointed.

Stop Key

Operation of the *Stop Key* halts the computer at the end of the current instruction.

Continue

Continue sets the central processor into automatic run without modifying any active registers. This allows a program that has been stopped with the Single Step Key or Stop Key to resume operation with all registers and conditions undisturbed. The Continue Key can be used in conjunction with the stop key to allow manual progression through a program one instruction at a time.

Fetch Stop

The computer will halt when the address of the word accessed during the *FETCH CYCLE* matches the address in the *Left Switch.*

Execute Stop

The computer will halt when the address of a word accessed in *any cycle* except Fetch matches the address in the Left Switches.

NOTE

Depressing Continue after either a Fetch Stop or Execute Stop will allow the program to continue automatically, until the Fetch Stop or Execute Stop condition is again qualified, at which time the computer will again halt.

Auto

The *Auto Key* activates an "Automatic Button Pusher" that works in conjunction with a number of keys on the console. The speed at which the "Button Pushing" is done is determined by the *Coarse* and *Fine* controls on the data terminal behind the door.

Auto with Fill Step

Allows the filling of sequential memory locations without having to continually press *Fill Step.*

Auto with Step Exam

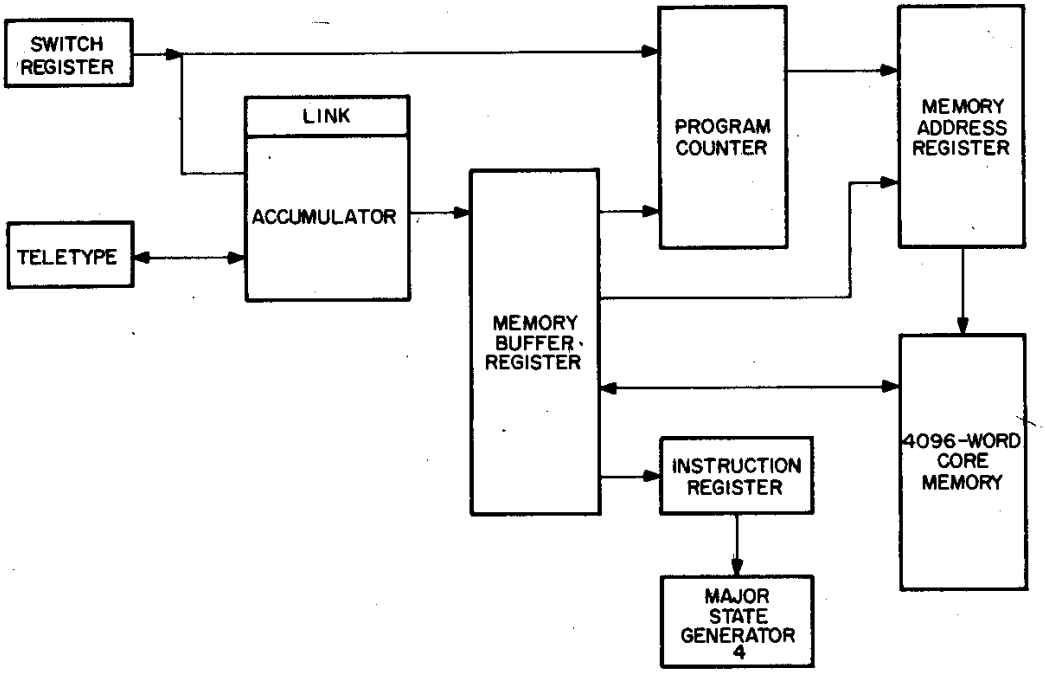
Examining of sequential memory locations can be performed with this operation.

Auto with Single Step and Continue

Allows progress through a program one cycle at a time at a speed determined by the coarse and fine adjustments for auto.

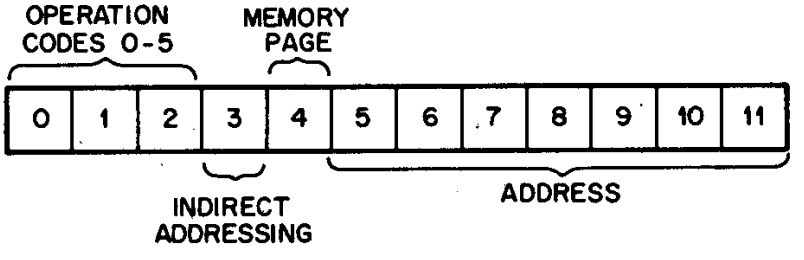
PDP-8 MODE PROGRAMMING

In the *8-mode* of operation, the PDP-12 takes on the internal configuration of a PDP-8I. A block diagram of this configuration is shown below.



Block Diagram of 8-Mode

The standard set of instructions for the 8 mode includes eight basic instructions. The first six of these are called the Memory Reference Instructions *(M.R.I.)* the format of an M.R.I. is shown below.



Memory Reference Instruction Bit Assignments

The three most significant bits (0, 1, 2) determine which of the eight basic instructions are to be performed.

The six memory reference instructions are listed below with their mnemonic and octal equivalents as well as their memory cycle times.

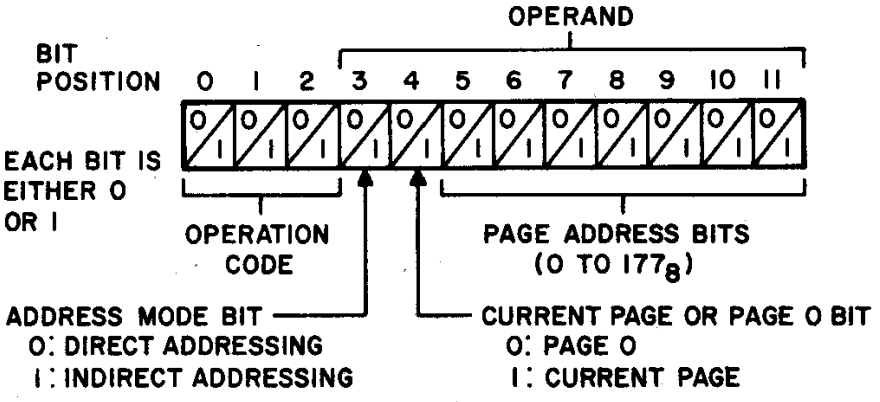
Instruction	Mnemonic[2]	Octal Value	Memory Cycles[1]
Logical AND	AND	0nnn	2
Two's Complement Add	TAD	1nnn	2
Deposit and Clear the Accumulator	DCA	3nnn	2
Jump	JMP	5nnn	1
Increment and Skip if Zero	ISZ	2nnn	2
Jump to Subroutine	JMS	4nnn	2

The remaining nine bits are used to address the operand of the instruction. However, a full twelve bits are needed to address the 4096 (10,000 octal) locations that are contained in memory. To make the best use of the available nine bits, the 8 mode utilizes a logical division of memory into sections (pages) of $200_{(8)}$ LOCATIONS each as shown in the following table.

Page	Memory Locations	Page	Memory Locations
0	0-177	20	4000-4177
1	200-377	21	4200-4377
2	400-577	22	4400-4577
3	600-777	23	4600-4777
4	1000-1177	24	5000-5177
5	1200-1377	25	5200-5377
6	1400-1577	26	5400-5577
7	1600-1777	27	5600-5777
10	2000-2177	30	6000-6177
11	2200-2377	31	6200-6377
12	2400-2577	32	6400-6577
13	2600-2777	33	6600-6777
14	3000-3177	34	7000-7177
15	3200-3377	35	7200-7377
16	3400-3577	36	7400-7577
17	3600-3777	37	7600-7777

Notice that a new page starts every $200_{(8)}$ locations.

Since there are 200_8 locations on a page and seven bits can represent 200_8 different numbers, seven bits (5 through 11 of the MRI) are used to specify the page address. Before discussing the use of the page addressing convention by an MRI, it should be emphasized that memory does not contain any physical page separations. The computer recognizes only absolute addresses and does not know what page it is on, or when it enters a different page. But, as will be seen, page addressing allows the programmer to reference all of the $4{,}096_{10}$ locations of memory using only the nine available bits of an MRI. The format of an MRI is shown below.



Format of a Memory Reference Instruction

As previously stated, bits 0 through 2 are the operation code for the MRI. Bits 5 through 11 identify a specific location on a given page, but they do not identify the page itself. The page is specified by bit 4, often called the *current page or page 0 bit*. If bit 4 is a 0, the page address is interpreted as a location on page 0. If bit 4 is a 1, the page address specified is interpreted to be on the current page (the page on which the MRI itself is stored). For example, if bits 5 through 11 represent 123_8 and bit 4 is a 0, the location referenced is absolute address 123_8. However, if bit 4 is a 1 and the current instruction is in a core memory location whose absolute address is between $4,600_8$ and $4,777_8$, the page address 123_8 designates the absolute address $4,723_8$. Note that, as shown in the following example, this characteristic of page addressing results in the octal coding for two TAD instructions on different memory pages being identical when their operands reference the same relative location (page address) on their respective pages.

	Content		
Location	Mnemonic	Octal	Explanation
200	TAD 250	1250	TAD 250 and TAD 450 both mean add the contents of location 50 on the current page (bit 4=1) to the accumulator.
.	.		
.	.		
400	TAD 450	1250	

Except when it is on page 0, a memory reference instruction can reference 400_8 locations directly, namely those 200_8 locations on the page containing the instruction itself and the 200_8 locations on page 0, which can be addressed from any memory location.

NOTE

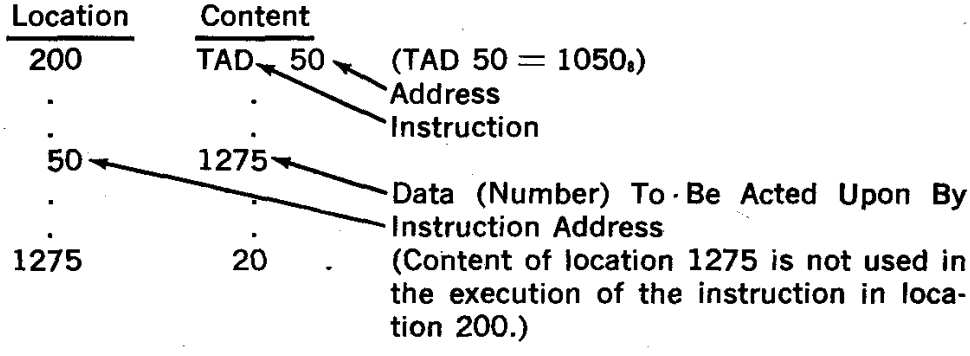
If an MRI is stored in one of the first 200_8 memory locations (0 to 177_8), current page *is* page 0; therefore, only locations 0 to 177_8 are directly addressable.

Indirect Addressing

In the preceding section, the method of directly addressing 400_8 memory locations by an MRI was described — namely those on page 0 and

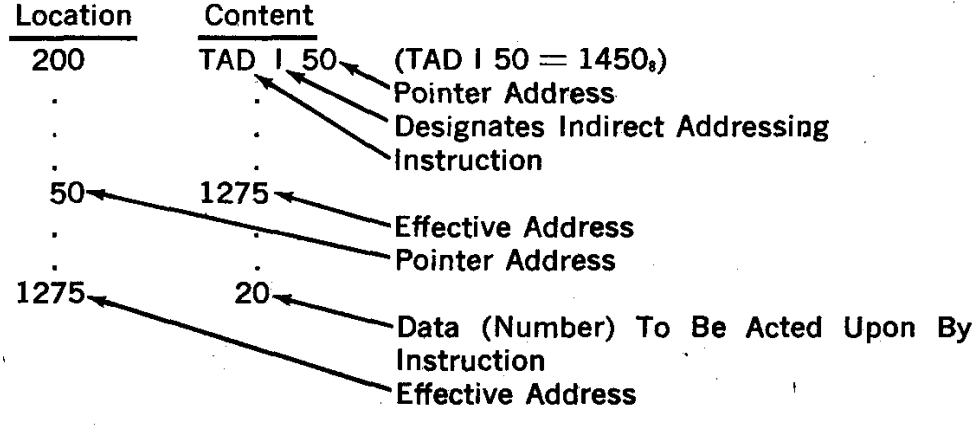
those on the current page. This section describes the method for addressing the other 7400_8 memory locations. Bit 3 of an MRI designates the address mode. When bit 3 is a 0, the operand is a direct address. When bit 3 is a 1, the operand is an indirect address. An indirect address (pointer address) identifies the location that contains the desired address (effective address). To address a location that is not directly addressable, the absolute address of the desired location is stored in one of the 400_8 directly addressable locations (pointer address); the pointer address is written as the operand of the MRI; and the letter I is written between the mnemonic and the operand. (During assembly, the presence of the I results in bit 3 of the MRI being set to 1.) Upon execution, the MRI will operate on the contents of the location identified by the address contained in the pointer location.

The two examples shown illustrate the difference between direct addressing and indirect addressing. The first example shows a TAD instruction that uses direct addressing to get data stored on page 0 in location 50; the second is a TAD instruction that uses indirect addressing, with a pointer on page 0 in location 50, to obtain data stored in location 1275. (When references are made to them from various pages, constants and pointer addresses can be stored on page 0 to avoid the necessity of storing them on each applicable page.) The octal value 1050, in the first example, represents direct addressing (bit 3 = 0); the octal value 1450, in the second example, represents indirect addressing (bit 3 = 1). Both examples assume that the accumulator has previously been cleared.



NOTE

AC = 1275 after executing the instruction in location 200.



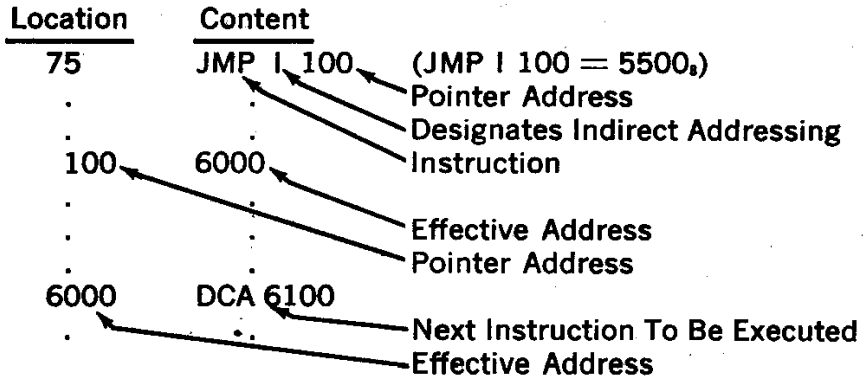
NOTE

AC = 1275 after executing the instruction in location 200.

Comparison of Direct and Indirect Addressing

The following three examples illustrate some additional ways in which indirect addresing can be used. As shown in example 1, indirect addressing makes it possible to transfer program control off page 0 (to any desired memory location). (Similarly, indirect addressing makes it possible for other memory reference instructions to address any of the $4{,}096_{10}$ memory locations.) Example 2 shows a DCA instruction that uses indirect addressing with a pointer on the current page. The pointer in this case designates a location off the current page (location 227) in which the data is to be stored. (A pointer address is normally stored on the current page when all references to the designated location are from the current page.) Indirect addressing provides the means for returning to a main program from a subroutine, as shown in example 3. Indirect addressing is also effectively used in manipulating tables of data as described and illustrated in conjunction with autoindexing.

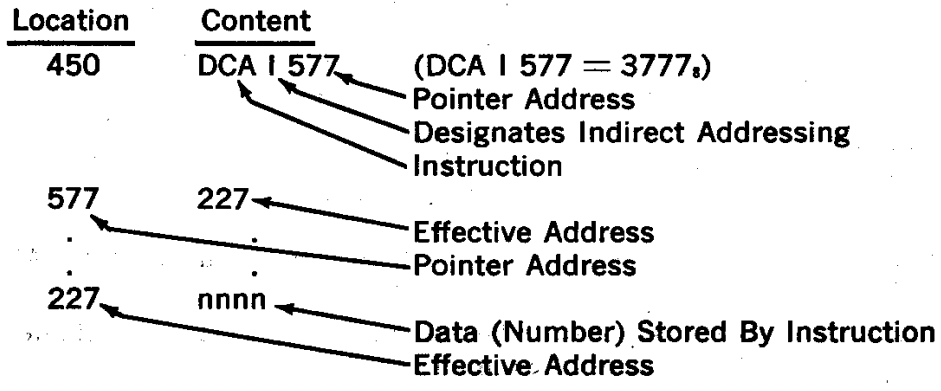
EXAMPLE 1



NOTE

Execution of the instruction in location 75 causes program control to be transferred to location 6000, and the next instruction to be executed is the DCA 6100 instruction.

EXAMPLE 2

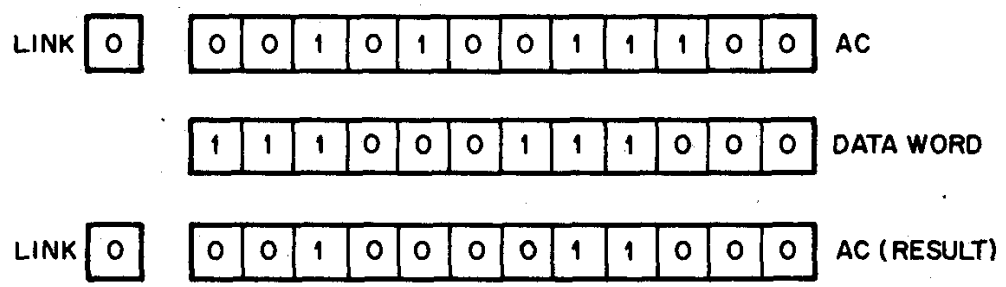


NOTE

Execution of the instruction in location 450 causes the contents of the accumulator to be stored in location 227.

AND ($0nnn_8$)

The AND instruction causes a bit-by-bit Boolean AND operation between the contents of the accumulator and the data word specified by the instruction. The result is left in the accumulator as illustrated below.

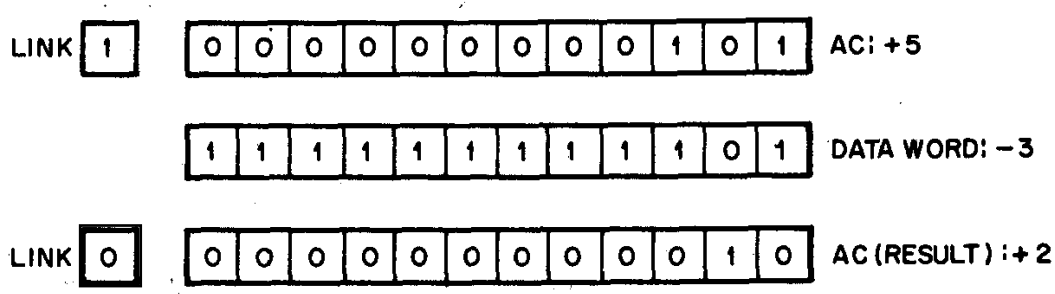


The following poins should be noted with respect to the AND instruction:

1. A 1 appears in the AC only when a 1 is present in both the AC and the data word (The data word is often referred to as a mask);
2. The state of the link bit is not affected by the AND instruction; and
3. The data word in the referenced location is not altered.

TAD ($1nnn_8$)

The TAD instruction performs a binary addition between the specified data word and the contents of the accumulator, leaving the result of the addition in the accumulator. If a carry out of the most significant bit of the accumulator should occur, the state of the link bit is complemented. The add instruction is called a Two's Complement Add to remind the programmer that negative numbers must be expressed as the two's complement of the positive value. The following figure illustrates the operation of the TAD instruction.



The following points should be remembered when using the TAD instruction:

1. Negative numbers must be expressed as a two's complement of the positive value of the number;

2. A carry out of the accumulator will complement the link; and
3. The data word in the referenced location is not affected.

DCA (3nnn$_8$)

The DCA instruction stores the contents of the AC in the referenced location, destroying the original contents of the location. The AC is then set to all zeroes. The following example shows the contents of the accumulator, link, and location 225 before and after executing the instruction DCA 225.

DCA 225

	AC	Link	Loc. 225
Before Execution	1234	1	7654
After Execution	0000	1	1234

The following facts should be kept in mind when using the DCA instruction:

1. The state of the link bit is not altered;
2. The AC is cleared; and
3. The original contents of the addressed location are replaced by the value of the AC.

The program below adds the contents of locations 4712 and location 274 together in the accumulator and stores the results in memory location 50.

Memory Address		Instruction	Code	Effect
Start	200	CLA	7200	Clears AC
	201	TAD 274	1274	Location 274 contains first number to be brought into the AC
	202	TAD I 377	1777	Location 377 contains address of word to be added to AC.
	203	DCA 50	050	Results is stored in AC
	.	.	.	
	.	.	.	
	274	XXXX	XXXX	First number to be added to AC.
	.	.	.	
	377	4712	4712	Address of 2nd number to be added to AC.

JMP (5nnn$_8$)

The JMP instruction loads the effective address of the instruction into the program counter, thereby changing the program sequence since the PC specifies the next instruction to be performed. In the following example, execution of the instruction in location 250 (JMP 300) causes the program to jump over the instructions in locations 251 through 277 and immediately transfer control to the instruction in location 300.

Location	Content	
250	JMP 300	(This instruction transfers program
.	.	control to location 300.)
.	.	
.	.	
300	DCA 330	

NOTE

The JMP instruction does not affect the contents of the AC or link.

ISZ ($2nnn_8$)

The ISZ instruction adds a 1 to the referenced data word and then examines the result of the addition. If a zero result occurs, the instruction following the ISZ is skipped. If the result is not zero, the instruction following the ISZ is performed. In either case, the result of the addition replaces the original data word in memory. The example in Figure 2-2 illustrates one method of adding the contents of a given location to the AC a specified number of times (multiplying) by using an ISZ instruction to increment a tally. The effect of this example is to multiply the contents of location 275 by 2. (To add the contents of a given location to the AC twice, using the ISZ loop requires more instructions than merely repeating the TAD instruction. However, when adding the contents four or more times, use of the ISZ loop requires fewer instructions.) In the first pass of the example, execution of ISZ 250 increments the contents of location 250 from 7776 to 7777 and then transfers control to the following instruction (JMP 200). In the second pass, execution of ISZ 250 increments the contents of location 250 from 7777 to 0000 and transfers control to the instruction in location 203, skipping over location 202.

CODING FOR ISZ LOOP

Location	Content
200	TAD 275
201	ISZ 250
202	JMP 200
203	DCA 276
.	.
.	.
.	.
250	7776
.	.
.	.
.	.
275	0100
276	0000

SEQUENCE OF EXECUTION FOR ISZ LOOP

		Content After Instruction Execution			
Location	Content	AC	250	275	276
FIRST PASS					
200	TAD 275	0100	7776	0100	0000
201	ISZ 250	0100	7777	0100	0000
202	JMP 200	0100	7777	0100	0000
SECOND PASS					
200	TAD 275	0200	7777	0100	0000
201	ISZ 250	0200	0000	0100	0000
202	JMP 200	(Skipped during second pass)			
203	DCA 276	0000	0000	0100	0200

ISZ Instruction Incrementing a Tally

The following points should be kept in mind when using the ISZ instruction:

1. The contents of the AC and link are not disturbed;
3. The original word is replaced in main memory by the incremented value;
3. When using the ISZ for looping a specified number of times, the tally must be set to the negative of the desired number; and
4. The ISZ performs the incrementation first and then checks for a zero result.

JMS ($4nnn_8$)

A program written to perform a specific operation often includes sets of instructions which perform intermediate tasks. These intermediate tasks may be finding a square root, or typing a character on a keyboard. Such operations are often performed many times in the running of one program and may be coded as subroutines. To eliminate the need of writing the complete set of instructions each time the operation must be performed, the JMS (jump to subroutine) instruction is used. The JMS instruction stores a pointer address in the first location of the subroutine and transfers control to the second location of the subroutine. After the subroutine is executed, the pointer address identifies the next instruction to be executed. Thus, the programmer has at his disposal a simple means of exiting from the normal flow of his program to perform an intermediate task and a means of return to the correct location upon completion of the task. (This return is accomplished using indirect addressing, which is discussed later in this chapter.) The following example illustrates the action of the JMS instruction.

Location	Content	
PROGRAM		
200	JMS 350	(This instruction stores 0201 in location 350 and transfers program control to location 351.)

```
201          DCA 270      (This instruction stores the contents of
                          the AC in location 270 upon return
                          from the subroutine.)
 .             .
 .             .

SUBROUTINE
350          0000         (This location is assumed to have an
                          initial value of 0000; after JMS 350 is
                          executed, it is 0201.)
351          iii          (First instruction of subroutine)
 .             .
 .             .
 .             .
375          JMP I 350    (Last instruction of subroutine)
```

The following should be kept in mind when using the JMS:

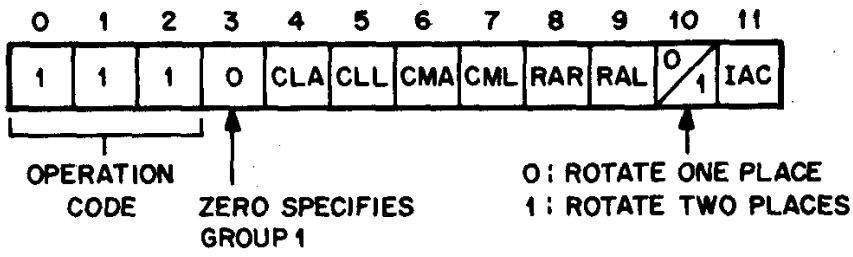
1. The value of the PC (the address of the JMS instruction +1) is always stored in the first location of the subroutine, replacing the original contents;
2. Program control is always transferred to the location designated by the operand +1 (second location of the subroutine);
3. The normal return from a subroutine is made by using an indirect JMP to the first location of the subroutine (JMP I 350 in the above example); (Indirect addressing, as discussed later in this chapter, effectively transfers control to location 201.);
4. When the results of the subroutine processing are contained in the AC and are to be used in the main program, they must be stored upon return from the subroutine before further calculations are performed. (In the above example, the results of the subroutine processing are stored in location 270.)

OPERATE MICROINSTRUCTIONS

The operate instructions (octal operation code = 7) allow the programmer to manipulate and/or test the data that is located in the accumulator and link bit. A large number of different instructions are possible with one operation code because the operand bits are not needed to specify an address as they are in an MRI and can be used to specify different instructions. The operate instructions are separated into two groups: Group 1, which contains manipulation instructions, and Group 2, which is primarily concerned with testing operations. Group 1 instructions are discussed first.

Group 1 Microinstructions

The Group 1 microinstructions manipulate the contents of the accumulator and link. These instructions are microprogrammable; that is, they can be combined to perform specialized operations with other Group 1 instructions.



The preceding diagram illustrates the manner in which a PDP-8 instruction word is interpreted when it is used to represent a Group 1 operate microinstruction. As previously mentioned, 7_8 is the operation code for operate microinstructions; therefore, bits 0 through 2 are all 1's. Since a reference to core memory is not necessary for the operation of microinstructions, bits 3 through 11 are not used to reference an address. Bit 3 contains a 0 to signify that this is a Group 1 instruction, and the remaining bits are used to specify the operations to be performed by the instruction. The operation of each individual instruction specified by these bits is described below.

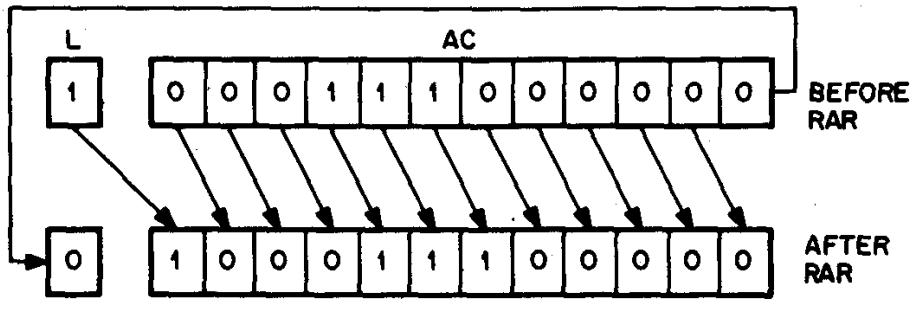
CLA *Clear the accumulator.* If bit 4 is a 1, the instruction sets the accumulator to all zeroes.

CLL *Clear the link.* If bit 5 is a 1, the link bit is set to 0.

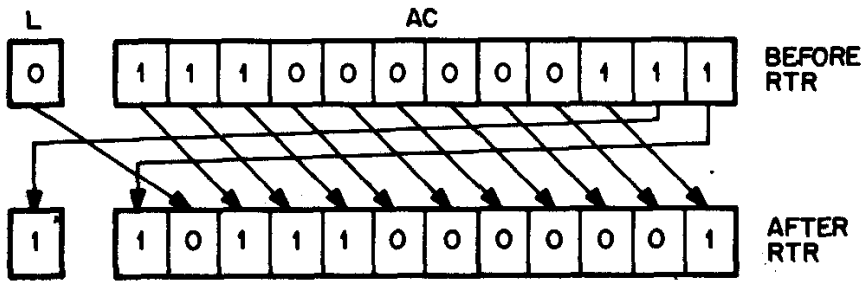
CMA *Complement the accumulator.* If bit 6 is a 1, the accumulator is set to the 1's complement of its original value; that is, all 1's become 0's, and all 0's become 1's.

CML *Complement the link.* If bit 7 is a 1, the state of the link bit is preserved.

RAR *Rotate the accumulator and link right.* If bit 8 is a 1 and bit 10 is a 0, the instruction treats the AC and L as a closed loop and shifts all bits in the loop one position to the right. This operation is illustrated by the following diagram.



RTR *Rotate the accumulator and link twice right.* If bit 8 is a 1 and bit 10 is also a 1, a shift of two places to the right is executed. Both the RAR and RTR instructions use what is commonly called a circular shift, meaning that any bit rotated off one end of the accumulator will reappear at the other end. This operation is illustrated below.



RAL *Rotate the accumulator and link left.* If bit 9 is a 1 and bit 10 is a 0, this instruction treats the AC and L as a closed loop and shifts all bits in the loop one position to the left, performing a circular shift to the left.

RTL *Rotate the accumulator and link twice left.* If bit 9 is a 1 and bit 10 is a 1 also, the instruction rotates each bit two positions to the left. (The RAL and RTL microinstructions shift the bits in the reverse direction of that directed by the RAR and RTR microinstructions.)

IAC *Increment the accumulator.* When bit 11 is a 1, the contents of the AC is increased by 1.

NOP *No operation.* If bits 0 through 2 contain operation code 7_8, and the remaining bits contain zeros, no operation is performed and program control is transferred to the next instruction in sequence.

A summary of Group 1 instructions, including their octal forms, is given below.

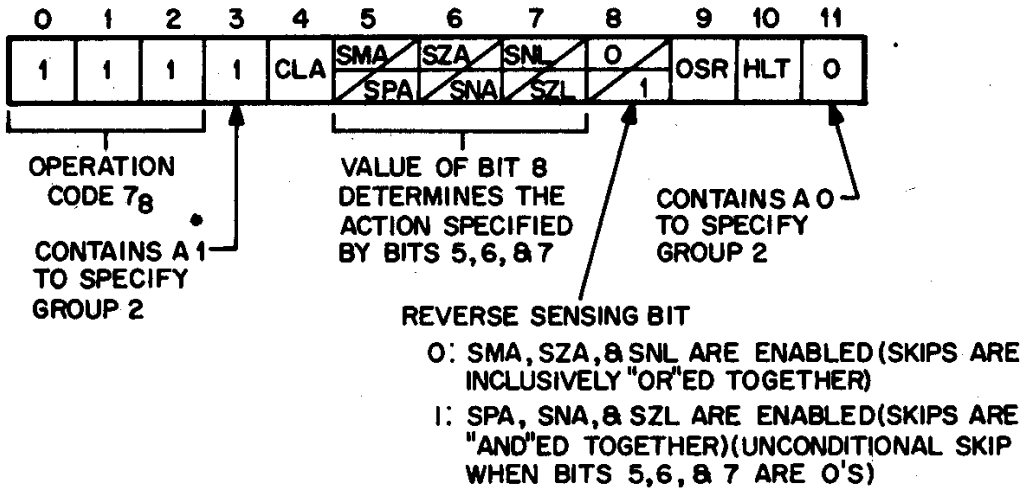
Mnemonic[1]	Octal[2]	Operation	Sequence[3]
NOP	7000	No operation	—
CLA	7200	Clear AC	1
CLL	7100	Clear link bit	1
CMA	7040	Complement AC	2
CML	7020	Complement link bit	2
RAR	7010	Rotate AC and L right one position	4
RAL	7004	Rotate AC and L left one position	4
RTR	7012	Rotate AC and L right two positions	4
RTL	7006	Rotate AC and L left two positions	4
IAC	7001	Increment AC	3

[1]Mnemonic code is meaningful to and translated by an assembler into binary code.
[2]Octal numbers conveniently represent binary instructions.
[3]Sequence numbers indicate the order in which the operations are performed.

Group 2 Microinstructions

Group 2 operate microinstructions are often referred to as the "skip microinstructions" because they enable the programmer to perform tests on the accumulator and link and to skip the next instruction depending upon the results of the test. They are usually followed in a program by a JMP (or possibly a JMS) instruction. A skip instruction causes the computer to check for a specific condition, and, if it is present, to skip

the next instruction. If the condition were not present, the next instruction would be executed.



The available instructions are selected by bit assignment as shown in the above diagram. The operation of each individual instruction specified by these bits is described below.

CLA — *Clear the accumulator.* If bit 4 is a 1, the instruction sets the accumulator to all zeros.

SMA — *Skip on minus accumulator.* If bit 5 is a 1 and bit 8 is a 0, the next instruction is skipped if the accumulator is less than zero. (bit 0 = 1)

SPA — *Skip on positive accumulator.* If bit 5 is a 1 and bit 8 is a 1, the next instruction is skipped if the accumulator is greater than or equal to zero. (bit 0 = 0)

SZA — *Skip on zero accumulator.* If bit 6 is a 1 and bit 8 is a 0, the next instruction is skipped if the accumulator is zero.

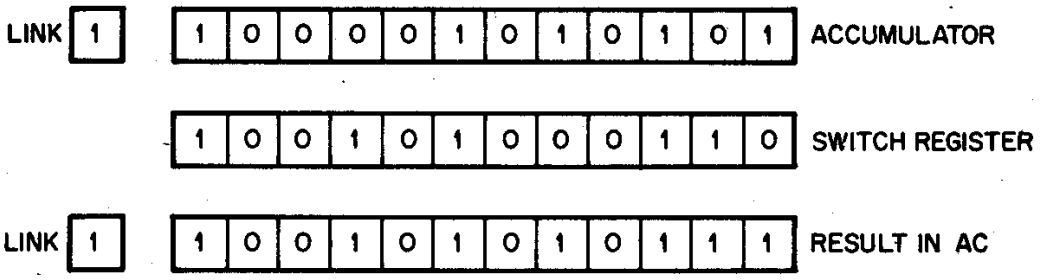
SNA — *Skip on nonzero accumulator.* If bit 6 is a 1 and bit 8 is a 1 also, the next instruction is skipped if the accumulator is not zero.

SNL — *Skip on nonzero link.* If bit 7 is a 1 and bit 8 is a 0, the next instruction is skipped when the link bit is a 1.

SZL — *Skip on zero link.* If bit 7 is a 1 and bit 8 is a 1, the next instruction is skipped when the link bit is a 0.

SKP — *Unconditional skip.* If bit 8 is a 1 and bit 5, 6 and 7 are all zeros, the next instruction is skipped. (Bit 8 is a reverse sensing bit when bits 5, 6 or 7 are used — see SMA, SPA, SZA, SNA, SNL, and SZL above.)

OSR — *Inclusive OR of switch register with AC.* If bit 9 is a 1, an inclusive OR operation is performed between the content of the accumulator and the console switch register. In short, the inclusive OR operation consists of the comparison of the corresponding bit positions of the two numbers and the insertion of a 1 in the result if a 1 appears in the corresponding bit position in *either* number. The action of the instruction is illustrated below.



HLT *Halt.* If bit 10 is a 1, the computer will stop at the conclusion of the current machine cycle.

A summary of Group 2 instructions, including their octal representation, is given in the following table.

Mnemonic	Octal	Operation	Sequence
CLA	7600	Clear the accumulator	2
SMA	7500	Skip on minus accumulator	1
SPA	7510	Skip on positive accumulator (or AC = 0)	1
SZA	7440	Skip on zero accumulator	1
SNA	7450	Skip on nonzero accumulator	1
SNL	7420	Skip on nonzero link	1
SZL	7430	Skip on zero link	1
SKP	7410	Skip unconditionally	1
OSR	7404	Inclusive OR, switch register with AC	3
HLT	7402	Halts the program	3

MICROPROGRAMMING

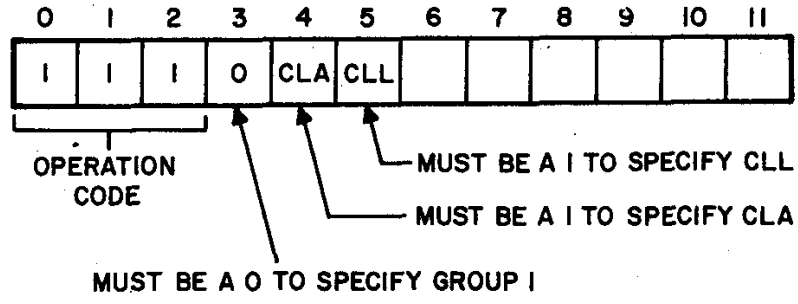
Because PDP-8 instructions of Group 1 and Group 2 are determined by bit assignment, these instructions may be combined, or microprogrammed, to form new instructions enabling the computer to do more operations in less time.

Combining Microinstructions

The programmer should make certain that the program clears the accumulator and link before any arithmetic operations are performed. To perform this task, the program might include the following instructions (given in both octal and mnemonic form).

CLA	7200 (octal)
CLL	7100 (octal)

However, when the Group 1 instruction format is analyzed, the following is observed.



Since the CLA and the CLL instructions occupy separate bit positions, they may be expressed *in the same instruction*, thus combining the two operations into one instruction. This instruction would be written as follows.

CLA CLL 7300 (octal)

In this manner, many operate microinstructions can be combined making the execution of the program much more efficient. The assembler for the PDP-8 will combine the instructions properly when they are written as above, that is, on the same coding line, and separated by a space.

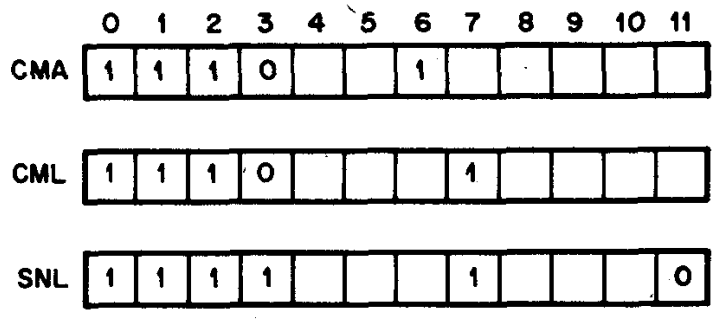
Illegal Combinations

Microprogramming, although very efficient, can also be troublesome for the new programmer. There are many violations of coding which are not executable.

One rule to remember is: "If you can't code it, the computer can't do it." In other words, the programmer could write a string of mnemonic microinstructions, but unless these microinstructions can be coded correctly in octal representation, they cannot be performed. To illustrate this fact, suppose the programmer would like to complement the accumulator (CMA), complement the link (CML), and then skip on a nonzero link (SNL). He could write the following.

CMA CML SNL

These instructions require the following bit assignments.



The three microinstructions cannot be combined in one instruction because bit 3 is required to be a 0 and a 1 simultaneously. Therefore, no

instructions may be used which combine Group 1 and Group 2 microinstructions because bit 3 usage is not compatible. The CMA and CML can, however, be combined because their bit assignments are compatible. The combination would be as follows.

```
CMA CML                    7060 (octal)
```

To perform the original set of three operations, two instructions are needed.

```
CMA CML                    7060 (octal)
SNL                        7420 (octal)
```

Because Group 1 and Group 2 microinstructions cannot be combined, the commonly used microinstruction CLA is a member of both groups. Clearing the AC is often required in a program and it is very convenient to be able to microprogram the CLA with the members of both groups.

The problem of bit assignment also arises when some instructions within a group are combined. For example, in Group 1 the rotate instructions specify the number of places to be rotated by the state of bit 10. If bit 10 is a 0, rotate one place; if bit 10 is a 1, rotate two places. Thus, the instruction RAL can not be combined with RTL because bit 10 would be required to have two different values at once. If the programmer wishes to rotate right three places, he must use two separate instructions.

```
RAR                        7010 (octal)
RTR                        7012 (octal)
```

Although he can write the instruction "RAR RTR", it cannot be correctly converted to octal by the assembler because of the conflict in bit 10; therefore, it is illegal.

Combining Skip Microinstructions

Group 2 operate microinstructions use bit 8 to determine the instruction specified by bits 5, 6, and 7 as previously described. If bit 8 is a 0, the instructions SMA, SZA, and SNL are specified. If bit 8 is a 1, the instructions SPA, SNA, and SZL are specified. Thus, SMA cannot be combined with SZL because of the opposite values of bit 8. The skip condition for combined microinstructions is established by the skip conditions of the individual microinstructions.

OR GROUP — SMA *OR* SZA *OR* SNL

If bit 8 is a 0, the instruction skips on the logical OR of the conditions specified by the separate microinstructions. The next instruction is skipped if *any* of the stated conditions exist. For example, the combined microinstruction SMA SNL will skip under the following conditions:

1. The accumulator is negative, the link is zero.
2. The link is nonzero, the accumulator is not negative.
3. The accumulator is negative and the link is nonzero.

(It will not skip if all conditions fail.) This manner of combining the test conditions is described as the logical OR of the conditions.

AND GROUP — SPA *AND* SNA *AND* SZL

A value of bit 8 = 1 specifies the group of microinstructions SPA, SNA, and SZL which combine to form instructions which act according to the

logical AND of the conditions. In other words, the next instruction is skipped only if *all* conditions are satisfied. For example, the instruction SPA SZL will cause a skip of the next instruction only if the accumulator is positive and the link is zero. (It will not skip if either of the conditions fail.)

NOTES

1. The programmer is not able to specify the manner of combination. The SMA, SZA, SNL conditions are always combined by the logical OR, and the SPA, SNA, SZL conditions are always joined by a logical AND.
2. Since the SPA microinstruction will skip on either a positive or a zero accumulator, to skip on a strictly (positive, nonzero) accumulator the combined microinstruction SPA SNA is used.

Order of Execution of Combined Microinstructions

The combined microinstructions are performed by the computer in a very definite sequence. When written separately, the order of execution of the instructions is the order in which they are encountered in the program. In writing a combined instruction of Group 1 or Group 2 microinstructions, the order written has no bearing upon the order of execution. This should be clear, because the combined instruction is a 12-bit binary number with certain bits set to a value of 1. The order in which the bits are set to 1 has no bearing on the final execution of the whole binary word.

The definite sequence, however, varies between members of the PDP-8 computer family. The sequence given here applies to the PDP-8/I and PDP-8/L. The applicable information for other members of the PDP-8 family is given in the Appendix. The order of execution for PDP-8/I and PDP-8/L microinstructions is as follows.

GROUP 1

Event 1 CLA, CLL — Clear the accumulator and/or clear the link are the first actions performed. They are effectively performed simultaneously and yet independently.

Event 2 CMA, CML — Complement the accumulator and/or complement the link. These operations are also effectively performed simultaneously and independently.

Event 3 IAC — Increment the accumulator. This operation is performed third allowing a number in the AC to be complemented and then incremented by 1, thereby forming the two's complement, or negative, of the number.

Event 4 RAR, RAL, RTR, RTL — The rotate instructions are performed last in sequence. Because of the bit assignment previously discussed, only one of the four operations may be performed in each combined instruction.

GROUP 2

Event 1 *Either* SMA *or* SZA *or* SNL when bit 8 is a 0. *Both* SPA *and* SNA *and* SZL when bit 8 is a 1. Combined microinstructions specifying a skip test are performed first. The microinstructions are *combined* to form one specific test, therefore, skip instructions are effectively performed simultaneously.
Because of bit 8, only members of one skip group may be combined in an instruction.
The actual skip however, will not occur until all other directives of the group 2 instruction are performed.

Event 2 CLA — Clear the accumulator. This instruction is performed second in sequence thus allowing different arithmetic operations to be performed after testing (see Event 1) without the necessity of clearing the accumulator with a separate instruction before some subsequent arithmetic operation.

Event 3 OSR — Inclusive OR between the switch register and the AC. This instruction is performed third in sequence, allowing the AC to be cleared first, and then loaded from the switch register.

Event 4 HLT — The HLT is performed last to allow any other operations to be concluded before the program stops.

This is the order in which all *combined* instructions are performed. In order to perform operations in a different order, the instructions must be written separately as shown in the following example. One might think that the following combined microinstruction would clear the accumulator, perform an inclusive OR between the SR and the AC, and then skip on a nonzero accumulator.

```
CLA OSR SNA
```

However, the instruction would not perform in that proper manner, because the SNA would be executed first. In order to perform the skip last, the instructions must be separated as follows.

```
CLA OSR
SNA
```

Microprogramming requires that the programmer carefully code mnemonics legally so that the instruction does in fact do what he desires it to do. The sequence in which the operations are performed and the legality of combinations is crucial to PDP-8 programming.

The following is a list of commonly used combined microinstructions, some of which have been assigned a separate mnemonic.

Instruction		Explanation
—	CLA CLL	Clear the accumulator and link.
CIA	CMA IAC	Complement and increment the accumulator. (Sets the accumulator equal to its own negative.)

LAS	CLA OSR	Load accumulator from switches (Loads the accumulator with the value of the switch register.)
STL	CLL CML	Set the link (to a 1).
—	CLA IAC	Sets the accumulator to a 1.
STA	CLA CMA	Sets the accumulator to a −1.

In summary, the basic rules for combining operate microinstructions are given below.

1. Group 1 and Group 2 microinstructions cannot be combined.
2. Rotate microinstructions (Group 1) cannot be combined with each other.
3. OR Group (SMA, SZA, or SNL) microinstructions cannot be combined with AND Group (SPA, SNA, or SZL) microinstructions.
4. OR Group microinstructions are combined as the logical OR of their respective skip conditions. AND Group microinstructions are combined as the logical AND of their respective skip conditions.
5. Order of execution for combined instructions (PDP-8/I and PDP-8/L only) is listed below.

Group 1	Group 2
1. CLA, CLL	1. SMA/SZA/SNL or SPA/SNA/SZL
2. CMA, CML	2. CLA
3. IAC	3. OSR
4. RAR, RAL, RTR, RTL	4. HLT

PDP-8 MODE INPUT/OUTPUT PROGRAMMING

Being able to program a computer to do calculations is of little use if there is no way of getting the results of calculations from the machine. Likewise, the programmer often must supply the computer with information to be processed. A programmer must be provided with the means to transfer information between the computer and the peripheral devices that supply input or that serve as a means of output.

Before a transfer of information can be executed, a control function must be supplied to specify when the exchange will occur, with what peripheral device the exchange will occur, and where in core storage the information will be stored (or obtained from). In general, this control function may be served by either the PDP-8 or the peripheral device itself.

There are three basic methods for the transfer of information between input/output (I/O) devices and the PDP-12. The first two methods provide for computer control over the transfer. One method is programmed transfer, in which instructions are included at some point in the program to accept or transmit information. Thus, programmed transfers are program initiated and are under program control.

Information may also be transferred through *program interrupt*, a standard feature of the PDP-8 computer family that provides for devices to signal the PDP-12 when they are ready to transfer information; the pro-

gram will then interrupt its normal flow and jump to a routine to process the information, after which it will return to the point in the main program at which it was interrupted. Thus, program interrupt transfers are device initiated but are under program control.

These first two methods (i.e., programmed transfers and program interrupt) use the accumulator as the *buffer*, or storage area in the computer, for all data transfers. Therefore, only one 12-bit word of input or output may be transferred at one time by a programmed transfer, or by program interrupt.

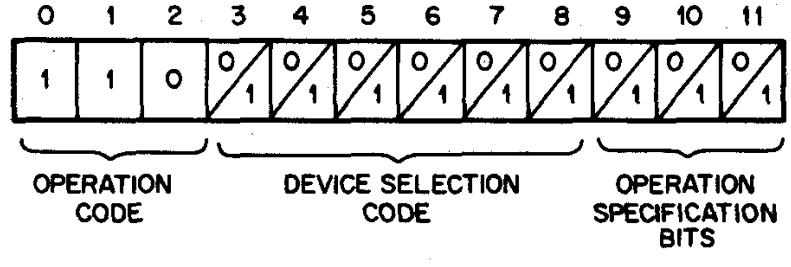
The third method of information transfer is *data break*, standard for the PDP-12 computer. Data break is essentially device controlled and allows for direct exchange of large quantities of information between the device and the PDP-8 memory. It differs from the previous two types of transfer in that there are no program instructions to handle the transfer and the accumulator is not used as a buffer. Data break transfers are device initiated and device controlled.

INPUT/OUTPUT INSTRUCTIONS

As the name implies, programmed transfers of information are accomplished with a set of program instructions. The instructions are similar to the operate microinstructions in that there is no need to specify an address in memory. The operation code 6_8 is used to specify an input/output transfer (IOT) instruction. All programmed transfers are between the accumulator and the device. Since many different devices could be connected to one computer and each device may at some time transfer information, the instruction must identify the proper device for each transfer. The instruction must also specify the exact nature of the function to be performed.

IOT Instruction Format

An IOT instruction is a 12-bit word that is in the following format. The first three bits represent the operation code 6_8. The remaining nine bits may be either binary 0's or 1's.



The IOT Instruction

The IOT instruction is divided into three parts: operation code, device selection code, and operation specification bits.

Device Selection

The device selection code is transmitted to *all* peripheral equipment whenever the IOT instruction is executed. A *device selector* within each

peripheral device monitors the device codes. When the device selector recognizes a device code as the device's assigned code, the device receives the last three bits of the instruction. Each of the last three bits specifies an action associated with the device. When one of the last three bits is set to a 1, the specified action is performed. Since there are three bits, only three different actions can be specified for each device code, although microprogramming is possible. When more instructions are necessary for a given device, more than one code is assigned to the device.

Checking Ready Status

Because there is a great difference in the processing speed of a computer and the speed of most peripheral devices, the computer must check the readiness of a device before any transfer of information is performed. The input device must signal the computer that it has completely assembled the information and is now ready to transfer the information to the computer memory. The output device must signal its readiness to accept the next piece of information from the computer. Without such signals, the computer would input and output information at a faster rate than the device could process it and some information would be lost.

To prevent any loss of information, the computer program checks the ready status of the transmitting or receiving device as part of preparing for a normal data transfer. The ready status is usually checked with a skip instruction such that if the device is ready, the following instruction is skipped. The ready status is signaled through a system of *flags*, which are 1-bit registers within the device. All I/O devices have a device flag which is set to a 1 when the device is ready; that is, when it can be used (if it is an output device), or when it has information (if is an input device). If the flag is cleared (set to 0), the device is *busy*. If a program initiates a device action, the flag associated with that device will be set to a 1 when the device action is completed.

Instruction Uses

In general, for each device there are three instructions:

1. An instruction to transfer information and operate the device.
2. An instruction to test the ready status of the device and skip on the ready (or not-ready) status of the device.
3. An instruction to clear the device flag.

The above instructions may be microprogrammed. In particular, the instructions to clear the flag and to operate the device often are combined.

The specific instructions for devices are given in the following sections. The Teletype unit is described in depth to explain the fundamentals of programming data transfers. The general techniques developed for the Teletype unit may be extended to handle other devices.

ASCII Code

The ASCII (U.S.A. Standard Code for Information Interchange) is presented below. Many of the programs written in this chapter use this code to transmit information to the PDP-12. The fact that the ASCII code for the octal digits 0 through 7 is the sum of that digit plus 260_8 should be observed.

THE 8-BIT ASCII[1] CODE

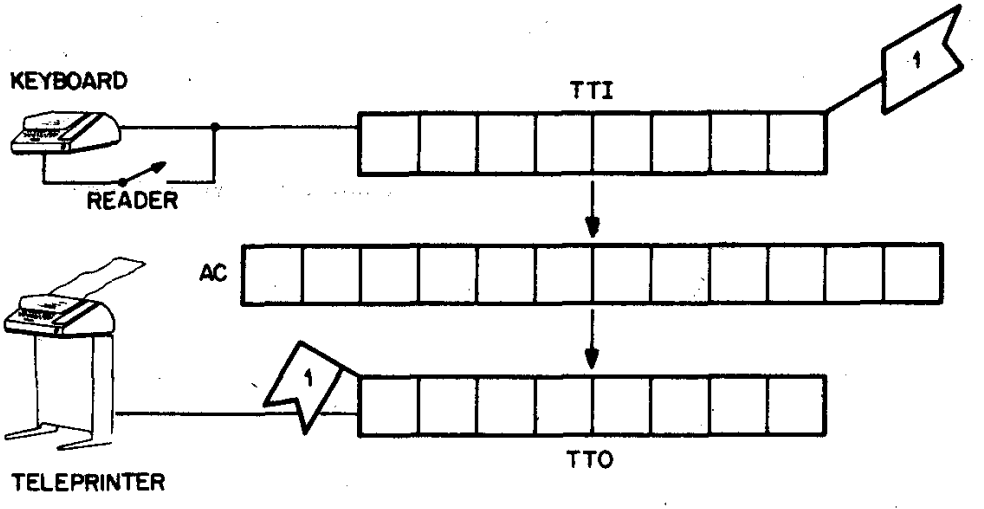
Character	8-Bit Octal	Character	8-Bit Octal
A	301	!	241
B	302	"	242
C	303	#	243
D	304	$	244
E	305	%	245
F	306	&	246
G	307	'	247
H	310	(	250
I	311	)	251
J	312	*	252
K	313	+	253
L	314	,	254
M	315	−	255
N	316	.	256
O	317	/	257
P	320	:	272
Q	321	;	273
R	322	<	274
S	323	=	275
T	324	>	276
U	325	?	277
V	326	@	300
W	327	[	333
X	330	\	334
Y	331	]	335
Z	332	↑	336
0	260	←	337
1	261	Leader/Trailer	200
2	262	LINE FEED	212
3	263	Carriage RETURN	215
4	264	SPACE	240
5	265	RUBOUT	377
6	266	Blank	000
7	267	BELL	207
8	270	TAB	211
9	271	FORM	214

[1]An abbreviation for USA Standard Code for Information Interchange.

PROGRAMMING THE TELETYPE UNIT

One of the most common I/O devices is the Teletype unit, which contains a keyboard, printer, paper tape reader, and paper tape punch.

The following is a diagram of how the teletype is connected to the PDP-12.



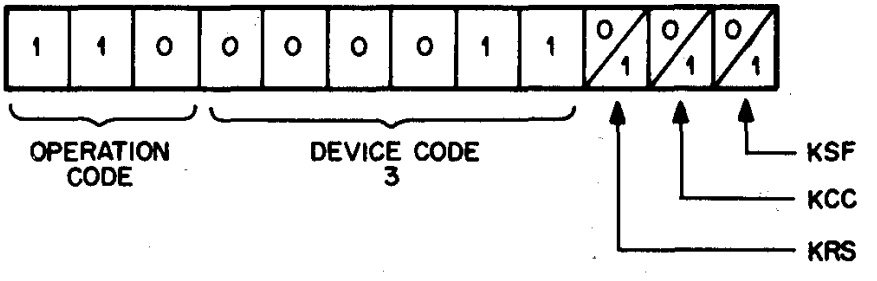
The Teletype unit can use either the keyboard or the paper tape reader to input information to the computer and can use either the printer or the paper tape punch to accept output information from the computer. The Teleype unit is therefore assigned two device codes.

Teletype Input/Output Transfer Instructions

Functioning as an input device, the keyboard/reader is assigned the device code 03_8, and functioning as an output device, the printer/punch is assigned the device code 04_8.

KEYBOARD/READER INSTRUCTIONS

The instruction format for the keyboard/reader is shown below. The mnemonic instructions generated by bits 9, 10, and 11 are noted. The sequence in which the mnemonic instructions are executed when microprogrammed is noted below.



Teletype Keyboard/Reader Instructions

Sequence	Mnemonic	Octal	Effect
1	KSF	6031	Skip the next instruction when the keyboard buffer register is loaded with an ASCII symbol (causing the keyboard flag to be raised).
2	KCC	6032	Clear AC, clear keyboard flag.
3	KRS	6034	Transfer the contents of the keyboard buffer into the AC.
2,3	KRB	6036	Transfer the contents of the keyboard buffer into the AC, clear the keyboard flag.

The fourth instruction (KRB) is a microprogrammed combination of the mnemonics KCC and KRS. If the paper tape reader is loaded with a paper tape and switch to START, the KRB instruction accepts one character from the reader.

A program using the above instructions to read in one ASCII character from the keyboard or paper tape reader is shown below. Note that this program does *not* type the character on the teleprinter, it merely stores the ASCII code for the character in the location STORE.

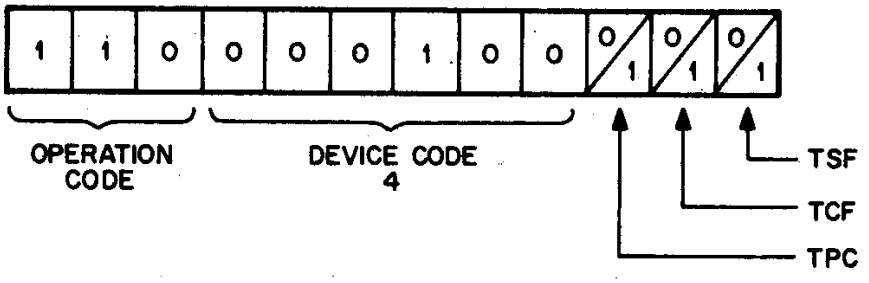
```
*200
INPUT,      KCC                 /CLEAR KEYBOARD FLAG
            JMS LISN
            DAC STORE
            HLT
LISN,       0
            KSF                 /SKIP ON KEYBOARD FLAG
            JMP .-1
            KRB                 /READ KEYBOARD BUFFER
            JMP I LISN
STORE,      0
$
```

Coding to Accept One ASCII Character

The main program begins with KCC. In general, the main program should begin by clearing the flags of all devices to be used later in the program. If the above program is started at location 200, it will proceed to the KSF, JMP .−1 loop, and stay in this loop endlessly until a key on the Teletype unit is pressed or a paper tape is loaded into the reader. When the ASCII code for the character is assembled in the keyboard/reader buffer register, the flag will be set to a 1 and the program will skip out of the loop. The contents of the buffer will be transferred into the accumulator, and the buffer and flag will be cleared.

PRINTER/PUNCH INSTRUCTIONS

The instruction format for the Teletype printer/punch IOT instructions is given below. The mnemonic instructions generated by pits 9, 10, and 11 are discussed on the following page.



Teletype Printer/Punch Instructions

Sequence	Mnemonic	Octal	Effect
1	TSF	6041	Skip the next instruction if the printer flag is set to 1.
2	TCF	6042	Clear the printer flag.
3	TPC	6044	Load the printer buffer register with the contents of the AC, select and print the character. (The flag is raised when the action is completed.)
2,3	TLS	6046	Clear the printer flag, transfer the contents of the AC into the printer buffer register, select and print the character. (The flag is raised when the action is completed.)

The last instruction is a microprogrammed combination of TPC and TCF, such that the flag is cleared, the character is printed, and then the flag is again raised. Whenever the paper tape punch is turned on, the character is punched on paper tape as well as printed on the teleprinter.

The chart below illustrates a program to print out one ASCII character which is held in a memory location.

```
*200
OUTPUT,     CLA CLL
            TLS
            TAD HOLD
            JMS TYPE
            HLT
TYPE,       0
            TSF              /SKIP ON TELEPRINTER FLAG
            JMP .-1
            TLS              /PRINT THE CHARACTER
            CLA CLL
            JMP I TYPE
HOLD,       0
$
```

Coding to Print One ASCII Character

The program shown above begins by clearing the accumulator and executing a TLS instruction (which has the effect of clearing the printer buffer), after which the printer flag will be set, thereby signifying readiness to accept a character. If the initial TLS instruction were not executed, the flag would not be raised (the START key clears all flags), and the program would remain in the +SF, JMP .— 1 loop endlessly. In the previous case, however, the program uses the printer with a cleared accumulator such that no character is printed. However, the flag is set when this action is complete enabling the printing of meaningful information in the TYPE subroutine. The TYPE subroutine clears the accumulator since the TLS instruction does not. It is advisable to clear the accumulator after any subroutine unless meaningful data is contained in it.

Format Routines

Input and output routines are very often written in the form of subroutines, as the TYPE subroutine in the previous example. The example shown below is a carriage return/line feed subroutine that calls the TYPE subroutine to execute a carriage return and line feed on the printer, thus advancing to a new line for the printing of information.

```
CRLF,   0
        TAD K215
        JMS TYPE
        TAD K212
        JMS TYPE
        JMP I CRLF
K215,   215             /ASCII FOR CARRIAGE RETURN
K212,   212             /ASCII CODE FOR A LINE FEED
TYPE,   0
        TSF
        JMP .—1
        TLS
        CLA CLL
        JMP I TYPE
```

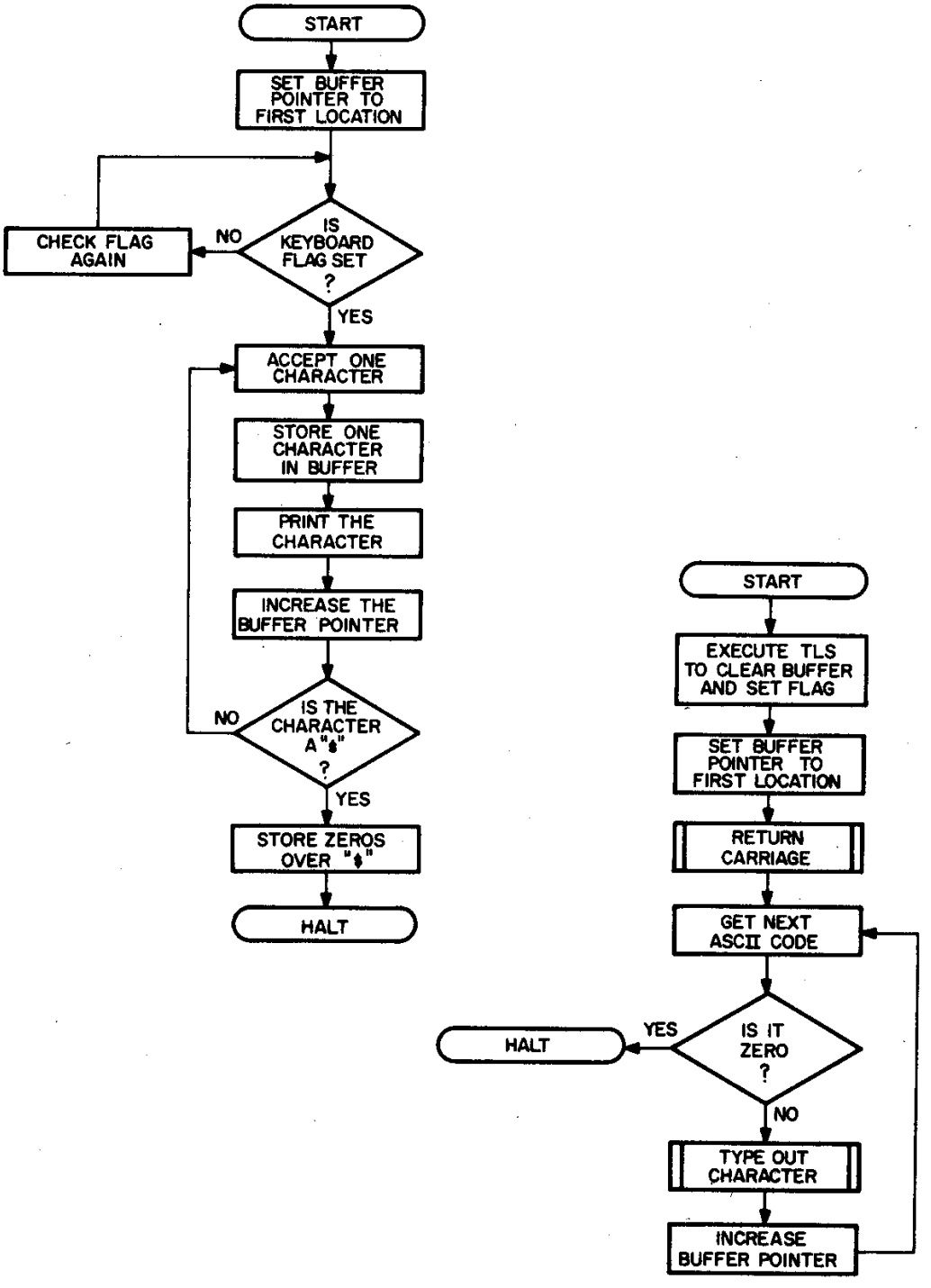
Carriage Return/Line Feed Subroutine

Subroutines similar to the one shown above could be written to tab space the carriage a given number of spaces, or to ring the bell of Teletype Model ASR 33 by using the respective codes for these nonprinting characters. Such subroutines, if commonly used in a program, should be placed on page 0 (or else a pointer word to the subroutine should be placed on page zero) to facilitate reaching the routine from all memory locations.

Text Routines

The examples shown below may be expanded to accept and type more than one character. Figures on the following page illustrate one expansion. These two programs are compatible in that the characters accepted by the first program are typed out by running the second program. The program to accept characters will continue to accept character input until a dollar sign ($) is struck on the keyboard, at which time the program will store all zeros in the next location and then halt. The program

will type the characters whose ASCII code was stored by the first program. The second program will halt when a location with contents equal to zero is reached. Both programs use locations beginning with 2000 as the buffer for the storage of ASCII characters. The following flowcharts introduce the techniques used in the program coding.



```
*5000
START,  CLA CLL
        TAD BUFF        /SET UP BUFFER SPACE.
        DCA BUFFPT
LISN,   KSF             /KEYBOARD STRUCK YET?
        JMP .-1         /NO: CHECK AGAIN.
        KRB             /YES: READ CHARACTER.
        TLS             /ACKNOWLEDGE IT ON  PRINTER.
        DCA I BUFFPT    /STORE CHARACTER.
        TAD I BUFFPT    /CHECK FOR TERMINAL $.
        TAD MDOLAR
        SNA
        JMP DONE        /CHARACTER IS A $.
        ISZ BUFFPT      /CHARACTER IS NOT A $.
        JMP LISN        /GET ANOTHER CHARACTER.
DONE,   CLA CLL         /STORE 0 IN LAST LOCATION.
        DCA I BUFFPT
        HLT
BUFF,   2000
BUFFPT, 0
MDOLAR, 7534
$
```

```
*5200
START,  CLA CLL
        TLS             /TLS TO SET PRINTER FLAG.
        TAD BUFF        /SET UP BUFFER SPACE.
        DCA BUFFPT
        JMS CRLF        /RETURN CARRIAGE.
CHRTYP, TAD I BUFFPT    /GET A CHARACTER.
        SNA             /IS IT ALL ZEROS?
        HLT             /YES: STOP.
        JMS TYPE        /NO: TYPE OUT THE CHARACTER.
        ISZ BUFFPT      /INCREMENT BUFFER POINTER.
        JMP CHRTYP      /TYPE ANOTHER CHARACTER.
CRLF,   0               /CARRIAGE RETURN & LINE FEED.
        TAD K215        /TYPE CR FIRST.
        JMS TYPE
        TAD K212        /TYPE LINE FEED.
        JMS TYPE
        JMP I CRLF
TYPE,   0               /SUBROUTINE TO TYPE CHARACTER.
        TSF             /PRINTER READY YET?
        JMP .-1         /NO: CHECK AGAIN.
        TLS             /YES: TYPE CHARACTER.
        CLA             /CLEAR  ASCII FROM AC.
        JMP I TYPE
BUFF,   2000
BUFFPT, 0
K215,   215
K212,   212
$
```

The program to print characters may be specialized to print a specific word. The example is a subroutine which uses autoindex registers in place of the ISZ instruction. The subroutine types "HELLO!"

```
HELLO,  0                   /HELLO SUBROUTINE
        CLA CLL
        TLS                 /TLS TO SET PRINTER FLAG.
        TAD CHARAC          /SET UP INDEX REGISTER
        DCA IRI             /FOR GETTING CHARACTERS.
        TAD M6              /SET UP COUNTER FOR
        DCA COUNT           /TYPING CHARACTERS.
NEXT,   TAD I IRI           /GET A CHARACTER.
        JMS TYPE            /TYPE IT.
        ISZ COUNT           /DONE YET?
        JMP NEXT            /NO: TYPE ANOTHER.
        JMP I HELLO         /YES: RETURN TO MAIN PROGRAM.
TYPE,   0                   /TYPE SUBROUTINE
        TSF
        JMP .-1
        TLS
        CLA
        JMP I TYPE
CHARAC, .                   /USED AS INITIAL VALUE OF IRI
        310                 /H
        305                 /E
        314                 /L
        314                 /L
        317                 /O
        241                 /!
M6,     -6
COUNT,  0
IR1=10
```

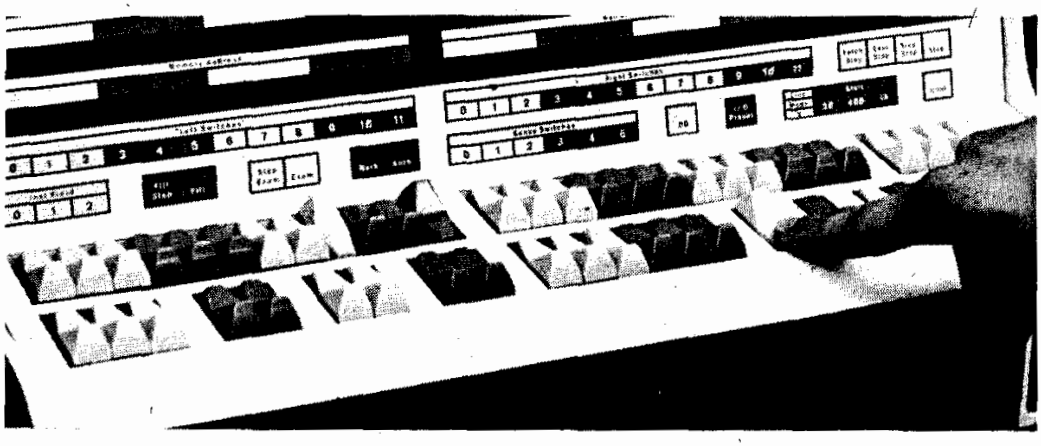
LINC MODE INSTRUCTIONS

Simple Instructions

LINC mode instructions themselves are encoded as binary numbers and held in various registers. The simplest of these instructions, namely those which operate only on the accumulator, are described first with reference to the LEFT SWITCHES.

Raising the DO lever (DO means "do toggle instruction") causes the PDP-12 to execute the instruction whose binary code number is held in the LEFT SWITCHES. The PDP-12 then halts. For example, if the LEFT

SWITCHES are set to the code number for the instruction CLEAR, which happens to be 0011_8, and the DO lever is then momentarily depressed, the accumulator lights all go out as does the LINK bit light, so that C(ACC) = 0, and C(L) = 0. In setting a switch, down corresponds to 1.



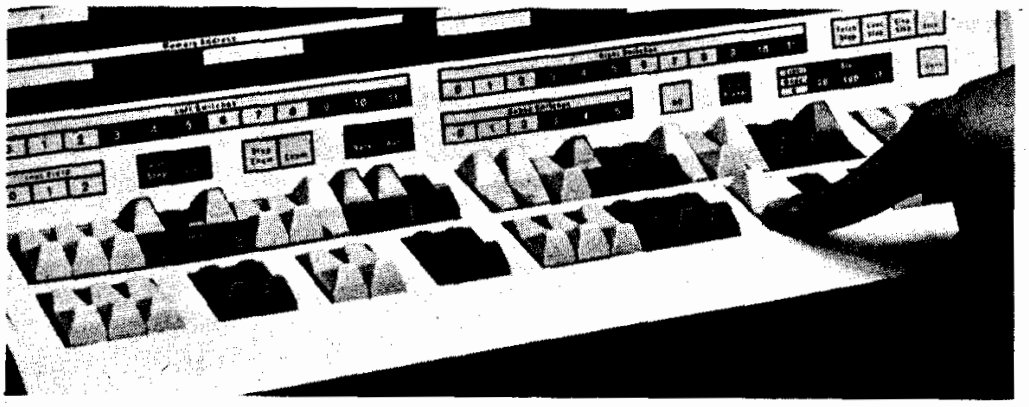
Briefly: If C(LEFT SWITCHES) = 0111_8, DO has the effect 0→C(ACC) and 0→C(L). *(Read "0 replaces the contents of the accumulator." etc.)*

Clear (or CLR) is an instruction of the class known as miscellaneous instruction. A second miscellaneous class instruction, COM (complement), with the code number 0017_8, directs the LINC to complement the contents of the accumulator and therefore has the effect C(ACC)→C(ACC). (Read: "the complement of the contents of the accumulator replaces the contents of the accumulator.")

Two other instructions of this class transfer information between the accumulator and the relay register. The relay register, displayed on the control console, operates six relays which can be used to control or run external equipment. An instruction with the code 0014_8, called ATR (accumulator to relay), directs the LINC to transfer the contents of the right half of the accumulator, i.e., the rightmost six bits, into the relay register. The accumulator itself is not changed when the instruction is executed. Another instruction, called RTA (relay to accumulator), 0015_8, causes the LINC to clear the accumulator and then transfer the contents of the relay register into the right half of the accumulator. In this case the relay register is not changed and the left half of the accumulator remains cleared (i.e., contains 0's).

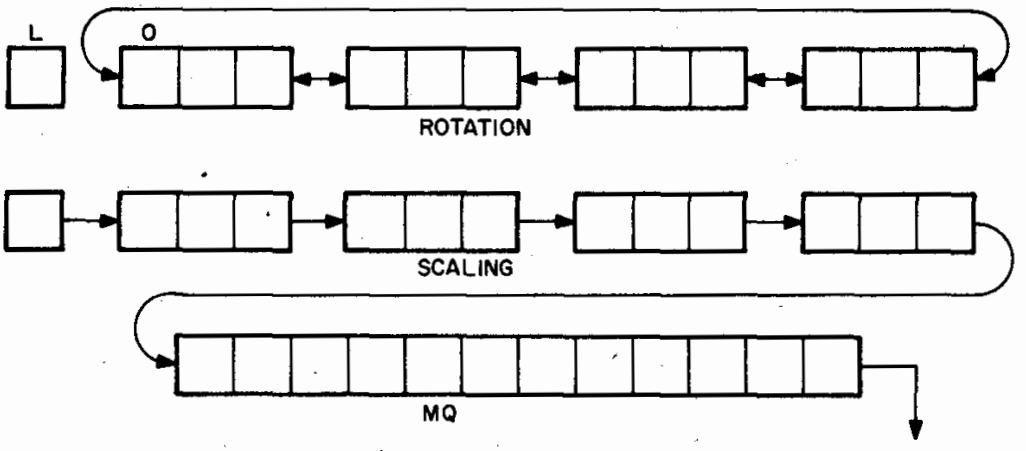
Another instruction called RSW (right switches), 0516_8, directs the PDP-12 to copy the contents of the RIGHT SWITCHES into the accumulator. By setting the LEFT SWITCHES to 0516_8, the RIGHT SWITCHES to whatever value wanted in the accumulator, and then momentarily depressing the DO lever, the operator can change the contents of the accumulator to

any desired new value. The drawing shows how the switches should be set to put the number 6451_8 into the accumulator:



Shifting

After a number has been put into the accumulator it can be repositioned (shifted) to the right or left. There are two ways of shifting: rotation, in which the end-elements of the accumulator are connected together to form a closed ring, and scaling, in which the end-elements are not so connected.



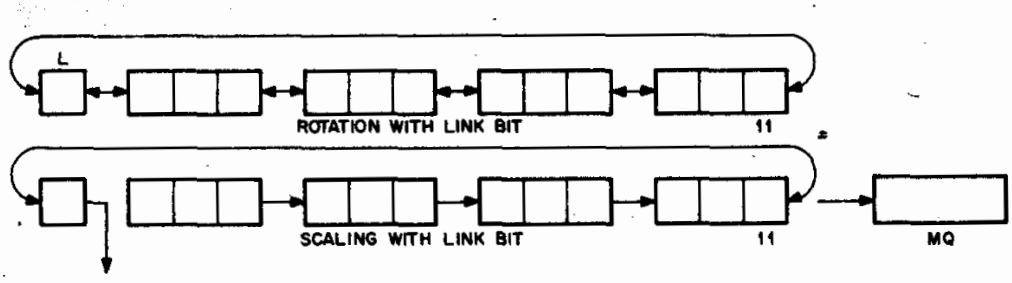
Examples of shifts of one place:

	Effect of rotating *right 1 place*	Effect of scaling *right 1 place*
Before	000 000 011 001	000 000 011 001 = +25 (decimal)
After	100 000 001 100	000 000 001 100 = +12
Before	111 111 100 110	111 111 100 110 = −25 (decimal)
After	011 111 110 011	111 111 110 011 = −12

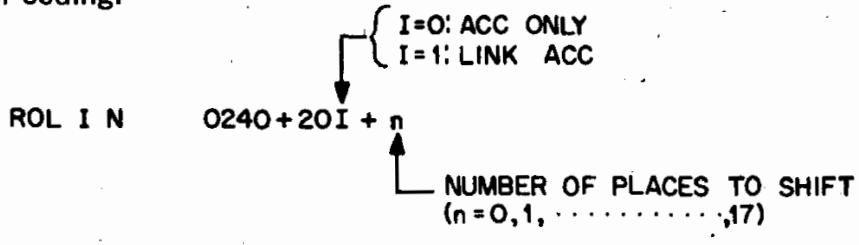
Note that, in scaling, bits are lost to the right, which amounts to an error of rounding off; the original sign is preserved in the sign bit and replicated in the bit positions to the right of the sign bit. This has the effect of reducing the size of the number by powers of two (analogous to moving the decimal point in decimal calculations).

The PDP-12 LINC mode has three instructions, called the shift class instructions, which shift the contents of the accumulator: rotate right, rotate left, and scale right. Unlike the simple instructions considered so far, the code number for a shift class instruction includes a variable element which specifies the number of places to shift. For example, write ROL n (rotate the contents of the accumulator n places to the left), where n can be any number from $0\text{-}17_8$.

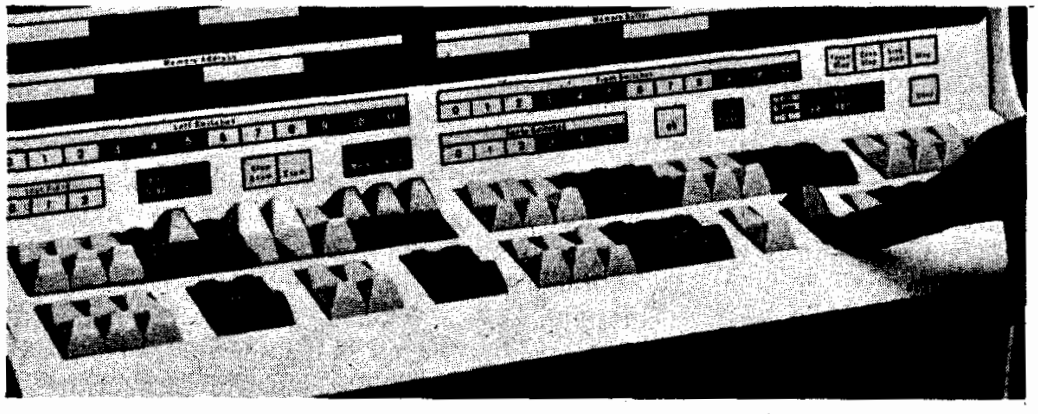
As a further variation of the shift class instructions, the link bit can be adjoined to the accumulator during rotation to form a 13-bit ring as shown below, or to bit 11 of the accumulator during scaling to preserve the low order bit scaled out of the accumulator:



The code number of a shift class instruction, e.g., rotate left, therefore includes the number of places to shift and an indication of whether or not to include the link bit. Use the full expression ROL I n, which has the octal coding:



so that, for example, rotate ACC left 3 places has the code 0243, and rotate ACC with link left 7 places has the code 0267. Note the correspondence between the code terms and bit positions of the binary coded instruction as it appears, for example, in the LEFT SWITCHES:



Similar coding is used with ROR I N (rotate right), 300 + 20I + N, and SCR I N (scale right), 340 + 20I + N.

LINC Mode Memory and Memory Reference Instructions

Before proceeding to other instructions, it is necessary to introduce the LINC Mode memory. This memory is to be regarded as a set of 1024_{10} registers each holding 12-bit binary numbers in the manner of the accumulator. These memory registers are numbered 0, 1, ..., 1023_{10}, or 0, 1,.... 1777_8, and reference is made to "contents of register, C(3), "the contents of register X," C(X), etc., referring to "3" and "X" as memory address.

The memory actually consists of a remotely-located array of magnetic storage elements with related electronics, but for introductory purposes view of terms of two registers of lights, namely the memory address register and the memory buffer register:

By using these two registers in conjunction with the LEFT SWITCHES it is possible to find out what values the memory registers contain. For example, to find the contents of register 3, set the LEFT SWITCHES to memory address 0003 and then operate the key labeled EXAM. As 0003 appears in the memory address register, the contents of register 3 appear in the memory buffer register. By setting the LEFT SWITCHES to a memory address register, the contents of register 3 appear in the memory buffer register. By setting the LEFT SWITCHES to a memory address and pushing EXAM, the contents of any register in the LINC memory may be examined.

The contents of any selected memory register may be changed by using both the LEFT and RIGHT SWITCHES and the key marked FILL. For example to make the memory register whose address is 700 contain −1 (i.e., 7776_8) set memory address 0700 into the LEFT SWITCHES. Set the RIGHT SWITCHES to 7776 and operate the FILL key. A 0700 appears in the memory address register and 7776 appears in the memory buffer register, indicating that the contents of the register 700 are now 7776. What ever value register 700 may have contained before FILL was pushed is lost, and the new value takes its place. In this way any register in the LINK MODE memory can be filled with a new number.

None of the mode instructions make explicit reference to the memory address register or memory buffer register; rather, in referring to memory register X, an instruction may direct the PDP-12 implicitly to put the address X into the memory address register and the contents of register X, C(X), into the memory buffer register.

The Store-Clear Instruction (400 + X)

Now it is possible to describe the first of the memory reference instructions, STC X(store-clear X), which has the code number 4000 + X, where $0 \leq X \leq 1777_8$.(From now on only octal numbers will be used for addresses.) Execution of STC X has two effects: 1. the contents of the accumulator are copied into memory register X, C(ACC)→C(X), and 2) the accumulator is then cleared, 0→C(ACC). (The link bit is not cleared.) Thus, for example, if C(ACC) = 0503 and C(671) = 2345, and the code number

for STC 671, i.e., 4671, is set into the LEFT SWITCHES, depressing the DO switch puts 0 into the accumulator and 0503 into register 671. The original contents of register 671 are lost.

It will be clear that the memory can be filled with new numbers at any time either by using the FILL key and the switches, or by loading the accumulator from the RIGHT SWITCHES with the RSW instruction and the DO switch and then storing the accumulator contents with the STC X instruction and the DO switch.

The ADD Instruction and Binary Addition (2000 + X)

STC is one of three full-address class instructions. Another instruction in this class, ADD X, has the code number 2000 + X where $0 \leq X \leq 1777$. Execution of ADD X has the effect of adding the contents of the accumulator, i.e., C(X)+C(ACC)→C(ACC). If the accumulator is first cleared, ADD X has the effect of merely copying to the accumulator the contents of memory register X, i.e., C(X)→C(ACC). In any case, the contents of memory register X are unaffected by the instruction.

The addition itself takes place in the binary system, within the limitations of the 12-bit registers. The basic rules for binary addition are simple: $0 + 0 = 0$; $1 + 0 = 1$; $1 + 1 = 10$ (i.e., zero, with one to carry). A carry arising from the leftmost column (end-carry) is brought around and added into the rightmost column (end-around carry). Some examples (begin at the rightmost column as in decimal addition):

```
001   111   010   001
000   010   111   001
 11   111     1     1 (CARRIES)
010   010   011   010 (SUM)
```

```
      111   100   010   011
      001   010   010   000
 1     11           1
      000   110   100   011
                        ↓
                        1 (END-AROUND
      CARRIES — — — —  11 CARRY)
      000   110   100   100 (SUM)
```

The reader should try some examples of his own. Verify the fact that adding a number to itself with end-around carry is equivalent to rotating left one place. With signed-integer interpretation, some other examples are:

```
000   000   000   101   = +5
111   111   111   100   = −3
111   111   111
000   000   000   001
                    ↓1
                    1
000   000   000   010   = +2
```

```
111   111   111   010   = +5
111   111   111   100   = −3
111   111   11
111   111   110   110
                    ↓1
                    1
111   111   110   111   = −8 (DECIMAL)
```

It can be seen that subtraction of the number N is accomplished by addition of the complement of N. Of course, if either the sum or difference is too large for the accumulator to hold, the result of the addition may not be quite the desired number. For example, adding 1 to the largest

positive integer in this system ($+3777_8$) results in the largest negative integer (-3777_8). This is sometimes called overflowing the capacity of the accumulator.

Instruction Location Register

It is clear that the code numbers of a series of different instructions can be stored in consecutive memory registers. The PDP-12 is designed to execute this stored program of instruction by returning and carrying out each instruction in sequence, using a register called the program counter, to hold the address of the next instruction to be executed. Using the FILL key and the LEFT and RIGHT SWITCHES already discussed can, for example, put into memory registers 20-24 the code numbers for a series of instructions with which divide by 8 the number held in memory register 30 and store the result in memory register 1:

Memory Address		Memory Buffer		Effect
Start	→ 20	CLR	0111	Clear the accumulator
	21	ADD 30	2030	Add the contents of register 30 to the accumulator.
	22	SCR 3	034	Scale C(ACC) right 3 places to divide by 8.
	23	STC 1	4031	Store in register 1.
	24	HLT	0000	Halt the computer.
	30	N	N	Number to be divided by 8.
	31	N/8	N/8	Result.

Simple Sequence of Instructions

Use the FILL Key and the LEFT and RIGHT SWITCHES to put the code numbers for the instructions into memory registers 20-24 and the number to be divided into register 30. Operating the console keys labeled IO Preset and START 20 directs the PDP-12 to begin executing instructions LINC Mode memory register 20. That is, the value 20 replaces the contents of the instruction register. As each instruction of the stored program is executed, the instruction location register is increased by 1, C(P) + 1→C(P). When the instruction location register contains 24, the computer encounters the instruction HLT, code 0000, which halts the machine. To run the program again, merely operate IO Preset and START 20 key. (The code numbers for the instructions stay in memory registers 20-24 unless they are deliberately changed.)

The Jump Instruction (6000 + X)

The last full-address instruction, JMP X, code 6000 + X, has the effect of setting the instruction location register to the value X; X→C(P). That is, the PDP-12, instead of increasing the contents of the instruction location register by one and executing the next instruction in sequence, is directed by the JMP instruction to get its next instruction from memory register X. In the above example having a JUMP to 20 instruction, code 6020, in memory register 24 (in place of HLT) would cause the computer to repeat the program endlessly. If the program were started with the IO Preset and START 20 switch, the instruction location register (P) would

hold the succession of values: 20, 21, 22, 23, 24, 20, 21, etc. (Later instructions will be introduced which increase C(P) by extra amounts, causing it to skip.)

JMP X has one further effect: if JMP 20, 6020, is held in memory register 24, then its execution causes the code for JMP 25 to replace the contents of register 0; i.e., 6025→C(0). More generally, if JMP is in any memory register p, $0 \leq p \leq 1777$, then its execution causes JMP p + 1→C(0).

Memory Address	Memory Buffer		Effect
0	JMP p+1	6000 + P+1	
.	.	.	
→p	JMP X	6000 + X	X C(P), and JMP p+1 C(0).
p + 1	.		
.	.		
X	→–		Next instruction.

This JMP p+1 code replaces the contents of register 0 every time a JMP X instruction is executed unless X=0, in which case the contents of 0 are not changed. Use of memory register 0 in this way is relevant to a programming technique involving subroutines which is described later.

The following programming example illustrates many of the features described so far. It finds one-fourth of the difference between two numbers N_1 and N_2, which are located in registers 201 and 202, and leaves the result in register 203 and in the accumulator. After filling consecutive memory registers 175-210 with the appropriate code and data numbers, the program must be started at memory register 175. Since there is no START 175 key on the console, this is done by setting the LEFT SWITCHES to 4175 and operating the console keys labeled IO Preset and Start LS (start RIGHT SWITCHES).

Memory Address		Memory Buffer		Effect
Start →	175	CLR	0011	0→C(ACC).
	176	ADD 210	2201	N_1→C(ACC).
	177	COM	0017	Forms - N_1 -
	200	JMP 204	6204	Jumps around data; 204→C(P), and JMP 201→C(0).
	201	N_1	N_1	Data and result.
	202	N_2	N_2	
	203	$(N_2-N_1)/4$	$(N_2-N_1)/4$	
	204	→ADD 202	2202	(N_2-N_1)→C(ACC).
	205	SCR 2	0342	Divides by 4.
	206	STC 203	4203	Stores result in 203;→C(ACC). C(203);→0 C(ACC).
	207	ADD 203	2203	Recovers result in ACC.
	210	HLT	0000	Halts the LINC.

Simple Sequence Using the Jump Instruction

In executing this program, the instruction location register holds the succession of numbers: 175, 176, 177, 200, 204, 205, 206, 207, 210.

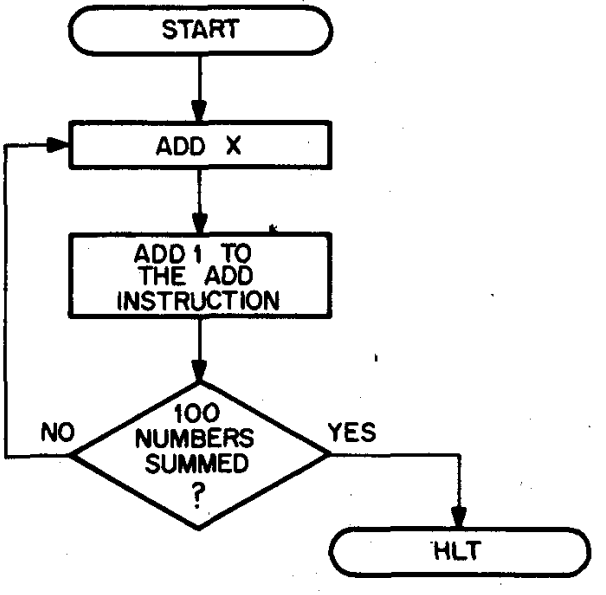
Address Modification and Program Loops

Frequently a program of instructions must deal with a large set of numbers rather than just one or two. For example, suppose one wishes to add 100_8 numbers and that the numbers are stored in the memory in registers 1000-1077. The sum is to go into memory registers 1100. It is possible, of course, to write out all the instructions necessary to do this.

	Memory Address	Memory Buffer		Effect
→	20	CLR	0011	0 C(ACC); 0 C(L).
	21	ADD 1000	3000	Add 1st number.
	22	ADD 1001	3001	Add 2nd number.
	23	ADD 1002	3002	Add 3rd number.
	24	ADD 1003	3003	Add 4th number.
		etc.	etc.	etc.

But it is easy to see that the program would be more than 100_8 registers long. A more complex, but considerably shorter, program can be written using a programming technique known as address modification. In this case the computer first executes an ADD 1000 instruction; the program then adds one to the ADD instruction itself and restores it, so that it is now ADD 1001. The program then jumps back to the location containing the ADD instruction and the computer repeats the entire process, this time executing an ADD 1001 instruction. In short, the program is written so that it changes its own instructions while running.

The process might be diagrammed:



This technique introduces the additional problem of deciding when all 100 numbers have been summed and halting the computer. In this con-

text a new instruction AZE (accumulator zero), code, should be introduced. This is one of a class of instructions known as skip instructions; it directs the PDP-12 to skip the instruction in the next memory register when C(ACC) = ± 0(0000_8 or 7777_8). If C(ACC) ≠ 0, the computer does not skip. For example, if C(ACC) = 7777, and one writes:

Memory Address	Memory Buffer	
p	AZE	0450
p + 1	—	—
p + 2	—	—

the computer takes the next instruction from p + 2. That is, when the AZE instruction in register p is executed, p + 2 replaces the contents of the instruction at p + 1. If C(ACC) ≠ 0, then p + 1→C(P) and the computer executes the next instruction in sequence as usual.

The following example sums the numbers in memory registers 1000-1077 and puts the sum into memory register 1100, using address modification and the AZE instruction to decide when to halt the computer. (Square brackets indicate whose contents change while the program is running.)

Memory Address		Memory Buffer		Effect
	10	ADD 1000	3000	Constants used by program.
	11	1	0001	
	12	−(ADD 1100)	4677	
	.	.	.	
	.	.	.	
Start →	20	CLR	0011	
	21	ADD 10	2010	Code for ADD 1000→C(25).
	22	STC 25	4025	0→C(ACC).
	23	STC 1100	5100	0→C(100), for accumulating sum.
	24	CLR	0011	
	25	(ADD X)	(2000+X)	Clear ACC and add C(X) to C(ACC).
	26	ADD 1100	3100	Sum so far + C(ACC)→C(ACC).
	27	STC 1100	5100	Sum so far C(1100).
	30	ADD 25	2025	ADD X instruction in register 25→C(ACC).
	31	ADD 11	2011	Add 1 to C(ACC) and replace in register 25.
	32	STC 25	4025	
	33	ADD 25	2025	C(25)+C(12) C(ACC).
	34	ADD 12	2012	If C(25) = ADD 1100, the C(ACC) = 7777.
	35	AZE	0450	Skip to register 37 if C(ACC) = 7777.
	36	JMP 24	6024	If not, return and add next num.

Memory Address	Memory Buffer		Effect
37	HLT	0000	When C(ACC) = 7777, all numbers have been summed. Halt the computer.
.	.	.	
.	.	.	
1000	N_1	N_1	Numbers to be summed.
1001	N_2	N_2	
.	.	.	
1076	N_{77}	N_{77}	
1077	N_{100}	N_{100}	
1100	(Sum)	(Sum)	

Summing a Set of Numbers Using Address Modification.

The instructions at locations 20-22 initially set the contents of memory register 25 to the code for ADD 1000. At the end of the program, register 25 will contain 3100, the code for ADD 1100. Adding (in registers 33 and 34) C(25) to C(12), which contains the complement of the code for ADD 1100, results in the sum 7777 only when the program has finished summing all 100_8 numbers. This repeating sequence of instructions is called a loop, and instructions such as AZE can be used to control the number of times a loop is repeated. In this example the instructions in locations 24-36 will be executed 100_8 times before the computer halts.

The following program scans the contents of memory registers 400 through 450 looking for registers which do not contain zero. Any non-zero entry is moved to a new table beginning at location 500; this has the effect of packing the numbers so that no registers in the table contain zero. When the program halts, the accumulator contains the number of non-zero entries.

Memory Address	Memory Buffer		Effect
4	ADD 400	2400	Constants used by the program.
5	STC 500	4500	
6	1	0001	
7	−(ADD 451)	5326	
10	−(STC 500)	3277	
.	.	.	
Start → 100	CLR	0011	Code for ADD 400 C(106).
101	ADD 4	2004	
102	STC 106	4106	
103	ADD 5	2005	Code for STC 500 C(112).
104	STC 112	4112	
105 →	CLR	0011	
106	(ADD 400)	(2000+K)	C(X) C(ACC).
107	AZE	0450	If C(ACC) = 0, skip to location 111.

Memory Address	Memory Buffer		Effect
110	JMP 112	6112	C(ACC) ≠ 0, therefore JMP to location 112.
111	JMP 116	6116	C(ACC) = 0, therefore JMP to location 116.
112	(STC 500)	(4000+X)	Store non-zero entry in new table.
113	ADD 6	2006	Add 1 to the STC instruction in register 112.
114	ADD 112	2112	
115	STC 112	4112	
116	ADD 6	2006	Add 1 to the ADD instruction in register 106.
117	ADD 106	2106	
120	STC 106	4106	
121	ADD 106	2106	C(106)+C(7) C(ACC). If C(106) = ADD 451, then C(ACC) = 7777.
122	ADD 7	2007	
123	AZE	0450	If C(ACC) = 7777, skip to location 125.
125	ADD 112	2112	If C(ACC) = 7777, then number of non-zero entries C(ACC) and computer halts.
124	JMP 105	6105	If not, return to examine next number.
126	ADD 10	2010	
127	HLT	0000	

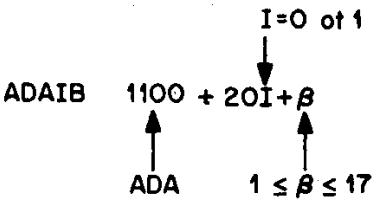
Packing a Set of Numbers

At the end of the program, register 106 contains the code for ADD 451, and all numbers in the table have been examined. If, say, 6 entries were found to be non-zero, registers 500-505 will contain the non-zero entries, and register 112 will contain the code for STC 506. Therefore by adding C(112) to the complement of the code for STC 500 (in registers 125-126 above, the accumulator is left containing 6, the number of non-zero entries.

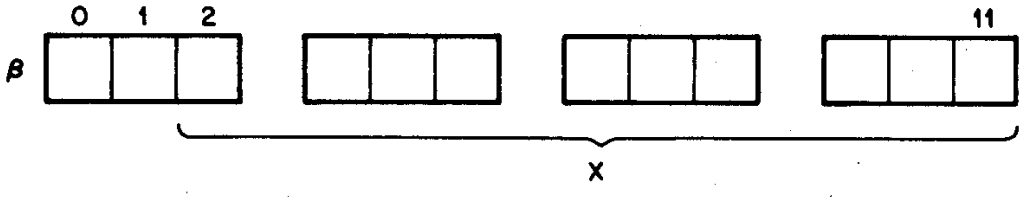
INDEX CLASS INSTRUCTIONS I

Indirect Addressing

The largest class of LINC Mode instructions, index class, addresses the memory in a somewhat involved manner. The instructions ADD X, STC X, and JMP X are called full address instructions because the 10-bit address X, $0 \leq X \geq 1777$, can address directly any register in the 2000_8 register memory. The index class instructions, however, have only 4 bits reserved for an address, and can therefore address only memory registers 1_8-17_8. The instruction ADA I β (add to accumulator), $1100_8 + 201 + \beta$, is typical of the index class:



Memory register β should be thought of as containing a memory address, X, in the rightmost 10 bits,



and X(β), as meaning the right 10-bit address part of register β. The leftmost bit can have any value, and, for the present, bit 1 must be 0. In addressing memory register β, an index class instruction tells the computer where to find the memory address to be used in executing the instruction. This is called indirect addressing.

For example, to add the value 35 to the contents of the accumulator, with 35 held in memory register 270, use the ADA instruction in the following manner:

Memory Address	Memory Buffer	Effect	
B ---→ 0270	0270	Address of register containing 35.	
0270	0035	0035	
p ADA B	1100 + β	C(270) + C(ACC)	C(ACC).

Note that the ADA instruction does not tell the computer directly where to find the number 35; it tells the computer instead where to find the address of the memory register which contains 35. By using memory registers 1-17 in this way, the index class instructions can refer to any register in the memory.

Two other index class instruction, LDA I β (load accumulator), and STA I β (store accumulator), are used in the following program which adds the contents of memory register 100 to the contents of register 101 and stores the result in 102. The LDA I β instruction, code 100 + 201 + β, clears the accumulator and copies into it the contents of the specified memory registers. STA I β, code 1040 + 201 + β, stores the contents of

the accumulator in the specified memory register; it does not, however, clear the accumulator. Addition with ADA uses 12-bit end-around carry arithmetic.

Memory Address		Memory Buffer		Effect
	10	X_1	0100	Address of N_1.
	11	X_2	0101	Address of N_2.
	12	X_3	0102	Address of (N_1+N_2).
	.	.	.	
	.	.	.	
Start →	30	LDA 10	1010	N_1, i.e., C(100), C(ACC).
	31	ADA 11	1111	N_2, i.e. C(101), +C(ACC) C(ACC).
	32	STA 12	1052	N_1+N_2 C(102)
	33	HLT	0000	
	.	.	.	
	.	.	.	
	100	N_1	—	
	101	N_2	—	
	102	(N_1+N_2)	(–)	

Indirect Addressing

INDEX REGISTERS AND INDEXING

When I is used with an index class instruction, that is, when I=1, the computer is directed to add 1 to the X part of memory register β before it is used to address the memory. This process is called indexing, and registers 1-17 are frequently referred to as index registers. In the example below, −6 is loaded into the accumulator after index register β is indexed from 1432 to 1433 by the LDA I B instruction.

Memory Address	Memory Buffer		Effect
β	(X)	(1432)	Address minus 1 of register
.	.	.	containing 7771.
.	.	.	
.	.	.	
→ p	LDA I β	$1020+\beta$	X+1, i.e., 1433,→C(β), and
.	.	.	C(1433)→C(ACC).
.	.	.	
.	.	.	
1432	—	—	
1433	−6	7771	

When the LDA β instruction is executed, the value $X(\beta)+1$ replaces the address part of register β (the leftmost 2 bits of register β are unaffected). This new value, 1433, is now used to address the memory. Note that if the LDA instruction at p were repeated, it would deal with the contents of the register 1434, then 1435, etc. Utility of index registers in scanning tables of numbers should be obvious.

Indexing involves only 10-bit numbers, and does not involve end-around carry. Therefore the address following 1777 is 0000. (The same kind of indexing takes place in the instruction location register, which counts from 1777 to 0000.)

The following example using indexing introduces another class instruction, SAE I β (skip if accumulator equals), 1440 $+201+\beta$. This instruction causes the LINC to skip one register in the sequence of programmed instructions when the contents of the accumulator exactly match the contents of the specified memory register. If there is no match, the computer goes to the next register in sequence as usual. The program example clears (stores 0000) in the set of memory registers 1400-1777; th SAE instruction is used in decide whether the last 0000 has been stored.

Memory Address	Memory Buffer		Effect
3	(X)	(1377)	Initial address minus 1 for the STA instruction.
4	356	0356	Address of test number.
.	.		
.	.		
350	CLR	0011	Clear the accumulator.
Start → 351	STA I 3	1063	Index the contents of register; store C(ACC) in the memory register whose address=X( ).
352	ADD 3	2003	C(3) C(ACC).
353	SAE 4	1444	Skip to 0355 if C(ACC)=C(356).
354	JMP 350	6350	If not, return to store 0000 in next register.
355	HLT	0000	Halt the computer.
356	1777	1777	

Example — Indexing to Clear a Set of Registers

When the program halts at register 355, register 3 will contain 1777. The SAE instruction is used here (as the AZE instruction was used in earlier examples) to decide when to stop the computer. The instructions in registers 350-354, the loop, are executed 400_8 times before the program halts. A 0 is first stored in register 1400, next in 1401, etc.

Another program scans the memory to see if a particular number, Q, appears in any memory register 0-1777. Q is to be set in the Right Switches, and the address of any register containing Q is to be left in the accumulator.

Memory Address	Memory Buffer		Effect
17	(X)	(–)	Address of register whose contents are to be compared with *RIGHT SWITCH*.

Memory Address		Memory Buffer		Effect
Start →	20	RSW	0516	C(RS) C(ACC).
	21	SAE I 17	1477	Index register 17, and compare C(ACC) with C(X).
	22	JMP 21	6021	If not equal, return or next test.
	23	CLR	0011	
	24	ADD 17	2017	If equal, clear ACC, copy address of register containing Q into ACC, and halt.
	25	HLT	0000	

Memory Scanning

If no memory register 0-1777 contains the number Q, the program will run endlessly. The location of the first register to be tested depends on the initial contents of index register 17.

An index class instruction, ADM I β, (add to Memory), code 1140 + 201 + β, adds the contents of the specified memory register to C(ACC), using 12-bit end-around carry arithmetic (as ADD or ADA). The results is left, however, not only in the accumulator but in the specified memory register as well. The bit clear instruction, BCL I β, code 1540 + 201 + β, is one of three index class instructions which performs a so-called "logical" operation. BCL is used to clear selected bits of the accumulator is set to 0.

In the folowing program two sets of numbers are summed term by term. The first set of numbers, each 6 bits long, is in registers 500-577, bits 6-11; bits 0-5 contain unwanted information. The second set of numbers is in registers 600-677, and the sums replace the contents of registers 600-677.

Memory Address		Memory Buffer		Effect
	3	(X)	(0477)	Initial address minus 1 of first set.
	4	0410	0410	Address of BCL pattern.
	5	$(X)_2$	(0577)	Initial address minus 1 of second set.
	6	0411	0411	Address of test number for halting.
	.			
	.			
Start →	400	LDA I 3	1023	Index X(3) and load number from first set into AC.
	401	BCL 4	1544	Clear the left 6 bits of the ACC.
	402	ADM I 5	1165	Lndex X(5). Add number from second set to C(ACC), and replace in memory.
	403	CLR	0011	
	404	ADD 3	2003	Check to see if finished.
	405	SAE 6	1446	

Memory Address	Memory Buffer		Effect
406	JMP 400	6400	C(3)≠ C(411), i.e., ≠0577.
407	HLT	0000	C(3)=0577; halt the program.
410	7700	7700	BCL pattern for clearing left half of ACC.
411	0577	0577	Test number for halting.

Summing Sets of Numbers Term by Term

Logic Instructions

The three logic instructions, BCL I β, BSE I β, and BCO I β, are best understood by studying the following examples. These instructions affect only the accumulator; the memory register M containing the bit pattern is unchanged.

BCL I β bit clear code 1540 + 20I + β

Clear corresponding bits of the accumulators:

If C(M)	=010	101	010	101
and C(ACC)	=111	111	000	000
then C(ACC)	=101	010	000	000

BSE I β bit set code: 1600 + 20I + β

Set to 1 corresponding bits of the accumulator:

If C(M)	=010	101	010	101
and C(ACC)	=111	111	000	000
then C(ACC)	=111	111	010	101

BCO I β bit complement code: 1640 + 20I + β

Complement corresponding bits of the accumulator:

If C(M)	=010	101	010	101
and C(ACC)	=111	111	000	000
then C(ACC)	=101	010	010	101

These instructions have a variety of applications, some of which will be demonstrated later.

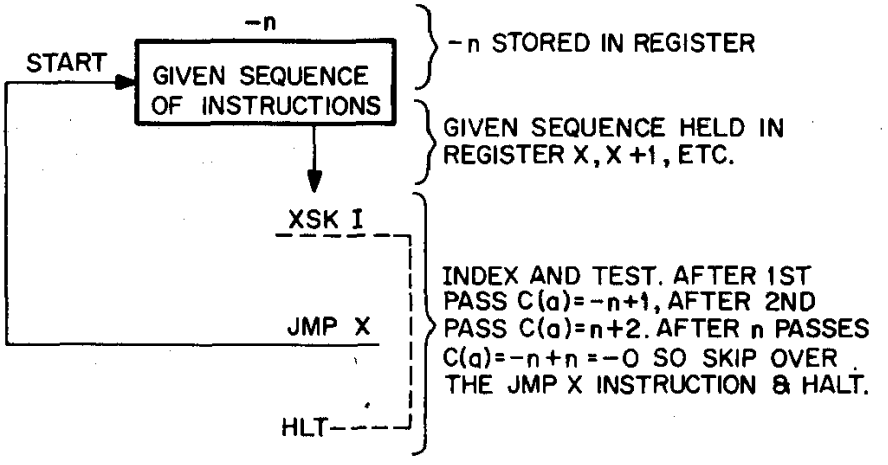
SPECIAL INDEX REGISTER INSTRUCTIONS

Before continuing with the index class, two special instructions which facilitate programming with the index class instructions will be introduced. These instructions do not use the index registers to hold memory addresses; rather they deal with the index registers and are used to change or examine the contents of an index register.

The Index and Skip Instruction

The index and skip instruction XSK I α, refers to registers 0-17 (0 a 17.)* It tests to see whether the address part of register a has its maximum value, i.e., 1777, and directs the LINC to skip the next register in the instruction sequence if 1777 is found. It will also, when I=1, index the

address part (X) of register a by 1. Like the index class instruction, XSK indexes register a before examing it, and it indexes from 1777-0000 without affecting the leftmost 2 bits. These 2 bits can therefore have any value. In particular, both can be set to the value 1 and XSK I can assumed to have the effect of skipping the next instruction when it finds the number 7777, (−0), in register α. Now it is easy to see how to execute any given sequence of instructions exactly n times, where n 1777 (octal):



For example, to store the contents of the accumulator in registers 350-357, using registers 6 to count, the following short program can be written.

Memory Address		Memory Buffer		Effect
	5	(X)	(0347)	Initial address minus 1 for STA instruction.
	6	(−10)	(7767)	−n, where n = number of times to store C(ACC).
Start →	200	STA I 5	1065	Index register 5 and store C(ACC).
	201	XSK I 6	0226	Index register 6 and test for X(6) = 1777.
	202	JMP 200	6200	X(6) ≠ 1777, return.
	203	HLT	0000	X(6) = 1777, halt.

*cf. β, $1 < \beta < 17$, which does not refer to register 0.

Index Registers Used as Counters

THE SET INSTRUCTION

The second special instruction which is often used with the index class instruction is SET I α, code $40 + 201 + \alpha$ where α again refers directly to the first 20_8 memory registers, $0 \leq \alpha \leq 17$. In some of the examples presented earlier, the contents of index registers are changed, either as counter values or as memory addresses, while the program is running. Therefore, in order to return the program, the index registers must be reset to their initial values.

The SET instruction directs the LINC to set register α to the value in any specified memory register. It is different from the instruction so far presented in that the instruction itself always occupies two consecutive memory registers, say p and p + 1:

Memory Address	Memory Buffer	
p	SET I	40+20I +a
p + 1	c	c
p + 2	—	—
.	.	.
.	.	.

The computer automatically skips over the second register of the pair, p + 1; that is the contents of p + 1 are not interpreted as the next instruction. The next instruction after SET is always taken from p + 2.

The I-bit in the SET instruction does not control indexing. Instead, it tells the LINC how to interpret the contents of register p + 1. When I = 0, the LINC is directed to interpret C(p + 1) as the memory address for locating the value which will replace C(α). That is, register p + 1 is thought of as containing X,

Memory Address	Memory Buffer		Effect
10	(N)	(—)	
.	.	.	
.	.	.	
→ p	SET 10	0050	C(X), i.e., N,→C(10).
p + 1	X	X	
.	.	.	
.	.	.	
X	N	N	

and the contents of register X replace the contents of 10, C(X)→C(10). In this case X is the rightmost 10 bits, the address part, of register p + 1; the leftmost bit of C(p + 1) may have any value and, for the present, bit 10 must be 0.

In the second case, when I=1, the LINC is directed to interpret C(p + 1) as the value which replaces C(α). Thus, below, C(p + 1) C(5):

Memory Address	Memory Buffer		Effect
5	(N)	(—)	
.	.	.	
.	.	.	
→ p	SET I 5	0065	C p + 1), i.e., N. C(5).
p + 1	N	N	

The following program scans 100_8 memory registers looking for a value which matches C(ACC). It halts with the location of the matching register in the accumulator if a match is found, or with −0 in the accumulator if a match is not found. The numbers to be scanned are in registers 1000-1077.

	Memory Address	Memory Buffer		Effect
	3	(−100)	(7677)	−(number of registers to scan).
	4	(X)	(0777)	Scanning address.
	.	.	.	
	.	.	.	
Start →	400	SET I 3	0063	C(401), i.e., −100,→C(3).
	401	−100	7677	
	402	SET I 4	0064	C(403), i.e., 777,→C(4).
	403	777	0777	
	404	SAE I 4	1464	Index X(4) and compare C(X) with C(ACC).
	405	JMP 411	6411	C(ACC) ≠ C(X), jump to 411.
	406	CLR	0011	
	407	ADD 4	2004	C(ACC) = C(X), copy location of matching register into ACC and halt.
	410	HLT	0000	
	411	XSK I 3	0223	Index register 3 and test for X(3) = 1777.
	412	JMP 404	6404	X(3) ≠ 1777, return.
	413	CLR	0011	
	414	HLT	0017	X(3) = 1777; all numbers have been scanned so −0→C(ACC) and halt.
	415	HLT	0000	

Setting Initial Index Register Values

The two SET instructions are executed once every time the program is started at 400; initially registers 3 and 4 may contain any value since the program itself sets them to the correct values.

Suppose the programmer had wanted to SET two index registers to the same value, say −100. He could write either:

	Memory Address	Memory Buffer		Effect
	11	(−100)	(7677)	
	12	(−100)	(7677)	
	.	.	.	
	.	.	.	
→	20	SET I 11	0071	C(21), i.e., −100,→C(11).

Memory Address	Memory Buffer		Effect
21	−100	7677	
22	SET 12	0052	C(21), i.e., −100,→C(12).
23	21	0021	

or:

Memory Address	Memory Buffer		Effect
20	SET I 11	0071	C(21), i.e., −100,→C(11).
21	−100	7677	
22	SET 12	0052	C(11), i.e., −100,→C(12).
23	11	0011	

The programmer could also, of course, have written SET I 12 in register 22 with −100 in register 23, but there are applications appropriate to each form.

INDEX CLASS INSTRUCTIONS II

Double Register Forms

The index class instruction have been thought of as addressing an index register $\beta, 1 \leq \beta \leq 17$, which contains a memory address X to be used by the instruction. They have been presented as single register instructions (unlike SET). However, when an index class instruction is written with $\beta=0$, it becomes a double register instruction like SET, whose operand address depends on I and p+1. These two interpretations are shown for STA.

Case: I=0, $\beta=0$

Memory Address	Memory Buffer		Effect
450	STA I	1040 + 20(0) + 0	C(ACC)→C(330)
451	330	0330	

When I=0, the LINC is directed to use C(p + 1), i.e., C(451) as the memory address at which to store C(ACC). The leftmost bit of C(p + 1) may have any value, and, for the present, bit 1 must be 0.

Case: I=0, $\beta=0$

Memory Address	Memory Buffer		Effect
450	STA I	1060	C(ACC)→C9451)
451	(–)	(–)	

When I=1, the LINC is directed to use p + 1, i.e., 451, directly as the memory address, and the contents of the accumulator are stored in 451. Note that when $\beta=0$ in an index class instruction, it does not refer to

memory register 0. In fact, when $\beta=0$, no reference is necessarily made to the index registers. As with SET, the computer automatically takes the next instruction from register p + 2.

Index class instruction may be thought of as having four alternative ways of addressing the memory, which depends on I and B, and which are summarized below:

Index Class Address Variations				
Case	I,B	Example	Form	Comments
1	I=0 $\beta\neq0$	LDA I β	Single Register	Register β holds operand address.
2	I=1 $\beta\neq0$	LDA β	Single Register	First, index register B by 1. Then, register β holds operand address.
3	I=0 $\beta=0$	LDA X	Double Register	Second register holds operand address.
4	I=1 $\beta=0$	LDA I N	Double Register	Second register hold operand.

The next programming example scans memory registers 1350-1447, counting the number of instances in which register contents are found to exceed some threshold value, T. In other words if C(X) T, X=1350, 1351, . . . , 1447, then C(CTR) + 1→C(CTR), where CTR is a memory register used as a counter, initially set to 0. The count, N, is to appear in the accumulator upon program completion.

Memory Address		Memory Buffer		Effect
	14	(X)	(–)	Address of register to be tested.
	15	(−n)	(–)	−(number of registers to test.)
Start	30	SET I 14	0074	Set index register 14 to initial address minus 1.
	31	1347	1347	
	32	SET I 15	0075	SET index register 15 to −100.
	33	−100	7677	
	34	CLR	0011	Clear CTR; o→C951).
	35	STC 51	4051	
	36	LDA I	1020	C(37), i.e., −T→C(ACC).
	37	−T	−T	
	40	ADA I 14	1134	Index the address in register 14 and form c(X) −T in ACC.
	41	BCL I	1560	Clear all but the sign in ACC;
	42	3777	6777	C(42=the bit pattern for clearing. Then if C(X)>T, C(ACC) = 0000 but if C(X)<T, C(ACC) = 40000.

Memory Address	Memory Buffer		Effect
43	SAE I	1460	Does C(ACC) = C(44)? If so,
44	0000	0000	skip to 46.
45	JMP 52	6052	If not, C(X)≤T. Jump to 52.
46	LDA I	1020	If so, C(X)>T; 1→C(ACC).
47	1	0001	
50	ADM I	1160	C(ACC)+C(51), i.e., N,→C(51) and→C(ACC).
51	(N)	(–)	
52	XSK I 15	0235	Index register 15 and test for 7777 C(15)≠ 7777. Return to check next register.
53	JMP 36	6086	
54	HLT	0000	C(15)= 7777, therefore halt. C(CTR) i.e., C(51), left in ACC.

Scanning for Values Exceeding a Threshold

Note that since the SAE instruction in locations 43 and 44 is written as a double register instruction, the LINC skips to location 46 (not 45) when the skip condition is satisfied. The next instruction in sequence is, in this case, at location 45.

Note also that if a double register instruction is written following a skip instruction such as XSK, the LINC mode tries to interpret the second register as an instruction:

Memory Address	Memory Buffer	Effect
.	.	
.	.	
.	.	
P	XSK I β	Go to p + 1 when X(β) ≠ 1777.
p + 1	LDA I	Go to p + 2 when X(β) = 1777.
p + 2	3	
.	.	
.	.	
.	.	

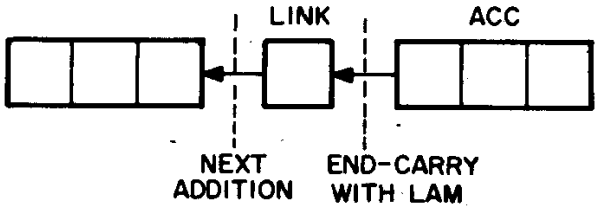
Since the XSK instruction sometimes directs the LINC to skip to P + 2, care must be taken to make sure that the LINC does not skip or jump to the second register of a double register instruction.

It is interesting to compare the above statement of the program made in rather detailed machine language with the following compact but entirely adequate restatement:

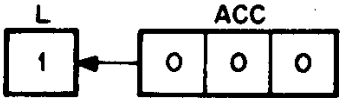
1. 0→C(CTR).
2. If C(X)>T then C(CTR) + 1 C(CTR), for X = 1350, 1351, . . . 1447.
3. C(CTR)→C(ACC).
4. HALT

Multiple Length Arithmetic

An index class instruction, LAM I β (link add to memory), code 1200 + 20I + β, makes arithmetic possible with numbers which are more than 12 bits long. Using LAM, one can work with 24-bit numbers for example, using 2 memory registers to hold right and left halves. It should be remembered that addition with ADD, ADA, or ADM always involves end-around carry. With LAM, however, a carry from bit 0 of the accumulator during addition is saved in the link bit; it is not added to bit 11 of the accumulator. This carry, then, could be added to the low-order bit of another number, providing a carry linkage between right and left halves of a 24-bit number. For simplicity, the illustration uses 3-bit registers; the principles are the same for 12 bits:



If, for example, the number in this 3-bit accumulator is 7 (all 1's) and C(L) = 0, and 1 is added with LAM, the link bit and accumulator will then look like:



Furthermore, LAM is an add-to-memory instruction, so that the memory register to which the LAM instruction refers will now contain 0 (as does the accumulator).

In addition to saving the carry in the link bit the LAM instruction also adds the contents of the link bit to the low order bit of the accumulator. That is, if, when the LAM instruction is executed C(L) = 1, then 1 is added to C(ACC). Using the result pictured above, add 2, where 2 is the contents of some memory register M:

	L	ACC	M
GIVEN:	1	000	010

Using LAM, the LINC is directed first to add C(L) to C(ACC), giving:

L	ACC	M
0	001	010

There is no end-carry this operation, so the link bit is cleared. The LINC then adds C(ACC) to C(M), giving:

L	ACC	M
0	011	011

which replaces both C(ACC) and C(M). Again there is no end-carry so the link bit is left unchanged.

The operation of LAM may be summarized:

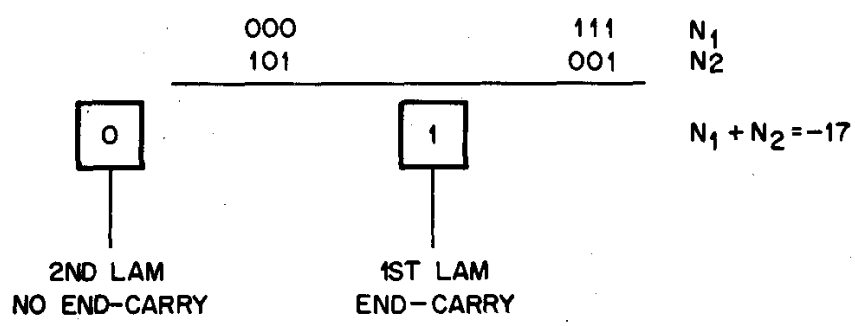
1. C(L) + C(ACC)→C(ACC).
2. End-carry→C(L). If no end-carry, 0→C(L).
3. C(ACC) + C(M)→C(ACC), and→C(M).
4. End-carry→C(L). If no end-carry, the link bit is left unchanged.

As an example of double length arithmetic, postulate 2 numbers, N_1 and N_2, each 6 bits long, which occupy a total of four 3-bit memory registers M_1 through M_4:

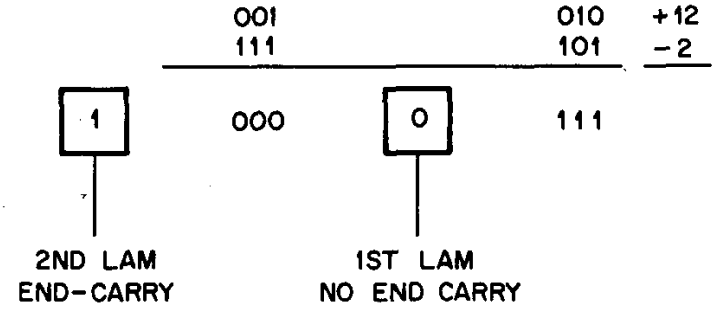
M_2	M_1	
000	111	$N_1 = +7$
M_4	M_3	
101	001	$N_2 = -26$

The sum (octal) of +7 and −26 is −17. Using the LAM instruction to get this:

1. Clear the link bit.
2. Add $C(M_1)$ to $C(M_3)$ with LAM, saving any carry in the link bit. This sums the right halves of N_1 and N_2.
3. Add $C(M_2)$ to $C(M_4)$ with LAM, which also adds in any carry from step 2. This sums the left halves of N_1 and N_2. Any new carry again replaces C(L).



Note that only the first LAM produced an end-carry.
To complete the illustration, consider a case in which the final carry appears in the link bit, as in the addition of +12 and −2.



whose sum, in 1's complement notation is 001/000, or $+10_8$, but which with LAM results in +7 and an end-carry in the link bit. Since 1's complement representation depends on end-around carry, some extra programming must be done to restore to a true 1's complement number. This is, of course, the equivalent of adding 1 to the 2-register result. Assuming that the result is in M_1 and M_2

L	M_2	M_1
1	000	111

again the use of the LAM instruction. First clear the accumulator without clearing the link bit (this can be done with an STC instruction). Then execute LAM with C(M_1) which gives

L	M_2	M_1
1	000	111

producing a new end-carry in the link bit. Again clear the accumulator (but not the link bit) and execute LAM with C(M_2) which gives

L	ACC	M_2
0	001	001

The result in M_2 and M_1 now looks like:

M_2	M_1
001	000 = 10(OCTAL)

It should be clear to the reader that adding in a final end-carry as an end-carry cannot itself give rise to a new final end-carry.

The following program illustrates the technique of double length arithmetic with tables of numbers; similar techniques would be used for other multiples of 12. Assume that 100_8 24-bit numbers, N_0, N_1, . . . N_{77}, are to be added term by term to 100_8 numbers, R_0, R_1, . . . R_{77}, so that $N_0 + R_0 = S_0$, $N_1 + R_1 = S_1$, etc. All numbers occupy 2 registers: the left halves of N_0, N_1, . . . N_{77} are in registers 100-177, the right halves in 200-277. The left halves of R_0, R_1, . . . R_{77} are in 1000=1077, the right halves in 1100-1177. The left halves of the sums, S_0, S_1, . . . S_{77}, replace the contents of 1000-1077, the right halves replace the contents of 1100-1177.

Memory Address	Memory Buffer		Effect
10	(X_1)	(−)	
11	(X_2)	(−)	
12	(X_3)	(−)	
13	(X_4)	(−)	
14	(−n)	(−)	
.	.	.	
.	.	.	
.	.	.	
377	(−)	(−)	
→400	SET I 10	0070	
401	77	0077	
402	SET I 11	0071	Set index registers to initial ad-
403	177	0177	dresses minus 1 for the 4 tables.
404	SET I 12	0072	
405	777	0777	
406	SET I 13	0073	
407	1077	1077	
410	SET I 14	0074	Set index register 14 as a counter for 100 loop repetitions.
411	−100	7677	
412	→CLR	0011	0→C(ACC); 0→C(L).
413	LDA I 11	1031	Right half of N_1→C(ACC).
414	LAM I 13	1233	Righ half of N_1+right half of R→ C(ACC), and→ right half of R_1, End-carry→C(L).
415	LDA I 10	1030	Left half of N_1→C(ACC).
416	LAM I 12	1232	C(L) + C(ACC) + left half of R_1 →C(ACC), and→Left half of R_1. End carry→C(L).
417	STC 377	4377	Clear accumulator by storing in 377. Do not clear link bit.
420	LAM 13	1213	C(L) + right half of S_1→C(ACC), & right half of S. End-carry→C(L).
421	STC 377	4377	Clear accumulator.
422	LAM 12	1212	C(L) + left half as S_1→C(ACC), and left half of S_1.

Memory Address	Memory Buffer		Effect
423	XSK I 14	0234	Index 14 and test for 7777.
424	JMP 412	6412	C(14)≠ 7777, return to form next sum.
425	HLT	0000	C(14) =7777, so halt.

Summing Sets of Double Length Numbers
Term by Term

The instructions in locations 412-416 produce an initial 24-bit sum leaving any final carry in the link bit. The instructions in locations 417-422 then complete the sum by adding in the final end-carry. The link bit always contains 0 after the computer executes the last LAM in location 422. Register 377 is used simply as a "garbage" register so that the accumulator can be cleared without clearing the link bit.

MULTIPLICATION

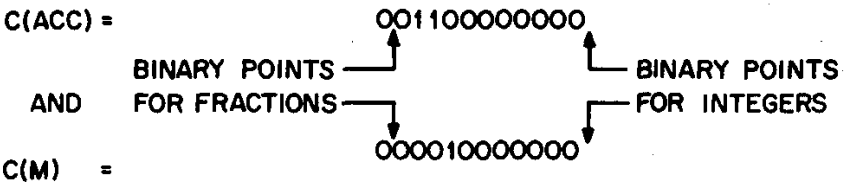
Another index class instruction which needs special explanation is MUL I β (multiply), code 1240 201 + β. This instruction directs the LINC mode to multiply C(ACC) by the contents of the specified memory register, and to leave the result in the accumulator. The multiplier and multiplicand are treated as signed 11-bit 1's complement numbers, and the sign of the product is left in both the accumulator (bit 0) and the link bit. The LINC may be directed to treat both numbers either as integers or fractions; it may not, however, be directed to mix a fraction with an integer. The leftmost bit (bit 0) of register β is used to specify the form of the numbers.

When bit 0 of register β contains 0, the numbers are treated as integers; that is, the binary points are assumed to be the right of bit 11 of the accumulator and the specified memory register. Given C(ACC) = −10, C(β)=400 (bit 0 of register β=0), and C(400) = +2, the instruction MUL β leaves −20 in the accumulator, and 1 in the link bit. Overflow is, of course, possible when the product exceeds ±3777. Multiplying +3777 by +2, for example, produces +3776 in the accumulator; note that the sign of the product is correct, and that the overflow effectively occurred from bit 1, not from bit 0.

When bit 0 of register β contains 1, the LINC treats the numbers as fractions; that is, the binary point is assumed to be to the right of the sign bit (between bit 0 and bit 1) of the accumulator and the specified memory register. Given C(ACC)=+.2 C(β) = 5120 (bit 0 of register β=1), and C(1120=+.32, execution of MUL β leaves +064 in the accumulator and 0 in the link bit.

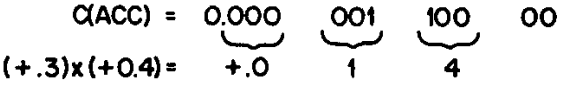
When the LINC multiplies two 11-bit signed numbers, a 22-bit product is formed. For integers the rightmost, or least significant, 11 bits of this product are left with the proper sign in the accumulator, and for fractions

the most significant 11 bits of the product are left with the proper sign in the accumulator. If, for example,



then C(ACC) can be thought of as either $+.3_8$ or $+1400_8$, and C(M) can be thought of as either $+.04_8$ or $+200_8$. The 22-bit product of these numbers looks like:

and if bit 0 of register β contains 1, the most significant 11 bits with the proper sign are left in the accumulator:



Had bit 0 of register β contained 0, the accumulator would be left with +0 as the result of multiplying (1400) x (200). It is the programmer's responsibility to avoid integer overflow by programming checks on his data and/or by scaling the values to a workable size.

Use of bit 0 of register β is new to the concept of index registers and should be noted in connection with the four memory addressing alternatives which index class instructions employ. When $\beta \neq 0$ then bit 0 of C(β), that is, bit 0 of the register which contains the memory address, is used. The same is true when I = 0 and $\beta = 0$, as in:

Memory Address	Memory Buffer	
p	MUL	1240
p + 1	h, X	4000 h + X

That is, bit 0 of C(p + 1), the register containing the memory address, is used. This bit is sometimes called the h-bit, whether in an index register or in register p + 1. When, however, I=1 and β=o, it will be recalled that p + 1 is itself the memory address:

Memory Address	Memory Buffer	
p	MUL I	1240
p + 1	N	N

There isn't any memory register which actually contains the memory address, and therefore there is no h-bit. The computer always assumes in this case that H=0, and the operands are treated as integers.

In the following program, registers 1200-1377 contain a table of fractions whose values are in the range ±.0176, that is, whose most significant five bits after the sign (bits 1-5) duplicate the sign. Each number

is to be multiplied by a constant, −.62, and the products stored at locations 1000-1177. To retain significance, the values, are first shifted left 5 places.

Memory Address	Memory Buffer		Effect
6	(X_1)	(−)	
7	(X_2)	(−)	
10	(−n)	(−)	
.	.	.	
.	.	.	
→ 500	SET I 6	0066	Initial address minus 1 of table
501	1177	1177	of fractions→C(6).
502	SET I 7	0067	Initial address minus 1 for STA instruction→C(7).
503	777	0777	
504	SET I 10	0070	−n→C(10).
505	−200	7577	
506	→LDA I 6	1026	Fraction→C(ACC).
507	ROL 5	0245	C(ACC).2^5→C(ACC).
510	MUL	1240	Multiply, as fractions, C(ACC) by C(516).
511	4000+516	4516	
512	STA I 7	1067	Store product.
513	XSK I 10	0230	
514	JMP 506	6506	If not finished, return.
515	HLT	0000	If finished, halt.
516	−.62	4677	

Multiplying a Set of Fractions by a Constant

The ROL instruction at location 507 rotates 0's or 1's, depending on the sign, into the low-order 5 bits of the acumulator. Since this amounts to a scale left operation, it introduces no new information which might influence the product. The reader should also note that the original values remain unchanged at locations 1200-1377.

Another example demonstrates the techniques of saving both halves of the product. Fifty (octal) numbers, stored at locations 1000-1047, are to be multiplied by a constant, +1633. The left halves of the products (the most significant halves) are to be saved at locations 1100-1147; the right halves (the least significant halves) at locations 1200-1247.

Memory Address	Memory Buffer		Effect
3	(X_1)	(1077)	Addresses of products.
4	(X_2)	(1177)	
5	$(4000+X_3)$	(4777)	Addresses of multiplier as fraction and integer.
6	(X_3)	(0777)	
7	(−n)	(7727)	Counter.
.	.	.	
.	.	.	

Memory Address	Memory Buffer		Effect
→ 1400	SET I 3	0073	Set addresses for storing products.
1401	1077	1077	
1402	SET I 4	0074	
1403	1177	1177	
1404	SET I 5	0075	Set 5 to address multiplier as fraction.
1405	4000+777	4777	
1406	SET I 6	0076	Set 6 to address multiplier as integer.
1407	777	0777	
1410	SET I 7	0077	
1411	−50	7727	Form left half of product, in accumulator.
1412	LDA I	1020	
1413	1633	1633	
1414	MUL I 5	1265	
1415	SCR I 1	0361	C(bit 11 of ACC)→C(L).
1416	STA I	1063	Store left half of product.
1417	STC 1434	5434	0→C(ACC).
1420	ROR I 1	0321	C(L)→C bit 0 of ACC).
1421	STC 1427	5427	4000 or 0000→C(1427).
1422	ADD 141	3413	Form right half of product; in accumulator.
1423	MUL I 6	1266	
1424	BCL I	1560	Clear bit 0 of right half.
1425	4000	4000	
1426	BSE I	1620	C(bit 11 of left half)→C(bit 0 of right half).
1427	(−)	(−)	
1430	STA I 4	1064	Store right half of product.
1431	XSK I 4	0227	Return if not finished.
1432	JMP 1412	7412	
1433	HLT	0000	
1434	(−)	(−)	

Multiplication Retaining 22-Bit Products

The instructions at location 1415, 1420-1421, and 1427 have the effect of making two halves of the product contiguous; the sign bit value of the right half is replaced by the low-order bit value of the left half, so that the product may be subsequently treated as a true double length number.

Through the use of another instruction, QAC it is possible to do a double precision multiplication using only one MUL instruction. When the LINC mode performs a multiplication, it uses three basic registers: The accumulator for the multiplicand, the memory buffer register for the multiplier, and the MQ register for partial containment of the initial 22-bit plus sign answer. The LINC then decides if the multiplication was fractional or integer and puts into the accumulator the correct half of the answer properly signed. In a fractional multiply, the most significant bits

of the product are found in the accumulator; however, the low-order portion of the product is not lost but is still in the MQ register as an unsigned number. By executing a MQ to AC instruction, QAC (MSC 005), Code 0005, the accumulator is cleared, and the contents of the MQ register are copied into bits 1-11 of the accumulator. Bit 0 is always o and the number is unsigned. However, the link contains the sign of the product so that, if necessary, the low-order portion of the product may be complemented.

Since bit 0 of the low-order portion does not contain a significant bit after a QAC instruction (unless the product was 3777 or less), it is useful to transfer bit 11 of the most significant portion of the product into bits 0 of the low-order portion. The following example multiplies the number in the LEFT SWITCHES by the number in the RIGHT SWITCHES and stores the double precision product in memory into two consecutive locations.

Memory Address	Memory Buffer		Effect
	100	(X_1) (C(R.S.))	Contents of RIGHT SWITCHES
	101	(X_2) (H.O.P.)	High order product
	102	(X_3) (L.O.P.)	Low order product
	.	. .	
	.	. .	
Start →	400	RSW 0516	Read RIGHT SWITCHES into A
	401	STC 100 4100	Store into location 100
	402	LSW 0517	Read LEFT SWITCHES into A
	403	MUL 1240	Multiply (fractional) C(p + 1) by C(A)
	404	(4000+0100) 4100	
	405	STC 101 4101	Store high order product into location 101
	406	ZTA 0005	C(Z)→C(A)
	407	LZE 0452	Was product positive?
	410	COM 0017	No, complement A
	411	STC 102 4102	Store low order product in location 102
	412	ADD 101 2101	Get back high order product
	413	ROR I 1 0321	Rotate bit 11 into link (sign bit) into bit 0.
	414	STC 101 4101	Store into 101
	415	ADD 102 2102	Get low order product
	416	ROL 1 0241	Rotate bit 0 into bit 11
	417	ROR I 1 0321	Rotate link into bit 0, bit 11 into link
	420	STC 102 4102	Store into 102
	421	HLT 0000	

Multiplication for 22 Bit-Product Using ZTA

There are two remaning index class instructions, SRO I β (skip rotate) and DSC I β (display character), which is discussed later in connection with programming the oscilloscope display.

HALF-WORD INSTRUCTIONS

The LINC mode has 3 instructions which deal with 6-bit numbers or half-words (word is another term for contents of a register). These instructions use the index registers and have the same four addressing variations as the index class, but specify in addition either the left or right half of the contents of memory register X as the operand. Think of LH(X) as meaning the contents of the left 6 bits of register X, and RH X), meaning the contents of the right 6 bits. Then it is possible to think of C(X)=LH/RH, or C(X)=100LH + RH.

Half-word instructions always use the right half of the accumulator. The load half instruction, LDH I β, code 1 00 + 201 + β, clears the accumulator and copies the specified half-word into the right half of the accumulator; which half of C(X) to use is specified by bit o, the h-bit, of register β.

When h=0, LH(X), RH(ACC). When h=1, RH(X) RH (ACC).

Memory Address	Memory Buffer		Effect
β	h, K	4000h + X	h=1.
.	.	.	
.	.	.	
p	LDH β	1300 + β	RH(X)→RH(ACC) and 0→LH(ACC).
.	.	.	
.	.	.	
X	LH/RH	100LH + RH	C(X) unchanged.

The same interpretation of the h-bit applies when I = 0 and β = 0, i.e., when the instructions occupies two registers:

Memory Address	Memory Buffer		Effect
40	LDH	1300	Since h = 1, RH(500),i.e., 76,→RH(ACC). 0→LH(ACC).
41	1,500	4,500	
.	.	.	
.	.	.	
500	32/76	3276	

If register 41 contained 500, i.e., h=0, then LH(500), or 32, would replace RH(ACC).

The store half instruction, STH I β, code 1340 + 20I + β, stores the right half of C(ACC) in the specified half of memory register X. C(ACC)

and other half memory register X are unaffected. To illustrate the case of I=1 and β=0, write:

Memory Address	Memory Buffer		Effect
1000	STH I	1360	RH(ACC)→LH(1001)
1001	6015	6015	

This case, it will be remembered, uses p+1 itself as the memory address. Since there is no h-bit, the computer assumes that h=0, and therefore the left half of C(1001) is affected. If, for example, C(ACC) = 5017, 17 replaces LH(1001), and the contents of register 1001 become 1715.

SHD I β (skip if half difiers), code 1400 + 20I + β, causes the LINC to skip one memory register in the program sequence when the right half of the accumulator does not match the specified half of memory register X. When it does match, the computer goes to the next Memory register in sequence for the next instruction. Neither C(ACC) nor C(X) is affected by the instruction. (If C(ACC) = 4371, and the programmer writes:

Memory Address	Memory Buffer		Effect
376	7152	7152	Skip to 402 if RH(376) ≠
→ 377	SHD	1400	RH(ACC).
400	4376	4376	
401	⌐ ¬ ⌐	—	
402	— ← ⌊	—	

The computer skips because RH(376), i.e., 52, ≠ RH(ACC), or 71. Had he written 376 in location 400, that is, h=0, RH(ACC) would equal LH(376) and the computer would not skip.

When β=0, and when I=1, the half-word class instructions cause the LINC to index the contents of memory register β, but in a more complex way than that used by the index class instructions. In order to have half-word indexing refer to consecutive half-words, the computer adds 4000 to C(β) with end-around carry. This has effect of complementing h(β) every time register β is indexed, and stepping X(β) every other time. Suppose, for example, that the instruction is LDH I 3, and that register 3 initially contains 4377, that is, it points to the right half of register 377. The computer first adds 4000 to C(3):

4377	Original C(3) = 1, 77
4000	Index H(3)
0377	
1	End-around carry
0400	New C(3) = 0,400

which leaves h=0 and X=400; C(3) now points to the left half of register 400. The computer therefore loads the accumulator from LH(400). Repeating the instruction, C(3) is indexed to 4400 and the accumulator is

loaded from RH(400). Continuing register would contain the following succession of values or half-word references:

4400	:	RH (400)
0401	:	LH (401)
4401	:	RH (401)
0402	:	LH (402)
4402	:	RH (402)
0403	:	LH (403)
etc.		etc.

Since the half-word indexing occurs before the contents of register β are used to address the memory, the memory address, when I = 1, can be described as

$$\bar{h}\ Xth$$

where $\bar{h}$ represents the indexed value of h, and Xth represents the indexed value of X. The succession of values which appear in register β can be written:

h, Xth
1, X + 0
0, X + 1
1, X + 1
0, X + 2
1, X + 2
etc.

The four address variations for half-word class instructions are summarized in the following table.

HALF-WORD CLASS ADDRESS VARIATIONS

Case	I β	Example	Form	Comments
1	I = 0 $\beta \neq 0$	LDH β	Single Register	Register β holds half-word operand address.
2	I = 1 $\beta \neq 0$	LDH I β	Single Register	First, index register β by 4000 with end-around carry. Then, register β holds half-word operand address.
3	I = 0 $\beta = 0$	LDH h, X	Double Register	Second register holds half-word operand address.
4	I = 1 $\beta = 0$	LDH I LH/RH	Double Register	Left half of second register holds half-word operand.

For h = 0, the operand is held in the left half of the specified memory register. For h = 1, the operand is held in the right of the specified memory register.

The following program will take a table or $100_{(8)}$ twelve bit numbers and pack them into a table as signed six bit numbers.

Memory Address		Instruction	Code	Effect
Start →	20	CLR	0011	Clear the Accumulator.
	21	SET I 5	0065	Initialize the location 5 to the
	22	777	0777	location-1, of the 12 bit table.
	23	SET I 6	0066	SET half-word pointer to right side.
	24	4477	4477	(h=1) of location 477.
	25	SET I 7	0067	Set counter to −100.
	26	−100	7677	
	27	LDA I 5	1025	Get, twelve bit number out of table.
	30	SCR 6	0346	Scale to 6 bits.
	31	STH I 6	1340	h-index location t and store data in half-word table.
	32	XSK I 7	0227	Done 100 words yet?
	33	JMP 27	6027	No, get next word.
	34	HLT	0000	Yes. Stop program.

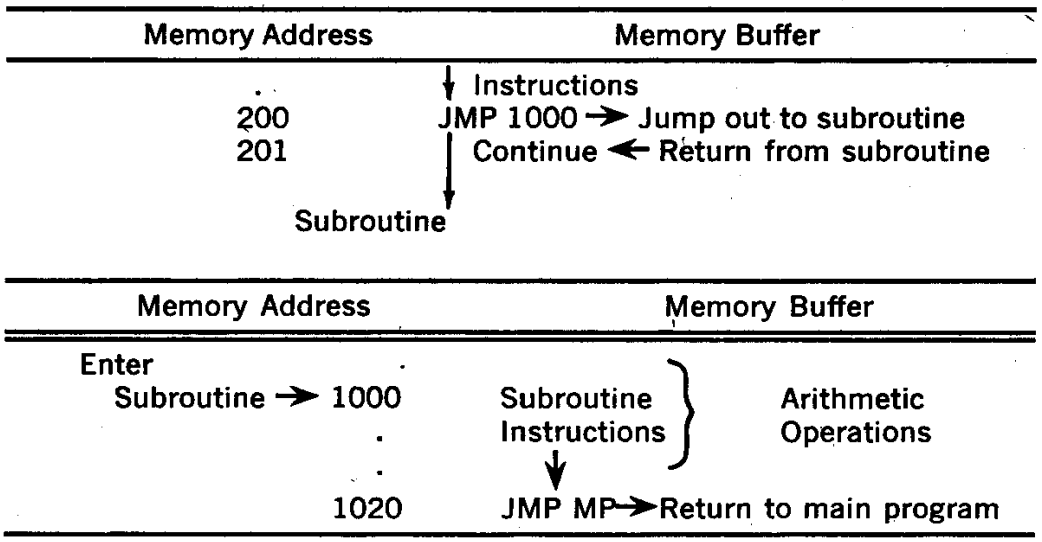
SUBROUTINE TECHNIQUES

Before describing the remaining instructions, some mention should be made of the technique of writing subroutines. Frequently a program has to execute the same set of instructions at several different places in the program sequence. In this case it is an inefficient use of memory registers to write out the same set of instructions each time it is needed. It is more desirable to write the instructions once as a separate, or "sub", routine to which the program can jump whenever these instructions are to be executed. Once the instructions in the subroutines have been executed, the subroutine should return control (jump back) to the main program.

For example, suppose that in two different places in a program we must execute the same set of arithmetic operations. Visualize the general structure of such a program as follows:

Main Program

Memory Address		Memory Buffer
Start →	100	Main
	.	Program
	.	Instructions
	.	↓
	150	JMP 1000 → Jump out to subroutine
	151	Continue ← Return from subroutine
	.	Main
	.	Program



It appears from this example that jumping to the subroutine from the main program (at locations 150 and 200) is straightforward. The subroutine must be able to return control to the main program, however, reentering it at a different place each time the subroutine is finished. That is, the JMP instruction at location 1020 must be changed so that the first time the subroutine is used it will return to the main program via a JMP 151 and the second time via a JMP 201.

It will be remembered that every time the computer executes a JMP instruction (other than JMP 0) at any location p, the instruction JMP p + 1 replaces the contents of register 0. Thus, when JMP 1000 is executed at location 150, a JMP is automatically stored in register 0, saving the return point for the subroutine. The subroutine might retrieve this information in the following way:

Subroutine

Memory Address		Memory Buffer	Effect
Enter Subroutine →	1000	LDA	C(0)→C(ACC); i.e., JMP p + 1→C(ACC).
	1001	0	
	1002	STC 1020	C(ACC)→C(1020).
	.	.	
	.	.	Execute arithmetic operation
	1020	(JMP p + 1)	Return to main program.

A simple JMP 0 in location 1020 clearly suffices when the subroutine does not, during its execution, destroy the contents of register 0. In this case, the instructions in locations 1000-1002 would be unnecessary.

A problem arises in the above example when the subroutine is not free to use the accumulator to retrieve the return point. Another method, using the SET instruction, is possible when there is an available p register.

Memory Address	Memory Buffer	Effect
Enter Subroutine → 1000	SET 10	C(0)→C(10) i.e., JMP p + 1 is saved in a free β register.
1001	0	
.	.	Execute arithmetic operations; the accumulator has not been disturbed. Return to main program by jumping to register 10.
.	.	
.	.	
1020	JMP 10	

A third possibility is the use of the "LINC" instruction "DJR".
DJR - 0006.

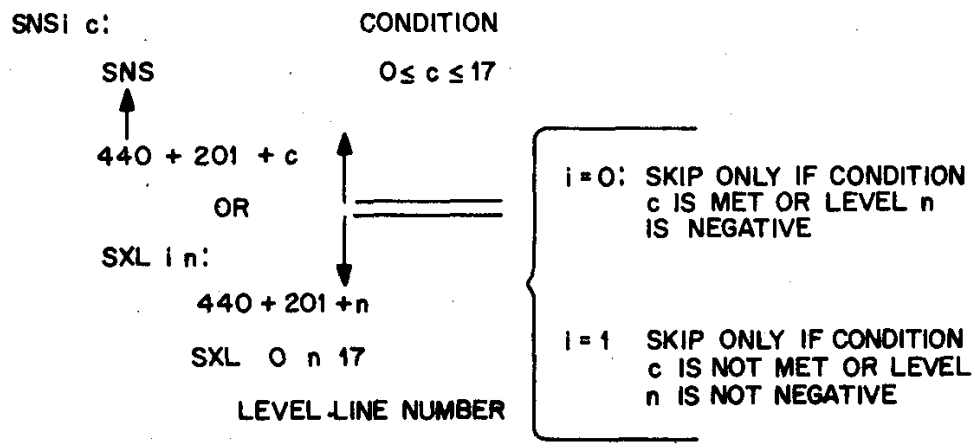
Disable the storing of the return jump (JMP PC) in memory *location zero* for the *next* "JMP" instruction.

The example below shows the use of a *DJR* instruction in a subroutine

Memory Address		Instruction	
Enter Subroutine	1000	.	
		.	
	1050	DJR	Inhibit saving PC in location 0 on next *JMP* instruction.
	1051	APO	Is AC Positive.
	1052	JMP 1035	No, go back to 1035. Do not save PC in LOC 0.
	1053	JMP 0	JMP to LOC 0 to return to main program.

THE SKIP CLASS INSTRUCTIONS

Instructions belonging to the skip class test various conditions of the accumulator, the keyboard, the tapes, and the external level lines of the data terminal module. Coding for these instructions includes the condition or level line to be checked and an option to skip or met skip when the condition is not or the external level is negative.



In these instructions the I-bit can be used to invert the skip decision. When I=0, the computer skips the next register in the instruction sequence when the condition is met or external level is negative. However, when I=1, the computer skips when the condition is not met or the external level is not negative. Otherwise the computer always goes to the next register in the sequence.

The four situations which may arise are summarized in the following table. The skip class instruction is assumed to be in register p.

BRANCHING IN SKIP CLASS INSTRUCTIONS

i	Condition met or level negative?	Location of next instruction
0	yes	p + 2 (skip)
0	no	p + 1
1	yes	p + 1
1	no	p + 2 (skip)

SNS IX instructions test 16 conditions, which, because of their variety, are described with different 3-letter expressions. Thus the AZE I instruction already presented is the same as SNS I 10. Another instruction, APO I, synonymous with SNS I 11, checks to see whether the accumulator is positive (bit 0 = 0):

Memory Address	Memory Buffer		Effect
p	APO	400+11	If C(bit 11 of ACC) = 0, go to
p + 1	←	—	p+2 for the next instruction;
p + 2	←	—	if C(bit 11 of ACC) = 1, go to p+1.

Case i=1

Memory Address	Memory Buffer		Effect
p	APO I	400+20+11	If C(bit 11 of ACC) = 1, go to
p + 1	←	—	p+2 for the next instruction;
p + 2	←	—	if C(bit 11 of ACC) = 0, go to p+1.

Other SNS variations which whether C(1) = 0, (LZE I, code 452 + 201, which is synonymous with SN8 I 2) or whether one of the 6 sense switches on the console is up (SNS I 0, SNS I 1, . . . , SNS I 5) SNS 16 is coded as the special instruction "SKP" (on-conditional skip)

The SKL I n instruct (skip on negative external level) checks for the presence of a −3v level on external line n, $0 \leq n \leq 13$, at the data terminal module. It is often used with the operate instruction, discussed in the next section, to help synchronize the LINC mode with external equipment.

The skip instruction KST I (key struck), code 415 + 20I, checks whether a keyboard key has been struck. SKT I is synonymous with SXL I 15.

To illustrate the use of these instructions, the following program counts the signal peaks above a certain threshold, 100_8' for a set of 1000_8 samples appearing on input line 13. The number of peaks exceeding the threshold will be left in the accumulator.

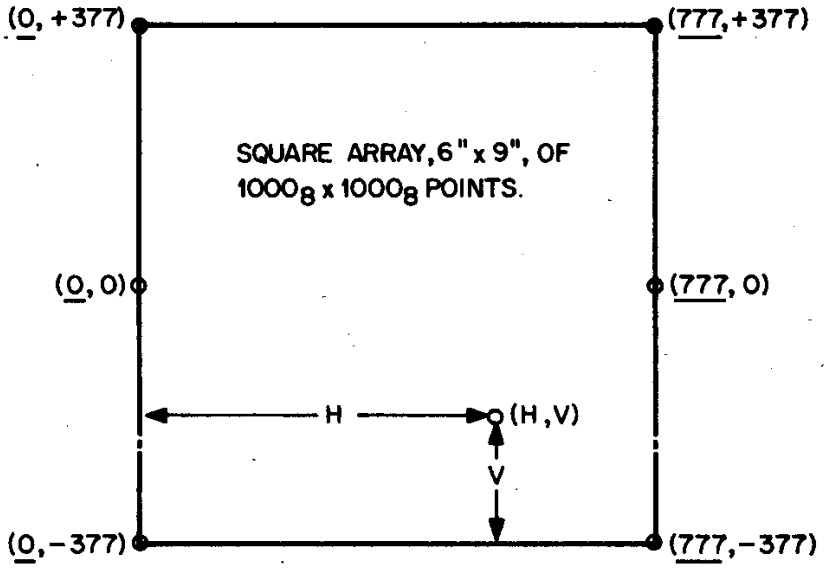
Memory Address	Memory Buffer		Effect
7	[−n]	[−]	Counter for 1000 samples.
10	[n]	[−]	Counter for number above 100_8.
.	.	.	
.	.	.	
.	.	.	
→ 1500	SET I 7	0677	Set register 7 to count 1000 samples.
1501	−1000	6777	
1502	SET I 10	0070	Clear register 10 to count peaks.
1503	0	0000	
1504	SAM 13	0113	Sample input line 13 and subtract 100 from the sample value.
1505	ADA I	1160	
1506	−100	7677	
1507	APO I	0471	Is the accumulator positive?
1510	XSK I 10	0230	If so, the value was above 100; add 1 to the counter. If not, skip the instruction in location 1510.
1511	XSK I 7	0227	Index register 7 and test.
1512	JMP 1504	7504	If 1000 samples have not been taken, return.
1513	IDA	1000	If 1000 samples have been taken, put the number of those above 100 into the accumulator and halt.
1514	10	0010	
1515	HLT	0000	

counting samples exceeding a threshold

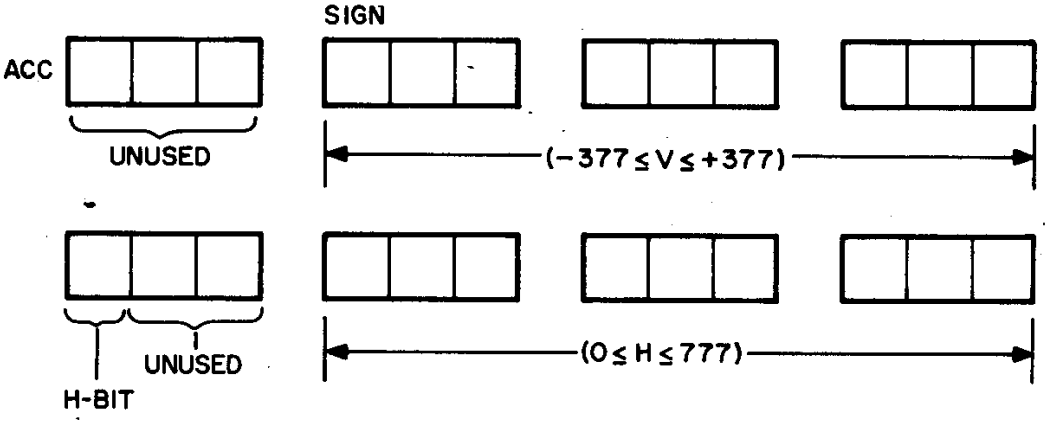
THE LINC SCOPES AND THE DISPLAY INSTRUCTIONS

The PDP-12 has a cathode ray tube display device called the VR-12, which is capable of presenting an array of 512_{10} by 512_{10}. spots (1000_8 by 1000_8). Special instruction, DIS I a (display), code 140 + 20I + a, monentarity produces a bright sopt at one point in this array. The horizontal (H) and vertical (V) coordinates are specified in the accumulator and in a. The vertical coordinate, $-377_8 \leq V \leq +377_8$, is held in the accumulator during a DIS I a instruction; the horizontal coordinate, $0 \leq H$

$\leq 777_8$, is held in register a, $0 \leq a \leq 17$. The spot in the lower left corner of the array has the coordinates (0, −377):



The coordinates are held in the rightmost 9 bits of register a and the accumulator,



so that if C(ACC) = 641, i.e. −136, and C(5) = 430, DIS 5 causes a spot to be intensified at (430, −136) on the scope, Both channels are positioned at the same time. The production of a bright spot on either channel depends upon the state of the leftmost bit (the H-bit) of register a and an external channel selector located on the face of the display scope. If h=0, then the spot is produced via display channel 1 0; if h=1, then the spot is produced via display channel 1. The scope may be manually set to intensify channel 1 0, channel 1, or both. The I-bit in DIS I a is used in the usual way to specify whether to index the right 10 bits of register a before brightening the spot. This indexing, of course, also increases the horizontal coordinate by one. To illustrate, the following program will display a continuous horizontal line through the middle.

Memory Address	Memory Buffer		Effect
5	[0, H]	[−]	Horizontal coordinate and channel selection.
.	.	.	
.	.	.	
.	.	.	
→ 20	SET i 5	0065	Set 5 to channel 0 and horizontal coordinate = 0.
21	0	0000	
22	CLR	0011	Vertical coordinate = 0→C(ACC).
23	→DIS i 5	0165	Index H (actually index entire rightmost 10 bits) and display. Repeat endlessly.
24	JMP 23	6023	

Horizontal Line Scope Display

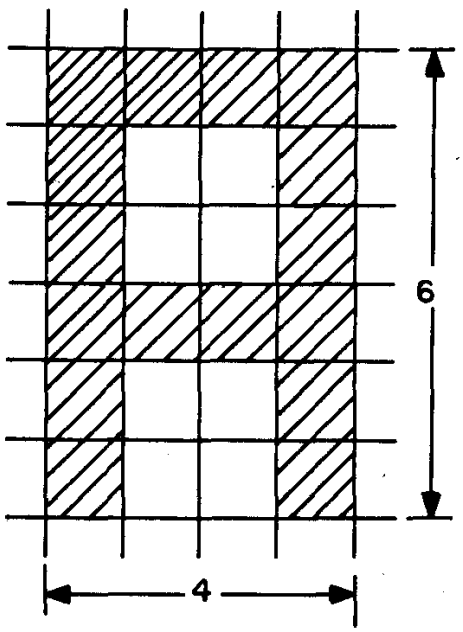
Another example displays as a curve the values found in a set of consecutive registers, 1400-1777. The vertical coordinates are the most signficant 9 bits of each value. Since these are only 400_8 points to display, the curve will be positioned in the middle of the scope. Channel 1 is used.

Memory Address	Memory Buffer		Effect
10	[X]	[−]	Address of vertical coordinates.
11	[1, H]	[4000+H]	Channel select and horizontal coordinate.
.	.	.	
.	.	.	
.	.	.	
→ 300	→SET i 10	0070	Set 10 to beginning address minus 1.
301	1377	1377	
302	SET i 11	0071	Set 11 to select channel 1 and to begin curve H = 200.
303	1 177	4177	
304	→LDA i 10	1030	Load ACC with value and scale right 3 places to position it as vertical coordinate.
305	SCR 3	0343	
306	DIS i 11	0171	Index the H coordinate and display.
307	XSK 10	0210	Check to see if X(10) = 1777.
310	JMP 304	6304	If 400_8 points have not been displayed, return to get next point.
311	JMP 300	6300	If X(10 = 1777, return to repeat entire display.

Curve Display of a Table of Numbers

CHARACTER DISPLAY

Display scopes are frequently used to display characters, for examples keyboard characters, as well as data curves. Character display is somewhat more complicated since the point pattern must be carefully worked out in conjunction with the vertical and horizontal coordinates for each point. For example, to display the letter A, the array of the scope might look like:

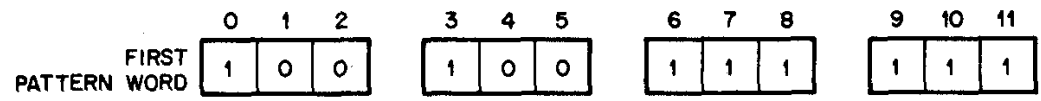


	FIRST WORD		SECOND WORD	
	6	0	6	0
	7	1	7	1
	8	2	8	2
	9	3	9	3
	10	4	10	4
	11	5	11	5

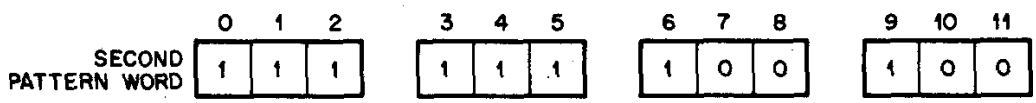
where the shaded areas of figure a represent point which are intensified, and the white areas points not intensified; the total area represented is 6 vertical positions by 4 horizontal positions. If, for example, the lower left point has the coordinates (400,0), then the upper right point has the coordinates (403, 5).

The programmer could, or course, store the H and V coordinates for every intensified point of the character in a table in the memory, but the letter A alone, for instance, would require 32_{10} registers to hold both coordinates for all the points which are intensified. Instead he arbitarily decides upon a scope format, say 4 x 6, and makes up a pattern word in which 1's points to be intensified and 0's points which are not intensified. To specify a 4 x 6 pattern of 24 bits requires 2 memory registers.

For efficiency of programming, the points are displayed in the order shown numerically in figure b, i.e., from lower left to upper right, column by column. Examining bit 11 of the pattern word first, bit 10 next, bit 9, etc., the pattern word for the left half of the letter A (the left two columns) looks like:



the pattern word for the right half of the letter looks like:



An index class instruction, SRO I B (skip rotate), code 1500 + 20I + B, facilitates character display with the kinds of pattern words described above. SRO i B directs the LINC to skip the next register in the instruction sequence when bit 11 of the specified memory register contains 0. If bit 11 contains 1, the computer does not skip. In either case, however, after examining bit 11, the contents of the specified memory register are rotated 1 place to the right. Therefore, repeating the SRO instruction (with reference to the same memory register) has the effect of examining first bit 11, then bit 10, bit 9, etc. Executing the SRO instruction twelve times, of course, restores the memory word to its original configuration.

The following example repeatedly displays the letter A in the middle of the scope, using register 7 to hold the address of the first pattern word and register 6 to hold the H coordinate. Since 4 x 6 contiguous points on the scope array define an area too small to be readable, a delta of 4 is used to space the points, so that if the first point is intensfied at coordinates (370, 0) the second point will be at (370, 4), the 7th point at (370, 0) etc. (This produces characters approximately 0.5 cm high).

Memory Address	Memory Buffer		Effect
6	[0, H]	[−]	Channel selection and H coordinate.
7	[X]	[−]	Address of pattern word.
.	.	.	
.	.	.	
.	.	.	
→ 60	SET I 6	0066	Set H coordinate = 370 for lower left point. Select channel 0.
61	0 370	0370	
62	SET I 7	0067	Set 7 to address of first half of pattern.

Memory Address	Memory Buffer		Effect
63	110	0110	
64	LDA I	1020	Initial V coordinate = −10→C (ACC).
65	−10	7767	
66	SRO 7	1507	Skip to location 70 if bit of pattern word is 0. Rotate the pattern word 1 place to right.
67	DIS 6	0146	If bit 11 of pattern word was 1, display one point.
70	ADD 75	2075	Add 4 to V coordinate in ACC.
71	SRO I	1520	Skip to location 74 when 6 bits of pattern word have been examined. Rotate C(72) 1 place to right.
72	3737	3737	
73	JMP 66	6066	Return to examine next bit of pattern word when bit 0 of C(72) = 1.
74	LDA I	1020	When bit 11 of C(72)=0, 6 points have been examined. Increase H coordinate by 4 to do next column.
75	4	0004	
76	ADM	1140	
77	6	0006	
100	SRO I	1520	Check to see if 2 columns have been displayed. Rotate C(101) 1 place to right.
101	2525	2525	
102	JMP 64	6064	Two columns have not been displayed; return to do next column.
103	XSK I 7	0227	Two columns have been displayed; index address of the pattern word.
104	SRO I	1520	Skip to 107 is both halves of pattern have been displayed.
105	2525	2525	
106	JMP 64	6064	Return to display 2nd half of pattern.
107	JMP 60	6060	Entire pattern has been displayed once. Return and repeat.
110	4477	4477	Pattern words for letter A.
111	7744	7744	

Character Display of the Letter A

The SRO instructions at locations 71, 100, and 104 determine when 1 columns, and 4 columns have been displayed. After each column the H coordinate is increased by 4 and the V coordinate reset to −10. After 2 columns the address of the pattern word is indexed by one, and after 4 columns the entire process is repeated. DSC I B (display character), code 1740 + 20I + B, is the last of the index class instructions; it directs the LINC to display the contents of one pattern word, or 2 columns of points. Register B holes the address of the pattern word and the I-bit is used in the usual way to index X (B). The points are displayed in the format

described above, i.e., 2 columns of 6 points each with a delta of 4 between points. The pattern word is examined from right to left beginning with bit 11 and points are plotted from lower left to upper right, as above.

When executing a DSC instruction the computer always takes the H coordinate and channel selection from register 1. The delta of 4 if automatically added to X (1) every time a new column is begun; furthermore, this indexing is done before the first column is displayed, so that if register 1 initially contains 0364, the first column is displayed at H = 370, the second at H = 374, and register 1 contains 0374 at the end of the instruction.

The vertical coordinate is, as usual, taken from the accumulator, and again + 4 is automatically added to C(ACC) between points. The rightmost 5 bits (bits 7-11) of the accumulator are always cleared at the beginning of a DSC instruction, so that if initially C(ACC) = + 273, the first point will be displayed at V −240, the second at V −244, etc. Characters can therefore be displayed using the DSC instruction only at vertical spacing of 40 on the scope, e.g., at initial vertical coordinates equal to −77, −37, 0, +40, +100, etc. The rightmost 5 bits of the accumulator always contain 30_8 at the end of a DSC instruction, so that if the initial C(ACC) = +273, the initial V equals +240 and C(ACC) equals +270 at the end of the instruction.

To display a character defined by a 4 x 6 pattern two DSC instructions are needed. The following example repeatedly displays the letter A in the middle of the scope, just as the program on page 48 (example 20) does, but with greater efficiency using the DSC instruction. Since an initial V = −10 is not possible with DSC, the program uses V=0.

Memory Address	Memory Buffer		Effect
1	[0, H]	[−]	Channel selection and H coordinate.
.	.	.	
.	.	.	
.	.	.	
7	[X]	[−]	Address of pattern word.
.	.	.	
.	.	.	
.	.	.	
→ 60	CLR	0011	Initial V = 0 → C(ACC).
61	SET I 1	0061	Set 1 to initial H coordinate minus 4, and select channel 0.
62	0364	0364	
63	SET I 7	0067	Set 7 to address of first half of pattern.
64	110	0110	
65	DSC 7	1747	Display, using 1st pattern word, the left 2 columns of the letter A, at initial coordinates of (370, 0).

Memory Address	Memory Buffer		Effect
66	DSC I 7	1767	Index address of pattern word, X(7), and display right 2 columns of the letter A at initial coordinates of (400, 0).
67	JMP 61	6061	Return and repeat.
.	.	.	
.	.	.	
.	.	.	
110	4477	4477	Pattern words for letter A.
111	7744	7744	

Character Display of the Letter A using DSC

After the first DSC instruction (at location 65), C (1) = 374 and C (ACC) = 30. After second DSC instruction, C (1) = 0404, C (7) = 0111, and C (ACC) = 30. C (110) and C (111) are unchanged. By adding more pattern words at locations 112 and following locations, and repeating the DSC 17 instruction, it is possible to display an entire row of characters.

The following program repeatedly displays a row of six digits. The pattern words for the characters 0-9 are located in a table beginning at 1000; i.e., the pattern words for the character 0 are at 1000 and 1001, for the character 1 at 1002 and 1003, etc. Keyboard codes for the characters to be displayed are located in a half-word table from 1400-1402 i.e., the first code value is LH (1400), the second RH (1400), etc. The program computes the address of the first pattern word for each character as it is retrieved from the table at 1400.

Memory Address	Memory Buffer		Effect
1	[1, H]	[−]	Channel selection and H coordinate.
2	[−n]	[−]	Counter for number of characters.
3	[h, X]	[−]	Address of keyboard code values.
4	[X]	[−]	Address of pattern word.
.	.	.	
.	.	.	
.	.	.	
→20	SET I 2	0062	Set 2 to count number of characters displayed.
21	−6	7771	
22	SET I 3	0063	Set 3 for loading code values beginning at LH (1400).
23	1, 1377	5377	
24	SET I 1	0061	Set 1 to initial H coordinate minus 4, and select channel 1.
25	1 344	4344	
26	LDH I 3	1323	Half-word index register 3 and put code value into accumulator.

Memory Address	Memory Buffer		Effect
27	ROL 1	0241	Compute address of pattern word by multiplying code value by 2 and adding beginning address of pattern table.
30	ADA I	1120	
31	1000	1000	
32	STC 4	4004	Address of pattern word → C (4); 0 → C (ACC).
33	DSC 4	1744	Display character at initial V = 0, and initial H = C (1) + 4.
34	DSC I 4	1764	
35	LDA I	1020	
36	4	0004	
37	ADM	1140	Increase H by 4 to provide space between characters.
40	1	0001	
41	XSK I 2	0222	Index X (2) and check to see if six characters have been displayed. If not, return to get next character. If so, return to repeat entire display.
42	JMP 26	6026	
43	JMP 20	6020	

Displaying a Row of Characters

Suppose, for example, that one of the six code values is 07. The pattern words for the character 7 are at locations 1016 and 1017. Multiplying the code value 07 by 2 ($7 \times 2 = 16_8$) and adding the beginning address of the pattern table ($16 + 1000 = 1016$) gives us the address of the first pattern word for the character 7. It should be clear that pattern words for all the keyboard characters could be added to the pattern table; by organizing the pattern table to correspond to the ordering of the keyboard code values, the same technique of "table look-up" using the code values to locate the pattern could be used to display any characters on the keyboard."

Half Size Characters

The previous discussion on display characters gave the parameters for full size characters. Bit 4 of the special functions register controls the character size. This bit is normally set to a one to provide full size. However, by clearing bit 4 of the special functions register, half size characters can be displayed. The only difference in displaying half size characters is the calculations required in determining where to place the characters on the display.

Instead of a delta of 4 between points, a delta of 2 is used. Each characters (2 x 6 pattern) is therefore, only 14 points high and 4 points wide.

The line spacing for a half size character is twenty points, and the incrementing of memory location 1 is by +2.

ANALOG INPUT AND THE SAMPLE INSTRUCTION

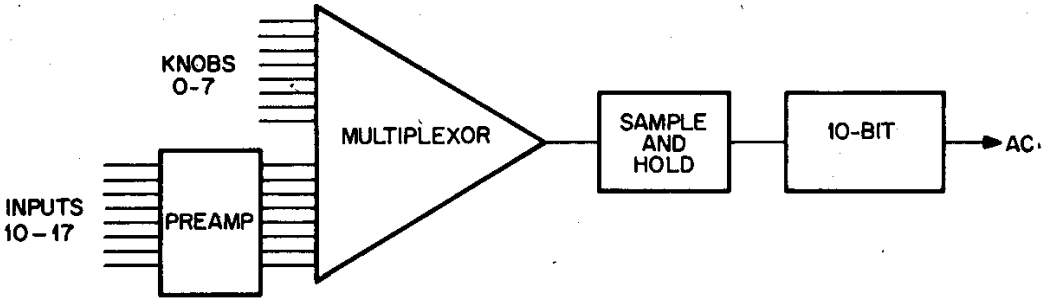
The AD12 Analog-Digital Converter provides for easy program access to a basic 16-input converter with a full 10-bit conversion range (-777_8 to $+777$).

Inputs

Eight channels are enternal inputs via phone jacks. Input range is ± 1 volt and feed through preamplifiers to the converter. Other input ranges of 0 to +2 volts, −5 to +5 volts, and −10 to +10 volts are available as a low-cost option.

The other eight channels are controlled by continuously variable knobs. The converter is sensitive to the middle seven turns of these ten-turn, lock-to-lock knobs. They can be used to establish parameters, set threshold levels., etc.

Functional Diagram



The input knobs are numbered 0-7, and the injut jacks are numbered 10-17.

Sample Instruction

The LINC-mode instruction SAM N (where N $0 \leq N \leq 17$), is used to select the input channel and initiate the conversion.

Normal Mode

In Normal Mode, when the SAM instruction is issued, the input channel is selected and sampled. Then the signal is converted and upon completion of the conversion process, the contents of the converter buffer is transferred to AC_{3-11}. The sign bit is placed in AC 0-2. The entire operation takes 18.2μs. During the time, the PDP-12 processor simply waits for the completion of the SAM operation.

The below example illustrates one of the uses of the potentiometers. This program plots the contents of a 512_{10} word segment of memory registers 0-1777. Location of the segment is selected by rotating knob 5, whose value is used to determine the address at which to begin the display. As the viewer rotates the knob, the display effectively moves back and forth across the memory.

Memory Address	Memory Buffer		Effect
12	[X]	[−]	
13	[1, H]	[−]	For channel selection, H coordinate, and counter.
.	.	.	
.	.	.	
.	.	.	
→ 20	SET I 13	0073	Set register 13 to select channel 1 and to begin displaying at H = 0.
21	4777	4777	
22	SAM 5	0105	Sample knob 5, add 400 to make the value positive, rotate left 1 place to produce an address for display, and store in register 12.
23	ADA I	1120	
24	400	0400	
25	ROL 1	0241	
26	STC 12	4012	
27	LDA I 12	1032	Index the address of the vertical co-ordinate, and put the coordinate into the ACC. Position it for display, index the H coordinate and display.
30	SCR 3	0343	
31	DIS I 13	0173	
32	XSK 13	0213	Check to see whether 510_{10} points have been displayed. (X(13) = 1777?).
33	JMP 27	6027	If not, return to display next point.
34	JMP 20	6020	If so, return to reset counter and get new address from knob 5.

Moving Window Display under Knob Control

At locations 23-25, a memory address is computed for the first vertical coordinate by adding 400 to the sample value. This leaves the value in the range +1 to + 777; it is then rotated left 1 place to produce an initial address in the range 2-1776 for the display.

A second example illustrates the technique of accumulating a frequency distribution of sampled signal amplitudes appearing on line 12, and displaying it simultaneously as a histogram. The distribution is compiled in a table at locations 1401-1777, and the sample values themselves from the addresses for table entry. Registers 1401-1777 are initially set to −377 so that the histogram will be from the bottom of the scope.

Note, at locations 104 and 105, because of using memory registers 1401-1777, the same index register (register 2) may be interpreted both as address (location 104) and counter (location 105). A separate counter is not needed because the final address (1777) will serve as the basis of the skip decision for the XSK instruction. The same is true at location 124 and 134.

Memory Address	Memory Buffer		Effect
2	[X]	[−]	Address of vertical coordinates.
3	[0, H]	[−]	Channel selection and H coordinate.
.	.	.	
.	.	.	
.	.	.	
100	SET I 2	0062	
101	1400	1400	
102	LDA I	1020	Initial routine to set registers
103	−377	7400	1401-1777 to −377.
104	STA I 2	1062	
105	XSK 2	0202	
106	JMP 104	6104	
107	SET I 2	0062	Set register 2 to initial address minus one of vertical coordinates.
110	1400	1400	
111	SET I 3	0063	Set register 3 to select channel 0 and begin display at H=201.
112	200	0200	
113	SAM 12	0112	Sample input line 12.
114	SCR 1	0341	
115	ADA I	1120	Add 1400+200 to the sample value
116	1600	1600	to form an address for recording
117	STC 123	4123	the event and store.
120	LAD I	1020	
121	1	0001	Add 1 to the contents of the register
122	ADM	1140	just located by the sample value to
123	[−]	[−]	record the event.
124	LDA I 2	1022	Index register 2 and put a histogram value in the accumulator.
125	DIS I 3	0163	Index the H coordinate and display.
126	DIS 3	0143	Display without indexing.
127	ADA I	1120	
130	−1	7776	Fill in the bar by decreasing the
131	SAE I	1460	vertical coordinate by 1 and
132	−400	7377	continuing the display until a point
133	JMP 126	6126	is displayed at V= −377.
134	XSK 2	0202	When bar is finished, check to see whether 377 values have been displayed. (X(2) = 1777?).
135	JMP 113	6113	If not, return to get next sample.
136	JMP 107	6107	If so, return to reset vertical coordinate address, H coordinate, and repeat.

Histogram Display of Sampled Data

FAST-SAMPLE MODE

In Fast-Sample Mode, the PDP-12 processor is allowed to proceed while the called for conversion operations is in program. The conversion still requires 18.2μs, but the order of events is changed. When the special function fast sample is enabled and a SAM N instruction is issued, the contents of the converter buffer is transferred to the accumulator first (total time 1.6 μs). The program then continues, and concurrently, a new sample and conversion is initiated on channel N. The results will not be sent to the accumulator until the execution of the next SAM instruction. This permits the program to handle incoming data as new data is being converted.

Fast-Sample Mode is initiated by setting bit 5 of the special function register to a one.

```
 LDA I
0100      /SET AC5=1
 ESF      /ENABLE FAST SAMPLE FUNCTION
```

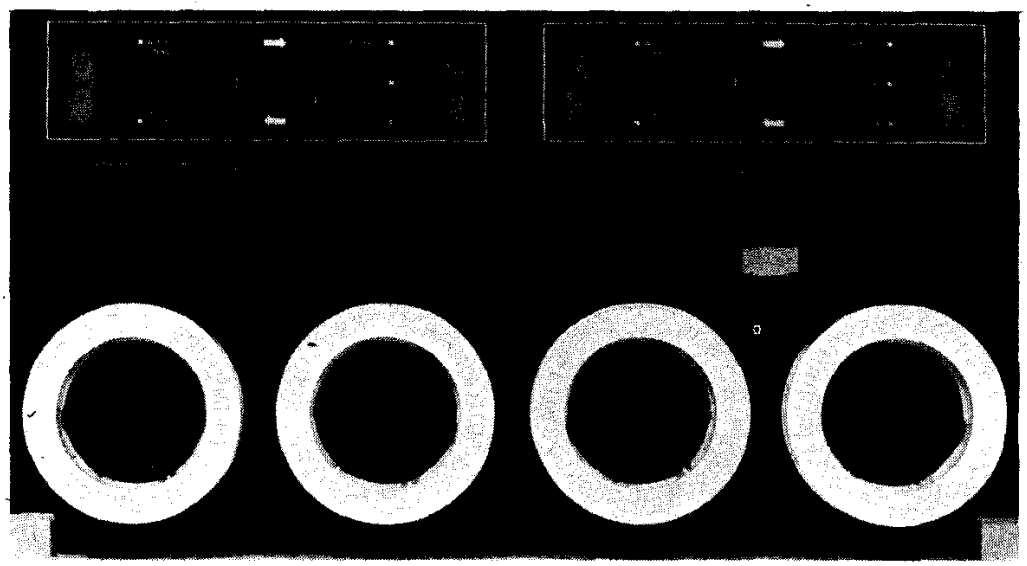
In either mode the maximum conversion rate is 18.2 μs per sample but fast sample doesn't hold up the CP while waiting for the conversion.

Below is an example that will sample 1000 data points and save them.

Memory Address	Instructions	Code	Effect
20	SET I 5	65	Set register t to -1000 to
21	-1000	6777	count no. of samples
22	SET I 6	66	Set register 6 to point to be-
23	777	777	ginning of table
24	SAM 15	115	Sample channel 15
25	STA I 6	1066	Store data in table
26	XSK I 5	0225	Count No. of samples
27	JMP 24	6024	Not done, sample another channel

The normal operation the sampling frequency of the above program would be $\approx 1/2912 \times 10^{-6}$ sec, or $\approx$ 34 KC.

If the fast sample mode is initiated, however, the time would be reduced to $1/18.2 \times 10^{-6}$ sec or about 54 KC. because the computer would not pause for the conversion sequence of the SAM instruction and the "JMP 24" instruction would be executed before the previous SAM would have finished. The computer now must wait only for the previous SAM to be completed.

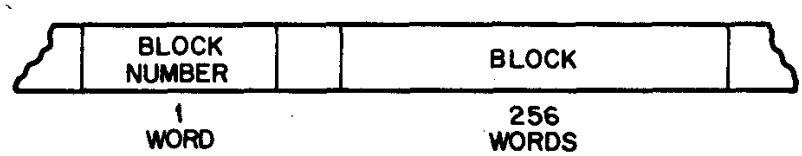


Dual DECtape Transport

Any magnetic tape instruction may refer to either the tape or unit 0 or the tape on unit 1; which unit to use is specified by the instruction itself; only one unit, however, is ever used at one time.

In the PDP-12, handling of magnetic tape and its instructions is done entirely by the computer hardware. This hardware is referred to as the TC-12.

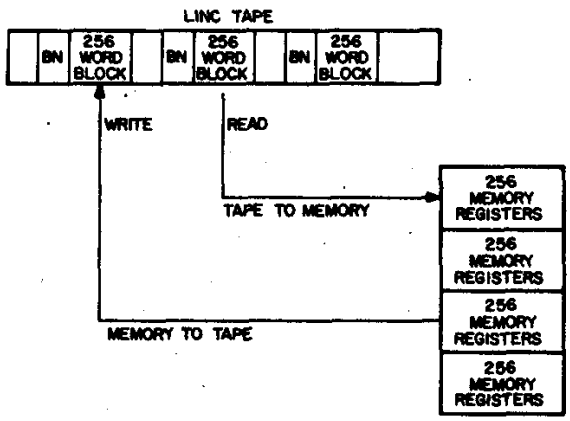
A LINCtape can hold 131,072, 12-bit words of information, or the equivalent of 128_{10} full LINC memories. It is divided into 512_{10} smaller segments known as blocks, each of which contains 256_{10} 12-bit words, a size equal to one-quarter of LINC memory. Blocks are identified on any tape by block numbers, $0\text{-}777_8$; magnetic tape instructions specify which block to use by referring to its block number. A block number (BN) on the tape permanently occupies a 12-bit space preceding the 256 words of the block itself:



There are other special words on the tape, serving other functions, which complete the tape format. Before describing these, however, look more specifically at one of the magnetic tape instructions, RDE I u (read tape)

Block Transfers and Checking

Read tape is one of six magnetic tape instructions which copy information wither from the tape into the LINC memory (reading), or from the memory onto the tape (writing). These are generally called block transfer instructions because they transfer one or more blocks of information between the tape and the Memory:



All magnetic tape instructions are double register instructions. RDE, typical of a block transfer instruction, is written:

Memory Address		Memory Buffer
p	RDEI n	
p + 1	MBLK/TBLK	702 + 20I + 10u
	3 bits/9 bits	1000 MBLK + TBLK

The first register of the instruction has two special bits. The u-bit (bit 8) selects the tape unit: when u = 0, the tape on unit 0 is used; when u = 1, the tape on unit 1 is used. Magnetic tape instructions require that the tape on the selected unit move at a speed of approximately 80 ips. Therefore, if the tape is not moving when the computer encounters a magnetic tape instruction, tape motion is started automatically and the tape control waits until the tape has reached the required speed before continuing with the instruction.

The I-bit (bit 7) specifies the motion of the tape after the instruction is executed. If I = 0, the tape will stop; if I = 1, it will continue to move at 80 ips. It is sometime more efficient to let the tape continue to move, as, perhaps, to execute several magnetic tape instructions in succession. If it stops, it is necessary to wait for it to start again at the beginning of the next tape instruction. Examples of this will be given later.

In the second register of the RDE instruction, the rightmost 9 bits hold the requested block number, BN; that is, they tell the computer which block on the tape to read into the memory. The left 3 bits hold the Memory Block number, MBLK, which refers to the memory. MBLK specifies which Memory Block of memory to use in the transfer. The Memory Blocks

of the LINC memory are numbered 0-7, and refer to the memory registers as follows:

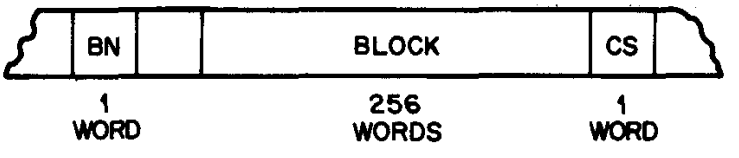
Memory Block	Memory Registers (octal)
0	0-377
1	400-777
2	1000-1377
3	1400-1777
4	2000-2377
5	2400-2777
6	3000-3377
7	3400-3777

Suppose, for example, the programmer wishes to transfer data stored on tape into memory registers 1000-1377. The data is in block 267 and the tape mounted on unit 1.

Memory Address	Memory Buffer		Effect
→200	RDE u	0712	Select unit 1; C (Tape block 267)
201	2/267	1000 x 2 + 267	C (Memory Block 2).

This instruction will start to move the tape on unit 1 if it is not already moving. It then reads block 267 on that tape into Memory Block 2 and stops the tape when the transfer is completed. The computer will go to location 202 for the next LINC instruction. After the transfer the information in block 267 is still on the tape; only memory registers 1000-1377 and the accumulator are affected. Conversely, writing affects only the tape and the accumulator: The memory is left unchanged.

Another special word on the tape, located immediately following the block, is called the checksum, CS:



The checksum, a feature common to many tape systems, checks the accuracy of the transfer of information to and from the tape. On a LINC tape, the checksum is the 2's complement of the sum of the 256 words in the block. Such a number is formed during the execution of another block transfer instruction, WRI I u (write tape). This instruction writes the contents of the specified memory block in the specifled block of the selected tape:

Memory Address		Memory Buffer
p	WRI I u	706 + 20I + 10u
p + 1	MBLK/TBLK	1000 MBLK + TBLK

During the transfer the words being written on the tape are added together without end-around carry in the accumulator. This sum is then 2's complemented and written in the CS space following the block on the tape. After the operation the checksum is left in the accumulator and the computer goes to p + 2 for the next LINC instruction. MBLK, TBLK, I, and u are all interpreted as for RDE.

One means of checking the accuracy of the transfer is to form a new sum and compare it to the checksum on the tape. This happens during RDE; the 256 words from the block on the tape are added together without end-around carry in the accumulator while they are being transferred to the memory. This uncomplemented sum is called the data sum. The checksum from the tape is then added to this data sum and the result, called the transfer check, is left in the accumulator. If the information has been transferred correctly, the transfer check will equal − 0 (7777): the block "checks". Thus, by examining the accumulator after an RDE instruction, the programmer can tell if the block was transferred correctly. The following sequence of instructions does this and reads block 500 again if it does not check:

Memory Address	Memory Buffer		Effect
→300	→RDE	0702	Read block 500, unit 0, into memory block 3. Leave the transfer check in the accumulator and stop the tape.
301	3/500	3500	
302	SAE I	1460	Skip to location 305 if C (ACC) = 777, i.e., if the block checks. If C (ACC) ≠7777, return to read the block again.
303	7777	7777	
304	JMP 300	6300	
305	—	—	
.	.	.	
.	.	.	

The remaining block transfer instructions check transfer automatically. RDC I u (read and check), does in one instruction exactly what the above sequence of instruction does. That is, it reads the specified block of the selected tape into the specified quarter of memory and forms the transfer check* does not equal 7777, the instruction is repeated (the block is reread, etc.). When the block is read correctly, 7777 is left in the accumulator and the computer goes on to the next LINC instruction at p + 2. The RDC instruction is written:

Memory Address		Memory Buffer
p	RDC I u	700 + 20I + 10u
p + 1	MBLK/TBLK	1000 MBLK + TBLK

One of the most frequent uses of instruction which read the tape is to put LINC programs stored on tape into the memory. Suppose the programmer is given a tape, for example, which has in block 300 a program he wants to run. The program is 100_8 registers long, starting in register 1250. He can mount the tape on either unit and then set and execute RDE or RDC in the LEFT and RIGHT SWITCHES. If he uses RDE, he should look at the ACCUMULATOR lights after the transfer to make sure the transfer check = 7777. When double register instructions are set in the toggle switches, the first word is set in the LEFT SWITCHES, and the second in the RIGHT SWITCHES. If the tape is on unit 1, to use RDC the toggle switches should be set as follows:

Console Location	Contents	
LEFT SWITCHES	RDC u	0710
RIGHT SWITCHES	2/300	2300

MBLK = 2 because the program in block 300 must be stored in memory registers 1250-1347, which are located in MBLK 2. Depressing the DO Key will cause the PDP-12 LINC mode to read the block into the proper Memory Block and check it. Start at 5250 from the console, using I/O Preset, Start LS.

The remaining block transfer instructions will be described later.

A non-transfer instruction, CHK I u (check tape), makes it possible to check a block without desroying information in the memory. This instruction does exactly what RDE does, except that the information is not transferred into the memory; that is, it reads the specified block into the accumulator only, forms the data sum, adds it to the checksum from the tape, and leaves the result, the transfer check, in the accumulator. Since this is a non-transfer instruction, MBLK is ignored by the computer. Otherwise this instruction is written as are the other instructions:

Memory Address	Memory Buffer	
p	CHK I u	707 + 201 + 10u
p + 1	TBLK	TBLK

The following program checks sequentially all the blocks on the tape on unit 0. The program starts at location 200. If a block does not check, the program puts its block number into the accumulator and halts at location 221. To continue checking, reenter the program at location 207. The program will halt at location 216 when it has checked the entire tape.

Memory Address	Memory Buffer		Effect
Start→200	CLR	0011	Store 0 in register 203 as first block number.
201	STC 203	4203	
202	CHK I	0727	Check the block specified in register 203; transfer check C (ACC); the tape continues to move.

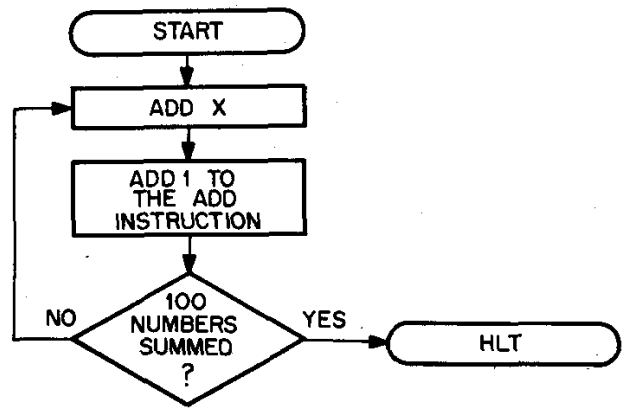
Memory Address		Memory Buffer		Effect
	203	TBLK ←	(−)	
	204	SAE I	1460	If the transfer check = −0, skip to location 207.
	205	7777	7777	
	206	JMP 217	6217	If the block does not check, jump to location 217.
Reenter→	207	LDA I	1020	Add 1 to the block number in register 203, and leave the sum in the accumulator.
	210	1	0001	
	211	ADM	1140	
	212	203	0203	
	213	SAE I	1460	
	214	1000	1000	If all blocks have been checked, skip to location 215. Otherwise return to check next block.
	215	JMP 202	6202	
	216	HLT ←	0000	
	217	LDA	1000	
	220	203	0203	Load the block number of the block which failed into the accumulator, and halt.
	221	HLT	0000	

Simple Check of an Entire Tape

A block transfer instruction, WRC I u, (write and check), combines the operations of the instructions WRI and CHK, and, like read — and — check, repeats the entire process if the check fails. That is, WRC writes the contents of the specified memory block in the specified tape block, forms the checksum in the accumulator and writes the checksum onto the tape. It then checks the tape block just written. If the resulting transfer check does not equal −0, the tape block is rewritten and rechecked. When the tape block checks, 7777 is left in the accumulator and the computer goes on to the next LINC instruction at p + 2. WRC is written:

Memory Address		Memory Buffer
p	WRC I u	704 + 20I + 10u
p + 1	MBLK + TBLK	1000MBLK + TBLK

This write-and-check process may be diagrammed:



The following sequence illustrates the use of some of the tape block transfer instructions. Since the LINC mode memory is small, a program must frequently be divided into sections which will fit into tape blocks, and the sections read into the memory as needed. This example saves (writes) the contents of Memory Blocks 2 of memory (registers 1000-1377) on the tape. It then reads a program section from the tape into memory blocks 1, 2, 3 (register 400-1777) and jumps to location 400 to begin the new section of the program. Assume that the tape is on unit 0. Memory Block 2 will be saved in tape block 50; the program to be read from the tape is in blocks 201-203:

Memory Address	Memory Buffer		Effect
→100	WRCI	0723	C (quarter 2) → C (block 50); transfer is checked, and the tape continues to move.
101	2050	2050	
102	RDC I	0702	C (block 201) →C (MBLK 1), and
103	1/201	1201	C (block 202) C (MBLK 2); transfers are checked and the tape
104	RDC I	0720	
105	2/202	2202	continues to move.
106	RDC	0720	C (block 203) →C (MBLK 3); transfer is checked and the tape stops.
107	3/203	3202	
110	JMP 400	6400	Jump to the new section.
	.	.	
	.	.	
400	→(—)	(—)	

Dividing Large Programs Between Tape and Memory

At the end of the above sequence, the contents of memory registers 400-1777 and tape block 50 have been altered.

Another program repeatedly fills memory block 3 with samples from input line 14 and writes the data in consecutive blocks on tape beginning at block 200. The number of blocks of data to collect and save is specified by the setting of the RIGHT SWITCHES. When the requested number has been written, the program saves itself in block 177 and halts. The tape is on unit 1.

Memory Address	Memory Buffer		Effect
10	(X)	(—)	Memory address for storing samples.
11	(—n)	(—)	Counter.
.	.	.	
.	.	.	
→ 1000	RSW	0516	C(RIGHT SWITCHES) C(ACC).
1001	COM	0017	Complement the number and store
1002	STC 11	4011	in register 11.

Memory Address	Memory	Buffer	Effect
1003	SET I 10	0070	Set register 10 to store samples beginning at 1400.
1004	1377	1377	
1005	SAM 14	0114	Sample input line 14, store value and repeat 400₈ samples have been taken.
1006	STA I 10	1070	
1007	XSK 10	0210	
1010	JMP 1005	7005	
1011	WRC u	0714	When MBLK 3 is full, write it on take and check the tape stops.
1012	(3/200)	(−)	
1013	LDA I	1020	Add 1 to the TBLK in register 1012.
1014	1	0001	
1015	ADM	1140	
1016	1012	1012	
1017	XSK I 11	0231	Index the counter and skip if the requested number has been collected.
1020	JMP 1003	7003	If not, return.
1021	WRC u	0714	If so, write this program in tape block 177, and check the transfer and stop of the tape.
1022	2/177	2177	
1023	HLT	0000	Halt the computer.

Collecting Data and Storing on Tape

Since the program saves itself when finished, the operator can continue to collect data at a later time by reading tape block 177 memory block 2, and starting at 1000. Since the TBLK in location 1012 will have been saved, the data will continue to be stored in consecutive blocks.

GROUP TRANSFERS

Two other blocks instructions, similar to RDC and WRC, permit a program to transfer as many as 8 blocks of information with one instruction. These are called the group transfer instructions; they transfer information between consecutive quarters of the memory and a group of consecutive blocks on the tape. Suppose, for example, that we want to read 3 blocks from the tape into memory blocks 1, 2 and 3. The 3 tape blocks are 51, 52, and 53. Using the instruction RCG I u(read and check group), write:

Memory Address		Memory Buffer
p	RCG I u	701 + 20I + 10u
p+1	2051	2051

The first register specifies the instruction, the tape unit, and the tape motion as usual. The second register, however, is interpreted somewhat differently. It uses TBLK to select the first block of the group. In addition, the rightmost 3 bits of TBLK specify also the first memory block of the group. That is, block 51 will be read into memory block 1, (block 127

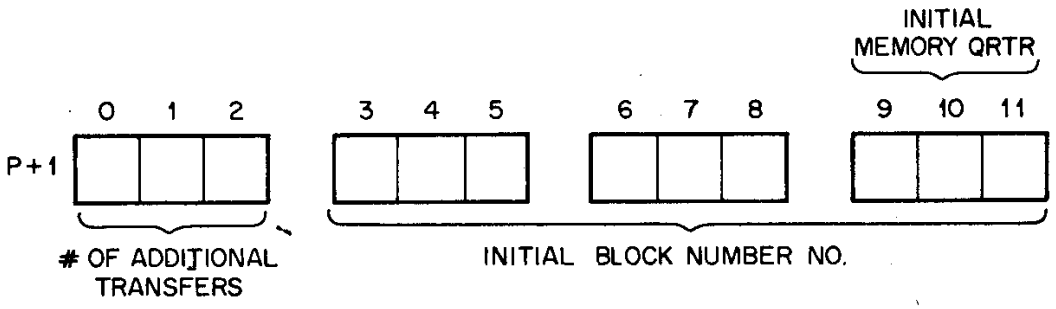
would be read into memory block 7, etc.). The leftmost 3 bits (usually MBLK) are used to specify the number of additional tape blocks to transfer. In the above example tape block 51 is read into memory block 1, and 2 additional tape blocks are transferred: Tape block 52 into memory block 2(and tape block 53 into memory block).

The format of WCG I u (write and check group) is the same as for RCG:

Memory Address	Memory Buffer	
p	WCG I u	705 + 20I + 10u
p+1	3/300	3300

The computer interprets the above example as: write and check MBLK 0 in tape block 300, and make 3 additional consecutive transfers: MBKL 1 into tape block 301, MBKL 2 into tape block 302, and MBKL 3 into tape block 303. When the leftmost 3 bits are 0, i.e., 0 additional transfers, the WCG instruction is like the WRC.instruction in that only 1 block is transferred.

The second word of a group transfer instruction may be diagrammed:



RCG and WCG always operate on consecutive memory blocks and tape blocks. Specifying 3 additional when the initial tape blocks is, say, 336, will transfer information between tape blocks 336, 337, 340, and 341, and memory blocks 6, 7, 0, and 1; that is, MBKL 0 succeeds, MBLK 7. The transfer are always checked; when a transfer does not check, the instruction is repeated starting with the first block. With WCG, all the tape blocks and their checksums are first written and then all are checked. If any block fails to check, the blocks are rewritten starting with the first block and then all blocks are checked again. As with RDC and WRC, group transfer instructions leave − 0 in the accumulator and go to p + 2 for the next LINC instruction.

Using RCG instead of RDC, the program example on page 210 can be written more efficiently:

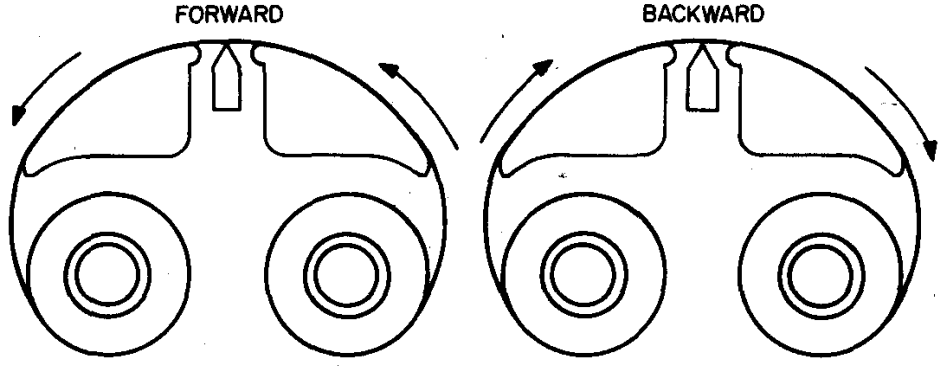
Memory Address	Memory Buffer		Effect
→ 100	WRC I	0724	C(quarter 2) C(block 50); transfer is checked and tape continues to move.
101	2/50	2050	
102	RCG	0701	Read blocks 201-203 into MBLKS 1-3; check the transfers and stop the tape.
103	2/201	2201	
104	JMP 400	6400	Jump to the new section.

Tape and Memory Exchange with Group Transfer

Tape Motion and the Move Toward Block Instruction

When the tape control (TC-12) is searching the tape for a required block, it looks at each block number in turn until it finds the corect one. Since the tape may be positioned anywhere when the search is begun, it must be able to move either forward or backward to find the block.

Forward means moving from the low block numbers to the high numbers; physically the tape moves onto the lefthand reel.

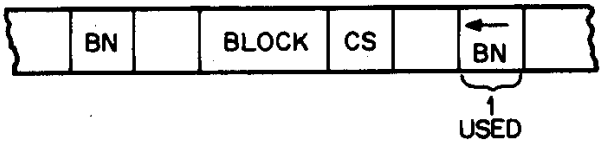


Backwards means moving from the high numbers to the low; the tape moves onto the righthand reel.

When searching for a requested block, the TC-12 decides whether the tape must move forward or backward by subtracting each block number it finds from the requested number, and using the sign of the result to determine the direction of motion. If the difference is positive, the search continues in the forward direction; if negative, it continues in the backward direction. This may, of course, mean that the tape has to reverse direction in order to find the required block.

Suppose, for example, that the TC-12 is instructed to read block 50, and that the tape is presently moving forward just below block 75. The next block number found will be 75. The result of subtracting 75 from 50 is −25, which indicates not only that the tape is 25 blocks away from block 50, but also that block 50 is below the present tape position. The tape will reverse its direction and go backward.

To facilitate searching in the backward direction a special word called a backward block number, BN, follows the checksum for each block.



When searching in the forward direction, the TC-12 looks at forward block numbers, BN; when searching in the backward direction, it looks at backward block numbers, BN. In either direction, each block number found is subtracted in turn from the requested number, and the direction reverse as necessary, until the result of the subtraction is −0 in the forward direction. Transfers and checks are made only in the forward direction.

Thus, in the above example, the tape will continue to move in the backward direction until the result of the subtraction is positive i.e., until the BN for block 47 is found and subtracted from 50, indicating that the tape is now below block 50. At the second block below the direction will be reversed; the computer will find 50 as a forward block number, BN, and the transfer will be made.

For all magnetic tape instructions, if the tape is not moving when the instruction is encountered, the computer starts the tape in the forward direction and waits until it is moving at the required speed before reading a forward block number, BN, and reversing direction if necessary. If the tape is in motion, however, (including coasting to a stop), the TC-12 does not change direction until block number comparison requires it.

For all tape transfer or check instruction with I = 1, the tape continues to move forward after the instruction is executed.

For all magnetic tape instruction stops are made in the backward direction. For transfer or check instructions this means that the tape always reverses before stopping. Furthermore, the tape then stops below the last block involved in the instruction, so that when the tape is restarted, this block will be the first one found. This reduces delay in programs which make repeated references to the same block.

The last magnetic tape instruction illustrates some to the tape motion characteristics. MTB I u (move toward block) is written:

Memory Address		Memory Buffer
p	MTB I u	703 + 20I + 10u
p+1	TBLK	TBLK

As in the other magnetic tape instruction, the u-bit selects the tape unit. The tape motion bit (the I-bit) and the second register, however, are interpreted somewhat differently. MTB directs the TC-12 to subtract the next block number it finds on the tape from the number specified in the second word of the instruction, and leave the result in the PDP-12 accumulator.

MBLK is ignored during execution of MTB. For example, if the block number in the second register of the instruction is 0, and if the tape is just below block 20 and moving forward, then −20, or 7757, will be left in the accumulator. The MTB instruction can thus be used to find out where the tape is at any particular time.

When I = 0, the tape is stopped as usual after the instruction is executed. When I = 1, however, the tape is left moving toward the specified block. The result of the subtraction is left in the accumulator, and the tape direction is reversed if necessary as the computer goes on to the next instruction. MBT I does not actually find the block; it merely orients the tape motion toward it.

The initial direction of motion and possible reversal are determined for MTB just as they are for all other magnetic tape instruction, as described above. Note, however, that since MTB I makes no further corrections to the direction of motion, the specified block may eventually be passed.

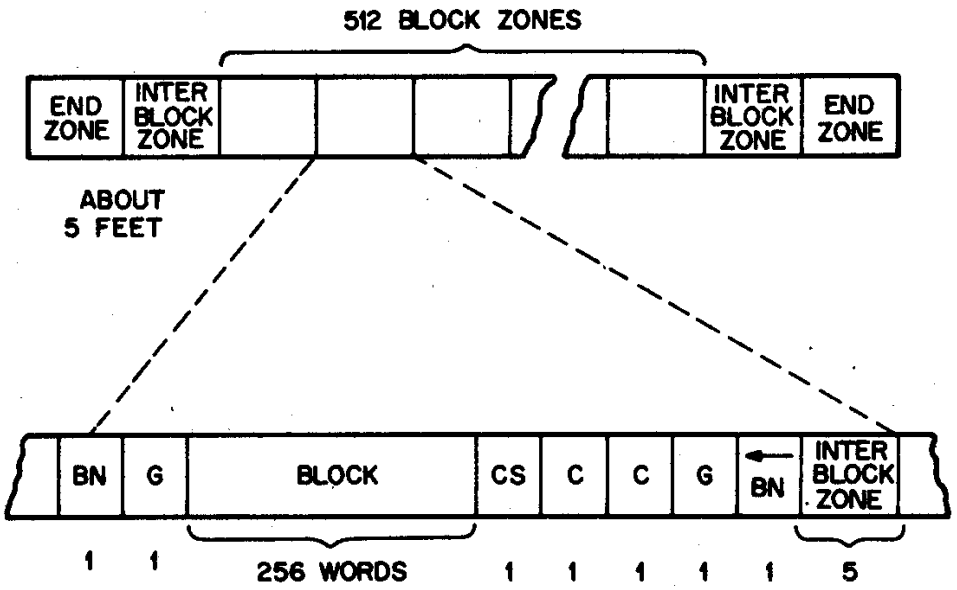
The move-toward-block instruction serves not only to identify tape position, but also to save time. If, for example, a program must read block 700, and then, at some later time, write in block 50, it is efficient to have the tape move toward block 50 in the interim while the program continues to run:

Memory Address	Memory Buffer		Effect
→100	RDC I	0720	C(block 700) → C (MBLK 3); tape moves forward
101	3/700	3700	
102	MTB I	0723	C(103) − next TBLK → C (ACC); tape reverses and moves backward toward block 50.
103	50	0050	Tape continues to move backward while program continues.
.	.	.	
.	.	.	
.	.	.	
300	WR1	0706	C (MBLK 0) → C (block 50); tape stops.
301	50	0050	

In this example it would be inefficient to stop the tape (I=0) with the RDC instruction at location 100 or to let it continue to move forward until block 50 is called for. Although the number left in the accumulator after executing the MTB at location 102 may not be of interest, the MTB does reverse the tape. Then, when block 50 is called for, the delay in finding it will not be so long.

Tape Format

Certain other facts about the tape format should be mentioned. Other special words on the tape are shown:



At each end of the tape is an area called end zone which provides physical protection for the rest of the tape. When a tape which has been left moving as the result of executing a tape instruction with I=1 reaches an end zone, the tape stops automatically. (This prevents the tape from being pulled off the reel). Words marked C and G above do not generally concern the programmer except insofar as they affect tape timing. The TC-12 uses words marked C to insure that the tape writers are turned off following a write instruction. Words marked G, called guard words, protect the forward and backward blocks numbers when the write curent is turned on and off.

Inter-block zones are spaces between the block areas which can be sensed by the skip class instruction, IBZ I, when either tape is moving either forward or backward. The purpose of such sensing is to make programmed block searching more efficient. For example, suppose that somewhere in a program block 500 must be read into MBLK 2; assume it does not matter when as long it is before the program gets to the instructions beginning at location 650. The following illustration uses a sub-routine to check the position of the tape and execute the read instruction if the tape is within 2 blocks of block 500. If the tape is not in an inter-block zone, the main program continues without having to wait for a block number to appear. For purposes of simplicity, assume that the tape (on unit 0) is moving. The program begins at location 400 and the subroutine at location 200.

Note that the following example works only if the tape is stopped by the RDC instruction in register 32. If the tape is not stopped here, subsequent jumps to the subroutine may continue to find the tape at the inter-block zone (locations 20-22) and block 500 may be read repeatedly. The test with the APO instruction at location 646, which signifies if the transfer has been made or not, is necessary to guarantee that the transfer will occur before location 650. At this point, if the transfer has not been made, the JMP 2 at location 647 will be executed.

Memory Address	Memory Buffer		Effect
20	IBZ	0453	Enter subroutine and sense tape position.
21	JMP 0	6000	Return if tape is not an inter-block zone.
22	MTB I	0723	If it is, subtract BN or BN from 500. Tape continues to move forward block 500.
23	500	0500	
24	APO	0451	Is result positive?
25	COM	0017	If negative, complement it.
26	ADA I	1120	Add −2 to see if tape is within 2
27	−2	7775	blocks of block 500.
30	APO I	0471	Is result positive?
31	JMP 0	6000	If result is positive, return to main program.
32	RDC	0700	If negative, tape is within 2 blocks of block 500. Make the transfer and stop the tape.
33	2500	2500	
34	STC 645	4645	Store the transfer check = −0 location 645 to indicate transfer had
35	JMP 0	6000	been made, and return.
→ 400	CLR	0011	Store positive 0 in location 645 to
401	STC 645	4645	indicate transfer has not been made.
402	JMP 20	6020	
500	JMP 20	6020	
600	JMP 20	6020	Jump to subroutine at these points; return to p + 1 and continue with main program.
644	LDA I	1020	Put test number (either 0000 or 7777) into accumulator.
645	(−)	(−)	
646	APO I	0471	Skip to location 650 if the transfer has been made; C (ACC) = 7777).
647	JMP 32	6032	If not, jump to subroutine to make transfer, and return to location 650.
650			

Block Search Subroutine

Tape Motion Timing

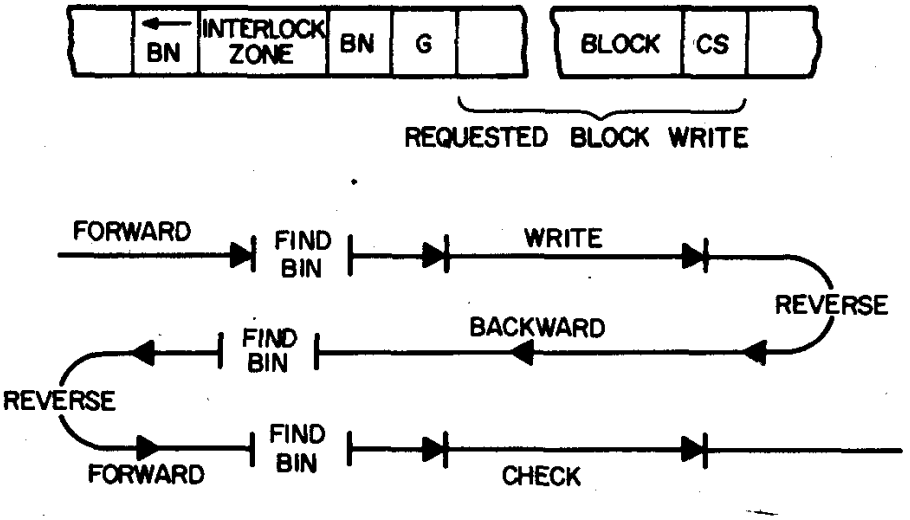
When a tape is moving at a rate of 80 ips, it takes approximately 3.2 msec.

to move from one forward block number to the next, or 120 usec per word. The following table summarizes some of the timing factors:

LINC TAPE MOTION TIME	
Start (from no motion to 80 ips)	0.1 sec
Stop (from 8 ips to no motion)	0.1 sec
Reverse Direction (from 80 ips to 80 ips on new unit)	0.2 sec
Change Unit (from no motion to 80 ips on new unit)	0.1 sec
BN to BN (at 80 ips)	3.2 msec
End Zone to End Zone (at 80 ips)	16 sec

Some methods of using the tape instructions efficiently become obvious from the above table. Generally speaking, tape instruction should be organized around a minimum number of tapes and a minimum amount of tape travel time. When dealing with only one tape unit, it is usually efficient to use consecutive or nearly consecutive blocks in order to reduce the travel time between blocks.

It is also efficient to request lower-numbered blocks before higher-numbered, avoiding unnecessary reversals. The write-and-check instruction, requiring two reversals, is thus costly. It first must find and write in the block in the forward direction; the tape must reverse and go backward until it is below, and then reverse a second time and go forward to find and check the block:



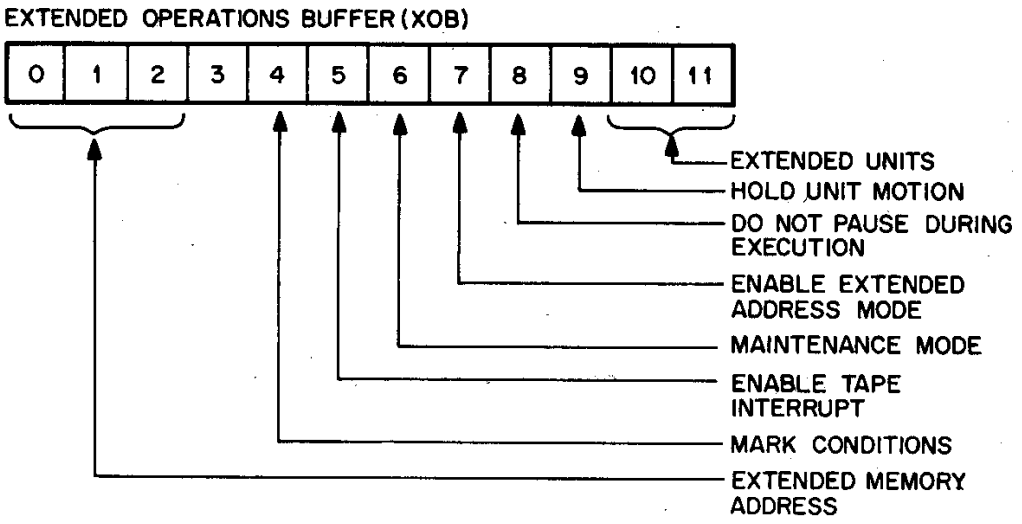
Because of these reversals it is sometimes more efficient to use two tape instructions, WR1 followed by CHK, than to use WRC. This is true, for example, when more than one block must be written and checked. For example, write quarters 1, 2, and 3 in blocks 100, 101, and 102, and check the transfers: using WRC, this would take a minimum of six reversals. The following sequence requires a minimum of two reversals:

Memory Address	Memory Buffer		Effect
→20m	→LDA	1000	Put the BN of the first block to be checked in register 32.
21	24	0024	
22	STC 32	4032	
23	WRI I	0726	Write consecutive blocks can the tape on unit 0 and leave the tape moving forward after each transfer.
24	1/100	1100	
25	WRI I	0726	
26	2/101	2101	
27	WRI I	0726	
30	3/102	3102	
31	→CHK I	0720	Check the blocks, beginning with block 100.
32	(TBLK)	(—)	
33	SAE I	1460	If a block does not check, repeat entire process.
34	7777	7777	
35	JMP 20	6020	
36	LDA I ←	1020	Add 1 to the BN in register 2. If the result #110, not all have been checked. Return and check the next block.
37	1	0001	
40	ADM	1140	
41	32	0032	
42	SAE I	1460	
43	1 103	1103	
44	JMP 31	6031	
45	MTB ←	0703	When all have checked, execute move toward —block to stop the tape, and halt.
46	0	0000	
47	HLT	0000	

In this example the two reversals will occur the first time the CHK instruction at location 31 is executed. Other reversals may be necessary when the computer initially searches for block 100, and when a block does not check, but careful handling of the tape instructions can reduce some of these delays. It should be noted that there are 9 words on the tape between any CS and the next BN in the forward direction. When the tape is moving at speed, it takes 1080 usec to move over these 9 words. Thus the program has time to execute several instructions between consecutive blocks, i.e., therefore the next TBLK appears. In the above example, then, there is no danger that the next block will be passed while the instructions at locations 33-44 are being executed.

EXTENDED TAPE OPERATIONS

The TC-12 (tape control) for the PDP-12 has been developed as subprocessor so that it may act independent of the central processor. To use the TC-12 most efficiently, the tape control can be programmed in its extended mode. The extended operations buffer register controls the extended mode. This register is shown below.



Extended Operations Buffer Bit Assignments

The Extended Operations buffer is controlled by two LINC instructions; XOA and AXO.

XOA — 0021 Extended tape operations buffer to the AC.

AXO-0001 AC to Extended tape operations buffer.

XOA is issued when it is necessary to interrogate the buffer to find out what entended conditions are present. AXO is issued when the extended operations buffer is to be loaded from the accumulator.

No Pause

Setting bit 8 of the extended operations buffer will allow the TC-12 to execute subsequent tape instructions without forcing the central processor into pause (wait for the tape instruction to be completed.) Programming can now continue while a tape operation is in process. The checksum calculated from a tape operation is not transferred to the AC because it would destroy any calculations in progress. The checksum can be retrieved however, by issuing the instruction TAC.

TAC 1 — 003 tape accumulator to AC.

Extended Addressing

The Extended Addressing mode allows the TC-12 to transfer information from tape to memory or vice versa without regards to specific memory blocks. The transfer can be accomplished to or from any section of available memory without changing fields. The TMA (tape memory address) register is used in extended addressing to determine the initial location used in the tape transfer. The TMA register is loaded from the AC with the LINC instruction TMA.

TMA-0023 load the tape memory address with the contents of the AC.

To determine when a tape instruction has been completed the instruction STD may be issued.

STD-0416 is the LINC instruction to skip on a tape done.

The tape done flag is up (set to a 1) whenever a tape instruction is not in progress. STD, therefore, tests for an IDLE condition within the tape control. If used AFTER a tape instruction is issued, STD can indicate a tape completion. The example below shows a tape instruction using no pause.

Memory Address	Instruction	Code	Effect
.	.	.	
.	.	.	
125	LDA I	1020	Set AC bit 8= 1 (No Pause).
126	10	0010	
127	AXO	0001	Set No Pause.
130	RDC	0700	Read tape block #250.
131	3250	3250	Into Memory block #3.
.	.	.	Do not pause to complete tape in-
.	.	.	struction. (Computer program
.			continues)
	.	.	
157	STD	416	Has tape instruction been completed?
160	JMP. −1	6157	No, do not go on until it has.
161	CLR	0011	Yes, Read tape accumulator into AC.
162	TAC	0003	
163	AZE	0450	Is AC = 0?
164	JMP X	6XXX	No, error in the tape operation.
165	APO I	416	Is AC = −0?
166	JMP X	6XXX	No, error in tape operation.
167	.	.	Tape checksum = 7777. Memory
	.	.	Block #3 may now be used.

Bits 0, 1, and 2 of the extended operations buffer are used to determine what 4K field the initial address is in. Only Non-Group instructions can be used with extended addressing. The example below reads tape block 475 into locations 4712 → 5311 of field 7 in a 32K PDP-12.

Memory Address	Instruction	Code	Effect
27	LDA I	1020	Set AC bits 0, 1, 2 to $111_{(2)}$
30	7030	7030	Set AC bits 7, 8 to $11_{(2)}$
31	AXO	0001	Set. Extended Memory Address to $7_{(8)}$ Set Extended Addressing and no pause.
32	LDA I	1020	Load AC with first address in core.
33	4712	4712	
34	TMA	0023	Load TMA with initial address.
35	RDC	0700	Read Tape Block #475 into Locations
36	475	0475	previously specified.
.	.	.	
.	.	.	
175	STD	0416	Has tape command been completed yet?

Memory Address	Instruction	Code	Effect
176	JMP. −1	6175	No, wait.
177			Yes, Data can now be used.
.	.	.	
.	.	.	
.	.	.	

Hold Unit Motion

Setting hold motion allows the maintaining of tape motion in units not currently selected. The procedure for multiple unit movement is to set bit 9 of the extended operations buffer, start a tape unit moving by issuing a tape instruction with the "I" bit set (Keep unit in motion at end of tape instruction) and issue a second tape command to a different unit. The first unit will remain in motion even though it is no longer selected. An example of this procedure is given below:

Memory Address	Instruction	Code	Effect
.	.	.	
27	LDA I	4	Set AC bit 9 to A "1".
30	0004	1020	
31	AXO	0001	Set Hold Unit Motion.
32	MTB I	0723	Move toward tape block #200 (tape unit 0).
33	200	200	Keeping tape in motion (tape unit 1).
34	RDC U	0700	Read tape block #373 (tape unit 1)
35	2/373	2373	into memory block #2 (tape unit 0 is
.	.	.	still moving, tape unit 1 stops at end
.	.	.	of RDC instructions.
36	CLR	0011	Clear accumulator.
37	AXO	0001	Clear hold until motion.
40	RDE	0702	Read tape block #200 into memory.
41	1/200	1200	Block #1 (tape unit 0 stops at end of
42			of RDE instruction).

Extended Units

The TC-12 can control as many as eight TU56 Dual DECtapes transports (each individually selected as tape unit 0 → 7). Within the TC-12 is a 3 bit unit select register. The two most significant bits of this register are controlled by bits 10 and 11 of the Extended Operation buffer. The least significant bit is controlled by the "U" bit of the tape instruction

itself. I. E. to select tape unit 5 for reading a clock of tape the following instructions would be given:

Memory Address	Instruction	Code	Effect
40	LDA I	1020	Set AC bit 10 = 1
41	0002	0002	
42	AXO	0001	Set bit 10 of Extended OP Buffer
43	RDC U	0710	Read Block No. 320 (Tape Unit 5)
	4320	4320	into memory block 4

Maintenance Mode

Setting bit 6 of the extended operations buffer enables the maintenance mode which allows complete control over every operation within the tape subprocessor. (Refer to *Users System Reference Manual* Section)

Mark Condition

The timing and mark track writers can be switched on if bit 4 of the extended operations buffer is set to a "1" while the *Mark Key* is being held down on the front console. This is used only when running a program which will format a blank Linc tape.

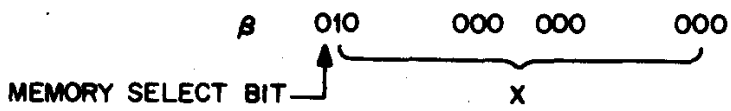
Enable Tape Interrupt

The tape done flag can be switched on to the interrupt line by setting bit 5 of the extend operations buffer. This allows monitoring of the tape status independent of the main program.

LINK DATA FIELD

The LINC has been presented as having a single 12-bit, 1024_{10} word memory. A second addressable memory provides another 1024_{10}, or 2000_8 12-bit words. This second memory is addressable for data storage and retrieval; it can not, however, be used to hold running programs.

Bit 1 of a register containing a memory address, e.i., a β register, is designated as the memory select bit. When this bit is 1, the second memory is addressed:



The addresses for the second memory may then be thought of as 2000 + X, where $0 \leq X \leq 1777$, as usual.

More simply it is referred to as memory registers $2000\text{-}3777_8$. While this scheme makes the memory address of the two memories continuous, they can not always be treated as such by the programmer. The memory address register, using only 10 bits, prohibits using the second memory to hold running programs, the next sequential instruction location after 1777 is always 0. Moreover, the full-address class instructions can address only registers 0-1777.

All other memory reference instructions have available a memory select select bit, and can address either memory. The instruction

P	LDA
p+1	2133

will load the accumulator with the contents of register 2133, i.e., register 133 of the second memory. It must be remembered, however, that all instructions which index the first 16 registers (index class, half-word class, XSK, and DIS) index 10 bits only, and thus indexes from 1777 to 0 without affecting the memory select bit. Therefore, by setting bit 1 the programmer can index through either memory he chooses, but he cannot index from one memory to the other, e.g.,:

Memory Address	Memory Contents	
3	(2000 + X)	(−)
40	SET I 3	0063
41	3777	3777
42	LDA I 3	1023
43	JMP 42	6042

In this example register 3 will contain the succession of values: 3777, 2000, 2001, ... 3777, 2000, etc., repeatedly scanning the second memory. In order for the first execution of the LDA instruction at location 42 to index register 3 to 2000, register 3 must be set initially to 2777, i.e., X(3) = 1777 and memory select bits = 1.

For many purposes this indexing scheme presents no disadvantages. Often, however, one would like to use both memories, for example to collect a large number of data samples. The following program fills memory registers 400-3777 with sample values of the signal on input line 10. The sample-and-stored part of the program is written as a subroutine (locations 31-40), and the sample rate is controlled by an SXL I N instruction:

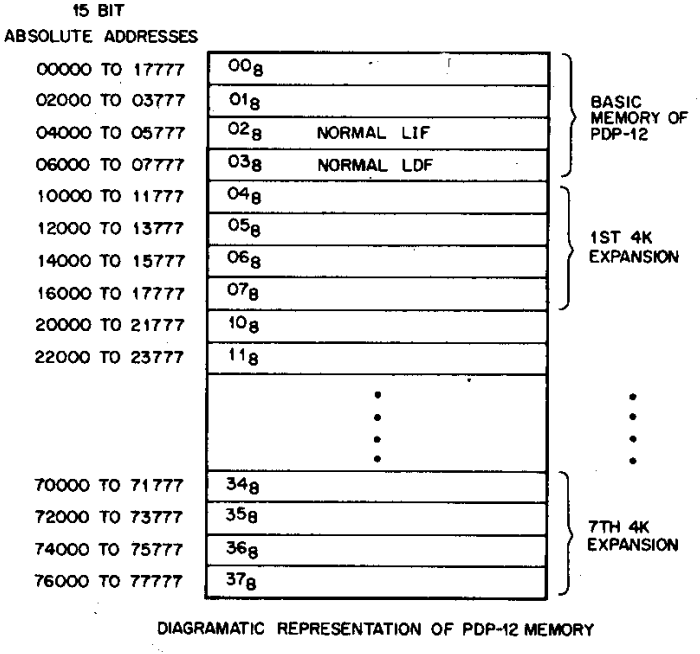
Memory Address	Memory Contents		Effect
7	(−)	(−)	For memory address.
10	(JMP X)	(−)	For return point.
20	SET I 7	0067	
21	377	0377	Set 7 to initial address minus 1 and jump to subroutine.
22	JMP 31	6031	
23	SET I 7	0067	Return from subroutine; set 7 to initial address minus 1 for sec. memory, and jump to subroutine.
24	3777	3777	
25	JMP 31	6031	Return from subroutine; write memory quarters 1 through 7 in blocks 31-37 and halt.

Memory Address	Memory Contents		Effect
26	WCG	0705	
27	6031	6031	
30	HLT	0000	
31	SET 10	0050	Enter subroutine and save return point in register 10.
32	0	0000	
33	SXL I 1	0521	Pause until restart signal appears on external level line 1.
34	JMP. −1		
35	SAM 10	0110	Sample input on line 10 and store.
36	STA I 7	1067	
37	XSK 7	0207	If X(7) = 1777, return to get next sample.
40	JMP 33	6033	
41	JMP 10	6010	When X(7) + 1777, return to main program via register 10.

Indexing Across Memory Boundaries

Changing Memory Fields

In actuality there are more than 2048_{10} words in the PDP-12 computer. The basic PDP-12 contains 4096_{10} words with the capacity of expansion up to $32{,}768_{10}$ words. From the LINC point of view, it is best to envision this 32K as 32 1K segments numbered from 00_8 to 37_8 (we will call these 1K segments LINC Memory Banks in this discusion). Normally, LINC mode programs reside in LINC Instruction Field 2 (instructions) and LINC data field 3 (data — see previous discussion). It is possible, however, to change this if such a condition is found desirable.

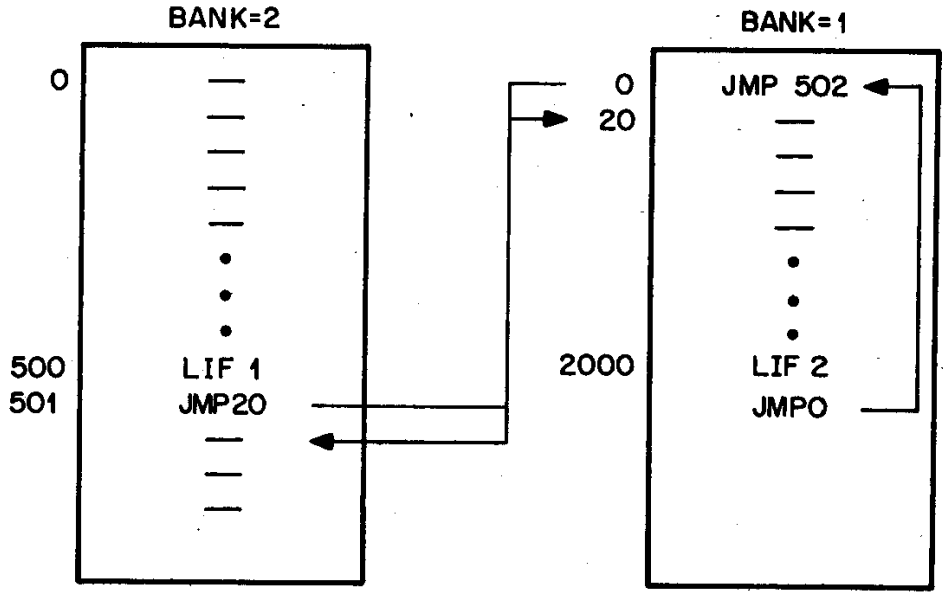


Diagramatic Representation of PDP-12 Memory

An example of such a condition would be a program that requires more than 1024_{10} data words to be in computer memory at one time. A decision is made to store the data in field 1 and field 3. The programming technique discussed will occupy field 2 (where it normally is) and the double memory programming technique discussed previously will be used to access this data. The accessing of the data in field 3 is no problem as field 3 is the "normal LINC Data Field." However, when it is desired to access the data in field 1 the "number" of the LINC Data Field must be changed. This is accomplished by executing the instruction LDF N (640 + N), change LINC Data Field to N. After this instruction is executed, all references to LINC Data Field will be to this memory field (1 in our example) until it is changed by another LDF N.

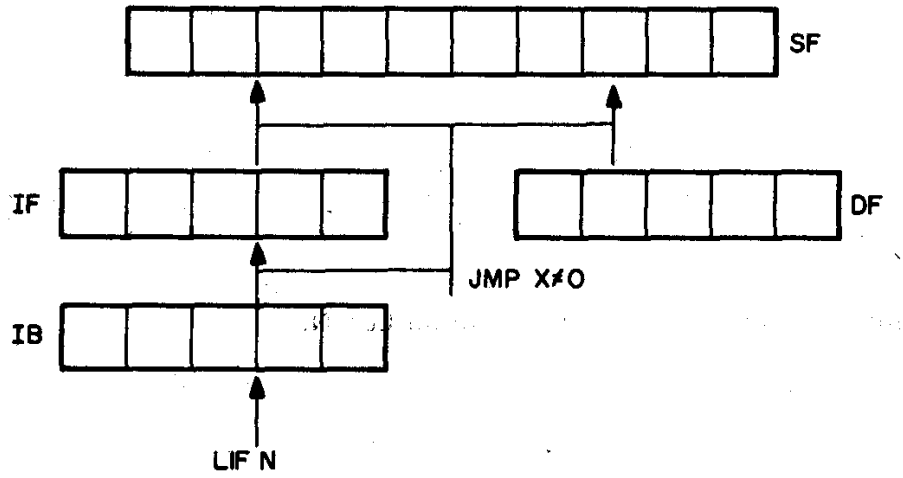
Another example of when it is desirable to change memory fields is if a program were too large to occupy one memory field. It is sometimes desirable to store a frequently used subroutine in a different memory field. Again, let us use memory field 1, this time to hold the subroutine. The main program is in memory field 2; the data, in field 3. To change memory field and transfer the control of a program, the instruction LIF N (600 + N), change LINC Instruction Field to "N" is given. Upon executing the next JMP X (X ≠ 0), control will be transferred to location X of field N (1 on our example) and JMP p + 1 will be stored in location 0 of the new memory bank. To exit field 1 to the original program in field 2, LIF 2 instruction is given, followed by a JMP 0.

THE RESULTS OF LIF/JMP X≠0

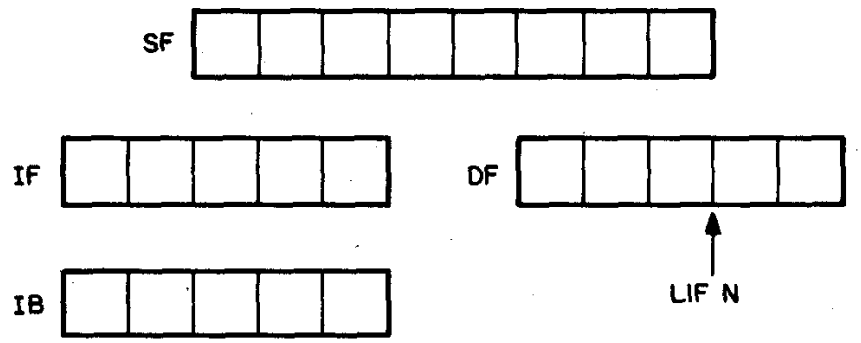


The LINC Data Field bank number is not affected by the LIF instruction (nor is it directly affected by the JMP X).

THIS SHOWS THE ACTION OF LIF/JMP≠0 ON THE EXTENDED MEMORY LOGIC



THIS SHOWS THE ACTION OF LDF N ON THE EXTENDED MEMORY LOGIC



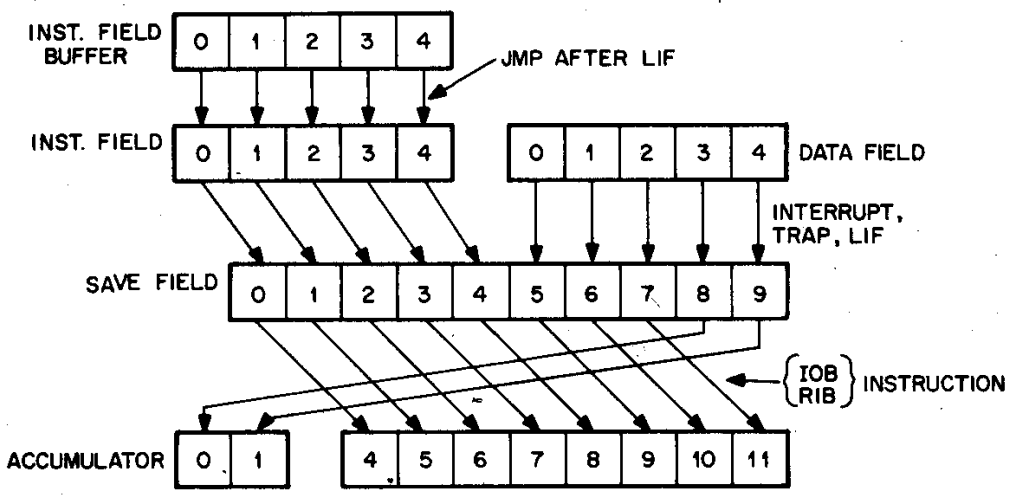
The contents of the Memory Field Registers can be manipulated by using the following IOB/IOT pairs.

IOB
RIF Read Instruction Field

Octal code:	0500 6224
Execution time:	5.9 μsec, including IOB
Operation:	The contents of the Instruction Field Register are placed in bits 6-10 of the AC. The remaining AC bits are unaffected and the contents of the IF are unchanged.

IOB
RDF Read Data Field

Octal code:	0500 6214
Execution time:	5.9 μsec, including IOB
Operation:	The contents of the Data Field Register are placed in bits 6-10 of the AC. The remaining AC bits are unaffected, and the contents of the DF are not changed.



Data Path: IB, IF, DF, and AC.

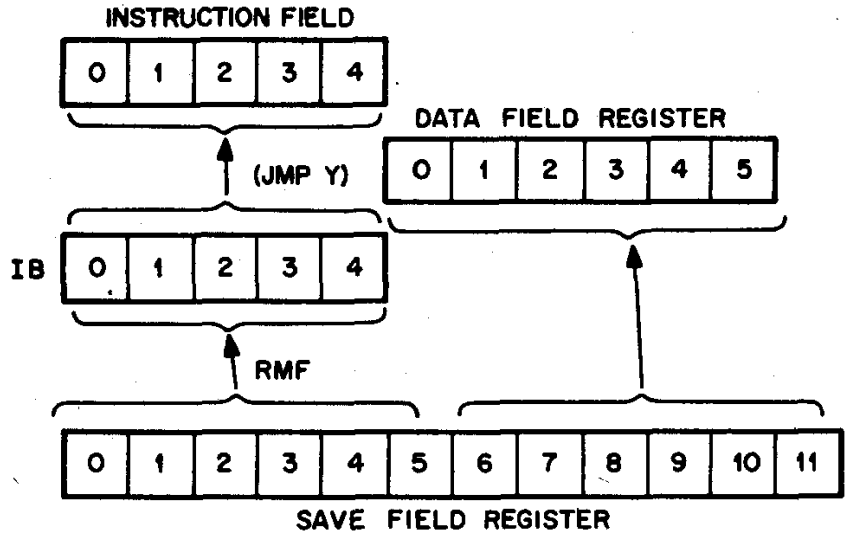
IOB
RMF Restore Memory Fields

Octal code: 0500 6244
Execution time: 5.9 μsec, including IOB
Operation: The contents of bits 5-9 of the SF are placed in the Data Field Register, and the contents of bits 0-4 of the SF are placed in the Instruction Field Buffer. At the next occurrence of a JMP Y instruction (Y ≠ 0000), the contents of the IB are transferred to the IF, effecting a return to the proper field after servicing an interrupt request. The data transfer path is shown in Figure 3-11.

IOB
RIB Read Interrupt Buffer

Octal code: 0500 6234
Execution time: 5.9 μsec, including IOB
Operation: The contents of the Interrupt Buffer (Save Field Register) are OR'ed into bits 0-1 and 4-11 of the AC, as shown in Figure 3-10. Bits 2 and 3 of the AC, and the contents of the SF are unchanged.

RIB is most commonly used immediately after a change of instruction field or a program trap, to save the record of the origin fields while the Program Interrupt is inhibited. (The first JMP instruction executed after a trap or change of instruction field restore the Interrupt; a waiting request would destroy the contents of the Save Field Register.)



Data Path, RMF Instruction

Processor Mode Changes

Switching the PDP-12 from LINC mode to PDP-8 mode to utilize the capabilities of both instruction sets is readily accomplished with a single instruction in each mode.

LINC-6141

The instruction LINC is a PDP-8 command. When issued the central processor immediately switches modes and begins interpreting the following codes as LINC commands.

Memory Address	Instruction	Code	Effect
4126	DCA 50	3050	Save last AC in location 50
4127	LINC	6141	Change to LINC mode
(4300) 300	LDA I	1020	Load the AC with the con-
(4301 301	3417	3417	tents of the next mem. lo- cation.

PDP-0002

The instruction PDP is a LINC COMMAND directing the PDP-12 to switch to PDP-8 mode. The effect is immediate and the following location are interpreted as PDP-8 commands.

Memory Address	Instructions	Code	Effect
(4027) 27	STC 250	4250	Store partial AC in loc. 250
(4030) 30	PDP	0002	Switch to PDP-8 mode oper- ation
4031	TAD 50	1050	Put first no in AC

PDP-8 MODE EXTENDED MEMORY

When additional 4096-word memory banks are attached to the PDP-12,

the Memory Extension Control provides access to the additional storage, both for programs and data. The registers of the Control are already built into the PDP-12; they are described in Chapter 8 p.225 in relation to LINC mode memory control. In PDP-8 mode, the functions of these registers are the same, but only a portion of each register is used. The Instruction Field (IF), Data Field (DF), and Instruction Field Buffer (IB) registers are each three bits long; the two low-order bits of the 5-bit total pertain only to LINC memory fields. The Save Field register (Interrupt Buffer) is only six bits long; in this case, the four high-order bits are unused.

Registers

INSTRUCTION FIELD REGISTER (IF), 3 BITS

These three bits serve as an extension of the PC for determining the 4096-word field from which executable instructions are to be taken. All direct memory references are made to registers in the Instruction Field, with one exception, all JMP and JMS instructions, whether direct or indirect, are registers within the Instruction Field. The exception is the first JMP or JMS executed after a CIF instruction is given. This causes the field to change.

DATA FIELD REGISTER (DF), BITS

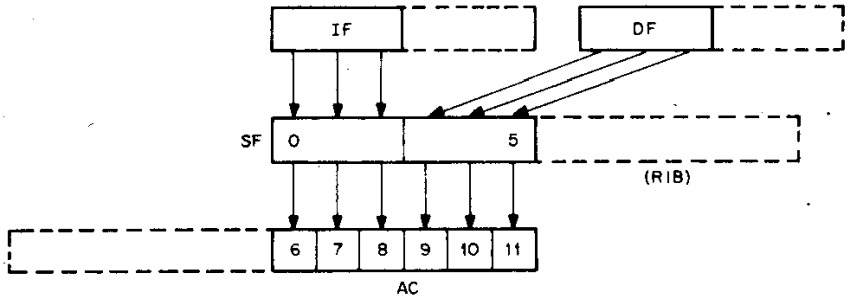
These three bits serve as an extension of the Memory Address register for determining which memory bank contains the operands to be accessed by indirect (only) memory references. The Data Field and Instruction Field may be set to the same bank.

INSTRUCTION FIELD BUFFER (IB), 3 BITS

This serves as an input buffer for the IF. Except for a direct transfer from the console switches, all transfers into the IF must pass through the IB. When a CIF or RMF instruction is executed, information going to the IF is first placed in the IB. At the next occurrence of a JMP or JMS, the contents of the IB are transferred to the Instruction Field register, and programming continues in the new field, starting in the target register of the jump.

SAVE FIELD REGISTER (SF), 6 BITS

Also called the Interrupt Buffer. When a program interrupt occurs the contents of the IF and DF are stored in the Save Field register, as shown in figure below. After the interrupt has been serviced, an RMF instruction will cause the contents of the SF to be restored to the DF and IB. The SF can be examined by using the RIB instruction.



Data Path to SF to AC

BREAK FIELD REGISTER (BF), 3 BITS

When an external device requires extended memory for the transfer of data using the Data Break Facility the contents of the BF specify the memory bank to be accessed. The use of the register is described in detail in Chapter .

Instructions

All Entended Memory IOT instructions 4.3 microseconds for execution.

CDF Change Data Field

Octal code: 62 N1, $0 \leq N \geq 7$
Operation: The quantity N is transferred to the Data Field register. All subsequent indirect memory references by AND, TAD, ISZ and DCA are to the new field.

CIF Change Instruction Field

Octal code: 62 N2, $0 \leq N \geq 7$
Operation: The quantity "N" is transferred to the Instruction Field Buffer. At the occurrence of the next subsequent JMP or JMS instruction, whether direct or indirect, the contents of the IB are transferred to the IF. The effective address of the jump is placed in the PC, and the program continues from that address in the new Instruction Field.

In both CIF and CDF, the number N occupies bits 6-8 of the instruction code.

RDF Read Data Field

Octal code: 6214
Operation: The contents of the Data Field register are placed in bits 6, 7, and 8 of the AC. The other bits of the AC are unaffected.

RIF Read Instruction Field

Octal code: 6224
Operation: The contents of the Data Field register are placed in bits 6, 7, and 8 of the AC. The other bits of the AC are unaffected.

RIB Read Interrupt Buffer

Octal code: 6234
Operation: The contents of the Save Field register (interrupt buffer) are transferred to the AC, as follows: Bits 0-2 (IF) are placed in AC_{6-8}; bits 3-5 (DF) are placed in AC_{9-11}.

RMF Restore Memory Field

Octal code: 6244
Operation: The contents of the Save Field are placed in the Instruction Field Buffer and D^F as follows: Bits 0-2 (original Instruction Field) are transferred to the Bits 3-5 (original

Data Field) are restored to the Data Field register. This instruction is used to restore the Memory Field registers after a program interrupt has been serviced. Normally, the next instruction after the RMF would be JMP I 0; the address of the interrupted program, stored in register 0000 of bank 0, is placed in the PC, and the contents of the IB are placed in the Instruction Field register; the program thus returns to the main program with the Memory Fields restored to their original values.

PDP-12 PROGRAM INTERRUPT

Monitoring I/O peripheral without dedicating the central processor to this task is a necessary part of efficient computer programming.

The PDP-12 program interrupt structure allows continuous monitoring of device flags independent of the main program, table is a partial listing of various device flags connected to the interrupt line.

The interrupt structure is enabled with the PDP-8 I/O command ION.
ION-6001

Enable program interrupt. When the interrupt structure is enabled the setting of any device flag hooked up to the interrupt line signals the interrupt hardware and a specific set of logic operations go into effect.

PDP-8 MODE INTERRUPT

Device	Device Flag	Comment
Teletype	Keyboard Teleprinter	Can be switcher on or off the interrupt line
LINCtape	Tape Done	Can be switched on or off the Interrupt Line
High Speed Reader	Reader Flag	Tied to Interrupt Line
High Speed Punch	Punch Flag	Tied to Interrupt
KW-12	Clock overflow or channel event	Overflow flag and channel flags tied together
DF3 2	Errors and Data Completion Flags	All tied together to interrupts

Table partial listings of I/O connected to Interrupt Line

If the PDP-12 is in the PDP-8 mode when an interrupt occurs, the following logic operations are performed:

1. The central processor enters the *Interrupt Major State* at the end of the current PDP-8 instruction.
2. The *Save Field* Register is loaded with the contents of the *current* instruction field and data field registers.
3. The Instruction Field and Data Field Registers are *set to zero.*
4. The Program Counter is stored in *absolute memory location 0.*

5. The *Next PDP-8 Programmed Instruction* is taken from *Absolute Location 1.*
6. The Interrupt Structure is *Turned off.*

The PDP-8 mode interrupt, as can be seen in the above steps, interrupts the main program and diverts the central processor to a specific section of memory; where a program capable of handling the I/O devices resides.

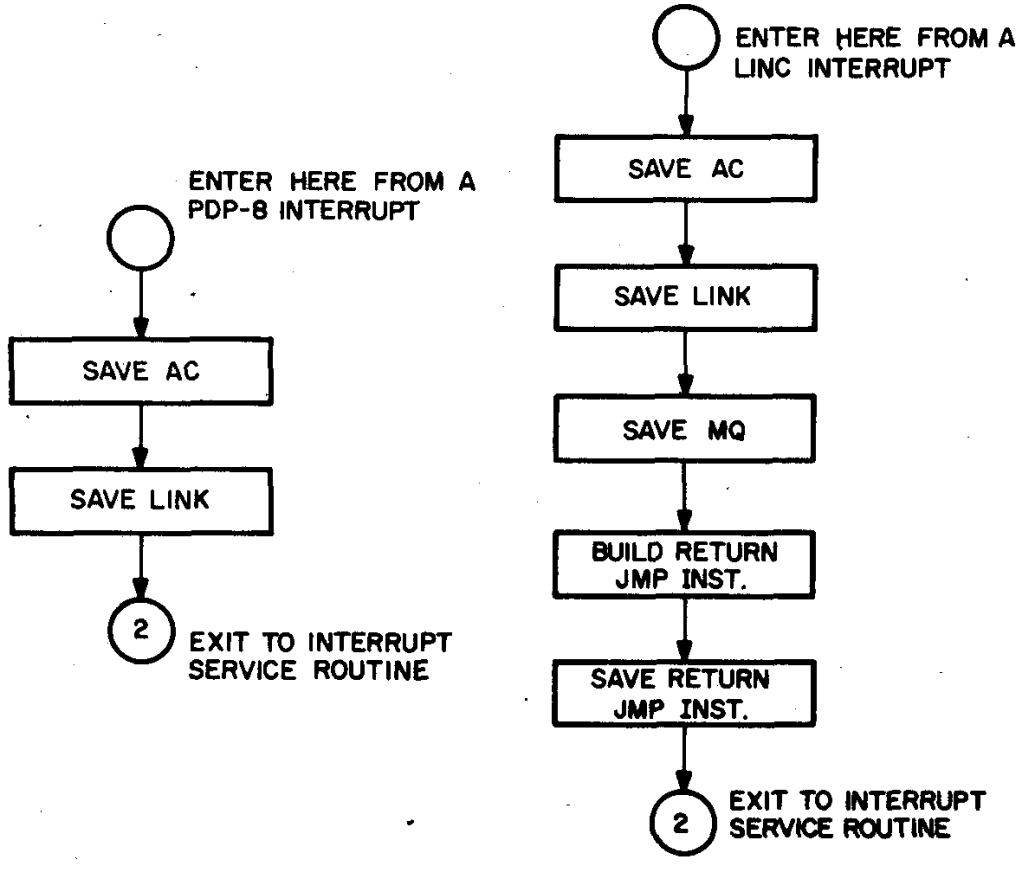
LINC MODE INTERRUPT

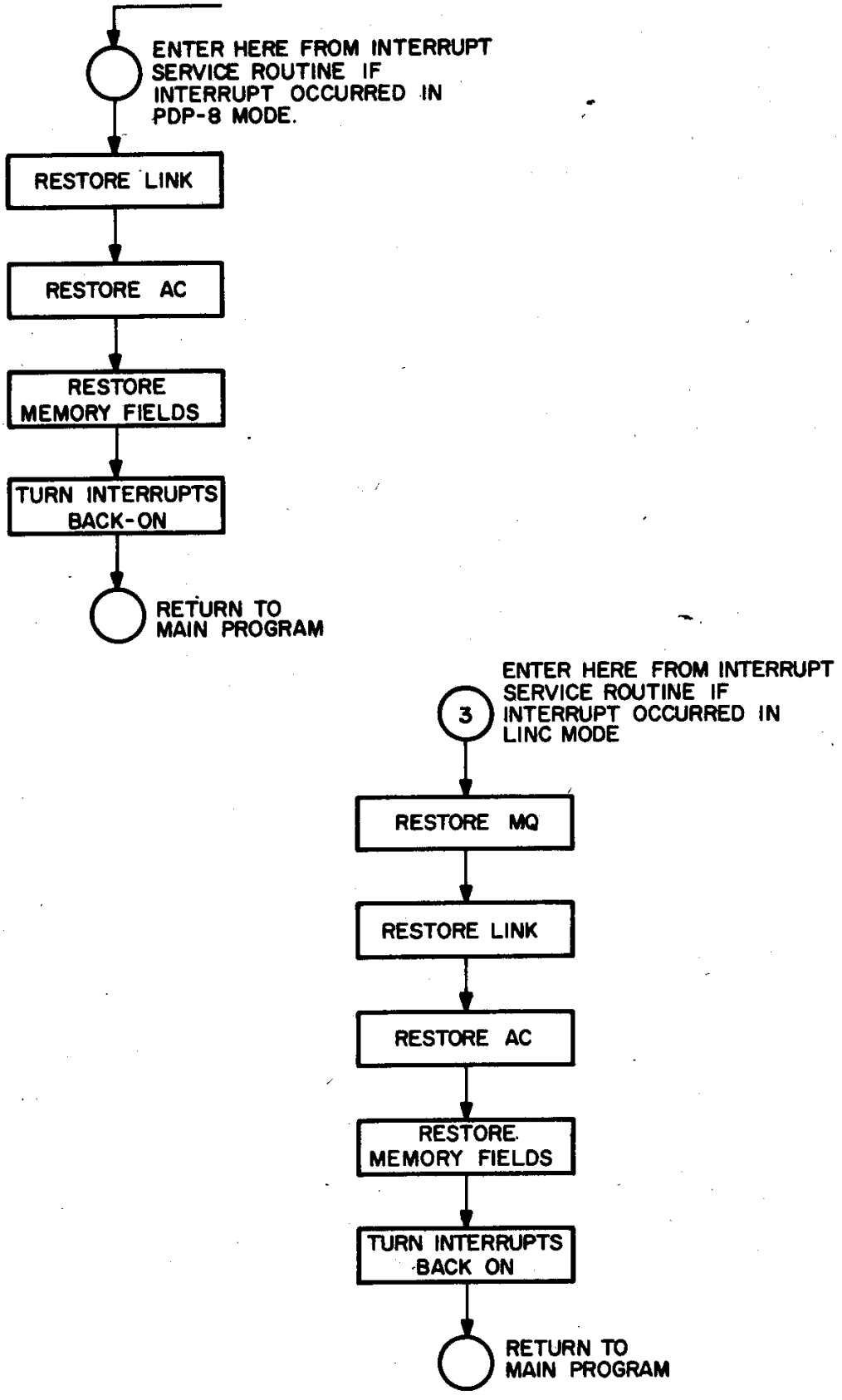
If the PDP-12 is in the LINC Mode, the Interrupt hardware diverts the computer to a different section of memory. The logic operations are as follows:

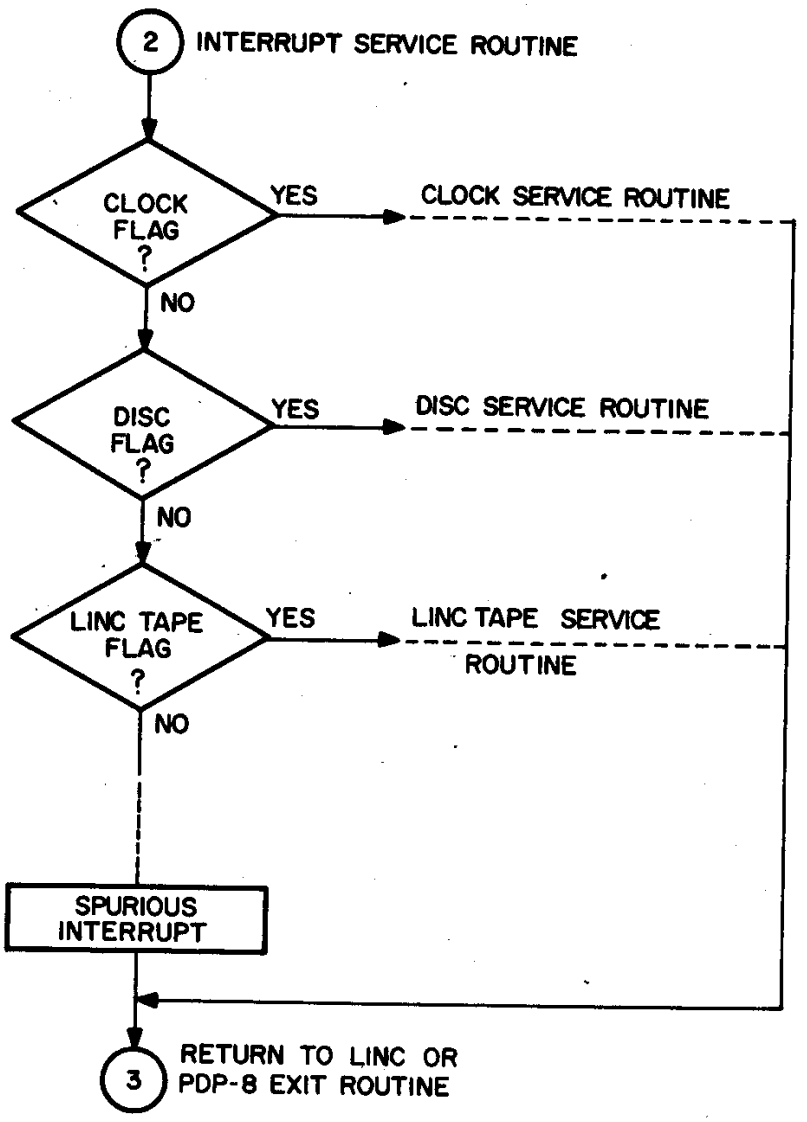
1. The central processor enters the *Interrupt Major State* at the *End* of the current LINC instruction.
2. The Save Field Register is loaded with the current contents of the *Current* instruction field and Data Field Registers/
3. The Instruction Field and Sata Field Registers *are set to zero.*
4. The Program Counter is stored in absolute memory location 40.
5. The Interrupt Structure is *Turned Off.*
6. The Next *LINC instruction* is taken from the *Absolute Location 41.*

Once the interrupt has occurred and control has been transferred to either location 1 (PDP-8 mode) or location 41 (LINC mode), it is up to the program to determine what to do next.

The following example shows a basic outline of an interrupt handler.




ENTER HERE FROM INTERRUPT SERVICE ROUTINE IF INTERRUPT OCCURRED IN PDP-8 MODE.
RESTORE LINK
RESTORE AC
RESTORE MEMORY FIELDS
TURN INTERRUPTS BACK-ON
RETURN TO MAIN PROGRAM
ENTER HERE FROM INTERRUPT SERVICE ROUTINE IF INTERRUPT OCCURRED IN LINC MODE
3
RESTORE MQ
RESTORE LINK
RESTORE AC
RESTORE MEMORY FIELDS
TURN INTERRUPTS BACK ON
RETURN TO MAIN PROGRAM


2
INTERRUPT SERVICE ROUTINE
CLOCK FLAG ?
YES
CLOCK SERVICE ROUTINE
NO
DISC FLAG ?
YES
DISC SERVICE ROUTINE
NO
LINC TAPE FLAG ?
YES
LINC TAPE SERVICE ROUTINE
NO
SPURIOUS INTERRUPT
3
RETURN TO LINC OR PDP-8 EXIT ROUTINE

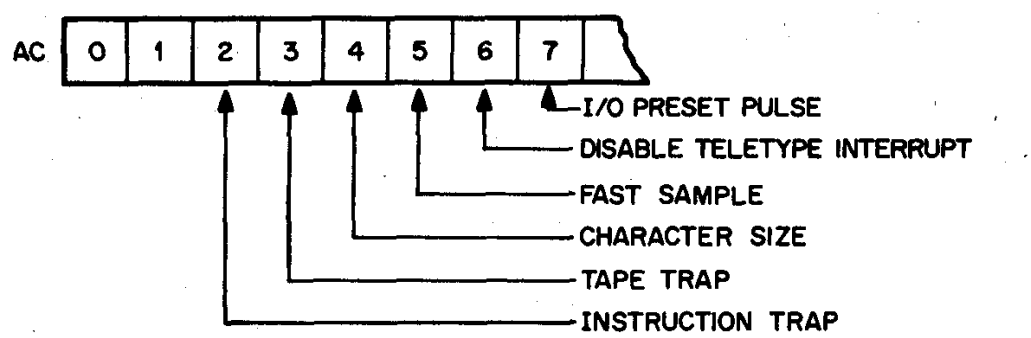
BASIC INTERUPT HANDLER

```
          Pmode
          *Ø
           Ø /PC saved here
          JMP EN8 /Go to PDP-8 Entrance

          * 100
          EN8, DCA ACSAVE  /SAVE AC
               RAR         /LINK to AC bit 11
               DCA LSAVE /SAVE LINK
               JMS INSERV  / Go to interrupt Service Routine
               TAD LSAVE   / Return  here from interrupt service routine
               RAL         /Restore LINK
               TAD ACSAVE  /Restore AC
                RMF        /Restore the memory fields
               ION         /Turn interrupts on again
               JMP I Ø     /Go back to main program
                           /
                           /
                           /
          *40
          Lmode
          Ø       /PC saved here
          STC ACSAVE  /SAVE AC
          ROL I 1     /LINK TO AC
          STC LSAVE   /SAVE LINK
          QAC         /MQ Ø → 10  to AC 1 → 11
          ROL  1      /AC 1 → 11  to ACØ  → 10
          QLZ / MQ11-Ø?
          ADD ONE     /NO, ADD ONE TO AC
          STC MQSAVE  /YES, MQ NOW RESTORED IN AC, SAVE MQ
          LDA         /GET PC
          40          /INTO AC
          BSE I       /BUILD A RETURN "JMP" INSTRUCTION
          6ØØØ
          STC JMRTN   /SAVE RETURN JUMP
          PDP         /CHANGE TO A PDP-8
          pmode
          JMS INSERV  /GO TO INTERRUPT SERVICE ROUTINE
          LINC        /RETURN HERE IF INTERRUPT
          LMODE       /OCCURRED IN LINC MODD AND CHANGE BACK TO LINC
          LDA I       /GET LINK INTO AC TO
LSAVE,     Ø          /RESTORE IT
          ROR I 1     /RESTORE LINK
          LDA  I      /GET MQ INTO AC
MQSAVE,   Ø
          ROR 14      /RESTORE MQ
          LDA I
ACSAVE    Ø           /RESOTE AC
          IOB
          RMF         /RESTORE MEMORY FIELDS
```

```
      IOB
      ION     /TURN INTERRUPTS BACK ON
MRTM,   Ø     /RETURN "JMP" RESIDES HERE TO GO BACK TO THE MAIN PROGRAM


      PMODE
INSERV, Ø   /PC SAVED HERE TO RETURN TO
            / PROPER ECIT SECTION
      CLSK  /KW-12 INTERRUPTS
      SKP   /NO, TRY NEXT FLAG
      JMP CLKSER /YES, SERVICE CLOCK
      DFSE  /DISK ERROR?
      SKP   /NO, GO ON
      JMP ERROR / YES, ERROR CONDITION
      DFSC      /DATA CONPLRTION FLAG ?
      SKP       /NO, TRY NEXT FLAG
      JMP DSCSER /YES, SERVICE DISC
      LINC       / CHANGE TO LINC MODE
      LONODE
      DJR        /INHIBIT STORING PC IN LOC Ø ( will destroy PDP-8 Exit)
      STD        /TAP COMPLETE FLAG?
      SKP        / NO, TRY NEXT FLAG
      JMP TAPSER/ YES, SERVICE TAPE
        .
        .
        .
      JMP ERROR /NO, FLAGS, SPURIOUS INTERRUPT
      JMP I INSERV  / COME BACK HERE FROM DEVICE SERVICE AND GO BACK TO
                      PROPER EXIT ROUTING.
```

SPECIAL FUNCTIONS

A set of six special functions allow the LINC programmer to establish any of five operating states, or generate an I/O Preset pulse. The special functions are determined by bits 2-7 of the AC, as shown in Figure 3-12.



SPECIAL FUNCTIONS

Setting the various conditions in the special functions register is accomplished by the instruction ESF.

ESF — 0004 —

Load the special functions register with the contents of the A.C. (AC remains unchanged.)

The status of the special functions register can be monitored using the instruction SFA.

SFA — 0024

Special Functions Register to A.C. (inclusive OR). The special functions provided are:

1. *Instruction trap enable*
2. *Tape trap enable*
3. Character size
4. Fast sample
5. *Disable teletype interrupt*
   Interrupt requests form the ASR-33 keyboard interrupt or printer are inhibited. No program interrupt will be produced, even if the interrupt facility has been enabled and either keyboard or printer flags have been set,
6. *Generate I/O Preset.*
   If this bit is set when the enabling instruction (ESF) is executed, an I/O preset pulse is generated which clears all device flags, disables the Interrupt, clears the Tape Extended Operations Buffer, and generates the TAPE PRESET pulse. The effect is the same as if the I/O PRESET key on the Operator's Console were passed, except that the active registers of the Central Processor are not affected, and the system continues to operate with RUN on.
   Other Special Functions are cleared, or in case of *Character Sizes*, reset to 1.

Tape Traps

The tape instructions (codes 700 — 737) can be made to trap rather than activate the LINC tape control, by setting "tape trap enable" (bit 3 of the special functions register) along with "instruction trap". With both of these conditions set, tape instructions encountered will not perform LINCtape instructions, instead they will be trapped to location 140 where a "trap handler" program would decide what to do with these instructions.

The tape trap feature could be used, for instance, if a programmer had written a trap handler program for a drum memory system that would make it look like a LINCtape and could be used with LINCtape instructions. No further programming of the drum memory would be required because all a user would have to do to operate with the drum would be to enable "tape trap" and "instruction trap", and issue tape instructions, the trap handler program would take care of the drum memory.

Instruction Trap

When the instruction trap bit (bit 2) of the special functions register is set to a "1" the *OPERATE, EXECUTE,* and *UNDEFINED* instruction will perform a trap if their codes are encountered in a program sequence.

A trap operation is similar to a program interrupt in that it starts a sequence of logical operations that transfers control to another section of memory. The logic operations that are executed are as follows:

1. The central procesor enters the *Interrupt Major State* at the end of the *Current Instruction* (in this case it is the trap instruction itself).
2. The *Save Field Register* is loaded with the contents of the *Current* instruction field and data field registers.

3. The instruction and data field registers are set to zero.
4. The program counter is stored in absolute location 140.
5. The interrupt structure is NOT turned off (if it was on it will stay on). Instead, interrupts are *inhibited* from occurring for the next two JMP instructions.
6. The next instruction in the program sequence is taken from location 141.

It is now up to the programmer to determine what to do once a trap has occurred.

An example of instruction traps is shown below where an instruction to decrement the accumulator is to be simulated:

Memory Address	Instruction	Code	Effect
Start ➤20	CLR	0011	Clear the accumulator
21	LDA I	1020	Set AC bit 2 = 1
22	1000	1000	
23	ESF	0004	Set "instruction trap"
24	LDA I	1050	Load the AC with
25	50	0050	$50_{(8)}$
26	DEC	0510	Decrement the accumulator
.	.	.	(Computer will trap here)
.	.	.	
.	.	.	
.	.	.	
.	.	.	

The trap handler for the above program would look like this:

Memory Address	Instruction	Code	Effect
0140			PC saved here
0141	STC ACSAVE	4XXX	SAVE AC
0142	ROL I 1	0261	LINK to AC
0143	STC LSAVE	4XXX	SAVE LINK
0144	LDA	1000	BRING PC INTO
0145	140	0140	AC
0146	ADA I	1120	SUBTRACT "1" FROM PC
0147	−1	7776	
0150	STC. +3	4154	STORE PC-1 AS INDIRECT REFERENCE
0152	PDP	0002	CHANGE TO PDP-8
0153	TAD I+1	1754	GET CODE THAT PRODUCED TRAP
0154			
0156	LINC	6141	CHANGE BACK TO LINK
0157	SAE I	1460	IS CODE DEFINED?
0160	0510	0510	
0161	JMP 167	6167	NO, GO BACK IMMEDIATELY (NOP)
0162	LDA	1000	YES, SIMULATE DECREMENTING
0163	AC SAVE	XXXX	THE AC BY GETTING AC SAVE
0164	ADA I	1120	AND ADDING "−1" TO IT

Memory Address	Instruction	Code	Effect
0165	−1	7776	
0166	STC ACSAVE	4XXX	SAVE AC BACK IN AC SAVE
0167	LDA	1000	
0170	140	0140	SET UP THE
0171	BSE I	1620	RETURN JUMP
0172	6000	6000	
0173	STC JMRTN	4203	
0174	LDA	1000	
0175	LSAVE	XXXX	RESTORE LINK
0176	ROR I 1	0321	
0177	LDA	1000	
0200	AC SAVE	XXXX	
0201	IOB	0500	RESTORE MEMORY FIELDS
0202	RMF	6244	
0203	JMRTN,	XXXX	GO BACK TO MAIN PROGRAM

TRAPS

There are many codes still unused as instructions in the LINC mode of operation. Rather than let these codes perform "NOP" thereby losing their potential use, these instructions can be "trapped" into a specific section of memory for decoding and software simulation.

The codes that have the ability to be trapped are:

501-515, 521-525 Operate
740 → 777 Execute
540 → 577 Undefined
1700 → 1737 Undefined
700 → 737 LINCtape instructions (special case)

CHAPTER 9
PDP-12 SOFTWARE

LAP6-DIAL

Introduction

LAP6-DIAL (hereafter referred to as DIAL for brevity) provides the PDP-12 user with a keyboard operating system that includes editing, assembling, and file handling capabilities. An interactive CRT display permits quick user response; a file index and peripheral device interchange program facilitate file manipulation.

The DIAL system is provided to the user on LINCtape.[1] These DIAL tapes, distributed by the DEC Program Library, contain two versions of DIAL: DIAL-V2 for 4K tape systems, and DIAL-MS for 8K and larger systems, particularly those using disks. Both versions of DIAL have the same fundamental system design; the main difference is the amount of available storage space and speed of accessing it on disk rather than tape.

LAP6-DIAL is an editor, filing system and assembler for use with the PDP-12 computer. The Editor and filing portion are derived from the basic LINC program LAP6[1] by Mary Allen Wilkes of Washington University. The assembly portion is derived from several programs used for the PDP-8 computer including PAL-D[2].

The Digital Equipment Corporation wishes to express to the author, Mary Allen Wilkes (Clark), and the Computer Research Laboratory of Washington University, St. Louis, Missouri, its appreciation for the development set forth in LAP6 as well as its thanks for permission to use parts of the LAP6 program.

System Concepts

A DIAL tape contains:

1. A reserved area occupied by DIAL.
2. A working area for temporary storage of user files.
3. A file area for permanent storage of user files.

The DIAL area of the tape contains the DIAL Monitor, Editor, Assembler, Utility routines, and a file index. User programs are saved as named files in the file area of the system tape(s) and/or disk(s).

A LINCtape containing DIAL must be designated as the system tape and assigned to tape transport 0. Most DIAL operations may be performed with only one LINCtape containing DIAL, but some procedures in DIAL-V2, such as assembling programs, require another tape on unit 1. Most efficient operation is achieved when both tapes contain DIAL systems. When the RK8 or RF08 disk is available, DIAL-MS operations are carried out using the disk only.

---

[1] A LINCtape contains 512_{10} blocks of 256 12-bit words each.

[2] M. A. Wilkes, *LAP6 Handbook*, Computer Research Laboratory Tech. Rep. No. 2, Washington University, St. Louis, May 1, 1967.

[3] PAL-D Assembler Programmer's Reference Manual DEC-D8-ASAA-D.

When the system is started, it automatically enters the text mode. A source program may then be typed into the source working area, character by character, via the Teletype keyboard in a symbolic language composed of PDP-8 and LINC mode instructions. The source program is displayed on the scope as it is entered and can then be altered with the Editor, stored as a named file, displayed on the scope or printed on the Teletype. About 400 characters can be displayed at a time on the scope. A line number ($1\text{-}777_8$) is automatically assigned by DIAL and appears to the left of each line on the scope to indicate the sequential location of that line in the source program. From 1 to 17_8 lines can be displayed at a time on the scope, as determined by the setting of the knob for A/D channel 7 which maintains an approximately constant number of lines on the scope.

As the program is typed in, it is placed in an input buffer in core. As the input buffer is filled, the text is written out to the source working area on tape or disk.

Every source display has a current line number. By definition, it is the last line number on the display. The current line is noted by an indicator (2 dashes) on the right-hand side of the scope. Each time a carriage return is typed to terminate a source line, the next sequential line number appears on the scope.

The display may be thought of as a window which may be moved anywhere in the working area. Its position is located according to the last line on the display — "the current line". Therefore, to "move the window", the user requests a new current line using either of the following methods:

a. Type →L where L+1 is the number of the line to be the new current line. (The right arrow indicates pressing the LINE FEED key and means press the RETURN key.) The display will now be positioned with line L as the last line displayed.

b. Type ALTMODE and then one of the following keys:

Key	Action
1	reposition the display forward one frame
2	reposition the display forward one line
Q	reposition the display backward one frame
W	reposition the display backward one line

These ALTMODE key combinations must be typed as the first characters on a new line. A frame is defined as the number of lines currently on the display, as set by the A/D knob for channel 7.

The DIAL system is file oriented. A program, either source or binary, is saved as a file in contiguous blocks of tape (disk) in the file area, in blocks 0 through 267 and 470 through 777. Every tape contains a file index in blocks 346 and 347 for the binary and/or source programs on that tape. File names, starting block, and length in blocks are recorded in the index. When a file is entered, the user gives it a name which must be 1 to 8

displayable keyboard characters[2] in length, of which at least one character is non-numeric. A full index can accommodate 63 different names; however, any name in the index can describe both a source and a binary program, thereby doubling the number of possible file entries to 126.

When a file is being saved, the unused file space nearest the index within the file area that is large enough to contain the file being saved is the next area used. Thus, the location of entries on the tape can be controlled by their order of filing. To minimize tape movement, the most frequently used files should be placed nearest to the index.

FEATURES OF DIAL-V2 AND DIAL-MS

In addition to the basic system structure just detailed, the following are the unique features of each version of DIAL.

DIAL-V2

1. Overlapped tape I/O to accept user input during a tape read or write operation. (When DIAL-MS is run on tape, input is not accepted while the tape is moving; using disk this is not noticeable).
2. 4K system.

DIAL-MS

1. Fully integrated tape-disk system with mass storage support for 1 or 2 DF32 disks, 1 to 4 RS08 disks or 1 RK08 disk.
2. Editor commands to clear the binary working area and to merge binary files, thus speeding assembling and debugging by facilitating the use of subroutines.
3. I/O routines to read, write, or move data.
4. Increased Assembler facilities for processing large programs and generating cross reference listings.
5. 8K system.

SYSTEM STARTUP

The following procedure is performed to start DIAL. For DIAL-MS, it is assumed that the tape and disk have been initialized (see below) so that the DIAL-MS system loaded from tape is aware of available disks and will use them appropriately.

1. Mount DIAL tape on tape unit 0.
2. Mount another LINCtape scratch on unit 1.
3. Set scope channel knob to position 1 & 2.
4. Set the swicthes of both tape units to REMOTE and WRITE ENABLE.
5. Set all disk units to WRITE ENABLE. A single DF32 disk must be set to 0; a second DF32, if present, must be set to 1.
6. Set the mode switch to LINC mode and press I/O PRESET.
7. Set the Left Switches to 0701 and the Right Switches to 7300 by pushing down the front part of the switches indicated by

[2]The characters slash, question mark, and comma should not be used. Only spaces in the middle or at the end are considered to be part of the name; leading spaces are ignored.

and pushing down the back part of those indicated by in the following diagram.

Left Switches Right Switches

8. Press the DO key.
9. Press the START 20 key when the tape has stopped spinning.

The version of DIAL on the tape is started and ready for any DIAL operation.

SYSTEM BUILD (for DIAL-MS only)

The following System Build procedure must be executed when DIAL-MS is to be used in order to "build" a DIAL-MS tape. Part of the procedure is running the program GENASYS, provided on the system tape.

1. Start the system by the system startup procedure.
2. Type→LO GENASYS, 0 to load GENASYS to specialize a tape for DIAL-MS on the system configuration.
3. After the message
   TAPE UNIT CONTAINING GENASYS:
   is printed, type 0.
4. GENASYS then asks
   TAPE UNIT FOR DIAL-MS:

   Type an octal digit to indicate the tape unit on which DIAL-MS is to be built. If the reply is not 0, GENASYS returns to DIAL and user input will be accepted. If the reply is 0, another message is printed on the Teletype:

   PRESS CONTINUE TO INITIALIZE DIAL-MS

   Press the CONTinue key. The last message is
   WHEN EDITOR DISPLAY APPEARS, TYPE (LINE FEED) EX (RETURN)
   After the message is printed, GENASYS initializes DIAL-MS. When that is done, the Editor display appears. Type EX and then press CONTinue. DIAL-MS is ready to accept user input.

The following error conditions can occur.

1. If GENASYS is not able to find 8K of core, the following message is printed.
   THIS MACHINE HAS ONLY 4K, DIAL-MS REQUIRES 8K.
   GENASYS returns to DIAL-V2.
2. If GENASYS is not able to find one of the four binary files needed to build DIAL-MS, the following message is printed:
   THIS TAPE DOES NOT CONTAIN BINARY DIAL-MSx NEEDED FOR GENASYS.
   where x is a digit, 1 to 4. Continue at step 4 above.
3. If one of the binary files is the wrong length, the following message is printed:
   LENGTH ERROR IN DIAL-MSx.
   where x is a digit, 1 to 4.

SYSTEM INITIALIZATION

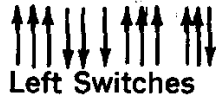
A system using DIAL-MS must be initialized. This procedure causes:

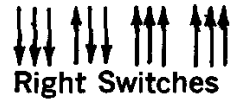
a. the initial tape to modify itself to become a startup tape by building a set of I/O routines for handling the user's particular disk configuration.

b. the initialization routine to copy the DIAL-MS system area (blocks 270 to 345 and 350 to 467) from tape 0 into the appropriate area on the disk.

System Initialization may be automatically carried out by GENASYS (by a reply of 0 in step 4) or may be performed when the equipment configuration changes, the contents of the disk are lost for any reason, or GENASYS does not perform the procedure. The procedure is:

1. Mount the DIAL-MS tape on tape unit 0.
2. Mount another LINCtape on unit 1 (required if configuration has no disk or only one DF32 disk).
3. Set the scope channel knob to the position 1 & 2.
4. Set the switches of both tape units to REMOTE and set unit 0 to WRITE ENABLE.
5. Set all disk units to WRITE ENABLE. A single DF32 disk unit must be set to 0. A second DF32, if present, must be set to 1.
6. Set the mode switch to LINC mode and press the I/O PRESET key.
7. Set the Left Switches to 0701 and the Right Switches to 7310 by pushing down the front part of the switches indicated by ▲ and pushing down the back part of those indicated by ▼ in the following diagram.


Left Switches


Right Switches

8. Press the DO key.
9. When the tape has stopped spinning, press the START 20 key.
10. Type EX) when part of the DIAL-MS program appears on the display in order to preserve the DIAL-MS pointers. Press CONTinue.
11. Set A/D knob 3 all the way to the right. A DIAL command may now be issued.
12. If PIP is to be used, refer to that section of the handbook.

At the completion of this operation, the tape on unit 0 has been modified for the user's particular configuration. It may be copied with the auxiliary "S" option of PIP if multiple copies are needed. The initialization procedure may be repeated at any time, but it is necessary only when the DIAL-MS system on the disk is lost or when the configuration changes.

Using The Editor

DIAL provides a powerful and flexible character Editor which is used in text mode to:

1. add or delete a single character
2. add or delete a text line

3. delete the current line
4. delete an entire portion of the display
5. add text at any location in the program

The Editor is controlled by a cursor that appears on the scope with the text as an inverted T (⊥). The cursor moves in its own alley below a line of text. The location of the cursor is at the current character; all editing operations occur at the current character. The exact location of the cursor is determined by the setting of A/D knob 3. That setting controls how many spaces back from the last character on the scope the cursor is to be placed. At any time, the location of the cursor may be changed by simply turning A/D knob 3. Rotating the knob clockwise moves the cursor to the right; rotating the knob counterclockwise moves the cursor to the left. For normal input of data, A/D knob 3 is initially rotated all the way to the right.

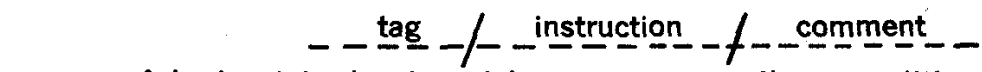
(For a PDP-12-B system without an AD12 and multiplexer, the Right and Left Switches are used instead of A/D knobs 7 and 3 respectively. The setting of Right Switches 8-11 determines the maximum number of lines displayed. The Left Switches can be set to values from 1 to 2047 to determine the position of the cursor. The value of the Left Switches locates the cursor that many characters from the end of the text.)

The Editor can be used to perform the following operation by the indicated sequence.

To:	Do:
delete a single character	Locate cursor under that character. Press RUBOUT.
insert a single character	Locate cursor under the character to precede the additional character. Type the character.
insert a new line	Issue a line call for line L-1 then type line L as the current line.
delete the current line	Type ALTMODE and D. (Location of cursor is immaterial.)
delete the section of the code to the left (right) of the cursor	Locate cursor under the last (first) character of text to be deleted. That character and all those to its left (right) seen on the display are then deleted by typing ALTMODE and L (R).
delete a large section of code	Turn knob 7 all the way to the right to maximize display size. Type a line call so that the last line to be deleted is the current line. Type ALTMODE and L as needed until the first line of the section of code to be deleted ap-

pears on the scope. Type ALT-MODE and D to delete the program through that line.

When a new line is typed in as the current line of displayed text, it is automatically formatted by the Editor. Each text line is considered to be composed of three fields; each field in a line has a number of displayable spaces designated to it by DIAL. Of forty spaces available, the first eight spaces are provided for the tag field, the next sixteen for the instruction field, and the last 16 for the comment field, as follows:



A horizontal tab takes eight scope spaces, thus permitting five tabs per line. When a new line is encountered, the first characters are displayed in the instruction field, unless the first character was a slash. If a comma is then typed, the preceding characters are moved to the tag field and subsequent input is displayed starting at the instruction field. If a slash is encountered as any character but the first one on a line, it is positioned in the comment field along with the characters typed after it and before a carriage RETURN. If a slash is the first character of a line, it is displayed in the tag field. Consider the following user input:

	DISPLAYED IN FIELD		
KEYS TYPED	TAG	INSTRUCTION	COMMENT
/L	/L		
P/L		P	/L
P,/L	P,		/L
P tab /L		P	/L

Assembling

The DIAL Assembler processes a DIAL source program by translating PDP-8 and LINC mode mnemonic operations codes into binary codes for corresponding machine instructions, relating symbols to their numeric values, assigning absolute core addresses for data and instructions and listing the program with error messages.

There are four fields in a DIAL statement; they are identified by the order of appearance in the statement, and by the separating or delimiting, character which follows or precedes the field. Statements are written in the general form:

label, operator operand/comment

A statement must contain at least one of these fields and may contain all four.

1. Labels

   A label is the symbolic name used in the source program to identify the position of the statement in the program. If present, the label is written first in a statement and terminated by a comma.

2. Operators

   An operator may be one of the mnemonic machine instruction codes or pseudo-op codes which direct assembly processing. Operators are terminated with a space if an operand follows, or with a semicolon, slash, or carriage return if no operand follow.

3. Operands

   Operands are usually the symbolic address of the data to be accessed when an instruction is executed, or the input data or arguments of a pseudo-op. In each case, interpretation of operands in a statement depends on the statement operator. Operands are terminated by a carriage return, semicolon, or slash.

4. Comments

   The programmer may add notes to a statement following a slash character. Such comments do not affect assembly processing or program execution, but are useful in the program listing for later analaysis or debugging.

There are two main groups of symbols.

1. Permanent Symbols

   The assembler has in its permanent symbol table definitions of its operation codes, operate commands, and many input/output transfer (IOT) microinstructions. Any symbol in the Assembler's permanent symbols may be used without prior definition by the user.

2. User-defined Symbols

   User-defined symbols, to be used as statement labels, operators, or operands, are composed according to the following rules:

   a. The character must be alphabetic (A-Z) or numeric (0-9).
   b. The first character must be alphabetic. Leading numeric characters are ignored.
   c. Only the first six legal characters of any symbol are meaningful to the Assembler; the remainder, if any, are ignored.
   d. The Assembler assigns values according to the following rules:

		Used After	
		LMODE	PMODE
Defined after	LMODE	10 bits[1]	12 bits
	PMODE	12 bits	12 bits

   e. The maximum number of symbols is 895.

   Symbols are used in three ways.

   1. To label an instruction or data word at any point in the program, the symbol must appear first in the statement and must be immediately followed by a comma.

2. As an operator, the symbol must be predefined by the Assembler or by the programmer.

3. Symbols used as operands should have a value defined by the user. This value may be symbolic references to previously defined labels where the arguments to be used by this instruction are to be found, or may be constants or character strings.

   For an example definition of the syntax, see the LAP6-DIAL Programmers Reference Manual (DEC-12-SE 2D-D).

CHARACTER SET

Keyboard	External (ASCII)	Internal
A	301	1
B	302	2
C	303	3
D	304	4
E	305	5
F	306	6
G	307	7
H	310	10
I	311	11
J	312	12
K	313	13
L	314	14
M	315	15
N	316	16
O	317	17
P	320	20
Q	321	21
R	322	22
S	323	23
T	324	24
U	325	25
V	326	26
W	327	27
X	330	30
Y	331	31
Z	332	32
[(SHIFT/K)	333	33
/ (SHIFT/L)	334	34
] (SHIFT/M)	335	35
↑	336	36
→	337	Illegal (not displayed)
SPACE	240	40
!	241	41
"	242	42
#	243	Illegal (not displayed)
$	244	44
%	245	45
&	246	46

'	247	Illegal (not displayed)
(	250	50
)	251	51
*	252	52
+	253	53
,	254	54
-	255	55
.	256	56
/	257	57
0	260	60
1	261	61
2	262	62
3	263	63
4	264	64
5	265	65
6	266	66
7	267	67
8	270	70
9	271	71
:	272	72
;	273	73
<	274	74
=	275	75
>	276	76
?	277	77
@	300	Illegal (not displayed)
LINE FEED	212	37
RETURN	215	43 (not displayed)
ALTMODE	375	None (not displayed)
RUBOUT	377	None (not displayed)
CONTROL/I (TAB)	211	47 (not displayed)

PERMANENT SYMBOLS

LINC SYMBOLS

Mnemonic	Octal	Operation
		ADD
ADD	2000	add memory to A (full address)
ADA	1100	add memory to A (index)
ADM	1140	add A to memory (sum also in A)
LAM	1200	add link and A to memory (sum also in A)
		MULTIPLY
MUL	1240	signed multiply
		LOAD
LDA	1000	load A, full register
LDH	1300	load A, half register

Mnemonic	Octal	Operation
		STORE
STC	4000	store and clear A (full address)
STA	1040	store A (index class)
STH	1340	store half A
		SHIFT/ROTATE
ROL N	0240	rotate left N places
ROR N	0300	rotate right N places
SCR N	0340	scale right N places
		OPERATE
HLT	0000	halt
NOP	0016	no operation
CLR	0011	clear AC and LINC
SET	0040	set register N to contents of register Y
JMP	6000	jump to register Y
QAC	0005	MQ transfer to AC
		LOGICAL OPERATIONS
BCL	1540	bit clear (any combination of 12-bits)
BSE	1600	bit set (any combination of 12-bits)
BCO	1640	bit complement (any combination of 12-bits)
COM	0017	complement AC
		SKIP
		Skip next instruction if:
SAE	1440	A equals memory register Y
SHD	1400	right half AC unequal to specified half of memory register Y
SNS N	0440+N	SENSE switch N is set
SKP	0056	unconditional skip
AZE	0450	AC equals 0000 or 7777
APO	0451	AC contains positive number
LZE	0452	link bit equals 0
FLO	0454	add overflow is set
QLZ	0455	bit 11 of Z register equals 0
SXL N	0400+N	external level N is preset
KST	0415	keyboard has been struck
SRO	1500	rotate memory register right one place; then if bit 0 of Y equals 0, skip next instruction
XSK	0200	contents of Y equal 1777; index memory register if I bit set
STD	0416	tape instruction completed

Mnemonic	Octal	Operation
	INPUT/OUTPUT	
ATR	0014	AC to relay buffer
RTA	0015	relay buffer to AC
SAM N	0100+N	sample analog channel N
DIS	0140	display point on oscilloscope
DSC	1740	display character on oscilloscope (2 x 6 matrix)
PDP	0002	change to PDP-8 mode
RSW	0516	RIGHT SWITCH register to AC
LSW	0517	LEFT SWITCH register to AC
IOB	0500	I/O bus enable
	MEMORY	
LIF	0600	change instruction field
LDF	0640	change data field
	LINC TAPE	
RDE	0702	read one block into memory
RDC	0700	read and check one block
RCG	0701	read and check N consecutive
WRI	0706	write one block on tape
WRC	0704	write and check one block
WCG	0705	write and check N blocks
CHK	0707	check one block of tape
MTB	0703	move tape toward selected block
XOH	0021	extended tape operations buffer to AC
	EXTENDED OPERATIONS	
ESF	0004	enable special functions
TAC	0003	tape control register to AC
TMA	0023	AC to tape control register
AXO	0001	A to extended operations buffer
DJR	0006	disable Jump Return Save
MSC	0000	miscellaneous
SFA	0024	special functions to AC

PDP-8 SYMBOLS

	MEMORY REFERENCE INSTRUCTIONS	
AND	0000	logical AND
TAD	1000	2s complement add
ISZ	2000	increment & skip if zero
DCA	3000	deposit & clear AC
JMS	4000	jump to subroutine
JMP	5000	jump
	GROUP 1 OPERATE MICROINSTRUCTIONS	
NOP	7000	no operation
IAC	7001	increment AC

Mnemonic	Octal	Operation
RAL	7004	rotate AC & link left one
RTL	7006	rotate AC & link left two
RAR	7010	rotate AC & link right one
RTR	7012	rotate AC & link right two
CML	7020	complement link
CMA	7040	complement AC
CLL	7100	clear link
CLA	7200	clear AC

GROUP 2 OPERATE MICROINSTRUCTIONS

HLT	7402	halts the computer
OSR	7404	inclusive OR switch register with AC
SKP	7410	skip unconditionally
SNL	7420	skip on nonzero link
SZL	7430	skip on zero link
SZA	7440	skip on zero AC
SNA	7450	skip on nonzero AC
SMA	7500	skip on minus AC
SPA	7510	skip on plus AC (zero is positive)

COMBINED OPERATE MICROINSTRUCTIONS

CIA	7041	complement & increment AC
STL	7120	set link to 1
GLK	7204	get link (put link in AC, bit 11)
STA	7240	set AC = −1
LAS	7604	load AC with switch register

IOT MICROINSTRUCTIONS

Program Interrupt

ION	6001	turn interrupt on
IOF	6002	turn interrupt off

Keyboard/Reader

KSF	6031	skip if keyboard/reader flag = 1
KCC	6032	clear AC & keyboard/reader flag
KRS	6034	read keyboard/reader buffer
KRB	6036	clear AC & read keyboard buffer & clear keyboard flag

Teleprinter/Punch

TSF	6041	skip if teleprinter/punch flag = 1
TCF	6042	clear teleprinter/punch flag
TPC	6044	load teleprinter/punch buffer, select & print
TLS	6046	load teleprinter/punch buffer, select & print, and clear teleprinter/punch flag

Mnemonic	Octal	Operation
Clock		
CLSK	6131	skip on clock interrupt
CLLR	6132	load clock control register 1
CLAB	6133	AC to buffer preset register
CLEN	6134	load clock control register
CLSA	6135	clock status to AC
CLBA	6136	buffer preset register to AC
CLCA	6137	counter to AC
Extended Memory (Type MC8/I)		
CDF	62n1	change to data field n
CIF	62n2	change to instruction field n
RDF	62n4	read data field into AC
RIF	6224	read instruction field into AC
RMF	6244	restore memory field
RIB	6234	restore instruction field
Processor Mode Change		
LINC	6141	change to LINC mode processing

OPERATORS AND SPECIAL CHARACTERS

Char	Mode	Operation
'	8/L	Assign symbolic address
*	8/L	Origin — dependent on mode (LINC or PDP-8)
=	8/L	Define parameters
+	8/L	Combine symbols or numbers
−	8/L	Combine symbols or numbers
.	8/L	Has value of current location counter
/	8/L	Comment
U	L	Add 10_8 to instruction
I	L	Add 20_8 to instruction
	8	Add 400_8 to instruction
;	8L	Terminate coding line
SPACE	8/L	IOR
&	8/L	Logical AND
!	8/L	Logical IOR
\	L	Operator $x\backslash y = 1000_8x+\overline{y}$ where x is a single octal digit and y is any expression

PSEUDO-OPERATORS

Pseudo-Op	Mode	Operation
ASMIFM n	8/L	Assemble if n is negative
ASMIFN n	8/L	Assemble if n ≠ 0
ASMIFZ n	8/L	Assemble if n = 0
ASMSKP n	8/L	Continue assembly after n lines
DECIMAL	8/L	Set decimal radix for integer input
EJECT	8/L	Print next line at top of next page of line printer
FIELD n	8/L	Defines each 4K of memory; n = 0 or 1
I	8	Indirect addressing
LIST	8/L	Negate NOLIST condition
LISTAPE n	8/L	Preserve header block if n is negative. List on unit n if n is positive
LMODE	8	Causes subsequent coding to be interpreted as LINC instructions
LODSYM	8/L	Load saved symbol table
NOLIST	8/L	Inhibit octal-symbolic listing
OCTAL	8/L	Set octal radix for integer input
PAGE n	8/L	Start new page at n • 200. If no parameter, start at next page ($0 \leqslant n \leqslant 40_8$)
PMODE	L	Causes subsequent coding to be interpreted as PDP-8 instructions
SAVSYM n	8/L	Save symbol table for later assembly (n = 1 or 2)
SEGMNT n	8/L	Start new segment at n • 2000. If no parameter, start at next segment ($0 \leqslant n \leqslant 7$)
TEXT	8/L	Pack two 6-bit words per cell
Z	8	Page zero reference

MONITOR COMMANDS

The DIAL system programs are requested through DIAL Monitor commands which cause DIAL to enter command mode. To issue a command:

1. Press the line feed key on the Teletype and observe the right arrow in lower left corner of the scope.
2. Type the command in the proper format.
3. Press the return key.

Improperly formatted or misspelled commands are ignored and automatically erased from the scope.

The Monitor commands are listed in the table below and described in detail later in the handbook. Items in parentheses are optional; note that if the file name is omitted, the user's program that was most recently manipulated is used. If no unit is specified, unit 0 is assumed. The unit

number required in DIAL commands can be between 0 and 7 for DIAL-V2 and between 0 and 17 for DIAL-MS, as follows:

DEVICE	ACCEPTABLE UNIT NUMBERS	LOGICAL DISK UNITS
DIAL-V2 and DIAL-MS:		
1 to 8 LINCtapes	0-7	
DIAL-MS only:		
1 RS08 disk	10-11	0-1
2 RS08 disks	10-13	0-3
3 RS08 disks	10-15	0-5
4 RS08 disks	10-17	0-7
1 RK08/disk	10-15	0-5

Each RS08 or RK08 disk is considered to be broken into smaller logical disks, each with its own directory. A logical unit is the equivalent of one LINCtape, 1000_8 blocks; thus, one RS08 disk is said to be made up of two logical disk units. When issuing a DIAL command, the logical disk unit is addressed by an acceptable unit number. Note that DF32 disks cannot be addressed; they are used only to hold DIAL-MS and the source and binary working areas.

All DIAL commands are issued in the form

→ COMMAND

Command	Function
AS (N, U)	Assemble
LI (L, L) (N, U)	Assemble and List
QL (L, L) (N, U)	Assemble and Quick List
LO (N, U)	Load Binary
SB N, U (,M) (FA)	Save Binary
AB (A, [F,]) N, U	Add Binary
SP N, U	Save Program (Source)
AP (L, L) N, U or B, U	Add Program (Source)
PS (L,) (L,) (N, U)	Print Source
DX (,U)	Display Index
PX (,U)	Print Index
CL	Clear Working Area
ZE	Zero Binary Working Area
PI	Peripheral Interchange
EX	Exit
MC X (Y), U	User's Monitor Command

Legend:

( ) indicates an optional parameter
N = File Name
U = Tape (0-7) or Disk (10-17) Unit
L = Line Number
M = Mode (L for LINC or P for PDP-8)
A = Address

F = Field (0 or 1)
B = Tape Block Number
X(Y) = Characters in Accumulator
→ = Line Feed
↘ = Carriage Return

Assemble Program

AS (NAME, UNIT)
NAME = name of filed program to be assembled
UNIT = tape or disk unit on which named file is to be found (0-17)

The Monitor command AS performs an assembly of the NAMEd source file on the specified UNIT. If no NAME is given, the source program in the working area on unit 0 is assembled. With the command AS, an assembly listing is not produced, but error messages with line numbers and a tag table are printed.

List

QL(LINE NUMBER1, LINE NUMBER2,) (NAME, UNIT)

The LIST command performs the same functions as the ASSEMBLE command but also generates an octal-symbolic listing on the Assembler output device (Teletype or line printer).

Quick List

LI(LINE NUMBER1, LINE NUMBER2,) (NAME, UNIT)

The QUICK LIST command performs the same functions as the LIST command, but its listing does not include line numbers and comments and tabs are printed as spaces, thereby greatly decreasing the time required to obtain a listing.

Load Binary

LO (NAME, UNIT)
NAME = name of binary file to be loaded
UNIT = unit from which file is to be loaded

If NAME is not specified, the program is loaded from the binary working area and the loader halts (at location 7774 for DIAL-V2 and at 7775 for DIAL-MS). If NAME is specified, but the designated program is not self-starting, the program is loaded and the loader halts as above. If the NAMEd program is self-starting, the loader will start the program after loading.

Save Binary

SB NAME, UNIT (,MODE) (ADDRESS)
NAME = name to be assigned to saved binary file
UNIT = unit on which binary is to be saved
MODE = L if program is to start in LINC mode
P if program is to start in PDP-8 mode
ADDRESS = starting address (5 digits)

The binary program most recently assembled with an AS, LI, or QL command can be saved with the SAVE BINARY command as file NAME on the specified UNIT. Two tapes are required for this operation when using DIAL-V2.

The SAVE BINARY command has a load and go option so that when a program is loaded into memory with the LOAD BINARY command, it will automatically be started. To use this option, the program mode must be specified. The standard starting addresses, which need not be specified, are 04020 if LINCmode, 00200 if PDP-8 mode. If the program is to be started elsewhere, a full five digit address must be specified. The data field is always set to three when the program is started.

If the SAVE BINARY command is terminated after UNIT with a carriage, the loader will halt after loading the program.

Add Binary (DIAL-MS only)

AB (ADDR, [FIELD,]) NAME, UNIT
ADDR = 4 digit (12-bit) address
FIELD = field number (0 or 1)
NAME = name of a binary file on UNIT
UNIT = unit on which file NAME may be found (0-17)

The binary file NAME is copied to the binary working area, omitting zeros. Typically, ADD BINARY will be used without specifying address and field, to combine standard subroutines with user written main programs. Many subroutines can thus be combined with the user's program, without necessitating reassembling each program. The result may be saved, with SAVE BINARY, as if the whole had been assembled together.

Note that, in general, for ADD BINARY to function properly it must be used in conjunction with the ZERO command. ZERO should always be used before assembling a program whose binary may later be "added", and before adding a binary or binaries to the working area.

Advanced users may want to use the address and field parameters to specify a new core location for the binary field. If address and field are not specified (or both are zero), it will be moved to its assembled address. If address is specified and field is not, it will be moved to address in field 0. Field can not be specified without address. No address adjustment within the assembled code is performed.

If address and field are specified such that any portion of the binary file would be moved to an address about 20000 (i.e., in the field 2 or higher), that portion of the file will be ignored.

A binary may not be moved to field 0, location 0, because this is the condition recognized as no relocation.

No address adjustment is performed by ADD BINARY when a binary is relocated. It is thus necessary that a program which is to be moved include self-relocating code, so that it can determine its location at execution time.

Save Program

SP NAME, UNIT
NAME = name to be assigned to saved program in file
UNIT = unit to contain the NAMEd program

DIAL saves the source program by NAME in one file on the UNIT specified. When saving a program, return may be typed at any time. This will interrupt the command and return to the source display, with no effect since DIAL has not updated the index.

Add Program

AP BN, UNIT
AP (LN1, LN2) NAME UNIT
BN = first block number of source program
NAME = name of filed program
UNIT = unit on which program is located

To add DIAL source to the current source at the current line in the source working area, or to call a previously stored source program into the working area for editing, the ADD PROGRAM command requires specifying only its starting block number, BN, or its NAME. Two line numbers may be specified to add that portion of the NAMEd program to the current source.

Print Source

PS (LINE NUMBER1,) (LINE NUMBER2,)
NAME = name of file to be printed
UNIT = unit on which named file is to located
LNI = starting line number
fLN2 = terminating line number

The NAMEd source program is printed on the Teletype from the specified UNIT. The source currently in the working area is printed when no NAME and/or UNIT are designed. If two line numbers are specified, that portion of the NAMEd file will be printed.

Line numbers, if specifled, provide inclusive bounds for the printout. When only one line number is specified, it is assumed to be the start of the printout and the rest of the source on unit 1 is printed.

Display Index

DX (,UNIT))
UNIT = unit whose index is to be displayed

The tape file index of the specified UNIT is displayed on the scope by the command DX. For each program on the tape, its name, source and/or binary, starting block number, and length in blocks is indicated. To view the entire index, use the following keys to modify the display. A frame is the number of lines defined by the setting of A/D knob 7.

Key	Action
1	Forward one frame
2	Forward one entry
Q	Backward one frame
W	Backward one entry

Print Index

PX (,UNIT))
UNIT = unit whose index is to be printed

The command PX prints out the contents of the specified index on the Teletype. Press RETURN at any time to stop the printout and to return to the source display.

Clear

CL

The source working area on tape unit 0 can be cleared by using the command CL. DIAL remains in core and is restarted with a clean buffer area.

Zero (DIAL-MS only)

The ZERO command fills the entire binary working area with zeros and clears the block map, guaranteeing that any location not used in a subsequent assembly will be zero.

There are three major application of the ZERO command:

a. 0000 is HLT in LINC mode. Therefore, filling the binary working area with zero is equivalent to filling unused core locations with HLT. Thus, a program being tested will halt if it jumps to an unused location.

b. The paper tape output option in PIP, DZ, when combined with the ZERO command, allows the user to assemble and punch short patches to binary programs, with the resulting tape only as long as the patch.

c. The ADD BINARY command depends on the use of the ZERO command in two instances:

   1. Before assembly of a program which will be save and later added.

   2. Before a group of ADD BINARY commands, the ZERO command is required because ADD BINARY does not copy zeros from the file in the working area. This is done to enable the user to make effective patches and overlays easily.

PIP

PI

This Monitor Command is provided to facilitate the loading of the Peripheral Interchange Program from the system unit: it is equivalent to LO PIP,0 for tape systems, to LO PIP,10 for users with a disk unit 10. Note that PIP is not included in the DIAL area and therefore must be loaded onto unit 10 by the user who wants to use this command. To implement the facility, perform the following steps after the system Initialization procedure described previously:

1. Type →LO PIP,0.
2. Respond to the PIP displays with A U LO RO in that order.

Exit

EX

The EXIT command completes the updating of the source working area from the memory buffers, thus assuring the user of leaving DIAL without losing the current source program in the working area. An EXIT command should be issued when concluding an editing session with the PDP-12. After EX DIAL halts. Press the CONTinue console switch to return to DIAL. The program that was in the working area when the EXIT command was issued is still there and any legal DIAL operation can now be performed.

User's Monitor Command

MC X (Y), UNIT

XY = argument(s) to be passed by the Editor to a user program via the AC (argument Y is optional)

UNIT = unit to be read

The USER's MONITOR command allows access to the free blocks of a DIAL tape. When the MC command is issued, block 270 of the UNIT is read into core locations 4000-4377; the arguments XY are placed in the AC and the Editor turns program control over to the initial free block of code at location 4020 in LMODE.

ADCON

(DEC-12-UW2A-D)

Description

ADCON converts the segmented data collected by ADTAPE onto a formatted LINCtape in contiguous blocks by channel number. Any or all of the A/D channels sampled by ADTAPE can be converted. ADCON requires user replies to six scope messages concerning the location of the stored data, the location where the contiguous data is to be placed, and that data channels to be so transferred. All channels stored, starting with the indicated location, can be converted or just specific channels may be moved with ADCON. As each requested channel is converted, its channel number and starting block number and length in blocks on the formatted LINCtape are printed on the Teletype. When one series of data has been converted, a display provides the options of continuing with the program and another set of data or leaving ADCON.

Minimum hardware

PDP-12A

Library Distribution

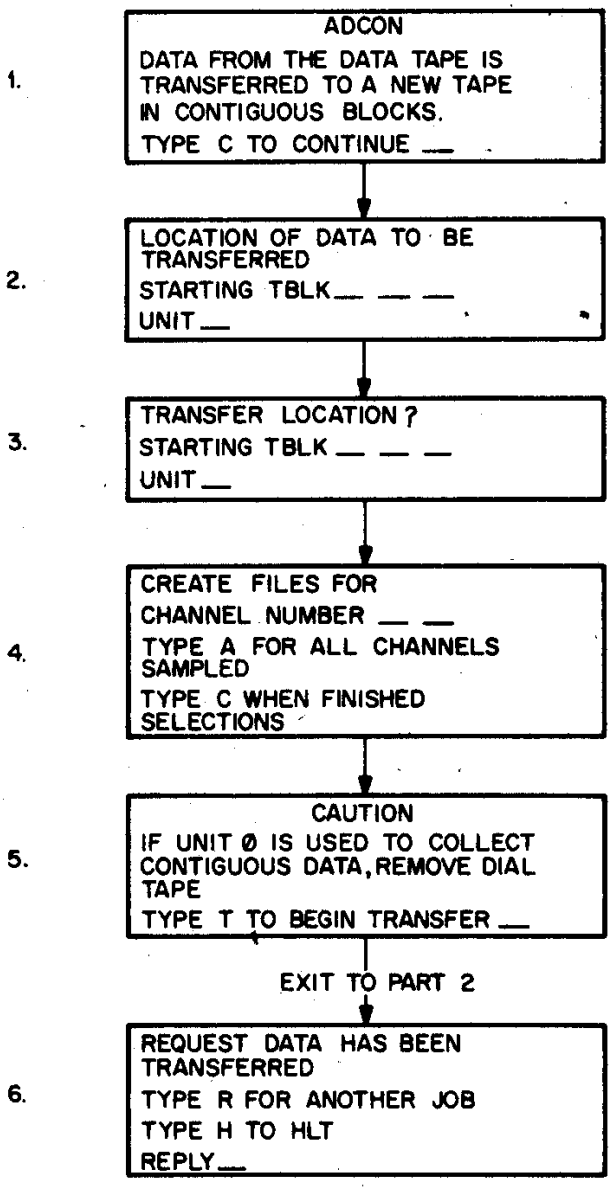
Source file on DIAL tape DEC-12-SE4A-UO, binary file on paper tape DEC-12-UW2A-PB. Described in document DEC-12-UW2AD.

Operating Instructions

Load ADCON by the command

LO ADCON, u
where u is the unit containing the program.

The series of messages is:



Example

Assume ADTAPE had collected 20_8 blocks of data from the 8_{10} channels numbered 3 through 12_8 and the data from channels 3, 5, and 10 are to be converted onto a formatted LINCtape beginning at block 10. The following printout would be on the Teletype when all conversions are completed.

CHAN	STBLK	NB
3	10	2
5	12	2
10	14	2

ADTAPE

(DEC-12-UW2A-D)

Description

ADTAPE consecutively samples and stores data from 1 to 16 A/D channels and can display 1 or 2 channels during sampling. Sampling rates may be as high as 1000 points/second and the time range can be from 1 millisecond/point to 40 seconds/point. Through a series of "question and answer" displays on the scope during initialization, the user specifies all experimental parameters, including channels to be sampled, sync and terminating pulses, sampling rate, and data storage location. The sync pulse can be on a sense switch external level or clock channel; the terminating pulse can be on any of the same devices or after a specified number of points have been collected. In addition, standard experiments can be defined and their parameters called from tape when required, eliminating respecification of the same values before each run.

When initialization is completed, ADTAPE enters visual mode and waits for the sync pulse to start the experiment. Data can be displayed on the scope as it is acquired during visual mode. A simple command causes ADTAPE to enter store mode so that the data will be stored on tape as well as displayed on the scope while it is being sampled. Pause mode can be called to stop temporarily sampling and writing on tape.

The end of the experiment is noted by a message printed on the Teletype. If the end of tape is reached before the terminating pulse has been received, that condition is also indicated by a printed message.

Minimum Hardware

PDP-12A

Library Distribution

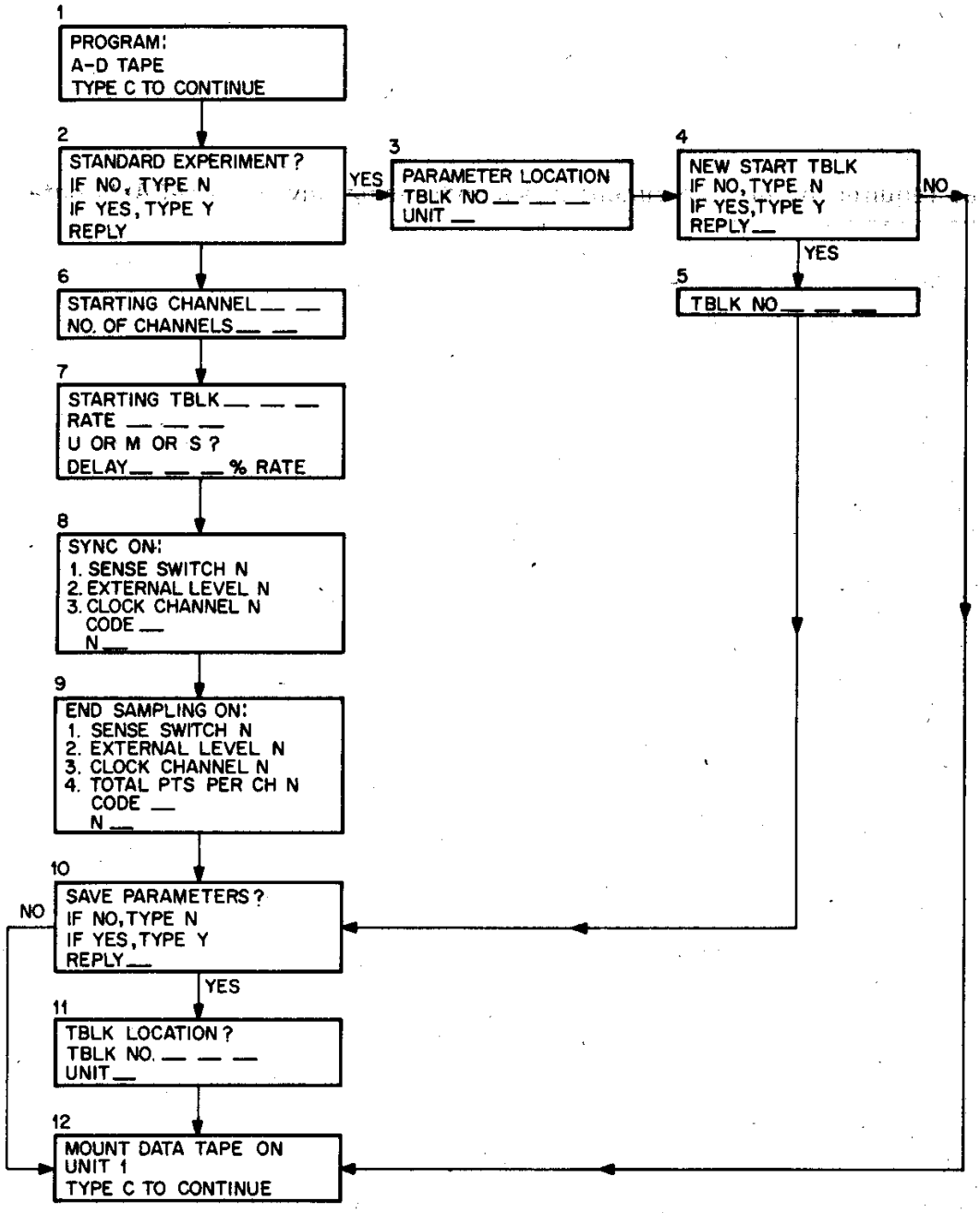
Source and binary files on DIAL tape DEC-12-SE4A-UO, binary file on paper tape DEC-12-UW2A-PB. Described in document DEC-12-UW2A-D.

Operating Instructions

Call the program from tape by a load command, e.g.,

```
→LO ADTAPE, 1)
```

The sequence of messages to specify parameters is as follows.



CATACAL

(DEC-12-UWIA-D)

Description

CATACAL is a box-car averager for data acquisition at rates from 35 seconds/point to 250 microseconds/point. Any analytical instrument or experiment supplying data within those limits can be interpreted easily and quickly using CATACAL's many facilities.

Initially, CATACAL accepts analog data from the interfaced analytical instruments, averages the information, and displays the averaged data on the PDP-12 oscilloscope. The scientist can then interpret the data as required by his experiment in seconds using any of CATACAL's data handling commands. Thus, a sloping baseline can be aligned, a spectrum can be scaled, many integrations can be performed, and two spectra can be compared simultaneously, each operation requiring only a single command.

Minimum Hardware

PDP-12A computer with 8K of core memory and KW12A clock. An X-Y Analog Recorder is recommended for hard copy. The program does not require, but will support, a high speed reader/punch.

Library Distribution

CATACAL is supplied in two versions — CATACAL and CATACALE. The only difference between the two is that the latter version uses EAE (Extended Arithmetic Element) for greater calculating speed. The binary for CATACAL is on DIAL tape DEC-12-SE2D-UD; CATACALE on DEC-12-SE5A-UO. The sources are on DIAL tape DEC-12-SE4A-UO. Described in document DEC-12-UWIA-D)

Operating Instructions

After a numeric value has been typed in response to a command, any character except 0 to 9, E, or . will terminate input for that entry. A space is recommended as the terminator. (Pressing RETURN does not automatically generate a LINE FEED.) In response to questions, only Y or N are acceptable answers. Any other response generates a question mark on the Teletype and is ignored. No terminator is required after a Y or N response.

In all cases, striking RUBOUT before a terminator will delete all input up to the preceding terminator to allow the correct value to be entered. A RUBOUT during decimal input echoes as an exclamation mark and during octal input as a question mark on the Teletype.

If, during scope display, a command unacceptable to CATACAL is typed, a ? is printed on the Teletype and the program returns to the same scope display.

If an operation must be halted immediately, press the console STOP switch. This should not be used haphazardly; if arrays were being modified the data will be lost. Routines requiring input parameters or initial dialogue can be halted during that stage and before the input is com-

plete. To restart CATACAL after an emergency stop, set the MODE switch to 8 mode, set STOP switch to run position, and press I/O PRESET and then the START 400 key. A new CATACAL command can be issued when the display is restarted. If the START 20 key is pressed instead of START 400, the program starts at the beginning.

Command Summary

At least the first two characters of a command must be typed before colon is typed.

INPUT-OUTPUT COMMANDS

TAPE:	Read or write LINCtape
AVERAGE:	Accept time averaged analog data
PAPERTAPE:	Input data from paper tape or keyboard
OUTPUT:	Print/punch paper tape
PLOT:	Plot data on X-Y analog recorder

PROCESSING COMMANDS

CALCULATE:	Calculate Lorentzian and/or Gaussian spectrum
ALTER:	Alter parameters input by previous CALCULATE command
COPY:	Copy CDC into NDC
XINVERT:	Invert X axis
YINVERT:	Invert Y axis
SCALE:	Scale to range of 0-1000
MULTIPLY:	Scale to arbitrary range
SMOOTH:	Apply eleven point digital filter
CURSORS:	Set up two cursors on scope
INTEGRATE:	Integrate between cursors or running integration (preceded by CURSOR command)
STRIP:	Strip out data or baseline (preceded by CURSOR command)
DERIVATIVE:	Form differences (derivatives)
SUBTRACT:	Subtract CDC from NDC; results in CDC
ADD:	Add NDC to CDC; results in CDC
SWAP:	Swap CDC and NDC
SQUEEZE:	Average adjacent points of displayed channels

SPECIAL COMMANDS

MODIFY:	ODT-like core modifier
TIME:	Set machine cycle time constant to calibrate AVERAGER
RESTART:	Restart program
DIAL:	Exit to DIAL Editor

CDC = currently displayed channel
NDC = non-displayed channel

CONVERT

(DEC-12-ESYB-D)

Description

Program CONVERT will perform the major tasks in translating a LAP6 or LAP6-3L source stored on a LINC tape to a suitable source for utilization by the DIAL system on the PDP-12.

Minimum Hardware

PDP-12B

Library Distribution:

Source file on DIAL tape distribution DEC-12-SE2D-UO; binary file on DIAL tape DEC-12-SE3B-UO.

Operating Instructions

1. Call program CONVERT by typing:

   → LO CONVERT, 0 ↵

2. Complete the questionnaire seen on the scope by inserting the starting Block Number of the program to be converted. Press the LINEFEED key upon completion.

3. Upon completion of the program, CONVERT rewinds the LAP6 tape on Unit 1, types out a message warning the user to remove the LAP6 tape from unit 1 and to type:

   → AP 370, 0 ↵

4. The user now has two alternative procedures:

   Choice 1 — if more than one program will be translated, leave the LAP6 tape on unit 1, type:

   → SP name, 0 ↵

   and file the source program ("name" is an eight-character name selected by the user). Then issue a:

   → LO CONVERT, 0 ↵

   to translate another source program.

Choice 2 — Replace the LAP6 tape on unit 1 with a DIAL system tape, and proceed.

WARNING

If a LI or AS command is issued without removing your LAP6 tape, the assembled binary will overlay filed programs on the LAP6 tape.

CREF12

(DEC-12-FRZA-D)

Description

In many situations of assembling, debugging, and modifying programs, a cross-reference listing of user-defined symbols is an invaluable aid to the programmer. CREF12 provides an alphabetical listing of all such symbols in the program with the value of each, the line number at which it was defined, and the line numbers at which references to it are made. Thus, the programmer can easily identify the various places in a program where a tagged location or equated symbol is used. Thoughtful examination of the cross-reference listing greatly simplifies debugging e.g. by indicating conflict in use of a temporary or an unintended recursion loop. Similiarly, code optimization is facilitated because unused locations and subroutines, as well as temporaries that can be used by more than one routine, can be easily noted.

Minimum Hardware

PDP-12A or B with 8K of core memory.

Library Distribution

Binary file on DIAL tape (DEC-12-SE5B—UO). Described in CREF12 document, DEC-12-FRZA-D.

Operating Instructions

To use CREF12, insert a LISTAPE n pseudo-op at the end of the program of interest, where n is an expression whose value is between 0 and 17_8 and is taken as the unit number of a scratch tape (0-7) or disk (10-17) on which the listing will be produced. This information is written on the specified unit starting at block 0. Thus unit n should be either a scratch tape or logical disk devoted to scratch work.

Set all Sense Switches equal to 1 and assemble the program with a LIST command. Then load CREF12 by typing → LO CREF12, 0 . Answer the message UNIT # ? with the unit on which the listing was written. The second message, LISTING [Y or N]?, asks if the full assembly listing is desired with the cross-reference.

Two error conditions can arise when using CREF12'.

BAD INPUT
CORE OVERFLOWED AT LINE NO. xxxx

Example

```
0000                    *20
0001                    /       EXAMPLE OF CROSS-REFERENCE USAGE
0002                    /
0003                            PMODE
0004                    /
0005                    /       SUBROUTINE TO MOVE DATA ACROSS FIELDS.
0006                    /       CALLING SEQUENCE IS:
0007                    /
0010                    /       JMS     MOVE
0011                    /       CDF             /FROM FIELD
0012                    /       FADDR           /FROM ADDRESS
0013                    /       CDF             /TO FIELD
0014                    /       TADDR           /TO ADDRESS
0015                    /       LEN             /NO OF WORDS TO MOVE
0016                    /
0017                    /
0020  4020  0000  MOVE, 0
0021  4021  1620        TAD I   MOVE    /THIS APPEARS IN CREF AS
0022                                    /A REFERENCE TO MOVE, NOT
0023                                    /THE DATA PICKED UP
0024  4022  3240        DCA     MVLOOP
0025  4023  2220        ISZ     MOVE
0026  4024  1620        TAD I   MOVE
0027  4025  3010        DCA     10      /THIS REFERENCE WILL NOT
0030                                    /APPEAR IN THE CREF,
0031                                    /BECAUSE IT IS ABSOLUTE
0032                                    /RATHER THAN SYMBOLIC
0033  4026  2220        ISZ     MOVE
0034  4027  1620        TAD I   MOVE
0035  4030  3242        DCA     MVLOOP+2/THIS APPEARS AS
0036                                    /A REFERENCE TO MVLOOP
0037  4031  2220        ISZ     MOVE
0040  4032  1620        TAD I   MOVE
0041  4033  3011        DCA     11
0042  4034  2220        ISZ     MOVE
0043  4035  7041        CIA
0044  4036  3247        DCA     TEMP
0045  4037  2220        ISZ     MOVE
0046  4040  0000  MVLOOP, 0              /CDF GOES HERE
0047  4041  1410        TAD I   10
0050  4042  0000        0               /CDF GOES HERE
0051  4043  3411        DCA I   11
0052  4044  2247        ISZ     TEMP
0053  4045  5240        JMP     MVLOOP
0054  4046  5620        JMP I   MOVE
0055                    /
0056  4047  0000  TEMP, 0
0057                    /
0060                            LISTAPE 12
```

```
SYMBOL  VALUE   DEF   REFERENCES
MOVE    4020    0020  0021 0025 0026 0033 0034 0037 0040 0042 0045 0054
MVLOOP  4040    0046  0024 0035 0053
TEMP    4047    0056  0044 0052
```

DISPLAY

(DEC-12-FLSA-D)

Description

DISPLAY enables a data display facility for those routines which do not require complex display processing or cannot sacrifice the core for such a display. The routine displays any contiguous section of core via a moving window, with a cursor and octal read out of cursor positions to facilitate operator interaction.

Minimum Hardware

A PDP-12A.

Library Distribution

Source and binary file on DIAL tape, DEC-12-SE5B-UO. Described in document DEV-12-FLSA-D.

Operating Instructions

INITIAL CALL

```
JMS I KIDORA
    FIELD
    CORE LOCATION
    FIELD
    CORE LOCATION
    Y OFFSET
    Y SCALE FACTOR
        .
        .
        .
        .
KIDORA, IDORA
```

REFRESH CALL

```
JMS I KRDORA
    .
    .
    .
KRDORA, RDORA
```

The refresh call displays 1000$_8$ points, sets arguments for the next refresh and returns to the location following the call in PDP-8 mode with the accumulator cleared and the data field unchanged. Note that the initial call to DISPLAY must be to IDORA; RDORA adways refreshes the buffer specified by the last call to IDORA.

OPERATOR INTERACTION

The operator controls the position of the window with knob 0: clockwise motion moves the window to the "right" or towards the end of the buffer; counter-clockwise, to the left. The midpoint reading on knob 0 causes the motion to stop.

Knobs 1 and 5 and Sense Switch (SSW) 5 control the cursor (an intensified dot). Depending on the setting of SSW5, the cursor may either move along the curve or be displayed independently. When SSW 5 = 0, the cursor moves along the curve and its position is controlled with knob 1: When knob 1 is turned to its furthermost clockwise position, the cursor sits upon the rightmost scope point; when knob 1 is positioned to its furthermost counter-clockwise position, the cursor sits on the leftmost scope point; intervening knob positions yield intervening cursor positions. When SSW5 = 1, the cursor is displayed independently of the curve. For this case, knob 1 controls the horizontal coordinate and knob 5 the vertical coordinate. Horizontal displacement of the cursor via knob 1 is identical to that described above. When knob 5 is at its furthermost clockwise position, the cursor is displayed at the top of the scope. When knob 5 is at its furthermost counter-clockwise position, the cursor is displayed at the bottom of the scope.

Associated with the cursor are four octal words displayed in the top left corner of the scope, one beneath the other.* The first two words are the absolute 15-bit core address of the cursor point. The third word is the contents of the displayed core address, i.e., the actual 12-bit value in the data buffer of the data word that corresponds to the cursor point. The fourth word is the scope Y coordinate of the cursor point and is a relative value and depends upon the Y scale factor and Y offset. If the data had been scaled to nine bits prior to display, the fourth word or Y coordinate would range from 0001 to 1000_8, where 0001 corresponds to the bottom of the scope and 1000 corresponds to the top.

To facilitate interaction with the calling program, the four displayed words described above are maintained in page 0 and may be accessed after the refresh return.

TAG	CONTENTS
XCURHI XCURLO	Fifteen bit address of the point in memory references by the cursor
CORVAL	Contents of memory
YCUR	Relative Y display coordinate

FRED

(DEC-12-FZFA-D)

Description

FRED (File Replacement, Entry, and Deletion) is a set of PDP-12 subroutines for manipulation of DIAL indices.

FRED occupies two tape blocks and, when in core, uses four LINC memory blocks (2000_8 words), including space for the index. The routines are segment-independent, but must be loaded at a segment boundary.

Minimum Hardware

PDP-12B

*The character size of the display depends upon the setting of the special functions register at the time the display is refreshed.

Library Distribution

Source and binary are on DIAL tape DEC-12-SE3B-UO. Described in document DEC-12-F2FA-D.

Operating Instructions

In this discussion, all locations are relative to the segment into which FRED is loaded. The user's program must load FRED from a DIAL tape, or assemble it with his program, at any memory address which is a multiple of 2000_8. It may then be reused until it is overlaid.

Entry points for the routines of FRED start at location 20 of the segment into which FRED is loaded, as follows:

```
20 - LOOKUP

22 - ENTER

24 - REPLACE

26 - DELETE

30 - READ

35 - WRITE
```

READ and WRITE are called as follows:

```
LIF X                 /SEGMENT INTO WHICH FRED IS LOADED
LDA I                 /LOAD AC WITH PARAMETER POINTER
RWPARM                /POINTER TO READ/WRITE PARAMETER LIST
JMP 30 (JMP 35)       /DO READ (WRITE)
.
.
.
.
.
.
RWPARM,  Y/UNIT       /HIGH-ORDER THREE BITS FOR FIELD
                      /LOW-ORDER THREE BITS FOR TAPE UNIT
BUFFER                /12-BIT MEMORY ADDRESS OF DATA
BLOCKNO               /BLOCK NUMBER OF FIRST TAPE BLOCK
COUNT                 /NO. OF BLOCKS TO READ/WRITE
```

The COUNT must not be zero.

Return is to the instruction following the JMP. If AC Bit 1 is 0, RWPARM is taken from the caller's instruction segment; if 1, the parameter list is in his data segment. Note: There is no check for attempts by the user to write over DIAL, nor is there a check to prevent reading over FRED.

LOOKUP, ENTER, and DELETE are called as follows:

```
LIF X                  /SEGMENT WITH FRED
LDA I                  /AC:  POINTER TO FILE DESCRIPTOR VECTOR

FDV                    /GO TO LOOKUP (ENTER, DELETE)
JMP 20 (JMP 22, 26)
.
.
.
.
.
FDV, UNIT              /LINC TAPE 0-7

TEXT "NAME????"        /FILE NAME, ENDING WITH 77'S

                       /TO FILL FOUR WORDS (8 CHARS)
TYPE                    /0023 FOR SOURCE, 0002 FOR BINARY

START                   /STARTING BLOCK NO. OF FILE:
                        /FILLED BY LOOKUP, ENTER, REPLACE,
                         OR DELETE
LEN                     /LENGTH OF FILE IN BLOCKS: FILLED IN
                        /BY LOOKUP, CALLER MUST SUPPLY IN
                        /ENTER-REPLACE, UNUSED BY DELETE
```

a) LOOKUP has two returns; the first, immediately following JMP 20, is taken if there is an error in the parameter list, or the named file is not found. The second, two words after JMP 20, is taken if the file is found, indicating that the information in the file descriptor vector is correct.

```
LIF X

LDA I

JMP LOOKUP             /GO FIND THE FILE

JMP NOFIND             /1ST RETURN FILE DOESN'T EXIST
                       /COME HERE WHEN FILE IS FOUND
```

b) ENTER has three returns; the first is taken if there already exists a file of the same name and type. The second is taken on errors in parameter list or insufficient space, either in the file or in the index. The third indicates successful updating of the index.

```
LIF X                  /SEGMENT CONTAINING FRED
LDA I                  /POINTER TO PARAMETER LIST
FDV
JMP ENTER              /GO ENTER FILE IN INDEX

JMP EXISTS             /1ST RETURN - FILE ALREADY EXISTS

JMP NOSPACE            /2ND RETURN - NO SPACE FOR FILE
                       /COME HERE ON SUCCESSFUL COMPLETION
```

c) DELETE has only one return, immediately following the JMP 26.

REPLACE may be called only immediately after a call to ENTER which took the second return. The parameter list need not be explicitly indicated — REPLACE uses that from the preceding ENTER, but the instruction field must be set again.

There are two returns; the first is taken on error in calling sequence or insufficient space. (This can never occur if the new file is smaller than or

equal to the old file). The second indicates successful replacing of the old file entry.

```
LIF X                   /SEGMENT CONTAINING FRED

JMP REPLAC              /ENTER FOUND A FILE OF SAME NAME

JMP NOSPAC              /NO SPACE FOR NEW ONE
                        /COME HERE ON SUCCESSFUL REPLACE
```

If REPLACE is not able to find space for a new file, the old file remains intact.

If the call to REPLACE is not immediately preceded by a call to ENTER which returns indicating the file exists, the machine will halt and FRED must be reloaded.

GENASYS

(SEE LAP6-DIAL MANUAL)

Description

To facilitate distribution, the DIAL LINCtape provided by the Digital Program Library contains DIAL-V2. For users with 8K (or larger) configurations, DIAL-MS is easily generated by running GENASYS just once to create DIAL-MS on the tape. Only two questions are asked, one to locate the tape unit containing the GENASYS files and one to specify the tape unit where DIAL-MS is to be built. The DIAL-MS system is customized for the configuration on which it is created, implementing all interfaced disks (DF32, RS08, and RK08) for faster processing.

GENASYS uses four short files, DIAL-MS1 through DIAL-MS4.

Minimum Hardware

PDP-12B with 8K of core memory.

Library Distribution

Binary file on DIAL tape DEC-12-SE2D-UO. Described in detail in LAP6-DIAL Programmer's Reference Manual, DEC-12-SE2D-D.

Operating Instructions

Load GENASYS by the DIAL command.

```
→LO GENASYS,0↓
```

The following messages are printed and require user answers:

```
TAPE UNIT CONTAINING GENASYS:

TAPE UNIT FOR DIAL-MS

PRESS CONTINUE TO INITIALIZE DIAL-MS

WHEN EDITOR DISPLAY APPEARS, TYPE (LINEFEED) EX (RETURN)
```

The first two messages are best answered by a reply of 0, causing DIAL-MS to be initialized for immediate use.

The following error messages can be printed:

```
THIS MACHINE HAS ONLY 4K, DIAL-MS REQUIRES 8K

THIS TAPE DOES NOT CONTAIN BINARY DIAL-MSX NEEDED
FOR GENASYS  (WHERE X = 1 TO 4)

LENGTH ERROR IN DIAL-MSX
```

L8SIM

(DEC-12-SI1B-D)

Description

The LINC-8 Simulator Trap Processor handles Teletype input and output for LINC-8 and classic LINC programs when they are run on the PDP-12. It must be loaded into the PDP-12 core memory with any LINC-8 or classic LINC program which uses the keyboard, or any classic LINC program which uses Teleprinter, in order for that program to run on the PDP-12.

The trap processor operates by using the PDP-12 Instruction Trap Facility to detect execution of either of the two LINC-8 Teletype input/output instructions by the user's program. It responds to user's execution of a Teletype instruction by executing coding to simulate the instruction's LINC-8 or classic LINC effect. After simulation of the instruction, the trap procesor returns control to the user program.

An important limitation of the trap processor is that it is not interruptible. It may not be operated when the PDP-12 Program Interrupt is enabled.

Minimum Hardware

PDP-12B

Library Distribution

Source file on DIAL tape DEC-12-SE2D-UO. Described in document DEC-12-SI1B-D.

Operating Instructions

Load the program into memory, the computer will halt. Press I/O Preset, and then START 400. The program will turn on the instruction Trap Enable Flip-Flop and halt with the Instruction Field set to 2 and the Data Field set to 3. Verify that the Instruction Trap Enable Flip-Flop is on by observing the console TRAP indicator. This indicator should be lit. If it is not, some kind of error has occurred. The error may be either a machine error or an operator error. Reload the trap processor and try again.

Now read in the user program. If the program is located on some specific block(s) of a LINCtape, mount the tape on either transport and execute an appropriate tape instruction from the console as if the machine were a LINC or a LINC-8. If the user program is a named file on a LAP-3L or GUIDE tape, mount the tape on unit 0, set the LOCAL-OFF-REMOTE switch to REMOTE and press CONT. GUIDE or LAP6-3L* will be loaded, and the user program may be recalled using the usual GUIDE or LAP6 program loading procedure.

If the user program is on paper tape, read it in and start it using the usual paper tape loading and starting procedures, as described in the Binary Loader operating iinstructions, DEC-08-LBAA-D.

Switch the processor mode to the PDP-8 mode by executing the PDP instruction (octal: 0002) before using the Binary Loader. Mode changing through use of I/O Preset in conjunction with the console Mode key should be averded because I/O Preset clears the Instruction Trap Enable Flip-Flop.

To automatically load and start a LINC-8 GUIDE or LAP-3L tape along with the trap processor, load the trap processor from the DIAL tape as directed above, and then press I/O PRESET, START 20, rather than I/O PRESET, START 400. A GUIDE or LAP-3L system will be read in from unit 0 and started.

This procedure duplicates the "START 400" procedure given above, with the exception that the computer does not halt between the trap processor initialization and the loading and starting of the GUIDE or LAP-3L system.

If the Instruction Trap Enable Flip-Flop has been cleared, it may be set again (providing the trap processor has been loaded into core as directed above) by starting at location 400 in memory segment 0 (absolute address 00400). Note that the START 400 key may not be used for this unless the Instruction Field (IF) is set of 0 because START 400 takes the high order 5 bits of the starting address from the IF. Set 0400 into the Left Switches and use START LS, rather than START 400. Use of this entry point sets the Trap Enable Flip-Flop and halts the computer. (Setting of the Trap Enable Flip-Flop may be confirmed by observing the console TRAP indicator.) Pressing continue after the computer has halted causes a transfer to location 400 in memory segment 2 (absolute address 04400), with the Data Field set to 3.

MAGSPY

(DEC-12-UZSA-D)

Description

This program provides a moving window for scanning data stored on a LINCtape. The data is scanned on the PDP-12 scope at a rate determined by the setting of the A/D knob. The data can be interpreted as waveforms or as packed ASCII characters.

Minimum Hardware

PDP-12A

Library Distribution

The source is on DIAL tape DEC-12-SE2D-UO; the binary, on DIAL tape DEC-12-SE3B-UO.

*LINC-8 and LINC users will recall that the GUIDE program starting procedure may be used with either GUIDE or LAP6-3L.

Operating Instructions

The program briefly displays the title then proceeds to the initial option questionaire. The user may choose one of three options (RUBOUT will erase the previously typed character; LINE FEED will execute the desired option).

EXPLANATION SLIDE (Option 1):

Selection of option one displays the various switch settings and waits for the operator to type a LINE FEED or RETURN indicating that he has read the displayed slide.

BLOCK/UNIT QUESTIONNAIRE (Option 2):

Type in the starting tape Block Number (BN), which may be 0 to 777. Leading zeros are not required so that BN 7 may be typed by: 7), 07), or 007). Upon observing the correct BN on the scope, press the RETURN key.

Type in the tape drive number which may be 0 through 7. If the correct entry is observed on the scope, type LINE FEED to begin displaying tape blocks. (This is QANDA — see Document DEC-12-FISA-D for operational details.)

CALL DIAL (Option 3):

This option recalls the DIAL system into core.

SUMMARY OF ACTIONS:

Procedure	Action
SW0 = 1	Display data as waveforms.
SW0 = 0	Display data as packed-ASCII Source.
SW1 = 0	Display data as small text if SW0 = 0.
SW1 = 1	Display data as large text if SW0 = 0.
A/D knob 7	Move toward the beginning of the tape.
A/D knob 7	Move toward the end of the tape.

MARK12

(DEC-12-YITA-D)

Description

To implement the block addressable feature of LINCtape, the tape must first be marked with timing (numbers of words per block) and block numbers. The MARK12 program writes these values onto the tape so that the PDP-12 hardware can read or write on any block. In addition, MARK12 writes a data test pattern onto each block of tape and reads it back to insure that there are no defects in the tape.

Two marking formats are available — 1000_8 blocks of 400_8 words each (used for DIAL) and 2702_8 blocks of 201_8 words each. The actual marking procedure is accomplished by a subroutine so that a unique format can easily be created by the user.

Minimum Hardware

PDP-12

Library Distribution

Source and binary files on DIAL tape DEC-12-SE2D-UO, described in document DEC-12-YITA-D.

Operating Instructions

MARK12 operates through a series of scope messages. After loading the program by issuing the command →LO MARK12, 0 , the following display appears.

```
     MARK12

THIS PROGRAM WILL FORMAT AND CHECK LINC TAPES FOR

THE PDP-12

SELECT OPTION AND PRESS LINE FEED ON THE CONSOLE

TELETYPE:

SELECT

  1            STD. LINC FORMAT

  P            129 WORD FORMAT
```

Reply by typing 1 or P and pressing LINE FEED. The next message is:

```
MOUNT TAPE TO BE

MARKED ON THE RIGHT

REEL OF UNIT 1
PLACE UNIT 1 IN

REMOTE WITH

WRITE ENABLED, THEN

PRESS THE MARK SWITCH
```

After the proper response the tape will be marked. One of the following displays will appear:

```
     GOOD TAPE

ALLOW MARKED TAPE TO REWIND

THEN SELECT OPTION AND TYPE

LINE FEED ON THE TELETYPE

SELECT

1    MARK ANOTHER TAPE

2    RESTART DIAL
```

```
     TAPE CHECK FAILED

SELECT

1    MARK ANOTHER TAPE

2    RESTART DIAL
```

If the "failed" message appears, the tape should not be used.

MILDRED

(DEC-12-FZDA-D)

Description

MILDRED (Multiple Index Lookup, Deletion, Replacement and Entry: Disk) is a set of PDP-12 subroutines for manipulation of DIAL-V2/DIAL-MS indices.

There are four levels of routines, with provision for a routine at any but the lowest level to call any routine of lower level. There is no provision for reentrance or recursion, but the routines are serially reusable (except for REPLACE, as explained later).

Locations 20 to 27 contain DJR, JMP pairs to the entry points of each major routine, so that the coding can be modified without changing calls in external routines.

MILDRED occupies two tape blocks and, when in core, uses four LINC memory blocks (2000_8 words), including space for the index. The routines are segment-independent, but must be loaded at a segment boundary. MILDRED requires that the DIAL-MS I/O routines (blocks 322 and 323 of DIAL) reside in field 1, 7600-7777.

Minimum Hardware

PDP-12B, normally with RS08 or RK8 disk.

Operating Instructions

The user's program must load MILDRED from a DIAL tape, or assemble it with his program, at any memory address which is a multiple of 2000_8. It may then be reused until it overlaid.

Entry points for the routines of MILDRED start at location 20 of the segment into which MILDRED is loaded, as follows:

```
LDF       7
FDC
6/322
RDC
7/323
```

These facilities are exactly analogous to the FRED facilities.

To read or write, the DIAL-MS I/O routines are used.

The DIAL-MS routines occupy blocks 322 and 323 of LINCtape 0 and may be loaded into locations 7000 through 777 of any field. A sequence to load the routines into locations 7000-7777 of field 1 follows.

Each of the three DIAL-MS routines, READ, WRITE, and MOVE, is called in the same manner. A typical call is:

```
CDF N
CIF M
JMS ROUTINE
ARGUMENTS
```

The CDF instruction sets the data field to the present instruction field so the routines know to which field to return. (If the data field is already set to the instruction field, this statement is not needed.) The CIF instruction sets the instruction field to the field of the routines. (If the routines are in the same field as the calling program, this statement may be removed.) The JMS (or JMS I) is the call.

READ and WRITE Routines.

The READ and WRITE routines are called in the same manner, as:

```
JMS READ
POINTER
                    /RETURN IS HERE
```

POINTER points to the following:

```
POINTER, UNIT NO
         CORELOC
         START
         BLOCKS
```

UNIT NO is the logical unit number of the I/O device for the READ or WRITE. CORELOC is the first location of transfer divided by 400; thus 13 refers to location 5400. START is the starting block number of the transfer. BLOCKS is the number of blocks to transfer.

A program to read in eight blocks from block 30 on disk 0 into location 0 is as follows:

```
   *500
   LINC
   LMODE
   LDF       7
   RDC
   6/322
   RDC
   7/323
   CLR
   PDP
   PMODE
   CDF       0
   CIF       10
   JMS I PREAD
   P1
   HLT
P1,10                /UNIT NO.
   0                 /CORE LOC.
   30                /TBLK
   10                /NO. BLOCKS
PREAD, READ
```

MOVE Routine

The MOVE routine transfers core locations from one area of core to another, as:

```
JMS MOVE
CDF FROMF
FROML
CDF TOF
TOL
WORDS
            /RETURN IS HERE
```

FROMF is the "from" field, FROML is the first "from" location, TOF is the "to" field, and TOL is the first "to" location. WORDS is the number of words to be transferred.

The MOVE entry point is at 7200, the READ is at 7774, and WRITE is at 7775.

NMRSIM

(DEC-12-UW5A-D)

Description

NMRSIM(E)* is designed to calculate theoretical spectra of compounds containing nuclei of spin one-half, including hydrogen, fluorine, and carbon-13. Chemical shifts for each nucleus and coupling constants between nuclei are input from the Teletype. Calculated line spectra are displayed on the VR12 scope. Twenty-five calibration points are displayed across the X axis. Output is to LINCtape and high- or low-speed paper tape punch, as well as to the Teletype. NMRSIM has options that allow varying all the parameters as well as offsets for enchancement of resolution. Chemical shifts and coupling constants may be adjusted continuously until the displayed theoretical spectrum is acceptable. Spectra may be read back and displayed from LINCtape or paper tape; several spectra may be merged in this mode of operation, thereby allowing the simulation of large spin systems or mixtures of compounds.

Minimum Hardware

PDP-12A with 8K core memory and KW12A clock.

Library Distribution

The binary for NMRSIM is on DIAL tape DEC-12-SE4A-UO; the binary for NMRSIME is on DEC-12-SE5B-UO. The source is on DIAL tape DEC-12-SE4A-UO. The program is described in document DEC-12-UW5A-D.

Operating Instructions

After the program is loaded, it prints a series of messages on the Teletype to specialize the parameters for the experiment. The user types a reply to each message and terminates the response by pressing the RETURN key. If an illegal response is typed, a ? is printed on the Teletype and the message is repeated.

*NMRSIME utilizes the EAE option for the PDP-12.

The RUBOUT key can be used to erase incorrectly typed characters before a terminator is typed. The messages are listed below with their acceptable responses.

"COMMENTS" — Text of titles, descriptions, etc.
Terminated with CTRL/A

"WANT PAPER TAPE I/O" — Answer Y or N

"NUMBER OF SPINS" — 1 thru 6 accepted

"OFFSET & WIDTH" — define scope display
OFFSET is value of first point and WIDTH is range in Hz

"CHEMICAL SHIFTS" — 1-6 values in Hz

"COUPLING CONSTANTS" — enter values in Hz, in order J_{12}, J_{13}, ... J_{1N}, J_{23} . . . . $J_{n-1,\,n}$

"BLOCK, U" — starting block and LINCtape unit to store data

SPECTRUM MODIFICATION & CONTROL COMMANDS
When displaying a spectrum, the Teletype will accept the following commands.

R-Restart
Calculate complete new spectrum

C — Coupline Constants
Recalculate spectrum with new set of coupling constants

O — Offset & Width
Allows any portion of the spectrum to be displayed

D — DIAL
Exit to LAP6-DIAL operating system

L — List
List energy and intensity of calculated transitions, either all transitions or only those in display.

P — Punch
Punch spectrum on high speed reader/punch

PATCH

(DEC-12-YU2A-D)

Description

PATCH allows the user to modify any location in any tape block on tape unit 1, thereby providing a quick method for making small patches to binary files. PATCH interacts with the user by a series of questions and answers in order to specify the tape block and location to be modified. The present contents of that location are printed and the new value is then typed.

Minimum Hardware

PDP-12B

Library Distribution

Source file on DIAL tape DEC-12-SE2D-UO, binary file on paper tape DEC-12-YU2A-PB. Described in document DEC-12-YU2A-D.

Operating Instructions

The tape to be modified must be on tape unit 1. The tape containing PATCH should be on unit 0 and is loaded by the command:

```
→LO PATCH,0↓
```

PATCH is a load and go program so the first message is displayed immediately after the program is loaded.

Sense Switch 0 is operable: raising it causes a return to DIAL. Therefore, be sure Sense Switch 0 is down before loading PATCH.

PIP

(SEE LAP-6-DIAL MANUAL; DEC-12-SE2D-D.)

Description

PIP, the Peripheral Interchange Program, is a DIAL system program that provides a flexible means for transferring data between peripheral devices, including LINCtape, Teletype and high speed paper tape reader/punch, line printer, card reader, and RS08 and RK08 disks. Symbolic and binary files, as well as absolute data, can be processed via PIP. PIP also has the facilities to make more than one copy at a time of a tape or disk, to copy just the DIAL area of a tape and to rewind and unload tapes. The I/O devices are monitored by PIP during a transfer; if an error occurs, three options are available to the user. The operation can be retried, the error can be bypassed (e.g., a bad area on tape), or the error can be ignored.

A brief series of scope mesages permit the user to specify the exact data transfer operation which is then performed immediately after answering the last display.

Minimum Hardware

None

Library Distribution

Binary file on DIAL tape DEC-12-SE2D-UO. Described in DIAL manual, DEC-12-SE2D-D.

Operating Instructions

The first scope message is:

PIP OPTIONS

```
A---AUXILIARY MODE
B---BINARY MODE
S---SOURCE MODE
```

Type the appropriate abbreviation and press RETURN.

If source or binary input was chosen, the second display is:

```
     INPUT DEVICE
C---CARD READER
H---HIGH SPEED READER
L---LINC TAPE
R---RSO8,RKO8 DISK
T---TELETYPE
```

Acceptable responses are:

Device	Response	Meaning
card reader	C	read columns 1 — 110.
	Caa; THRU, bb	read columns aa-bb
high speed reader tape must be terminated by CTRL/Z)	H	read tape and don't start program
	H; P (or L)	read tape and start program at loc. 200 in PDP-8 mode (or loc. 4020 in LINC mode)
	H1; P addr	read tape and start program in field 1 in PDP-8 mode at location "addr"
LINCtape	Ln; name	read file "name" from tape unit n
disk	Rn; name	read file "name" from logical disk unit n
Teletype	same as high speed reader, but use T instead of H in response.	

The output device message is displayed next.

```
   OUTPUT DEVICE

H---HIGHSPEED PUNCH
L---LINCTAPE
P---LINC PRINTER
R---RSO8,RKO8 DISK
T---TELETYPE
```

Acceptable responses are:

Device	Response	Meaning
high speed punch	H	Punch the file on the high speed punch
	H; DZ	Punch the file on the high speed punch, but don't punch zeros

LINCtape	Ln; name	Write file "name" on tape unit n
line printer	P	Print file on printer
disk	Rn; name	Write file "name" on logical disk n
Teletype	T	Punch the file on the low speed punch
	T; DZ	Punch the file on the low speed punch, but don't punch zeros

If auxiliary mode was chosen in the first display, the second message is:

```
       OPTIONS

C---COPY SPECIFIED BLOCKS
D---DUPLICATE TAPE 0 ONTO 1
S---COPY SYSTEM
U---COPY UNIT
```

Reply as follows:

Option	Response	Meaning
duplicate tape	D	Copy tape on unit 0 onto tape on unit 1
	Dn	Copy tape on unit 0 on tape units 1 through n

Options C, S, and U display the source and binary mode input and output device messages and must be answered as follows:

	input file	
copy blocks	Dn; fb, nb Ln; fb, nb	Copy nb blocks starting at block fb from tape (L) or disk (D) unit n
	output file	
	Dn; fb Ln; fb	Place file on disk (D) or tape (L) unit n starting at block fb
	input file	
copy system	Dn Ln	Copy blocks 300-345 and 350-370 from disk (D) or tape (L) unit n
	output file	
	Dn Ln	Write them on disk (D) or tape (L) unit n
	input file	

copy unit	Dn Ln	Copy tape on disk (D) or tape (L) unit n
	output file	
	Dn Ln	Write tape on disk (D) or tape (L) unit n

PRTC12-F
(DEC-12-YIYA-D)

Description
The PRTC12-F program operates the TC12-F tape option which is prewired in the PDP-12 computer and allows the user to read and write in the forward direction DECtapes that have been formatted on the PDP-8, PDP-9, PDP-10, or PDP-15 computers. The tape used on the PDP-12 is in LINC format and differs from other DECtapes in the following ways:

a. Tape direction over the tape head is reversed.
b. The polarity of the tape heads is reversed.
c. Channels one and three are reversed.
d. Data transfer has a different bit configuration. The following table is a 12-bit comparison of the two systems.

LINCTAPE FORMAT	0	1	2	3	4	5	6	7	8	9	10	11
DECTAPE FORMAT	2	5	8	11	1	4	7	10	0	3	6	9

e. The "mark track" on LINCtape is 4-bit oriented and on DECtape is 8-bit oriented. The TC12-F hardware has a special window register, but only the "block mark" (BM) is decoded. Data flags, bit shuffling, and the computation and verification of the checksum are all done with software.

Minimum Harware
PDP-12B with PRTC12-F option

Library Distribution
The source is on DIAL tape DEC-12-SE2D-40; the binary, on DIAL tape DEC-12-SE3B-UO. Described in document DEC-12-YIYA-D.

Operating Instructions
After it is loaded, the program displays an introduction followed by three sets of questions for the user to define the operation.

The first display is an introduction to the program. Press the LINE FFED or RETURN key to display the second message.

The READ questionnaire is displayed next.

```
READ          BLOCKS
TAPE FORMAT       UNIT
STARTING WITH BLOCK
FORMAT A -- PDP-8 201 WORDS/BLOCK
FORMAT B -- PDP-12 400 WORDS/BLOCK
FORMAT C -- OTHER (PDP-9, 10, 15, WITH 600 12-BIT
   WORDS/BLOCK)
```

Type in each value followed by a carriage RETURN and then press LINE FEED to advance to the next display.

The WRITE questionnaire must be answered.

```
WRITE THE RESULT
IN TAPE FORMAT      ON UNIT
STARTING AT BLOCK
FORMAT A -- PDP-8 201 WORDS/BLOCK
FORMAT B -- PDP-12 400 WORDS/BLOCK
FORMAT C -- OTHER (PDP-9, 10, 15, WITH 600 12-BIT
   WORDS/BLOCK
```

Again, type the correct values, each followed by a carriage RETURN. Press LINE FEED when completed to display the last message.

Respond to the PARITY questionnaire:

```
CHECK PARITY-
0 SPECIFIES NO
1 SPECIFIES YES
```

Type 0 or 1 and press LINE FEED. The requested operation is performed. (Be sure sufficient tape has been wound on th take-up reels before pressing line feed.)

QANDA

(DEC-12-FISA-D)

Description

QANDA is a PDP-12 subroutine written in LINC mode which allows a user to display textual information on the scope, ask questions of the viewer, allow editing of the input, and receive responses thereto.

Minimum Hardware

PDP-12

Library Distribution

The source and binary are on DIAL tape DEC-12-SE3B-UO. Described in document DEC-12-FISA-D.

Operating Instructions

The subroutine is called by the following format:

```
.       JMP QAINIT
.+1     TXTSTR      /POINTER TO TEXT STRING
                    (HALF WORD ADDRESS)
.+2     ANSWER      /POINTER TO ANSWER BUFFER
                    (HALF WORD ADDRESS)
.+3     REFRESH return
.+4     DONE return
```

The calling sequence must be in LINC mode.

. A JMP to QAINIT will initialize the subroutine and fill the answer buffer with the underline character ( ). The subroutine must be initialized at least once. QAINIT is located at the relative address 0 with respect to the beginning of the routine.

.+1 Points to the first character of the textual information to be displayed on the scope.

Characters in the text string which have special meaning are:

Character	Code	Meaning
RETURN	43	End of a line of display. Place next character on next line.
<	74	Interpret the decimal number immediately following < as the number of characters in the question field (range 1-9).
\	34	End of text string.
F	06	Treated as a special character only if it appears at the beginning of a line. If present, the entire line will be displayed in full-size character format. F will not appear on the scope.
H	10	Treated as a special character only if it appears at the beginning of a line. If present, the entire line will be displayed in half-size character format. H will not appear on the scope. If neither F nor H is present at the beginning of a line, half-size is assumed, and the first character of the line will appear on the scope. Intermixing of half and full-size characters between lines is legal.

The 6-bit character string must conform to the character set accepted by DIAL.

.+2

Points to the first character of the answer buffer. The answer buffer need not be set up in any special way; it must be in length at least the number

of half words equal to the sum of the number of characters in each question field plus one for each question field plus one.

Upon entry, QANDA will initialize the answer buffer. All characters in each answer field are initialized to the underline character (00). Code 74 precedes each answer field. Code 34 is the terminator. These codes are placed in the answer buffer by QANDA upon initialization.

Characters, as they are received by QANDA from the Teletype, replace code 00 from left to right in each answer field.

Conditions will always be such that the presence of a null value (00) in an answer field will guarantee that all remaining characters in that field will be set to 00. Note that an all-null field is possible.

If the responses to questions are dealt with as received, the same answer buffer may be used with various text strings in sundry calls to QANDA, since it will be initialized by QANDA upon each initialization entry; the area reserved must, of course, be of sufficient length to accommodate the requirement among the text strings.

.+3

QANDA will refresh the scope once and then will return to this address, provided a LINE FEED has not been typed. This return is provided so that the calling program may periodically check external conditions: e.g., a sense switch may be checked or the program may display a message while awaiting completion of a tape instruction which it may check following each refresh. Examining a partial answer buffer, however, is not recommended, because the answer buffer can be edited at any time by the typist.

To maintain the display on the scope, QANDA has another entry point, QARFSH, which will not re-initialize the answer buffer. QANDA must be entered at this point each time it is to be refreshed. It is located at the relative address QAINIT + 53.

QANDA will always return to .+3 or .+4 following the instruction JMP QAINIT, regardless of the address of the instruction JMP QARFSH. A common situation is to place the instruction JMP QARFSH at .+3 following JMP QAINIT.

.+4

QANDA will return to this location only if either

1. LINE FEED is struck, or
2. RETURN is struck and no question fields exists.

A return to this address signals that the typist has completed his input.

INPUT

Input is received from the Teletype keyboard. Legal keyboard characters

are converted to their 6-bit equivalent and displayed on the scope. The following input characters are not displayed, but are treated as special characters.

Character	Code	Meaning
\	34	Ingnored on input.
ALT MODE	36	The scope display is reinitialized. All answer fields are reinitialized to the underline character (—).
RUBOUT	37	A cursor will always appear on the scope in front of the next character to be typed (unless there are no question fields). Typing RUBOUT will delete the character preceding the cursor and will move all characters following the cursor one character to the left, unless the cursor is initially at the beginning of a question field.
RETURN	43	The cursor is moved to the beginning of the next question field. If the cursor is currently in the last field, it will be moved to the beginning of the first field when RETURN is typed. If there are no question fields, RETURN will have the same effect as LINE FEED.
LINE FEED	45	Causes QANDA to exist to the "DONE" return.
TAB	47	Ignored on input.
<	74	Moves the cursor left one position. Subsequent typing of another legal character will cause the present character on the scope to the right of the cursor to be replaced by the character just typed.
>	76	Moves the cursor right one position unless that character is the underline character (—).

OUTPUT

All output is to the scope, as described above.

QANDA is written in LINC code. Along with the keyboard input subroutine, GETKBD, it occupies two blocks (512 words) of binary LINCtape. It may be assembled with the calling program by adding the source to the program. If this is done, remove *1000 at the beginning of the subroutine.

It may also be called by reading the binary of the program into LINC location 1000 (memory blocks 2 and 3) and executing an effective JMP 1000 and JMP 1053 to refresh. Two blocks of the binary must be read into core memory.

SIGAVG

(DEC-12-UZ1A-D)

Description

SIGAVG is a multisweep signal averager that extracts an analog signal from a high noise external environment and displays the signal on the scope. Sampling rates can vary from 55 to 4095 microseconds/point/instrument for up to 5 instruments. Up to 4096 sweeps can be taken; the value of the sweeps, the sampling rate, and the delay after sync can be altered during data acquisition. In addition, the SIGAVG source can be modified by the user to customize the program to his particular needs.

Minimum Hardware

PDP-12A with KW12A clock.

Library Distribution

Source file on DIAL tape DEC-12-SE2D-UO and binary files on tape DEC-12-SE4A-UO. Described in document DEC-12-UZ1A-D.

Operating Instructions

Choose the appropriate version of SIGAVG to suit the instrument from the following chart:

Name	# Channels	Data Points
SIGAVG1	1	1000
SIGAVG2	2	500
SIGAVG3	4	250

Use the name of the program in that chart in the command

```
LO NAME,UNIT
```

where unit is the tape unit with the SIGAVG binaries. The following messages are printed and must be answered with values in the indicated ranges.

message	acceptable values
R: (sampling rate)	55 - 4095 (microseconds)
N: (no. of sweeps)	1 - 4096 (sweeps)
D: (delay after sync)	(times clock rate)

The following Teletype commands can be issued.

CTRL/A	Initialize averaging parameters
CTRL/D	Restart DIAL
CTRL/Q	Quit current operation.
CTRL/R	Rerun the last average.
CTRL/Z	Zero out all previous results.

Carriage RETURN	Argument terminator. Commence averaging. Position plotter pen. Commence plotting.
LINE FEED	Pause averaging and start view input mode (same as carriage RETURN in plot mode).
C	Contract averaged data by a power of two.
P	Enter Plot mode Terminate Plot mode
T	Type out average.
V	Switch from view input to view average or vice versa.
X	Expand averaged data by a power of two.
W	Write on LINCtape.

SINPRE

(DEC-12-UW4A-D)

Description

SINPRE converts a double precision file, such as that from the Signal Averager program, into a single precision file by scaling the double precision file to ±8 bits (scope limits). The single precision file can be created on the same LINCtape as the original double precision file; internal checks in SINPRE prevent the new file from overwriting the old file. The conversion operation is specified by replying to 2 scope messages, one to locate the double precision file and the other to determine the location of the single precision file.

Minimum Hardware

PDP-12B.

Library Distribution

Source file on DIAL tape DEC-12-SE2D-UO, binary file on DEC-12-SE4A-UO. Described in DEC-12-UW4A-D.

Operating Instructions

The following are the messages displayed by SINPRE requiring a response.

```
     SINPRE

CONVERT A DOUBLE PRECISION FILE TO

A SINGLE PRECISION FILE


TYPE C TO CONTINUE -

DOUBLE PRECISION FILE

SNGL PRECISION FKE


FIRST BLOCK ---

UNIT -
FIRST BLOCK ---

LAST BLOCK ---

UNIT -
MOUNT TAPES

ON PROPER UNITS

TYPE C TO CONVERT -
REQUESTED DATA

HAS BEEN CONVERTED

TYPE R FOR ANOTHER JOB

REPLY -
```

TISA

(DEC-12-UW3A-D)

Description

TISA (time independent spectrum acquisition) acquires asynchronous (or synchronous) data simultaneously from 1 to 5 interfaced instruments that transmit X-Y data at rates that do not exceed 2 milliseconds/point. Asynchronous data can be acquired via potentiometers or shaft encoders, such that one input transmits X-axis data (independent variable) and the other transmits Y-axis data (dependent variable). The X coordinate can be arbitrarily defined by the user; thus, for example, the data points in an acquired spectrum can be made to correspond to wave numbers. Acquired data is displayed on the scope and stored on LINCtape. The moving window display is controlled by A/D channel knobs 0 and 4 and includes a cursor with X-Y decimal readout. For a 32K machine, TISA can acquire 29,184 data points.

Once started, TISA is in setup mode, during which the user can tailor the program to his experimental requirements by specifying such parameters as the number of points to acquire and the sampling frequency of each instrument. In addition, frequently used parameter sequences can be saved on LINCtape and then specified by tape location.

Minimum Hardware

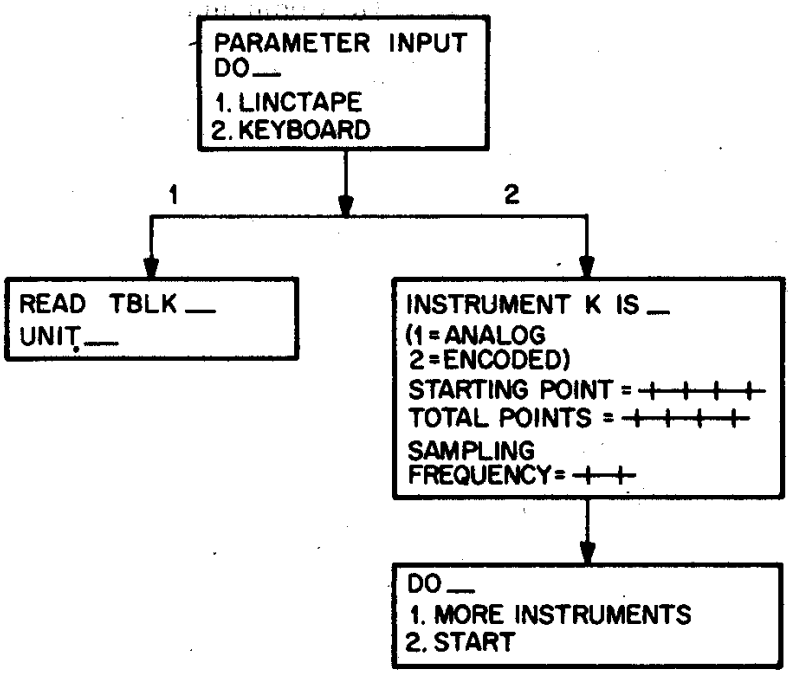
PDP-12B with KW12A clock.

Library Distribution

Binary file on DIAL tape DEC-12-SE5A-UO, paper tape DEC-12-UW3A-PB. Described in DEC-12-UW3A-D.

Operating Instructions

After loading (→LO TISA, unit ↓), TISA is in setup mode and requests experimental parameters by the following messages.



Setting Sense Switch 0 = 1 will erase all previous parameter input and initialize setup mode. After the parameters have been specified and data is being acquired, TISA is in A/D mode. When no data is being acquired, TISA is in pause mode. The following commands are operable.

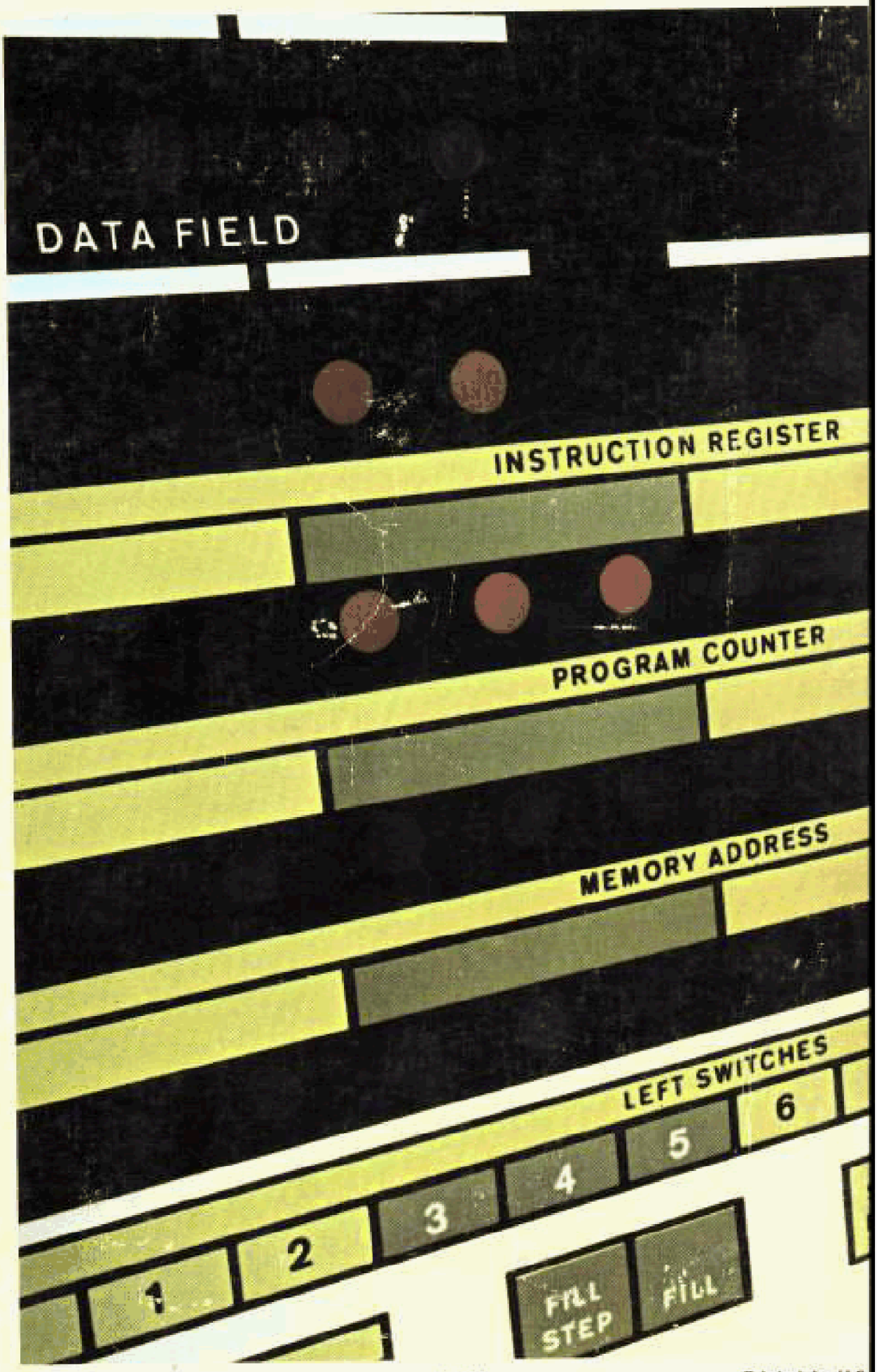
A/D MODE

CTRL/H	Halt all instruments
Hn	Halt instrument n
P	Polarize (invert) display
F	Freeze (half) display
D	Return to DIAL

PAUSE MODE

Gn	Go instrument n
S	Enter Setup mode
W	Write data buffers on LINCtape
C	Load CATACAL
M	Load MAGSPY
L	Load program from tape
P	Polarize (invert) display
F	Freeze (halt) display
D	Return to DIAL

INDEX


DATA FIELD
INSTRUCTION REGISTER
PROGRAM COUNTER
MEMORY ADDRESS
LEFT SWITCHES
6
5
4
3
2
1
FILL STEP
FILL