GENERAL REFERENCE MANUAL

SPURT

UP-3900

This manual is published by the UNIVAC® Division in loose leaf format as a rapid and complete means of keeping recipients apprised of UNIVAC Systems developments. The UNIVAC Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of hardware and/or software changes and refinements. The UNIVAC Division reserves the right to make such additions, corrections, and/or deletions as, in the judgment of the UNIVAC Division, are required by the development of its respective Systems.

PRINTED IN U.S.A.

PREFACE

This manual provides a comprehensive source of information concerning the UNIVAC 490 Real-Time System: its basic units; its wide variety of configurations and applications; its uniquely efficient instructions, and the programming packages which make the use of the system outstandingly practical and economical.

The bulk of the manual is addressed to those who have a good knowledge of general computer concepts and practices. The opening section, General Description, is intended for the use of anyone with an interest in the subject. In addition to a discussion of the system as a whole and of its principal components, this section deals with the relatively new concept of real time processing, of the reasons for it, of the manner in which the UNIVAC 490 Real-Time System, through concurrency of operations, combines the advantages of familiar batch processing methods with those of real time operations.

1. CONTENTS

2. INTRODUCTION

A. REAL TIME, BATCH PROCESSING, AND CONCURRENT OPERATIONS

1. Real Time

Real time computers were originally developed for military purposes: to aim, track, and guide ballistic missiles; to receive, store, analyze, and provide useful results from vast radar or other information gathering net works so as to provide the basis for timely decisions on the part of human beings – decisions which could, when required, be acted upon almost instantly at locations thousands of miles removed from their source.

The essential characteristics of real time computers are that they can store enormous masses of information; that they can access, or reach, and act upon, any item or group of items of the information within thousandths of a second; that they can receive and transmit data from and to small or large numbers of remote points in extremely short periods of time; that they can automatically assign priorities to many different operations so that action and response take place in proportion to the urgency of the need.

The expression *real* time is applied to computers of this kind because thay are capable of keeping pace in their operations with the occurrence of events which follow one another with great rapidity and which may even occur simultaneously.

The Univac Division of the Sperry Rand Corporation has been an outstandingly successful pioneer in the development of real time computers and realized very early the tremendous potentials of their application to business situations. The result of this realization and of the ensuing development work is the UNIVAC 490 Real-Time System which makes it possible for modern management to take the fullest advantage of modern management techniques. Because it provides proper input/output devices for every type of business transaction, business events may be entered into the system, stored, located, related to other events, acted upon at the time of occurrence. Management by exception is made possible on a truly current basis, in ample time to correct deviations from plan, whether these involve availability of working capital, sales forecasts, manning and machine scheduling, inventory levels, receipts and shipments of materials, operating costs, or controls.

The system, because of its unique mass storage and communications facilities, constitutes an excellent instrument for the centralized control of decentralized operations whether these are situated at a large central site or are dispersed over an area of hundreds or thousands of miles. Moreover, the system in many instances eliminates the need for original documents and their slow, expensive and frequently inaccurate conversion to a form suitable for input to a computer system.

The question is: how can a data processing system be made to provide all of these advantages in such a way as to approach the maximum utilization of the equipment employed, the highest return from the money expended for it, and therefore a correspondingly satisfactory increase in the profitability of the enterprise.

The answer is provided in the UNIVAC 490 Real-Time System in this manner:

2. Real Time and Batch Processing

The requirements for real time action are known frequently to occur in peaks and valleys. In many businesses these requirements tend to increase from early morning through the middle of the day and to taper off from then on. In other businesses the occurrence of these demands may be sporadic. It makes no difference. The UNIVAC 490 Real-Time System is so designed that it will automatically, as its facilities are freed from the dynamic demands of real time processing, load them up with the ordinary day to day back log of less urgent work of the familiar batch processing type - typically, the sequential processing of sequentially ordered files such as accounts receivable, payable, or payrolls.

3. Concurrency of Operations

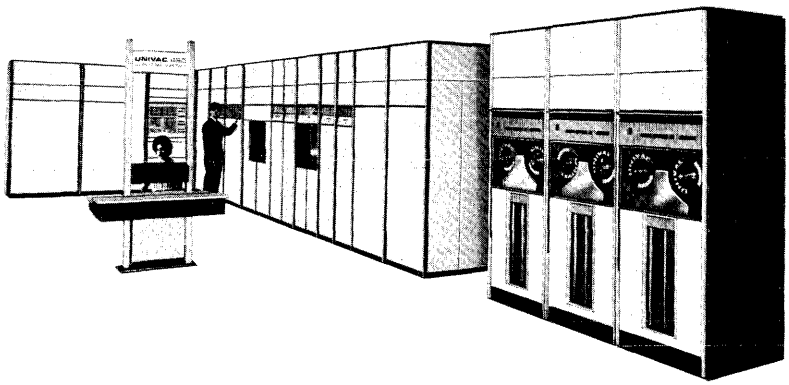
The great point is that the UNIVAC 490 Real-Time System is at no moment necessarily committed to real time operations or to batch processing operations exclusively. Both may proceed concurrently and several kinds of each may proceed concurrently under the control of an internally stored executive program. *But* the real time operations always have priority and the system will assign its facilities as these priorities require, relinquishing them to other activities, such as engineering calculations or normal business processing tasks, as soon as they are no longer needed to keep pace with real time events.

In this way maximum use may be made of the components of any desired configuration of the UNIVAC 490 Real-Time System; and the advantages of its enormous storage capacity, speed, flexibility, and communications capabilities may be obtained at a low cost per unit of work accomplished.

To sum up, it should be said that experience indicates that the UNIVAC 490 Real Time System will quite probably outperform by a wide margin any other system of its kind in a wide range of applications.

The fact that it can actually do so in any particular situation can be established beyond doubt only by investigation and analysis. This fact *has* already been established many times and *is being* established with greater and greater frequency as larger numbers of organizations examine the system's capabilities.

3. GENERAL DESCRIPTION



For years there has been an urgent need for a business data processing system specifically designed to meet the steadily increasing pressures of costs vs. profits, of competition, of the need for radically improved communications, for improved service to customers, for more efficient inventory management, for more effective controls -- for improved over all capability to cope with the constantly growing complexity, diversity, wide spread nature, and sheer size of modern business.

The UNIVAC 490 Real-Time System is the first and only system specifically designed to meet these needs and which has proved in daily use that it does meet these needs, and more, at an entirely reasonable cost. It is by a long way the most versatile and the most comprehensive data processing system yet produced.

The system provides the most effective instrument in existence for the centralized control of decentralized operations. The same characteristics which account for this, account also for its other advantages, which are many. These characteristics and advantages, stated very briefly, are:

- The system itself can communicate directly with central site or remote locations – points of sale or other customer contact, production or transportation facilities, warehouses, domestic or foreign branches or affiliates.
- The system can store large masses of information and almost instantly retrieve and deliver in useful form any item or group of items – data required for customer service, for the most advantageous stocking and distribution of materials or products, or for use by central or local management.

- Under automatic executive program control the system can concurrently execute several programs – the saving in processing time, and therefore in cost, is very large.

- The system operates in *real time* – that is, it responds to the need for action in a period of time proportional to the urgency of the need – *first* things are done *first.*

- The system can be depended upon to provide the information necessary to base this *minute's* or this *hour's* decisions on information up to date as of the *minute* or the *hour* – the value of information diminishes rapidly with the passage of time – *this* system, because the data it provides reflects the *present* facts, makes it possible to avoid many difficulties and to seize many opportunities which would otherwise be lost.

A. COMMUNICATION CAPABILITIES

The system is adaptable to *any* standard common carrier communication code, rate of transmission, or equipment. It may be connected to a few or hundreds of low cost communication lines.

B. MASS STORAGE FACILITIES

Drum storage capacity ranging from a few million to billions of characters – average access time 17 or 92 thousandths of a second – magnetic tape units compatible with other UNIVAC systems or with those of IBM – read/write speeds up to 125,000 characters per second; number of units, as in the case of other peripheral devices, expandable according to requirements. The combination of almost unlimited low cost communications facilities plus almost unlimited random access storage (at the lowest cost per character stored) provides service and economy never before approached in any system.

C. REAL TIME CENTRAL PROCESSOR DESIGN

Sixty two basic instructions which may be modified to provide unprecedented programming versatility; magnetic core storage capacity of 16,000 to 32,000 computer words of 30 bits each; 10 millionths of a second average instruction execution time; the ability to accomodate any standard communication code or speed; the ability to carry out several programs concurrently, to make most effective use of random access drum storage as well as magnetic tapes; *plus* the ability under automatic executive program control to perform all of its functions including control of or response to central or remote input/output devices in a time scale proportional to the need for action; *and* precision electronic clocks for use in compiling statistics for analysis and improvement of computer utilization, for initiation of action at specific times during the day, and for checking the proper execution of operations. In addition to all of this, the Central Processor may be connected to a satellite computer or to another entire 490 system complex.

D. INPUT/OUTPUT EQUIPMENT

In addition to its mass storage drums and tapes, the system may employ a variety of input/output devices including 600 card per minute readers; 150 card per minute punches; 700 to 922 lines per minute printers, punched paper tape systems (5 to 8 channel – 400 characters per second reading speed; 110 characters per second punching speed).

E. AN ADVANCED, COMPREHENSIVE PROGRAMMING SYSTEM

The UNIVAC 490 Real-Time System is provided with a rapid, efficient assembler, (SPURT), a COBOL compiler, and an executive routine (REX) which coordinates the operations of all elements of the system, making it entirely feasible to run several independent programs concurrently* and automatically to assign priority to most urgent requirements, such as those of a communications subsystem, as they arise. The programming package also includes an unusually efficient SORT/MERGE routine, utility, service and program testing routines, and routines for the most effective management of all peripheral units. Very large savings of time and money are realized by these means.

F. DEPENDABILITY OF OPERATION

This system has established an extraordinary record of operational dependability which results from the Univac Division's years of experience in the design of solid state computers of many kinds: military, scientific, and commercial. Reliability design goals have been exceeded, in actual performance, by well over 100%.

---

* *These programs may be written quite independently – the REX executive routine relieves the programmer of all concern with accomplishment of concurrent operations.*

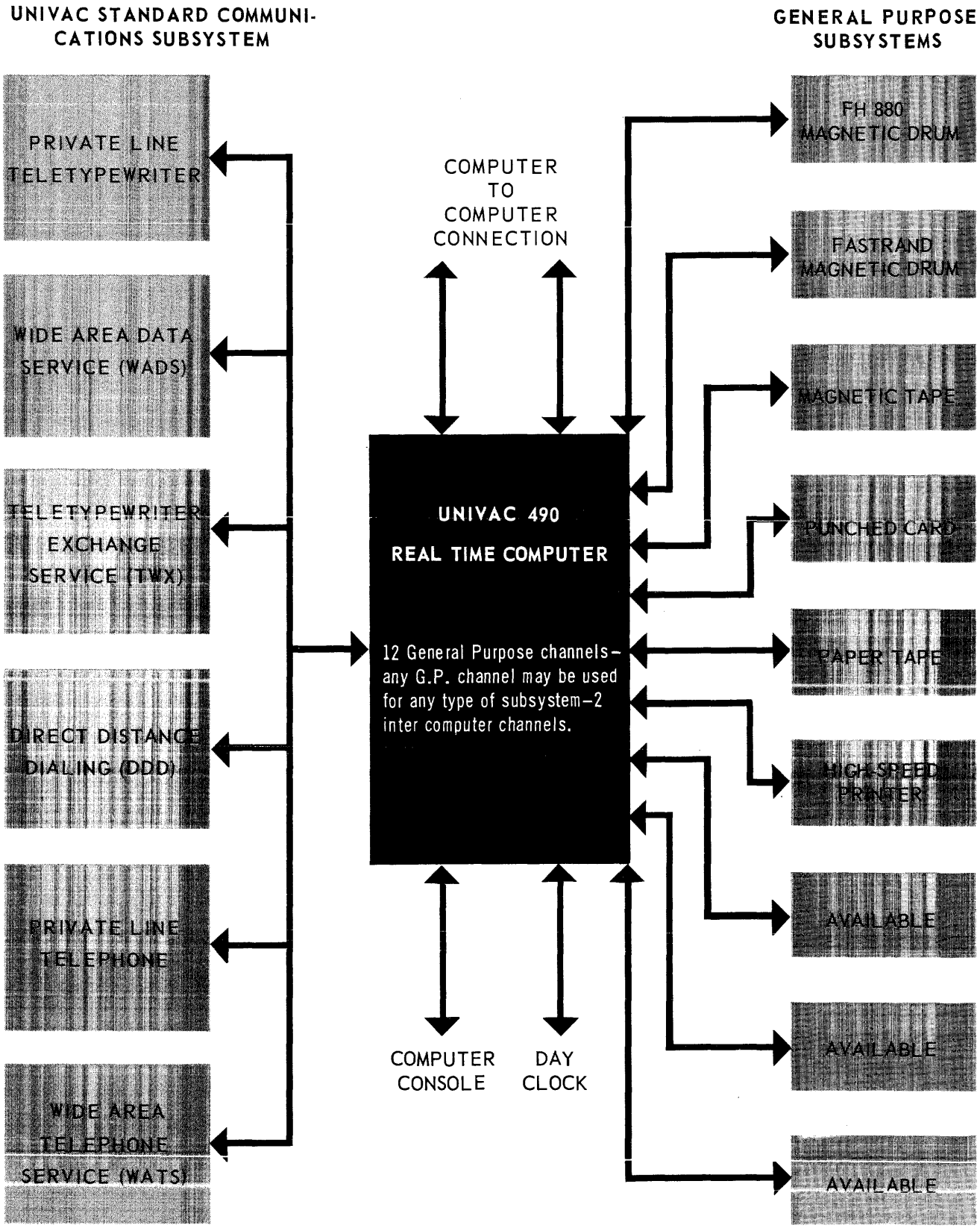


Figure 3-1. UNIVAC 490 Real-Time System – Simplified Block Diagram

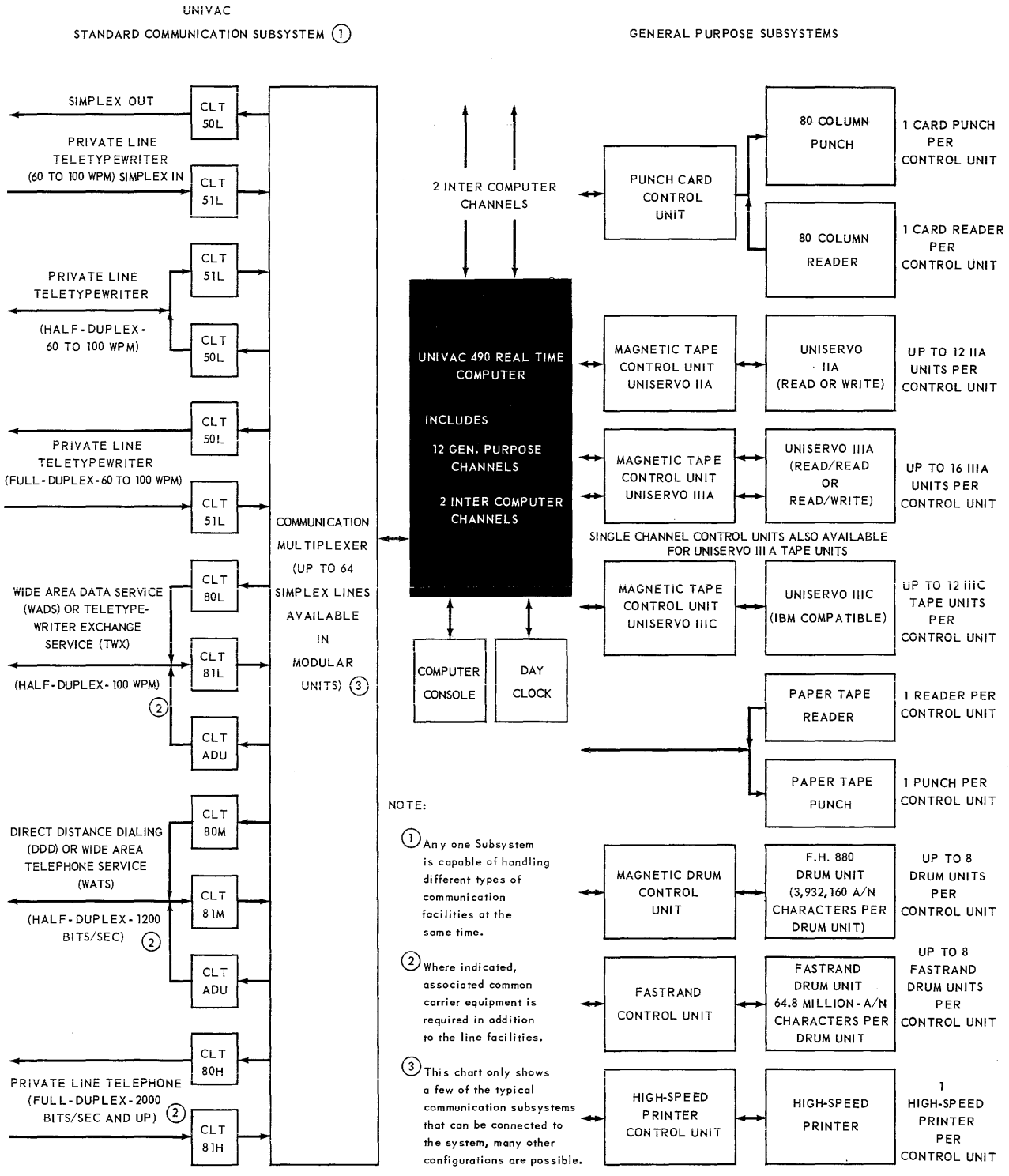


Figure 3-2. UNIVAC 490 Real-Time System (Expanded Block Diagram)

4. COMPONENT DESCRIPTION

A. CENTRAL PROCESSOR

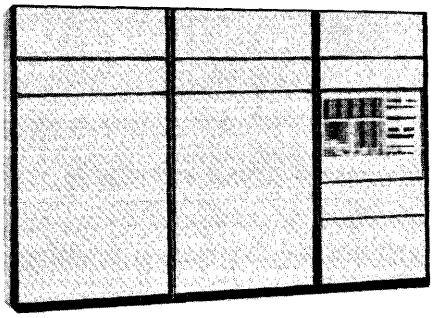


Figure 4–1. Central Processor

CHARACTERISTICS	
STORAGE CAPACITY	16, 384 words (30 bits/word) or 32, 768 words
WORD LENGTH	30 bits (may be accessed as 15 bit half word)
NUMBER OF I/O CHANNELS	6 or 14
AVERAGE ACCESS TIME	1.9 microseconds
CYCLE TIME	6 microseconds
AVERAGE INSTRUCTION EXECUTION TIME	10 microseconds

The UNIVAC 490 Real-Time System Central Processor is a binary computer designed for real time and batch processing, for concurrency of operations, for the employment of advanced programming concepts such as those incorporated in the SPURT assembler, the REX executive routine, the various input/output routines; and for the use of extremely extensive and rapid mass random access storage and communications subsystems as well as magnetic tape auxiliary storage, punched card equipment, and a variety of special peripheral devices.

It employs 62 basic instructions which, by automatic manipulation directed by the contents of the instruction word, can be made to perform practically any desired programming function.

A simplified diagram of the principal elements of the Central Processor is shown in Figure 4-2. For directness of presentation, only those elements referred to in program instructions are shown.

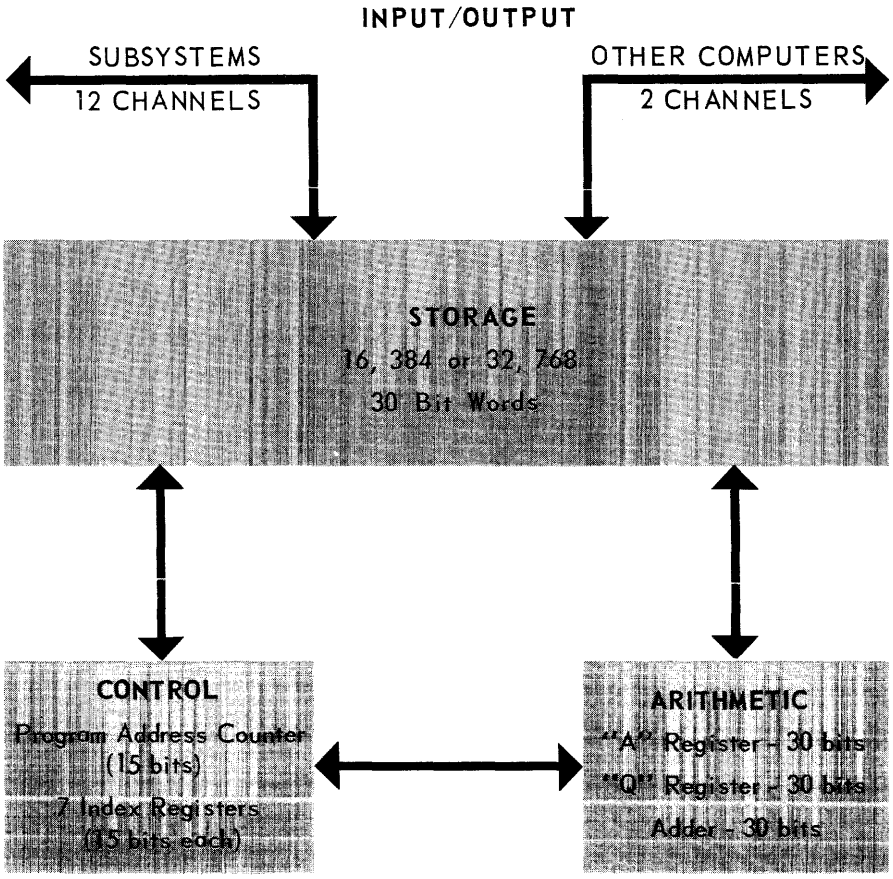


Figure 4-2. Central Processor - Simplified Block Diagram

- Control

  The control portion of the Central Processor coordinates the flow of data between the arithmetic and storage units. It contains a program address counter which holds the 15 bit address of the next instruction to be executed throughout the program and seven 15 bit index registers.

- Storage

  Storage consists of 16,384 or 32,768 30 bit words. Areas of storage are automatically allocated by the executive routine (REX) according to the requirements of operating programs.

- Arithmetic

  The arithmetic unit comprises a 30 bit "A" register, a 30 bit "Q" register and a 30 bit adder. The "A" and "Q" registers are used together in multiplication and division and, when desired, in 60 bit capacity. All arithmetic operations are performed in binary form. Subtraction is performed in the end around borrow manner; addition is performed by complementation of the addend and subtraction from the augend.

- Input/Output

All input/output data transfers are automatically controlled by signals emanating from the Central Processor or from the peripheral subsystems themselves. Any general purpose input/output channel can accommodate a UNIVAC 490 Real-Time System peripheral device. All I/O channels are buffered. Each input/output device has associated with it a Control Unit which, once activated by the Central Processor, carries out the entire operation of the instruction given it while the Central Processor proceeds with other tasks. This is invaluable. For example, in the case of a simple Print instruction, the Central Processor can, during the execution time – the printing of one line – execute more than 9,000 *other* instructions. Equally important is the fact that any input/output device can, through its Control Unit, instantly signal the Central Processor if it requires a response to an inquiry or if it has completed a task and is ready to begin another. Through REX, the executive routine, all activities, both internal and external to the Central Processor, are properly correlated, assigned the necessary areas of storage, and carried out in the most advantageous sequence.

B. CLOCKS

Processing timelines, an inherent characteristic of real time processing, requires the system to be extremely *time conscious*. Three precision electronic clocks provide the UNIVAC 490 Real-Time System with unequalled timing sensitivity.

1. Day Clock

A feature particularly suited to real time problems is the 24 hour Day Clock. This electronic clock interrupts the computer with the current time of day expressed in hours (0-23), minutes (0-59) and half minutes (0-1). A visual display of the time as indicated by the Day Clock is located on the operator's console.

The Day Clock can be used to initiate a variety of subroutines at specific times during the day. Error checking routines, trace routines, output conversions, management report programs, maintenance routines and memory dumps are some of the many routines which can be initiated by the Day Clock.

2. Real Time Clock

Of particular benefit to statistical analysis of computer utilization, the UNIVAC 490 Real-Time System contains an incremented clock called the Real Time Clock. This clock is a 15 bit register located in computer memory. Approximately once each millisecond the clock is incremented by one. Since 15 bits will allow a maximum count of 32,768, this clock will go through a complete cycle in approximately 32 seconds.

This clock can be used for precision timing of computer functions such as the elapsed time between the receipt of an inquiry and the transmission of a reply. The compilation of such statistics are an invaluable aid in analyzing and improving computer utilization.

3. Interval Timer

Like the Real Time Clock, the Interval Timer is a 15 bit register in computer storage which is incremented once each millisecond. The major difference between the two clocks is that the Interval Timer provides an unconditional interrupt to the computer upon reaching a full count of 32,768.

Erroneous input could perhaps cause a routine to initiate a repetitive logical path. The Interval Timer is set when a transaction is initiated. The time of interrupt is slightly in excess of the longest normal transaction. If an interrupt occurs, the real time program is prepared to enter a routine for analysis and possible reinitiation of the transaction which caused the interrupt.

C. SUBSYSTEMS

A subsystem consists of one or more peripheral units of the same type connected to an available input/output channel. Each subsystem is controlled by a channel synchronizer/control unit that interprets the control signals and instructions issued by the Central Processor, effects the transfer of data to or from the selected unit and the Central Processor, indicates to the Central Processor the status of the available peripheral units, and informs the Central Processor when errors or faults that affect the operation of the subsystem occur. The Central Processor and the subsystems have capabilities which lead to great efficiency in their mutual operations.

When the main program requires that the Central Processor employ a subsystem, the Central Processor issues control signals which select the proper subsystem and initiate the desired action. Once this is done the execution of the main program automatically continues until the subsystem has completed the required action. At this point the subsystem signals the Central Processor that the action is complete and the Central Processor now deals with the results of the action taken: for example the processing of data transferred from the subsystem.

In similar manner a subsystem signals the Central Processor its state of readiness to require action on the part of the Central Processor, such as response to an inquiry, and it also signals the Central Processor when its requirements have been met. These characteristics not only provide almost instantaneous availability of the services of the subsystems to the Central Processor, and those of the Central Processor to the subsystems, but they also reduce to at most a few thousandths of a second those Central Processor delays ordinarily associated with drum latency periods, magnetic tape reading or writing, or the employment of printing, punched card, or communications systems.

During the execution of input/output instructions the Central Processor proceeds with the main program, taking action on results of the operation only on receipt of a signal from the subsystem indicating that the operation is complete.

1. Flying Head-880 Magnetic Drum

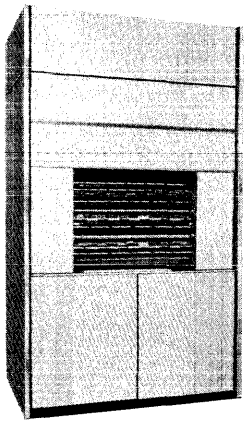


Figure 4-3. Flying Head-880 Magnetic Drum Unit

CHARACTERISTICS	
STORAGE CAPACITY	786,432 30 bit computer words 3.9 million alphanumeric characters (approximately)
AVERAGE ACCESS TIME	17 milliseconds
RECORDING DENSITY	518 bits per inch
DRUM SPEED	1,800 revolutions per minute
NUMBER OF READ/WRITE HEADS	880 (40 blocks with 22 heads in each block)

The Flying Head-880 Magnetic Drum Subsystem is a large capacity, high speed random access storage device used for program storage, temporary data storage, or file storage. As many as eight Flying Head-880 Magnetic Drum Units may be connected to any available input/output channel.

The magnetic drum units used in the Flying Head-880 Magnetic Drum Subsystem differ from conventional drum units in that the read/write heads float on a boundary layer of air created by the drum's rotation. Since the boundary is extremely thin, (less than 0.0005 inch) the head to surface distance is reduced and the read/write heads follow the contours of the drum in precise manner. This feature permits a greater recording density since it eliminates the disadvantages of the wider head to surface distance required in conventional drum units to compensate for surface irregularities.

The read/write heads are positioned around the drum in 40 groups of 22 heads each. Each head reads from or writes upon a recording track that revolves beneath it.

Each Flying Head-880 Magnetic Drum Unit has 128 6 bit track recording bands, each capable of storing 6,144 30 bit computer words; that is, 786,432 computer words (3.9 million alphanumeric characters) can be stored on a single unit.

The read/write heads record information on the drum at a density of 518 bits per inch while it is revolving at a speed of 1,800 revolutions per minute. Average access time is 17 milliseconds.

All read operations are parity checked. All search operations may be conducted off line; that is, once the search instruction is received by the drum synchronizer control unit, the Central Processor is free for other functions until the search is completed.

2. FASTRAND* Mass Storage Subsystem

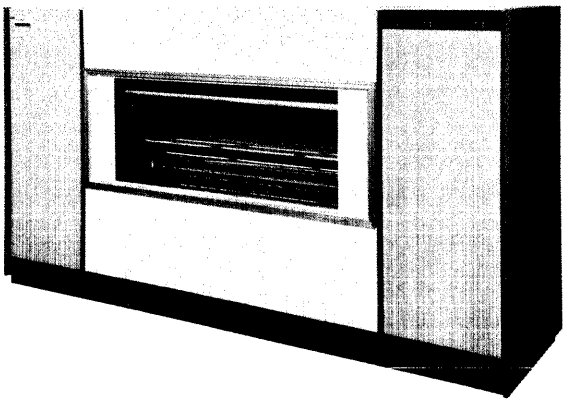


Figure 4-4. FASTRAND Mass Storage Unit

CHARACTERISTICS	
STORAGE CAPACITY	12,976,128 30 bit computer words 65 million alphanumeric characters (approximately)
AVERAGE ACCESS TIME	92 milliseconds
RECORDING DENSITY	1,000 bits per inch
DRUM SPEED	870 revolutions per minute
NUMBER OF READ/WRITE HEADS	64

The FASTRAND Mass Storage subsystem is an extremely large capacity random access mass storage device. As many as eight FASTRAND Mass Storage Units may be connected to any available input/output channel.

Each FASTRAND Mass Storage Unit in the subsystem contains two magnetic drums. These drums are similar to those used in the Flying-Head-880 Magnetic Drum Subsystem in that they employ flying read/write heads; however, these drums have an additional feature, the ability laterally to position the read/write heads.

On each drum there are 3,072 recording tracks each of which can store 2,112 30 bit computer words; that is, 6,488,064 computer words can be stored on one drum. Since there are two drums in a FASTRAND Mass Storage Unit the total storage capacity is 12,976,128 computer words per unit, or approximately 65,000,000 characters. The total words of storage per available channel is approximately 104,000,000; the total characters, 520,000,000.

* *Trademark of Sperry Rand Corporation*

a. Data Arrangement – Access Time

The data is recorded around the circumference of the drums in tracks. Each track will contain 2,112 words. These words are grouped into logical records called Sectors. A sector consists of 33 words.* Therefore, each track contains 64 sectors.

The 64 read/write heads are mounted on one head positioning bar and all move in unison. Each head will cover 96 tracks. The addressing logic is such that data from track 1 under head 1 is sequentially followed by data from track 1 under head 2 etc., through the entire 64 heads. Thus by one positioning of the heads (which may be accomplished by separate instructions or as part of a read or write instruction) 4096 sectors (135,000 words) are available without further head positioning. This feature when coupled with good systems design and data layout can result in reducing access time for many references to 35 milliseconds.

Among the valuable features of the FASTRAND system is a unique search function. The search may be initiated to search on the first word of each sector or it may be initiated to search all words of the sectors involved. When a find is made, the Search function automatically converts into a read operation as specified by instruction. In the process of searching the operation may be limited to search through only 64 sectors or to search through 4096 sectors. This variable search length feature is under program control.

All functions of the FASTRAND are buffered from the Central Processor so that the computer may continue processing while records are being accessed on the FASTRAND unit.

All read operations are parity checked.

** Information is recorded on FASTRAND in 33 word sectors. (a word contains 30 bits, five 6 bit characters)*

3. UNISERVO* Magnetic Tape Subsystems

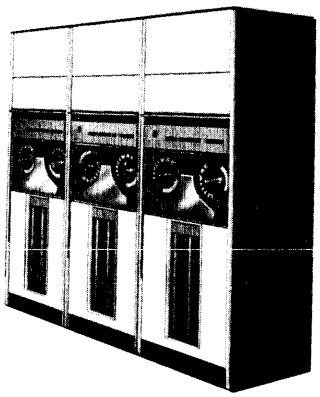


Figure 4–5. UNISERVO Tape Handling Unit

CHARACTERISTICS

	UNISERVO IIIA Tape Handling Unit	UNISERVO IIA Tape Handling Unit	UNISERVO IIIC Tape Handling Unit
TRANSFER RATE	100,000 or 125,000 characters/second	12,500 and 25,000 characters/second	22,500 and 62,500 characters/second
RECORDING DENSITY	1,000 or 1,250 6 bit characters /inch	125 and 250 6 bit characters/inch	200 and 556 6 bit characters/inch
TAPE SPEED	100 inches/second	100 inches/second	112.5 inches/second
TAPE WIDTH	0.5 inch	0.5 inch	0.5 inch
TAPE LENGTH	3,600, 2,400, 1,800	1,500 feet (metallic) 2,400 feet (plastic)	2,400 feet (plastic)
THICKNESS	1 mil	1 mil (metallic) 1.5 mils (plastic)	1.5 mils
BLOCK LENGTH	Variable	Variable	Variable
SPACE BETWEEN BLOCK	0.75 inch	1.2 inches	0.75 inch
CHANNELS ON TAPE	100KC-7 channels 6 data 1 parity 125KC-9 channels 8 data 1 parity	8 channels 6 data 1 parity 1 sprocket	7 channels 6 data. 1 parity
READ/WRITE	Reading in forward and backward directions; writing in the forward direction only.	Reading in forward and backward directions; writing in the forward direction only.	Reading and writing operations proceed in the forward direction only.

* *Trademark of Sperry Rand Corporation*

a. UNISERVO IIIA Tape Handling Unit

CHARACTERISTICS	
TRANSFER RATE	100,000 or 125,000 characters/ second
RECORDING DENSITY	1,000 or 1,250 6 Bit characters/inch
TAPE SPEED	100 inches/second
TAPE WIDTH	0.5 inch
TAPE LENGTH	3600, 2400, 1800
THICKNESS	1 mil
BLOCK LENGTH	Variable
SPACE BETWEEN BLOCKS	0.75 inch
CHANNELS ON TAPE	100KC-7 channels 6 data 1 parity 125KC-9 channels 8 data 1 parity
READ/WRITE OPERATION	Reading in forward and backward directions; writing in the forward direction only.

The UNISERVO IIIA Tape Handling Unit is a high speed magnetic tape storage device. The UNISERVO IIIA tape subsystem is connected to the UNIVAC 490 System through any available general purpose channel. Up to 16 tape units may be connected through one channel control unit. Several subsystems may be connected – one subsystem per channel.

Data is written or read to or from the UNISERVO IIIA Tape Handling Unit at a rate of 100,000 or 125,000 characters per second. The recording is done in the variable block length mode. The UNISERVO IIIA Tape Handling Unit has an automatic read after write feature to provide the maximum assurance that the data has been recorded correctly on the tape. Automatic recovery, bad spot detection and skipping are provided through a combination of hardware and executive program action.

Among the outstanding functions of the UNISERVO IIIA magnetic tape systems are: Read Forward, Read Backward, Write Forward, Masked Search, and High Speed Rewind.

As in the case of drum storage (of both types) and the other available magnetic tape units, the search function is conducted in the off line manner – that is, the control unit, once given the command by the Central Processor, takes over and the processor proceeds with the execution of other instructions in its program until the desired item is located.

The masked search function is extremely valuable in that it greatly simplifies programming because a search can be made upon any portion of an identifier word. The advantages of rapid rewind are obvious.

There are mechanical features of the UNISERVO IIIA Tape Handling Unit which contribute a good deal to its generally recognized status as the most advanced, reliable, and efficient magnetic tape system which now exists. It uses tape only 1 mil thick (most other tapes are 1.5 mils thick). Thus, more tape can be wound on a reel of normal size, and there can be more data per reel. Also, the tape is vacuum clutched which reduces slippage and tape wear and shrinks start/stop time to three milliseconds. Closing of the tape unit door automatically moves a newly mounted tape to its load point, ready for use.

When data is to be stored and manipulated by means of magnetic tape in large quantities and at high speed, UNISERVO IIIA Tape Handling Units provide the world's outstanding system, for the reasons cited above.

b. UNISERVO IIA Tape Handling Unit

CHARACTERISTICS	
TRANSFER RATE	12,500 and 25,000 characters/second
RECORDING DENSITY	125 and 250 6 bit characters/inch
TAPE SPEED	100 inches/second
TAPE WIDTH	0.5 inch
TAPE LENGTH	1,500 feet (metallic) 2,400 feet (plastic)
THICKNESS	1 mil (metallic) 1.5 mils (plastic
BLOCK LENGTH	variable
SPACE BETWEEN BLOCKS	1.2 inches
CHANNELS ON TAPE	8 channels 6 data 1 parity 1 sprocket
READ/WRITE OPERATION	Reading in forward and backward directions; writing in the forward direction only.

The UNISERVO IIA Tape Handling Unit is a moderate speed, low cost magnetic tape subsystem which may be connected to the UNIVAC 490 system, via a control unit through any available general purpose channel. Data is written or read to or from UNISERVO IIA units at rates of 12,500 or 25,000 characters per second. The subsystem can also read data from a magnetic tape produced by a UNITYPER*.

Up to 12 UNISERVO IIA units may be employed through one synchronizer control unit connected to one general purpose channel. As many units as the installation may require may be employed by the use of more than one general purpose channel.

Data may be written or read in either a fixed or variable block length. Fixed length is 720 characters. Variable length is entirely at the discretion of the user of the system. The only restrictions are that one block must comprise at least one computer word, and that it must not be so long as to exceed the system's storage capacity. The UNISERVO IIA Tape Handling Unit as a unit of the UNIVAC 490 System is completely compatible with the UNIVAC I, II, III computer systems, and the UNIVAC 1103A, 1105 and 1107 systems, as well as the UNIVAC File Computer, the Solid State and STEP systems, without off line tape conversion. Any tape unit on a given channel may communicate with the computer while the other units are rewinding.

Both read and search operations may be conducted when the tape is moving either forward or backward. The UNISERVO IIA subsystem can detect and signal end of file.(end of logically related blocks of data) when reading or searching. As explained under UNISERVO IIIA Unit, search is conducted in the off line manner.

** Trademark of the Sperry Rand Corporation*

c. UNISERVO IIIC Tape Handling Unit

CHARACTERISTICS	
TRANSFER RATE	22,500 and 62,500 characters/second
RECORDING DENSITY	200 and 556 6 bit characters/inch
TAPE SPEED	112.5 inches/second
TAPE WIDTH	0.5 inch
TAPE LENGTH	2,400 feet (plastic)
THICKNESS	1.5 mils
BLOCK LENGTH	Variable
SPACE BETWEEN BLOCKS	0.75 inch
CHANNELS ON TAPE	7 channels 6 data 1 parity
READ/WRITE OPERATION	Reading and writing operations proceed in the forward direction only.

The outstanding virtue of the UNISERVO IIIC subsystem is that it provides compatibility between the magnetic tapes of the world's two greatest producers of computer systems – The Univac Division of the Sperry Rand Corporation and IBM. There are many instances in which analysis shows that a UNIVAC computer such as the 490 is preferable to a marked degree, but the data to be processed is already on other than UNIVAC tapes. The UNISERVO IIIC Tape Handling Unit is the answer. When this subsystem is employed, offline conversion is utterly unnecessary. You simply take the old tapes, put them on the new machine, and go from there. The programming problems are insignificant because the circuitry of the UNISERVO IIIC Tape Subsystem control unit automatically deals with them.*

Up to 12 UNISERVO IIIC units may be connected to any available general purpose channel and, as with the other subsystems, more than one channel may be used for this purpose.

Data may be recorded at 200 or 556 characters per inch. The read/write speed is 22,500 or 62,500 characters per second. A tape may contain more than 3000 blocks of 1000 words each in high density binary recording, or 1000 blocks of 1000 words each in low density. Block length, subject only to storage capacity, is completely variable. For practical purposes, this is to say that it is completely at the will of the systems analyst or programmer.

* *UNIVAC punched card readers and punches also deal readily with 80 column punched cards. Translation from code to code is automatic.*

The UNISERVO IIIC has the further advantage that, unlike many earlier devices of its kind, it can write data on tape in what is known as the non stop mode. This means that block after block of data may be written without re-initiating the write instruction. The resultant simplification of programming is significant. As explained under UNISERVO IIIA Unit search is conducted in the offline manner.

4. High-Speed Printer Subsystem

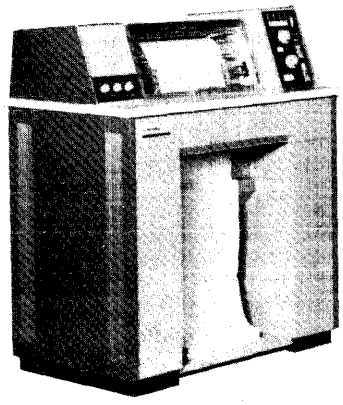


Figure 4–6. High-Speed Printer

CHARACTERISTICS	
LINES PER MINUTE	700 - 922
CHARACTERS PER LINE	132
LINES PER INCH (VERTICAL)	6 or 8
CHARACTERS PER INCH (HORIZONTAL)	10
NUMBER OF PRINTABLE CHARACTERS	63
PAPER STOCK	Any sprocket fed paper 4 to 27 inches wide, up to and including card stock thickness, either blank or preprinted forms.
NUMBER OF COPIES	At least 5 carbons and an original when 12 pound paper stock is used.

A High-Speed Printer subsystem may be connected to any available general purpose channel. Operating under program control, the subsystem produces single or multiple copy data at a rate of 700 lines of alphanumeric characters per minute, up to 922 lines of numeric characters per minute. The printed line may be up to 132 characters long.

Including the blank, or space, the standard character set provides 64 characters: the 26 letters of the alphabet, the digits 0 through 9: seven punctuation marks: comma, period, apostrophe, colon, semicolon, question mark, and exclamation point: and 20 special symbols [] right and left brackets, () right and left parenthesis, \ / right and left solidus, &, #, @, *, $, %, Δ, ¤, =, <, >, – (to represent either minus or dash), + and ±.

This is the standard character set.

Other character sets are available on special order to meet special needs.

The printer is designed to prevent a build up of static electricity, facilitating the proper stacking of paper which may be fed directly from the shipping container. Numbered calibrations on the printer enable the operator to record the positioning of a particular form and, at a later time, to set the same type of form to the necessary position. Fine adjustments are provided so that the operator may shift the paper horizontally or vertically the space of one character or line, or less, in either direction. The adjustment may be made while the printer is operating or while it it in the standby condition.

5. Card Subsystems

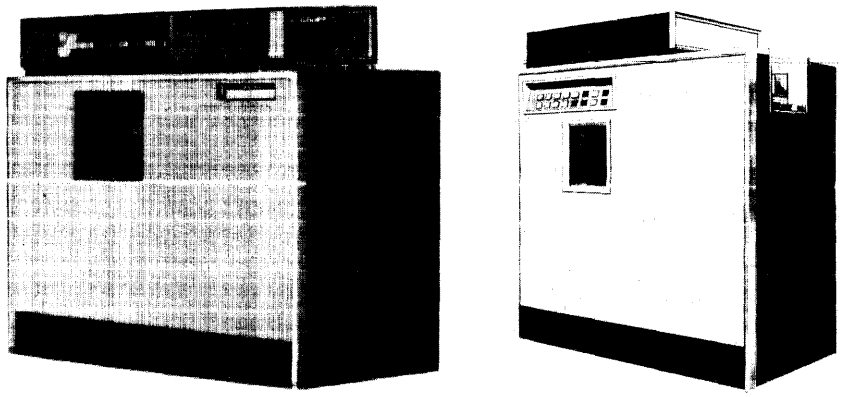


Figure 4–7. Card Reader and Punch Verifier

CHARACTERISTICS		
	CARD READER	PUNCH VERIFIER
CARDS PER MINUTE	600	150
NUMBER OF OUTPUT STACKERS	3	2

The Card Reader reads and checks 80 column cards at the rate of 600 cards per minute. Under direction of the program, cards are moved from the input hopper into the card channel and pass through two read stations. At the first read station the card image is read and stored. At the second read station the card is read again and compared to the image that was stored. If the comparison is equal, the data on the card is transferred to the Central Processor, and the card goes to the normal output stacker or the stacker by the program.

The Punch Verifier punches and checks 80 column cards at the rate of 150 cards per minute. Under direction of the program, a card is sent through the unit, information generated by the program is punched into the card at the punch station, the card is moved to the wait station, and then is moved to the third read station where it is verified. The verification is accomplished by reading the card and comparing the results hole by hole with the card image which is retained in the control unit until the checking is complete. When this has been completed the card is placed in the normal output stacker or the stacker selected by the program.

Translation from card code to internal machine code and vice versa is automatically accomplished within the control unit through the use of core storage and table look up techniques. By changing the contents of the table nonstandard translations may be performed.

6. UNIVAC Standard Communication Subsystem

The UNIVAC Standard Communication Subsystem enables the UNIVAC 490 Real-Time System to receive and transmit data via *any* common carrier in *any* of the standard codes and at *any* of the standard rates of transmission up to 4800 bits per second. It is the only communication system which can receive data from or transmit data to low speed, medium speed, or high speed lines in any combination.

The subsystem consists of two principal elements, the Communication Line Terminals (CLT's), which make direct connection with the communication facilities, and the Communication Multiplexer through which the CLT's deliver data to and receive data from the Central Processor. A Communicaiton Multiplexer may be connected to any general purpose computer channel or two or more multiplexers may be connected to two or more channels. If required, a number of multiplexers may be connected through a Scanner Selector to the same general purpose channel . The total number of multiplexers which can be connected to a general purpose channel is dependent on the number and speed of the communication systems linked to the multiplexers by their CLT's.

a. Communication Line Terminals (CLT's)

There are three basic kinds of input and output CLT's: low speed (up to 300 bps*), medium speed (up to 1600 bps) and high speed (2000 - 4800 bps). Each is easily adjusted to the speed and other characteristics of the type of line with which it is to operate - see CHARACTERISTICS at the beginning of this section. Each CLT requires one position, either input or output, of the Communication Multiplexer.

The CLT – Dialing is an output CLT which is employed to enable the Central Processor automatically to establish communications with remote points via the common carrier's switching network. Each CLT – Dialing requires one output position of a Communication Multiplexer. Since CLT – Dialing does not transmit data, it is always used in conjunction with an output CLT, an input CLT, or, for two way communications, both.

** bits per second*

b. Communication Multiplexer

The Communication Multiplexer functions as the link between the processor and the CLT's and is available in modules to handle 4, 8, 16, 32, or 64 CLT's. In each of these modules, an equal number of input and output CLT positions are provided. For example a 64 position Communication Multiplexer can accommodate up to 32 input and up to 32 output CLT's.

The CLT's may request access to the Central Processor via the Communication Multiplexer in random sequence, or several, or conceivably, all CLT's might request access simultaneously. The Communication Multiplexer automatically assigns priorities among CLT's requesting access and identifies to the Central Processor the particular CLT granted access.

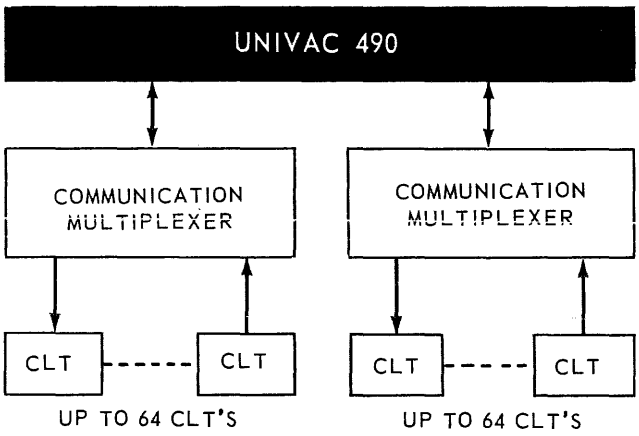


Figure 4-8. One Communication Multiplexer per General Purpose Channel

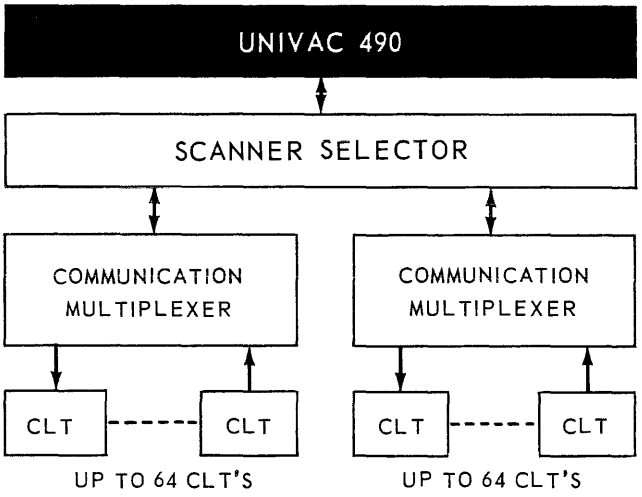


Figure 4-9. Multiple Communication Multiplexers per General Purpose Channel using Scanner Selector

CHARACTERISTICS
INPUT COMMUNICATION LINE TERMINALS (CLT's)

	LOW SPEED		MEDIUM SPEED		HIGH-SPEED
NAME	CLT51L	CLT81L	CLT81M	CLT81P	CLT81H
CODE	5 LEVEL	6,7,or 8 LEVEL	5,6,7,or 8 LEVEL	8 LEVEL	5,6,7,or 8 LEVEL
MODE	*ASYNCHRONOUS	ASYNCHRONOUS	ASYNCHRONOUS	***TIMING SIGNAL	**SYNCHRONOUS
	BIT SERIAL	BIT SERIAL	BIT SERIAL	BIT PARALLEL	BIT SERIAL
SPEED	UP TO 300 bps	UP TO 300 bps	UP TO 1600 bps	UP TO 75 cps	2000-4800 bps

OUTPUT COMMUNICATION LINE TERMINALS (CLT's)

	LOW SPEED		MEDIUM SPEED		HIGH-SPEED	† DIALING
NAME	CLT50L	CLT80L	CLT80M	CLT80P	CLT80H	CLT DIALING
CODE	5 LEVEL	6,7, or 8 LEVEL	5,6,7, or 8 LEVEL	8 LEVEL	5,6,7, or 8 LEVEL	4 LEVEL
MODE	ASYNCHRONOUS	ASYNCHRONOUS	ASYNCHRONOUS	TIMING SIGNAL	SYNCHRONOUS	TIMING SIGNAL
	BIT SERIAL	BIT SERIAL	BIT SERIAL	BIT PARALLEL	BIT SERIAL	BIT PARALLEL
SPEED	UP TO 300 bps	UP TO 300 bps	UP TO 1600 bps	UP TO 75 cps	2000-4800 bps	VARIABLE

COMMUNICATION MULTIPLEXER

NAME	FUNCTION
C/M-4	Connects 2 input and 2 output CLT's to General Purpose Channel
C/M-8	Connects 4 input and 4 output CLT's to General Purpose Channel
C/M-16	Connects 8 input and 8 output CLT's to General Purpose Channel
C/M-32	Connects 16 input and 16 output CLT's to General Purpose Channel
C/M-64	Connects 32 input and 32 output CLT's to General Purpose Channel

Types of Communication Service provided:

PRIVATE LINE TELETYPEWRITER
WIDE AREA DATA SERVICE (WADS)
TELETYPEWRITER EXCHANGE SERVICE (TWX)
WIDE AREA TELEPHONE SERVICE (WATS)
PRIVATE LINE TELEPHONE
DIRECT DISTANCE DIALING (DDD)

† *CLT – Dialing – This is an output CLT employed when the Central Processor is automatically to establish communications with remote points via the common carrier's switching network.*

* *ASYNCHRONOUS – Employs start and stop bit with each character to establish timing.*

** *SYNCHRONOUS – Uses timing characters at pre-determined intervals between data characters.*

*** *TIMING SIGNAL – Indicates the presence of a character at a Data Set.*

7. Punched Paper Tape Subsystem

a. Paper Tape Reader

CHARACTERISTICS	
READING RATE	400 characters/second
NUMBER OF CHANNELS	5, 6, 7, or 8
CHARACTERS PER INCH	10
TAPE SPEED	40 inches/second, free running (forward or reverse)
TAPE WIDTHS	11/16, 7/8, or 1 inch

b. Paper Tape Punch

CHARACTERISTICS	
PUNCHING RATE	110 characters/second
NUMBER OF CHANNELS	5, 6, 7, or 8 plus in line sprocket.
CHARACTERS PER INCH	10
TAPE SPEED	11 inches/second
TAPE WIDTHS	11/16, 7/8, or 1 inch

The Punched Paper Tape Subsystem enables the UNIVAC 490 Real-Time System to read or punch paper tape in all standard codes with programmed translation. A paper tape subsystem may be connected to any available input/output channel. Parity checking may be preformed in either reading from tape or in punching, under control of the Central Processor Program. The subsystem handles 5, 6, 7, or 8 channel tapes at a reading rate of 400 characters per second and at a punching rate of 110 characters per second.

Tape may be read, or punched, employing spools or individual unspooled strips of tape. When desired, numbers of strips can be spliced together and spooled for reading. When spools of tape are used as input to the reader, a supply reel and a take up reel are employed. Tape is read in a forward direction and may be back spaced a specified number of characters under program control.

5. PROGRAMMING IN SPURT

A. COMPUTER ELEMENTS RELATED TO PROGRAMMING

1. The Computer Word

The most fundamental level of storage in the computer is the internal data word. A data word is made up of 30 binary bit positions as shown in Figure 5-1. Each of these bit positions may represent a value of 0 or 1. When used for arithmetic operations, a value of 1 in bit position 29 will indicate a negative quantity; a value of 0 indicates that the value represented by the remaining bit positions is positive.

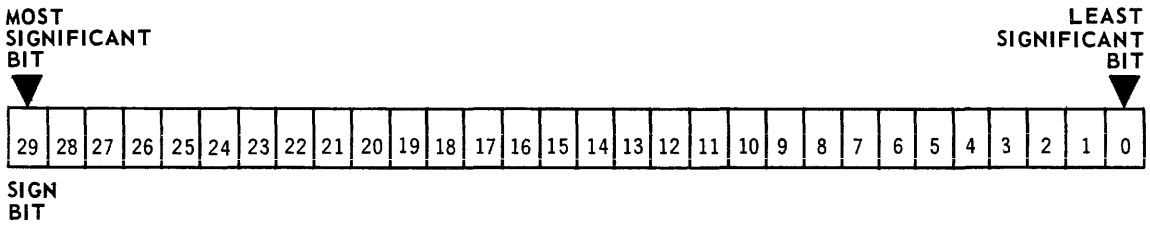


Figure 5-1. Basic Internal Data Word

Values may be expressed in binary notation for which the base is 2 instead of 10. The following equivalence exists:

BINARY	DECIMAL
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
. . .	. . .
10110101	181

The use of binary digits to represent large values is cumbersome. The use of octal notation for which the base is 8 is used for convenience. The following equivalence exists:

BINARY	OCTAL
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	10
. . .	. . .
111111	77
1100101	145

Binary values may be converted to octal notation by starting from the least significant (rightmost) digit. Each group of three binary digits is expressed as a digit from 0 to 7. By this method:

$$1\ \underbrace{100}\ \underbrace{101} = 145 \text{ (octal)}$$

$$\underbrace{111}\ \underbrace{101}\ \underbrace{000} = 750 \text{ (octal)}$$

A computer word containing 30 binary bits could be expressed in octal notation as:

7 7 7 7 7 7 7 7 7 7

Negative numbers are represented as the complement of positive numbers. A value of –3 is represented as:

7 7 7 7 7 7 7 7 7 4

The SPURT Assembly System will permit the user to express values as decimal numbers. These will be converted to their binary equivalents by the assembler. In order to distinguish between binary, octal, and decimal numbers, the following subscripting will be used in this manual:

n_2 = binary value of n

n_8 = octal value of n

n = decimal value (no subscript)

Examples:

$$11_2 = 3$$

$$10111_2 = 27_8$$

$$11_8 = 9$$

$$63 = 77_8$$

When the contents of a computer word are displayed, or if reference is made to a computer instruction word, octal notation will be assumed.

2. Addressing

Each word within the computer has a unique address. The available addresses in memory may range from 00000 to 37777_8 (if memory capacity is 16,384 words), or from 00000 to 77777_8 (if memory capacity is 32,768 words).

a. Data Addressing

Data is addressed by instructions that are themselves contained in the memory of the computer. When it is required to access data to complete an instruction, the instruction will contain an address portion capable of containing a maximum value of 77777_8.

b. Instruction Addressing

A basic computer instruction may be contained within one computer word. Instructions are accessed in memory, analyzed by the computer and then executed.

The next instruction is then accessed at the next sequential location unless a new sequence is specified.

c. Standard Locations

Memory locations 00000_8 through 00136_8 have special uses which will be fully explained in the appropriate context.

3. The Computer Instruction

A description of the basic computer instruction word and a summary of the numeric codes used to initiate computer functions are included in APPENDIX A. These instructions are made available as a reference for checking output from the SPURT Assembler. The SPURT Assembler provides a more convenient, easily remembered format for program coding. However, a general explanation of the basic instruction word will clarify such operations as address modification, operand modification, and jump interpretation. Reference will be made to these operations in the text dealing with the SPURT instructions.

The format of the basic computer instruction word is shown in figure 5-2.

f	j	k	b	y
29 24	23 21	20 18	17 15	14 0

Figure 5-2. Instruction Word

f — The value contained in these six bit positions determines the basic operation to be performed. The value is coded as two octal digits. For example, the code 20 specifies an addition of values.

y — This portion of the word may be a value representing the address of a 30 bit memory location at which the operand used in this operation may be found, or the 15-bit positions of this portion of the word may be the operand used by the instruction.

j — The most common use of the j portion of the instruction word is to specify a jump condition. If the condition (such as a negative sign and a value of zero in an arithmetic register) is present the next instruction will be skipped. This will permit the user to control program sequence based upon the results of an operation. For those instructions that do not have jump conditions, this portion of the instruction word may have other uses.

k — The k portion of the word will determine the size and format of the operand used by the instruction. Changes in the k portion will specify an operand which is in the upper 15-bit positions of a data word, the lower 15-bit positions, the entire 30-bit positions of a word with or without additional modification, or the lower portion of the instruction word itself.

b — The b portion of the instruction contains a value from 0 to 7. It refers to a B-register with a capacity of 15-bit positions that may be used for the non-destructive modification of the y portion of an instruction. For example, if the b portion of an instruction contains a value of 2, the contents of B-register number 2 will be used to modify the instruction. If the y portion of the instruction contains the value 00150, and B-register number 2 contains the value 00055, these octal values will be added as follows:

```
  00150
+ 00055
  00225
```

The address from which the operand is obtained is 00225. The y portion of the instruction remains unchanged.

An end-around carry will result when a carry is generated in the addition of the high order bits. This carry is added to the lowest order bit position of the sum. End-around carry is illustrated as follows:

```
77775 = 111111111111101  (y portion of instruction)
00005 = 000000000000101  ( contents of B-register)
00002 = 000000000000010
                      1  (end-around carry)
00003 = 000000000000011  (resulting operand address)
```

There are seven actual B-registers numbered 1, 2, 3, 4, 5, 6, and 7. The specification of a value of 0 in the b portion of a word has the same effect as adding 00000 to the y portion of the instruction.

Note:

B-register modification cannot be used to generate a resulting operand address of 00000, unless both the y portion of the instruction and the contents of the B-register equal 00000. This is due to the nature of end-around carry addition.

B. CENTRAL PROCESSOR

The central processor is a stored program binary computer that is designed to process large quantities of data in both batch processing and real-time modes. The central processor contains a storage section, a control section, an arithmetic section, an input/output section, and registers which, under direction of the program, perform the functions that effect the input, processing, and output of data.

The relationship of these sections to each other and the paths over which the data flows in going to and from each section is shown in Figure 5-3.

1. Storage Section

   At the option of the user, the central processor is available with a storage capacity of 16,384 or 32,768 computer words. Since information can be randomly selected, (the location in storage that a word is inserted into or removed from has no bearing on the time it takes to perform these operations) access to all locations in storage is the same; that is, words can be inserted into or removed from any location in storage at a rate of six microseconds per word.

2. Control Section

   The control section coordinates the flow of data between the arithmetic and storage sections and governs the operations that take place during the sequential execution of the instructions.

3. Arithmetic Section

   The arithmetic section performs arithmetic and logical operations, and communicates with the arithmetic registers, the adder or the other operational registers.

4. Input/Output Section

   The input/output section transfers data to and from the central processor via 14 input/output channels. Two of these channels are designed for communication with other computers. The remaining 12 channels are used for communication with the various peripheral units. The transfer of data to and from the central processor via these channels proceeds simultaneously with processing and is normally in the form of successive 15-bit half words or 30-bit computer words.

5. Registers

   The central processor contains a number of registers (15 or 30-bit areas apart from the storage section) that hold data during processing. These registers are designated by a letter or letter-numeral and are interconnected by parallel transmission paths. During processing,data flows to and from the registers via these paths.

   The registers fall into two categories: operational and transient.

   a. Operational Registers

      The operational registers are referred to by instructions in the program. Data that is placed in these registers is retained until it is replaced by new data.

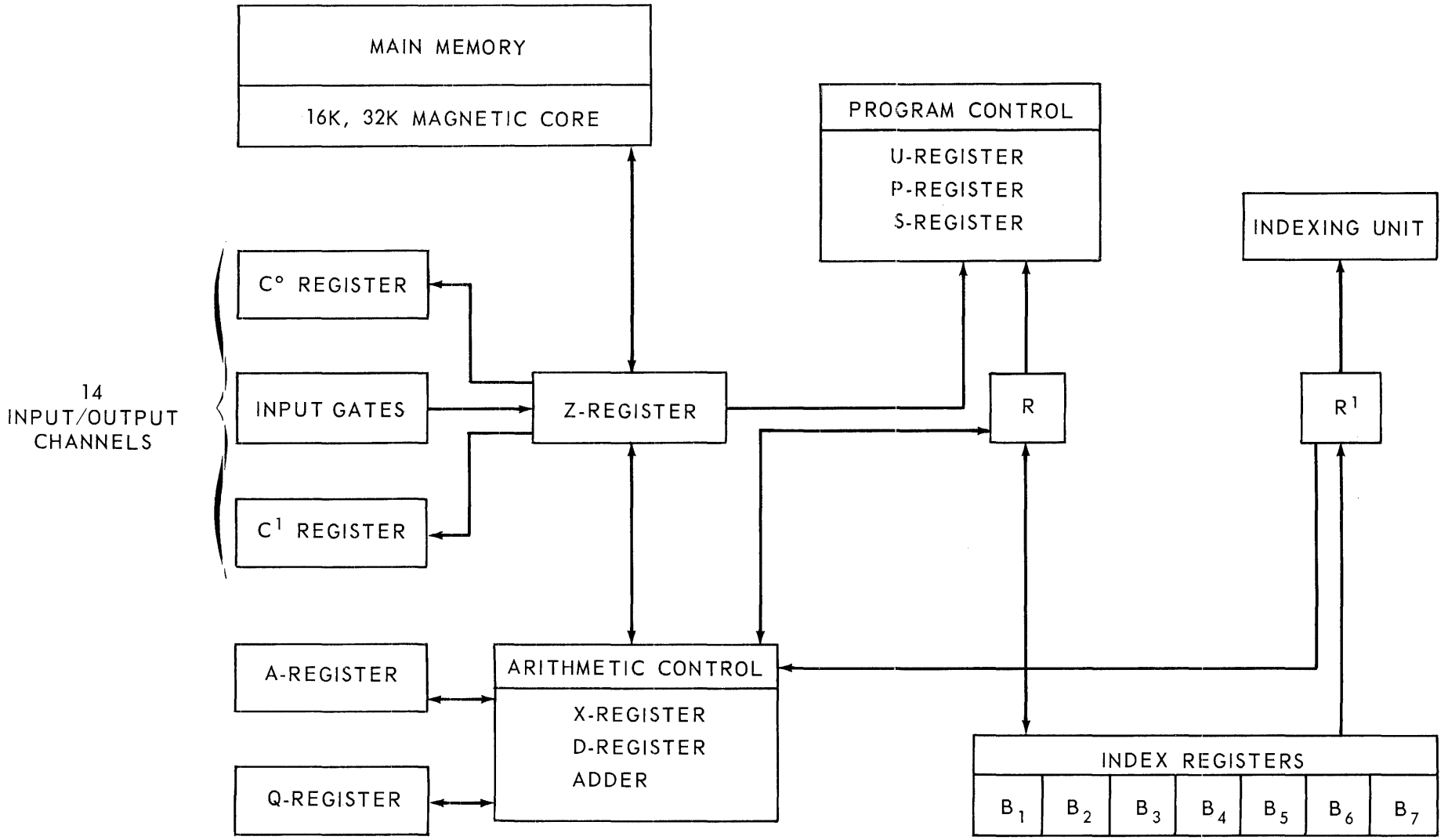


Figure 5-3. Simplified Logic Diagram of the UNIVAC 490 Real-Time Computer

(1) A-Register

The A-register is the principal 30-bit arithmetic register. It is used primarily for arithmetic and shifting operations (the moving of the contents of the register a specified number of positions to the right or left). In arithmetic operations, the result is usually retained in the A-register for use in later program steps. For example, after addition the sum is retained in the A-register, after subtraction the difference is retained, after multiplication the most significant half of the product (the upper 30-bits of the 60-bit product) is retained, and after division the remainder is retained.

(2) Q-Register

The Q-register is a 30-bit auxiliary arithmetic register that has adding, shifting, and logical properties.

(3) A- and Q-Registers in Combination

Multiply, divide, and certain shift operations utilize the A- and Q-registers as a single 60-bit register. For example, the Q-register holds the multiplier at the beginning of a multiply operation and when completed the 60-bit product is in the AQ-register; that is, the most significant half of the product (the upper 30-bits) is in the A-register and the least significant half of the product (the lower 30-bits) is in the Q-register. In a divide operation the AQ-register holds the dividend at the beginning and at the end, the quotient is in the Q-register and the remainder is in the A-register.

In shift operations, the 60-bit contents of the AQ-register may be shifted right or left in the same manner as either the A-register or the Q-register.

(4) B-Registers (Index Registers)

The B-registers, numbered B1 through B7, are 15-bit registers whose contents are used to increment the operand address before execution of an instruction if such modification is specified in the instruction word.

(5) P-Register (Program Address Counter)

The P-register is a 15-bit register that holds the address of the next sequential instruction throughout the program. As each instruction is executed, the contents of the P-register are usually increased by one. The contents may be changed to a non-sequential value by some instructions.

b. Transient Registers

The transient registers are used for the manipulation of instruction and data words during processing. They are not accessible to the user and do not retain information from one operation to the next. These registers are discussed in order to present a complete picture of the logical flow of data through the central processor.

(1) X-Register

The X-register, a 30-bit register, is used as an arithmetic communication register. The X-register receives the operand from storage during all arithmetic operations. All communication between the A- and Q-registers and the other operational registers or the adder output takes place via the X-register.

(2) S-Register

The S-register is a 15-bit register which holds the storage address during storage references. At the beginning of the storage access period, the address is transferred to the S-register. The contents of the S-register are then translated to activate the storage selection system.

(3) Z-Register

The Z-register is a 30-bit register that serves as an operand buffer for storage references. During the read portion of the storage access period, the Z-register is cleared. The digit reading amplifiers are then sampled to set the contents of the Z-register corresponding to the bits in storage. During the write portion of the storage access period, the Z-register controls inhibit circuits in order to write or restore the disturbed storage address. Input data from the input channels is gated directly to the Z-register.

(4) K-Register

The K-register is a 6-bit register that functions as a shift counter for shift operations and all arithmetic operations involving shifts. The maximum shift count permitted is 59. Multiply and divide operations are controlled by presetting the K-register to 30. The K-register then counts the operational steps.

(5) U-Register

The U-register or program control register is a 30-bit register that holds the instruction word during the execution of an operation. The function code and the various instruction designators are translated from the appropriate sections of this register. If an address modification is required before execution, the contents of the appropriate B-register are added to the low order 15-bits of the U-register.

(6) R-Register

The R-register is a 15-bit register that functions as a communications register for all internal transmissions to the B-registers.

(7) R'-Register

The R'register is a 15-bit register that functions as a communications register for all internal transmissions from the B-registers. During address modification this register holds the incrementing quantity.

(8) C-Registers

The C-registers are 30-bit communication buffer registers through which data is synchronized. There are two C-registers, C0 andC1. The C0 register is used to communicate output data to peripheral devices on a maximum of 12 different channels. The C1-register is used to communicate output data on two different channels to other computers. Input data is gated directly to the Z-register.

(9) D-Register

The D-register is a 30-bit arithmetic register that holds the operand for presentation to the adder during the execution of the arithmetic operations.

C. THE SPURT ASSEMBLY SYSTEM

The SPURT Assembly System will accept a source program via one of the available input media. A source program is the definition of a problem written in a convenient form resembling a series of verbal statements. These statements are then translated into several intermediate forms. The output from the assembly process is an object program which is acceptable as instruction input to the UNIVAC 490 Central Processor.

1. The SPURT Statement

The source program is composed of statements having the following general format:

l	w	v
label	OPERATOR	operand(s)

l — The label identifies a particular statement. Only statements that will be referred to by other statements require a label. A label may consist of a maximum of ten characters composed from the numbers (0 through 9) or the letters (A through Z). Alphabetic O and zero are equated and spaces are discarded starting at the end of the label field up to the first valid character. The following restrictions must be observed:

- A label cannot begin with the letters O or X, or any of the numbers (0 through 9).
- A label cannot consist of only A, Q, B0 through B7, or C0 through C16.

Examples of valid labels are:

PRPY12, BAC, THAN, CAT, DOG, TABLE12

Examples of *invalid* labels are:

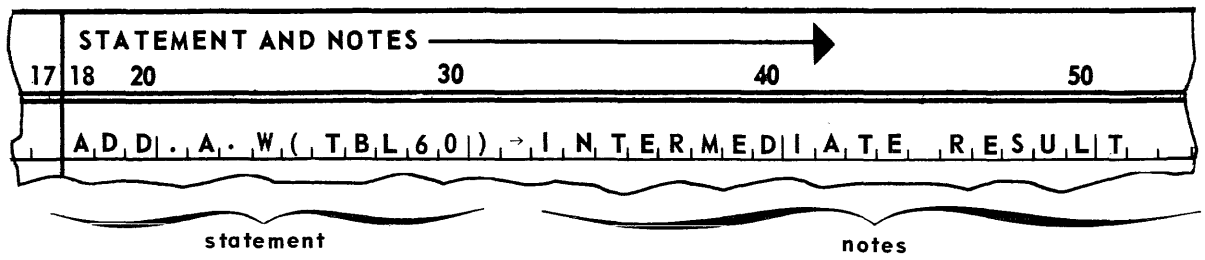
TABLE/12	(/is not a valid alphanumeric character)
B6	(B0 through B7 not permitted)
C11	(C0 through C15 not permitted)
XYZ	(begins with X)
TABLEΔ12	(internal spaces not permitted)

w — The operator is a mnemonic code which identifies the operation to be performed. It may cause the assembler to generate one computer instruction or a group of computer instructions.

v — One or a series of operands which are required for the operation. They will be referred to as v_0, v_1, v_2. . . . v_n.

. — A point separation symbol will be used to separate the operator from the first operand. If a statement requires more than one operand, this symbol will separate the operands.

A statement with notes will appear as follows:



Notes have no effect upon the assembly process. They will appear on some of the output listings and are used as a reference aid. A separation symbol (→) will separate the last operand from the notes.

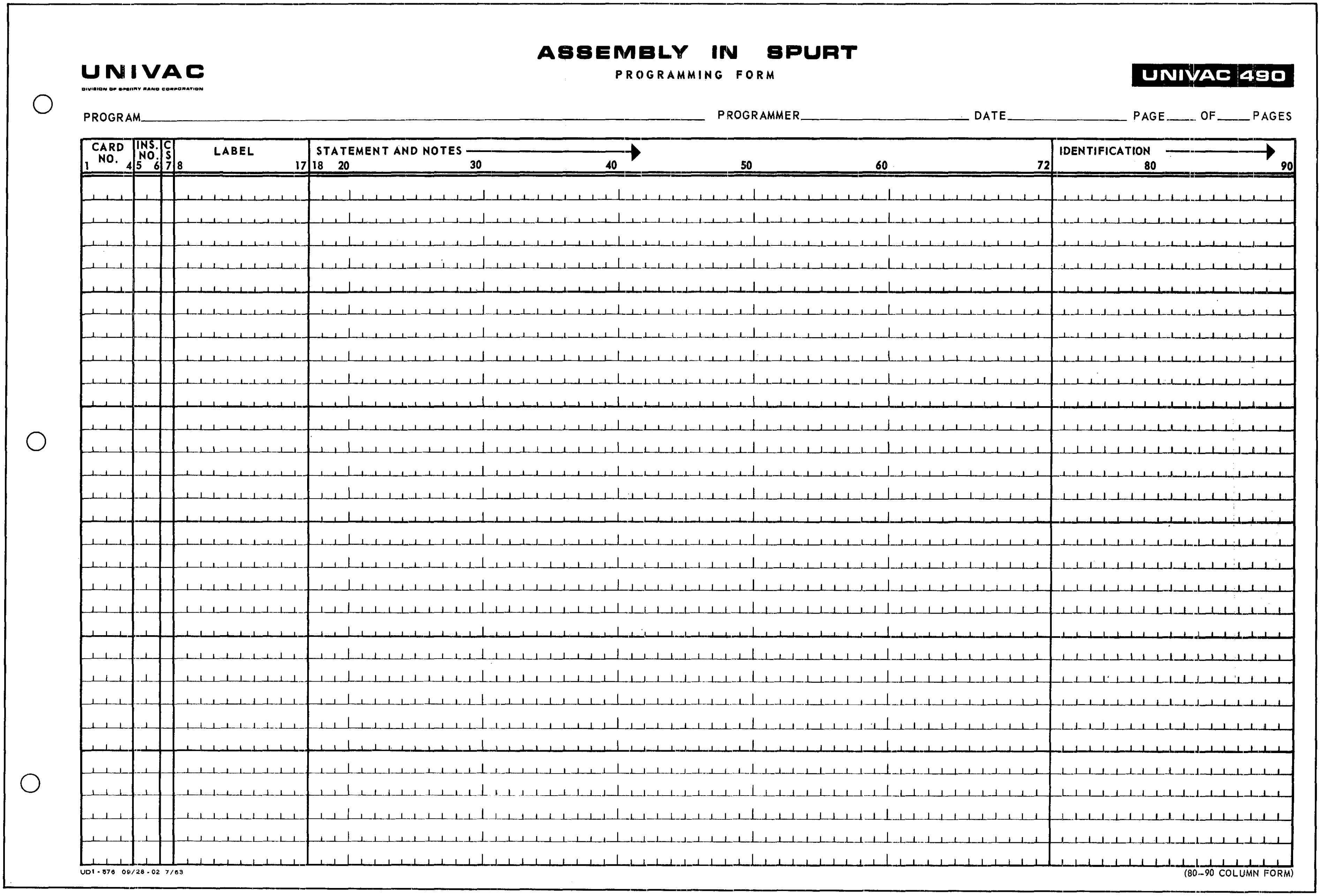
The actual codes which are the equivalents for each of the symbols are presented in APPENDIX B, SOURCE PROGRAM INPUT MEDIA.

2. The SPURT Coding Form

Figure 5-4 is a reduced representation of the 11" x 16" SPURT Coding Form. A line is provided at the top to identify the program, programmer, date, and page number. This information is not part of the source program input. Card input is recommended for ease of insertion and correction. The form may however, be used for other input media.

The line provided for the source program statement is arranged as follows:

COLUMN	USE
1-4	The card number provides an external sequencing criteria. It is not directly used by the assembler. The assembler will make a separate assignment in sequential order.
5-6	The insertion number is associated with the card number. It actually represents the two low order digits of the card number, which are not normally assigned initially. This permits sequential insertion.
7	A continuation symbol is used in this column of the additional lines, if more than one line is required for a statement.
8-17	If a statement is labelled. The label must appear within these columns, left-justified.
18-72	The statement and notes are written within these columns. Separation of the components of the statement is accomplished by the point separation symbol. The statement is separated from the notes by a separation symbol.
73-80 or 73-90	The remaining columns are used for additional remarks or identification. These entries have no effect upon the assembly process.



UNIVAC
DIVISION OF SPERRY RAND CORPORATION

ASSEMBLY IN SPURT
PROGRAMMING FORM

UNIVAC 490

PROGRAM________ PROGRAMMER________ DATE________ PAGE____ OF____ PAGES

CARD NO. 1–4	INS. NO. 5–6	CS 7	LABEL 8–17	STATEMENT AND NOTES → 18 20 30 40 50 60 72	IDENTIFICATION → 80 90

UD1-576 09/28-02 7/63

(80–90 COLUMN FORM)

Figure 5-4. SPURT Coding Form.

D. SPURT INSTRUCTION FORMAT

The basic SPURT instruction is defined by an operator, such as MUL to indicate a multiply operation. An allied operand may be used to further define a basic operation. For example, if the basic operator is STR, to indicate a store type of operation, the allied operand may modify the basic operation as follows:

STR · A means store the contents of the A-register in an operand location.

STR · Q means store the contents of the Q-register in an operand location.

Instructions within this section will be presented in the following format:

OPERATION	Y	J
operator	**operand**	**jump designation**

The operator, as described above, will appear under OPERATION. Both the operator and its allied operand will be included.

When an operand is required to complete an operation it will appear in the box under Y. The three general classes of operand will be defined below under THE GENERAL OPERAND. If an operand belongs to one of these classes, the class will be written in the box. Exceptions to the general classes will be noted and explained.

The box under J will refer to the jump designation. The word "normal" will be placed in this box if a normal jump designation applies to this operation. Exceptions to the normal jump designation will be noted and explained.

1. The General Operand

Basic SPURT instructions are of three general classes: read class, store class, and replace class. These classes are related to the k-designator of the computer instruction. Various modifications to the operand are performed depending upon the class of operand and the modification symbol that is specified. The modification may affect the high order 15-bits of the word, the low order 15-bits of the word, or the entire 30-bits of the word.

Two forms of the general operand that are used by the three classes of instructions are constants and operand labels.

a. Constants

A value may be assigned to a computer word by the specification of a constant. The values may be expressed as either an octal or a decimal value. Decimal values are followed by the character D. Positive values require no sign; negative values are preceeded by a minus sign (–).

The maximum decimal value that may be expressed as a positive or negative quantity is 536870911D or –536870911D.

The maximum octal value that may be expressed as a positive or negative quantity is 3777777777 or –3777777777.

A group of labelled constants would appear as follows:

C S	LABEL	STATEMENT AND NOTES	WORD REPRESENTATION GENERATED
	HIVALUE	1028D	00000 02004
	LWVALUE	-377	77777 77400
	TABLE	77.62D	00077 00076

Two constants may be placed in the upper and lower portions of a word with right justification and zero fill as shown in the example labelled TABLE. The two values must be separated by a point separation symbol and the maximum values defined may be 77777 or 32767D.

b. Operand Labels

The definition of a label may be found under THE SPURT STATEMENT. All restrictions apply to the labelling of a constant. An instruction may access a constant value by the label assigned to that value. The actual or relative memory address of the value will be assigned by the assembler.

References to the computer memory address that is assigned to a label may be modified by an increment or decrement. The increment or decrement may be a maximum of 77777_8 or 32767.

The computer memory address assigned to a label may also be modified by the contents of a B-register. This is accomplished by following the label with +B1, +B2, +B3, +B4, +B5, +B6, or +B7.

Both an increment or decrement and a B-register modification may be used.

Examples of modified labels are:

TAXRTE-5	(the address represented by TAXRTE will be decremented by 5.)
PRX12+B6	(the address represented by PRX12 and the value contained in B-register number 6 will be added. The result will be the address of the operand.)
LAB05+12D+B3	(the address represented by LAB05 will be incremented by 12 (decimal). The value contained in B-register number 3 will be added to the result. The final result will be the address of the operand.)

c. The Read Class Operand

A read class instruction involves the transfer of a value from a memory location to an arithmetic register, where an operation specified by the operator may be performed. An operand used in these operations may be modified when it enters the arithmetic register.

Operand modification is specified by enclosing an operand within parentheses. The operand may be incremented or decremented and/or B-register modification may be specified. An actual machine address may also be used, if the actual address of an operand is known. Any of these means of addressing an operand will be represented by x in the format definitions which follow.

L(x) the value is obtained from the lower 15-bits of the memory location. The upper 15-bits of the arithmetic register are filled with zeros.

U(x) the value is obtained from the upper 15-bits of the memory location. The value is right justified in the register and the upper 15-bits of the arithmetic register are filled with zeros.

W(x) the value is the whole word of the memory location.

LX(x) the numeric value is obtained from the lower 15-bits of the memory location. The most signification bit (sign bit) of the 15-bit value is extended through the upper 15-bit positions of the register.

UX(x) the numeric value is obtained from the upper 15-bits of the memory location. The value is right justified into the register and the most significant bit (sign bit) of the 15-bit value is extended through the upper 15-bit positions of the register.

Read class operands may also appear in the following forms:

(1) A Half-Word Constant

A constant value equal to or less than 5 octal digits will be stored in the address portion (lower 15-bits) of the computer word that is generated. When the instruction is executed, the value is right justified in the arithmetic register and the remainder of the register is zero filled.

Sign extension of a constant value equal to five octal digits, with a most significant digit of 4, 5, 6, or 7, may be specified by preceding the value with the symbol X. Examples of values of this type and the binary values that will be brought to the register are:

OPERAND	BINARY VALUE IN REGISTER	
40000	000000000000000	100000000000000
X40000	111111111111111	100000000000000
77773	000000000000000	111111111111011
X77773	111111111111111	111111111111011

The extensions referred to above for LX(x) and UX(x) are similar.

(2) A Register Mnemonic

The A-register and any of the B-registers may be addressed as an operand. For this type of addressing, one of the following symbols is used:

A, B1, B2, B3, B4, B5, B6, or B7

Only the simple unmodified form for the A-register may be used as an operand. A B-register specification may appear without a label; for example, L(B3) will access the lower 15-bit positions of the computer address represented by the value in B register number 3.

d. The Store Class Operand

A store class instruction involves the transfer of a value from an operational register to a register or a memory address. The operand, represented by x, may be in any of the forms described under GENERAL OPERAND.

L(x) The value is obtained from the lower 15-bit positions of the register and stored in the lower 15-bit positions of the memory location. The upper 15-bits of the memory location remain unchanged.

U(x) The value is obtained from the lower 15-bit positions of the register and stored in the upper 15-bit positions of the memory location. The lower 15-bits of the memory location remain unchanged.

W(x) The entire contents of the register are stored in the memory location.

CPL(x) The value is complemented and storage is as described for L(x) above.

CPU(x) The value is complemented and storage is as described for U(x) above.

CPW(x) The value is complemented and storage is as described for W(x) above.

The A-register and the Q-register may be specified as storage locations. For this type of addressing, one of the following symbols is used without further modification.

A or Q

e. The Replace Class Operand

A replace class instruction involves the transfer of a value to an arithmetic register. An operation is performed in the register. The resulting value will then replace the initial contents of the memory location.

An operand address, represented by x, may be in any of the forms described under GENERAL OPERAND. An operand address must always appear within parentheses in a replace class instruction. The operand modification formats are:

L(x) The value is obtained from the lower 15-bit positions of the memory location. After the basic operation is performed, the value contained in the lower 15-bit positions of the arithmetic register will replace the previous contents of the lower 15-bit positions of the memory location. The upper 15-bit positions of the memory location remain unchanged.

U(x) The value is obtained from the upper 15-bit positions of the memory location and normalized to the right in the arithmetic register. After the basic operation is performed, the value contained in the lower 15-bit positions of the arithmetic register will replace the previous contents of the upper 15-bit positions of the memory location. The lower 15-bit positions of the memory location remain unchanged.

W(x) The value is obtained from the entire 30-bit positions of the memory location. After the basic operation is performed, the value contained in the entire 30-bit positions of the register will replace the entire 30-bit positions of the memory location.

LX(x) The value is obtained from the lower 15-bit positions of the memory location with sign extension into the upper 15-bit positions of the arithmetic register. After the basic operation is performed, replacement is as described for L(x) above.

UX(x) The value is obtained from the upper 15-bit positions of the memory location, normalized to the right of the arithmetic register, with sign extension into the upper 15-bit positions of the arithmetic register. After the basic operation is performed, replacement is as described for U(x) above.

f. Operand Interpretation Summary

The descriptions of the operand interpretations for normal read, store, and replace class operands have been summarized in a diagrammatic form designed for easy reference. They may be folded out for easy access at the beginning of Section 5E.

g. Normal Jump Designation

The normal entries for J are as follows:

J MNEMONIC	SKIP CONDITION
(absent)	The next operation will be performed.
SKIP	Skip next operation unconditionally.
QPOS	Skip next operation if Q is positive.
QNEG	Skip next operation if Q is negative.
AZERO	Skip next operation if A is zero.
ANOT	Skip next operation if A is non-zero.
APOS	Skip next operation if A is positive.
ANEG	Skip next operation if A is negative.

Both the normal jump designations and some of the irregular entries are presented on an easily referenced foldout at the beginning of Section 5E.

2. Comments on the Presentation of the Basic SPURT Instructions.

The basic instructions will be presented in groups related to their function. The format of the instruction will be followed by an example written in SPURT language. The computer instruction that is generated from the SPURT mnemonics will be shown on the right.

When the label is used in the examples, it will be written as LABEL. The assembler assigned computer address to which this corresponds will be 01234. This will maintain a consistent point of reference in the examples. For actual coding any valid label could be used.

E. BASIC SPURT INSTRUCTIONS

1. Data Transfer Instructions

Transfer instructions either transfer data that is contained in a storage location to a register or store the contents of a register in a storage location.

- ENTER Q

This instruction transfers an operand to the Q-register.

OPERATION	Y	J
ENT · Q	**read class**	**normal**

Example:

ENT · Q · W (LABEL) 10030 01234

- ENTER A

This instruction transfers an operand to the A-register.

OPERATION	Y	J
ENT · A	**read class**	**normal**

Example:

ENT · A · L (B2 + 2) 11012 00002

- ENTER Bn

This instruction transfers a 15-bit operand to a selected B-register.

OPERATION	Y	J
ENT · Bn	**note 1**	**not used**

n may be 0, 1, 2, 3, 4, 5, 6 or 7 to specify the B-register.

Example:

ENT · B7 · 5 12700 00005

NOTE 1

A normal read class operand may be specified. If more than a 15-bit operand is indicated, only the low order 15-bits will be transferred into the B-register.

TYPE	MNEMONIC CODE		INSTRUCTION	OPERATION		EXECUTION TIMES * k:0,4	EXECUTION TIMES * k:7	EXECUTION TIMES * k:1,2,3,5,6	PAGE
TRANSFER	ENT•Q	10	Enter Q	Y→Q	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-1
	ENT•A	11	Enter Accumulator	Y→A	Rd				5E-1
	ENT•B_n	12	Enter B-Register	Y→B_n; B_o: No Op	Rd	6.0/3.6			5E-1
	STR•Q	14	Store Q	(Q)→Y	St	8.4/4.8	12/		5E-2
	STR•A	15	Store Accumulator	(A)→Y	St				5E-2
	STR•B	16	Store B-Register	(B)→Y	St				5E-2
SHIFT	RSH•Q	01	Shift Q Right	Shift Q right by Y_{05-00}, Sign Fill	Rd	8.4/	9.6/	13.2/	5E-3
	RSH•A	02	Shift A Right	Shift A right by Y_{05-00}, Sign Fill	Rd				5E-3
	RSH•AQ	03	Shift AQ Right	Shift AQ right by Y_{05-00}, Sign Fill	Rd	8.4/	9.6/	13.2/	5E-3
	LSH•Q	05	Shift Q Left	Shift Q left by Y_{05-00}, Circularly	Rd	8.4/	9.6/	13.2/	5E-4
	LSH•A	06	Shift A Left	Shift A left by Y_{05-00}, Circularly	Rd				5E-5
	LSH•AQ	07	Shift AQ Left	Shift AQ left by Y_{05-00}, Circularly	Rd	12/	13.2/	16.8/	5E-5
COM	COM•A•Q•AQ	04	Compare	A:Y or Q:Y or AQ:Y; and sense j	Rd	9.6/7.2	8.4/6.0	12/8.4	5E-6
	COM•MASK	43	Mask Compare	(A)-L⊂Y (Q)⊃; sense j A and Q Unchanged	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-7
JUMP	JP	60	Arithmetic Jump (Normal)	Y→P, per j; Y: 15 bits maximum	Rd	6.0	7.2	12	5E-8
	RJP	64	Arithmetic Return Jump	P+1→Y_{14-00}; Y+1→P if j is met	Rd	13.2	14.4	18	5E-8
MODIFY	RPT	70	Repeat	NI (Y) times per j; B_7=NEn	Rd		7.2		5E-9
	BSK•B_n	71	Index Skip	If (B_n)=(Y), skip NI, and CL (B_n)	Rd	7.2/	8.4/		5E-10
	BJP•B_n	72	Index Jump	If (B_n)≠(Y), take NI, B_n=(B+1) If (B_n)=0, take NI If (B_n)≠0, jump to Y; B_n=(B-1)	Rd	6.0/	7.2/		5E-11
SUBTRACTION	SUB•A	21	Subtract	(A) - Y→A	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-12
	SUB•Q	27	Q Subtract	(Q) - Y→Q	Rd	9.6/7.2	8.4/6.0	12/8.4	5E-12
	ENT•Y-Q	31	Subtract Q and Load A	Y -(Q)→A	Rd	8.4/6.0	7.2/4.8	12/9.6	5E-13
	STR•A-Q	33	Subtract Q and Store	(A) - (Q)→Y and A	St	8.4/4.8	12/	12/7.2	5E-13
	RPL•A-Y	25	Replace Subtract	(A) - Y→Y and A	Rp			18/	5E-13
	RPL•Y-Q	35	Replace Subtract Q	Y - (Q)→Y and A	Rp				5E-13
	RPL•Y-1	37	Replace Subtract 1	Y - 1→Y and A	Rp				5E-14
ADDITION	ADD•A	20	Add	(A) + Y→A	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-14
	ADD•Q	26	Q Add	(Q) + Y→Q	Rd	9.6/7.2	8.4/6.0	12/8.4	5E-15
	ENT•Y+Q	30	Add Q and Load A	(Q) + Y→Y	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-15
	STR•A+Q	32	Add Q and Store	(A) + (Q)→Y and A	St	8.4/4.8	12/		5E-15 5E-15
	RPL•A+Y	24	Replace Add	Y + (A)→Y and A	Rp			18/	5E-15
	RPL•Y+Q	34	Replace Add Q	Y + (Q)→Y and A	Rp				5E-16
	RPL•Y+1	36	Replace Add 1	Y + 1→Y and A	Rp				5E-16
MUL DIV	MUL	22	Multiply	(Q)•Y→AQ	Rd	(37.2	85.2) VARIABLE		5E-16
	DIV	23	Divide	(AQ) ÷ Y; Quot→Q, Rem→A	Rd	86.4			5E-18
LOGICAL & SELECTIVE	ENT•LP	40	Enter Logical Product	L⊂Y•(Q)⊃→A	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-27
	STR•LP	47	Store Logical Product	L⊂(A)•(Q)⊃→Y	St	8.4/4.8	12/		5E-28
	RPL•LP	44	Replace Logical Product	L⊂Y (Q)⊃→Y and A	Rp	/	/	18/12	5E-28
	ADD•LP	41	Add Logical Product	L⊂Y (Q)⊃+ (A)→A	Rd	8.4/6.0	7.2/4.8	12/7.2	5E-28
	SUB•LP	42	Subtract Logical Product	(A) - L⊂Y (Q)⊃→A	Rd				5E-28
	RPL•A+LP	45	Replace Add Logical Prod.	L⊂Y (Q)⊃+ (A)→Y and A	Rp			18/12	5E-29
	RPL•A-LP	46	Replace Subtract Log. Prod.	(A) - L⊂Y (Q)⊃→Y and A	Rp				5E-29
	SEL•SET	50	Selective Set	Set (A_n) for (Y_n) = 1	Rd	9.6/7.2	8.4/6.0	12/8.4	5E-29
	SEL•CP	51	Selective Complement	Complement (A_n) for (Y_n) = 1	Rd				5E-30
	SEL•CL	52	Selective Clear	Clear (A_n) for (Y_n) = 1	Rd				5E-30
	SEL•SU	53	Substitute	(Y_n)→(A_n) for (Q_n) = 1	Rd				5E-30
	RSE•SET	54	Replace Selective	Set (A_n) for (Y_n) = 1→Y and A	Rp			18/12	5E-30
	RSE•CP	55	Replace Selective Compl.	Complement (A_n) for (Y_n) = 1→Y and A	Rp				5E-31
	RSE•CL	56	Replace Selective Clear	Clear (A_n) for (Y_n) = 1→Y and A	Rp				5E-31
	RSE•SU	57	Replace Substitute	(Y_n)→(A_n) for (Q_n) = 1→Y and A	Rp				5E-31

NI: NEXT INSTRUCTION

CLASS: Rd = READ, St = STORE, Rp = REPLACE

*NOTE: LOWER TIME IN EACH BRACKET IS FOR REPEAT MODE TIMES

■ RIGHT SHIFT Q

This instruction shifts the contents of the Q-register to the right. The bits that are shifted off the right end are lost. The sign value (bit position 29) will be extended through the bit positions that are vacated at the left. If the shift count, contained in the low order six bits of the shift operand specified under Y, is greater than 59_{10}, an incorrect shift will result. If the shift count is equal to 29_{10}, or ranges between 29_{10} and 59_{10}, all bit positions of the Q-register will be filled with the original value of the sign position.

OPERATION	Y	J
RSH · Q	**read class**	**normal**

Examples:

RSH · Q · U (LABEL) 01020 01234

10010010111111101110000111010 1 (initial contents of Q) 001000 (shift count = 8_{10})

11111111100100101111111011100 0 (final contents of Q)

RSH · Q · 29D 01000 00035

01111111111111111111111111111111 (initial contents of Q) 011101 (shift count = 29_{10})

00000000000000000000000000000000 (final contents of Q)

■ RIGHT SHIFT A

Shifting takes place in the A-register; otherwise the functional description is the same as RIGHT SHIFT Q.

OPERATION	Y	J
RSH · A	**read class**	**normal**

Example:

RSH · A · L (LABEL) 02010 01234

■ RIGHT SHIFT AQ

This instruction shifts the contents of the A and Q-registers. Both A and Q may be considered as a single combined register, AQ, containing 60 bit positions. The shift count is specified in the lower six bits of the shift operand.

Bits that are shifted off the right end of the Q-register are lost; bits that are shifted off the right end of the A-register replace the shifted high order positions of the Q-register. The sign value (bit position 29 of the A-register) will be extended through the shifted high order positions of the A-register and into the Q-register.

- STORE Q

This instruction stores the contents of the Q-register in a storage location.

OPERATION	Y	J
STR · Q	**store class**	**normal**

Example:

STR · Q · W (LABEL) 14030 01234

- STORE A

This instruction stores the contents of the A-register in a storage location.

OPERATION	Y	J
STR · A	**store class**	**normal**

Example:

STR · A · U (LABEL) 15020 01234

- STORE Bn

This instruction stores the contents of a selected B-register in a storage location. The j-designator is used to specify the selected register; consequently, there can be no skip designation.

OPERATION	Y	J
STR · Bn	**store class**	**not used**

n may be 0, 1, 2, 3, 4, 5, 6 or 7 to specify the B-register.

Example:

STR · B7 · L (LABEL) 16710 01234

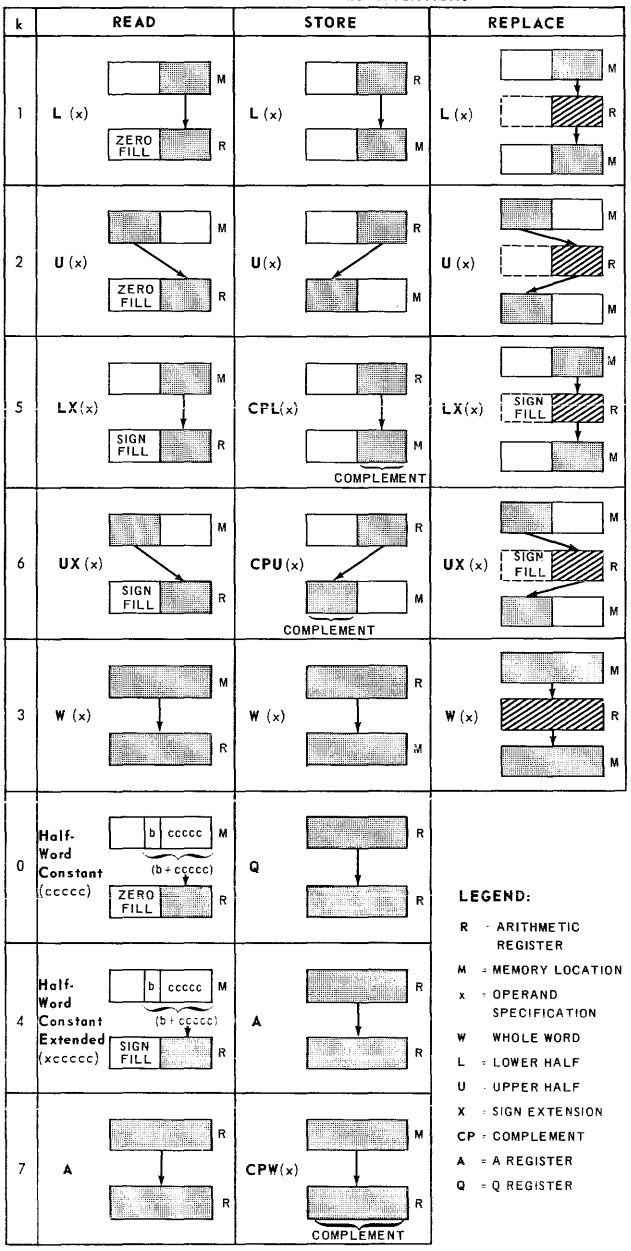
STR · B0 · L (LABEL) 16010 01234
(L(LABEL) is set to zero by the preceding instruction.)

STR · B0 · CPL (LABEL) 16050 01234
(L(LABEL) is set to all ones by the preceding instruction.)

2. Shift Instructions

Shift instructions shift the contents of a selected register to the right or left a specified number of bit positions. The shift count is contained in the six low order bit positions of a shift operand specified under Y. The maximum shift that may be specified is 73_8 or 59_{10}.

NORMAL Y OPERAND MODIFICATIONS



LEGEND:

- R = ARITHMETIC REGISTER
- M = MEMORY LOCATION
- x = OPERAND SPECIFICATION
- W = WHOLE WORD
- L = LOWER HALF
- U = UPPER HALF
- X = SIGN EXTENSION
- CP = COMPLEMENT
- A = A REGISTER
- Q = Q REGISTER

SPECIAL j-DESIGNATOR INTERPRETATION

j	COM•A≠Q≠AQ≠	ADD•Q/SUB•Q	ENT•LP,RPL•LP	DIV.	RPT	JP/RJP	JP(60)/RJP(64)
0	No skip	No skip	No skip	No skip	NE: y=y	Always jump	RIL/SIL Release/Set Interlock
1	SKIP Always skip	SKIP Always skip	SKIP Always skip	SKIP Always skip	ADY NE: y=y+1	KEY 1 Jump Key 1	RILJP/SILRJP Release/Set Interlock Jump
2	YLESS Y ≤ Q	APOS Skip (A) Pos.	EVEN Skip (A) even	NOOF Skip, no overflow	BACK NE: y=y−1	KEY 2 Jump Key 2	QPOS Jump (Q) Pos.
3	YMORE Y > Q	ANEG Skip (A) Neg	ODD Skip (A) odd	OF Skip, overflow	ADDB NE: y=y+B_b	KEY 3 Jump Key 3	QNEG Jump (Q) Neg.
4	YIN Q ≥ Y > A	QZERO Skip (Q) =0	AZERO Skip (A) =0	AZERO Skip (A) =0	R NE: y=y+(B_6)*	STOP Always stop	AZERO Jump (A)=0
5	YOUT Q < Y or Y ≤ A	QNOT Skip (Q) ≠0	ANOT Skip (A) ≠ 0	ANOT Skip (A) ≠ 0	ADVR NE: y=y+1+(B_6)*	STOP 5 Stop Key 5	ANOT Jump (A)≠0
6	YLESS Y ≤ A	QPOS Skip (Q) Pos	APOS Skip (A) Pos.	APOS Skip (A) Pos.	BACKR NE: y=y−1+(B_6)*	STOP 6 Stop Key 6	APOS Jump (A) Pos.
7	YMORE Y > A	QNEG Skip (Q) Neg.	ANEG Skip (A) Neg.	ANEG Skip (A) Neg.	ADDBR NE: y=y+$B_{\overline{b}}$+(B_6)*	STOP 7 Stop Key 7	ANEG Jump (A) Neg.

*(B_6) Increment if RPL CLASS

NORMAL j-DESIGNATOR		
	j = 0	NEXT INSTRUCTION
SKIP	j = 1	Skip NI
QPOS	j = 2	Skip NI if (Q) Pos.
QNEG	j = 3	Skip NI if (Q) Neg.
AZERO	j = 4	Skip NI if (A) = 0
ANOT	j = 5	Skip NI if (A) ≠ 0
APOS	j = 6	Skip NI if (A) Pos.
ANEG	j = 7	Skip NI if (A) Neg.

JUMP ADDRESS	
k = 0,4	Y = $\overline{y}$
k = 1,3,5	Y = (Y_L)
k = 2,6	Y = (Y_U)
k = 7	Y = (A_L)

If the shift count exceeds 59_{10}, an incorrect shift will result. If the shift count is 29_{10}, all bit positions of the A-register will contain the initial sign value. If the shift count is 59_{10}, all bit positions of both the A and Q-registers will contain the initial sign value of the A-register.

OPERATION	Y	J
RSH · AQ	**read class**	**normal**

Examples:

RSH · AQ · 12D 03000 00014

Initial contents of AQ:

(A)	(Q)
11101001111111111011010001100	00111010110101001010001100 1001

Final contents of AQ:

(A)	(Q)
11111111111111101001111111111	01101000110000111010110101 0010

RSH · AQ · U (00035 + B2) 03012 00035

The shift count will be the sum of 00035 and the contents of the B-register. The shift count will then be 59_{10}.

Initial contents of AQ:

(A)	(Q)
10111100001110101000000000 0001	11111111110000010101010101 0101

Final contents of AQ:

(A)	(Q)
11111111111111111111111111 1111	11111111111111111111111111 1111

- LEFT SHIFT Q

This instruction shifts the contents of the Q-register to the left. The bit positions shifted off the left end replace the shifted positions of the right end. If the shift count, contained in the low order six bits of the shift operand specified under Y, is greater than 59_{10}, an incorrect shift will result. If the shift count is 30_{10}, the Q-register will be restored to its initial condition.

OPERATION	Y	J
LSH · Q	**read class**	**normal**

Examples:

LSH · Q · L (LABEL + 1) 05010 01235

0011101011010100101000110010 01 (initial contents of Q) 001111 (shift count = 15_{10})

0101000110010010011101011010 10 (final contents of Q)

LSH · Q · 30D 05000 00036

0101111111000001100000100111 00 (initial contents of Q) 011110 (shift count = 30_{10})

0101111111000001100000100111 00 (final contents of Q)

- LEFT SHIFT A

Shifting takes place in the A-register; otherwise the functional description is the same as LEFT SHIFT Q.

OPERATION	Y	J
LSH · A	**read class**	**normal**

Example:

LSH · A · A 06070 00000

- LEFT SHIFT AQ

For this instruction, the A and Q-registers function as a single 60 bit register, AQ, in which high order bit positions are contained in the A-register. The bit positions shifted off the left end of the A-register replace the bit positions vacated from the right end of the Q-register. Bit positions shifted off the left end of the Q-register replace the bit positions vacated from the right end of the A-register. If the shift count exceeds 59_{10}, an incorrect shift will result. If the shift count is 30_{10}, the contents of the A-register and the Q-register will be interchanged.

OPERATION	Y	J
LSH · AQ	**read class**	**normal**

Examples:

LSH · AQ · 6D 07000 00006

Initial contents of AQ:

(A)	(Q)
111010011111111111011010001100	001110101101010010100011001001

Final contents of AQ:

(A)	(Q)
01111111111101101000110000111O	10110101001010001100100111101O

LSH · AQ · 30D 07000 00036

Initial contents of AQ:

(A)	(Q)
10111110101000110001001110110l	00000001111111101010101001100l

Final contents of AQ:

(A)	(Q)
00000001111111101010100110001	10111110101000110001001110110l

3. Comparing Instructions

- COMPARE

This instruction compares the signed value of an operand with the signed value contained in the A-register and/or the signed value contained in the Q-register. One of several skip conditions may be tested, based upon the result of the comparison.

If n represents any integer value, the comparison scale may be summarized as follows:

$+n > +0 > -0 > -n$

OPERATION	Y	J
COM · A	**read class**	**note 1**
COM · Q		
COM · AQ		

NOTE 1

One of the following must be present:

J MNEMONIC	SKIP CONDITION
YLESS	Skip the next instruction if the operand value is equal to or less than the value in the specified register.
YMORE	Skip the next instruction if the operand value is greater than the value in the specified register.
YIN (COM · AQ only)	Skip the next instruction if the operand value is less than or equal to the value in the Q-register and greater than the value in the A register. The Q and A-registers establish an upper and lower range respectively. The Y operand is within that range if a skip occurs.

YOUT
(COM · AQ only)

Skip the next instruction if the operand value is greater than the value in the Q-register or less than or equal to the value in the A register. The Q and A-registers establish an upper and lower range respectively. The Y operand is out of that range if a skip occurs.

Example:

COM · A · W (LABEL) · YMORE 04730 01234

- COMPARE MASKED

This instruction compares the contents of the A-register to a masked operand. The comparison is made by forming the logical product of the Q-register and the operand specified as Y. (See "Logical Product" under LOGICAL INSTRUCTIONS.) The result of the logical operation is substracted from the contents of the A-register to form a difference. A skip condition may be specified by the J operand depending on the results of the above operations. The result of the logical operation is then added to the A-register. There is no change in the contents of any of the operational registers as a result of this instruction.

OPERATION	Y	J
COM · MASK	**read class**	**normal**

Example:

COM · MASK · W (LABEL) · AZERO 43430 01234

10010000110011100010110111 0111 (Operand contents)

00000000000000000000000111111 (Contents of Q)

00000000000000000000000110111 (Result of logical operation)

00000000000000000000000111111 (Contents of A)

00000000000000000000000001000 (A - logical product)

The jump condition would not be present, i.e., A is not zero. The result of the logical operation is added to A to return the register to the initial condition.

4. Jump Instructions

Instructions are normally executed in sequential order. Jump instructions are used to depart from this sequential order and to specify a point in the program at which the sequential order will again be resumed. The jump may be unconditional or it may be based on various conditions.

- JUMP

A jump instruction may be unconditional or arithmetic (depending upon the contents of an arithmetic register). If the jump condition is not satisfied, control proceeds to the next sequential instruction.

OPERATION	Y	J
JP	note 1	note 2

NOTE 1

A read class operand may be specified with the following restrictions:

(1) only the forms x, L(x), U(x), W(x) are meaningful. W(x) has same result as L(x).

(2) if the A-register is specified, only the 15 low order positions will be meaningful.

(3) if a B-register is specified, no sign extension is permitted.

(4) If an actual computer address is used it cannot exceed 77777 or the decimal equivalent.

NOTE 2

The absence of a J entry indicates an unconditional jump; the remaining jump conditions are normal.

Examples:

JP · U (LABEL)	60020 01234
JP · L (LABEL) · QPOS	60210 01234
JP · LABEL · ANOT	60500 01234

- RETURN JUMP

The return jump may be unconditional or arithmetic (depending upon the contents of an arithmetic register). If an unconditional jump is specified, or if the jump condition exists, a jump is made to the address specified in Y incremented by 1, e.i., Y + 1. The address of the instruction immediately following the return jump is stored in the lower portion of the storage location at the address specified in Y. If the jump condition is not satisfied, the instruction immediately following the return jump instruction is executed.

OPERATION	Y	J
RJP	note 1	note 2

NOTE 1

See restriction under NOTE 1 of Jump Instruction.

NOTE 2

The absence of a J entry indicates an unconditional jump; the remaining jump conditions are normal.

Example:

```
RJP · LABEL                    64000 01234
```

If the RJP instruction shown above is located at address 23344, location 01234 will appear as follows:

```
LABEL                          00000 23345 (Y)
                               . . . .  (Y + 1)
```

A jump instruction could terminate the sequence of instructions and bring the program back to the instruction following the RJP instruction as follows:

```
JP · L (LABEL)
```

5. Modifying Instructions

The sequential execution of instructions may be varied by the instructions which follow. This program modification is controlled by counters contained in a B-register or in memory.

- REPEAT

This instruction causes the instruction immediately following it to be repeated the number of times specified by a 15 bit value in the Y operand. The value may be equal to or greater than zero but cannot be greater than 32767_{10}. The 15 bit value determined by the form of the Y operand is placed in B-register number 7. If the value is zero the instruction immediately following the Repeat instruction is skipped. If the value is not zero, the repeat mode which is determined by the J portion of the instruction is initiated. Any modifications to the initial instruction are performed in transient registers; the instruction as it is stored in the computer is not altered.

OPERATION	Y	J
RPT	**note 1**	**note 2**

NOTE 1

A read class operand. The form W(x) will refer to the low order 15 bits of the operand location.

NOTE 2

J MNEMONIC	REPEAT MODE
(absent)	An unmodified repeat of the next operation.
ADV	Advance the operand address of the repeated operation by an addition of 1 after each individual execution.
BACK	Decrease the operand address of the repeated operation by a subtraction of 1 after each individual execution.

ADDB — Add cumulatively the B-register indicated in the repeated operation to its operand during each execution.

R — Increase the operand of the repeated replace type operation by the content of B-register 6 for the store portion of the replace only.

ADVR — Increase the operand address of the repeated replace type operation by the content of B-register 6 for the store portion of the replace only; then increment the operand address of the repeated operation by one after each execution.

BACKR — Increase the operand address of the repeated replace type operation by the content of B-register 6 for the store portion of the replace only; then decrement the operand address of the repeated operation by one after each execution.

ADDBR — Add cumulatively the B-register indicated in the repeated replace type operation to the operand address during each execution; in addition, increase the operand of the repeated operation by the content of B6 for the store portion of the replace only.

Example:

In order to present a complete and satisfactory example, a small block of coding is presented. This coding is designed to find the square root of a number which is assumed to be in the A-register. A base value of 2 is entered into B2. The repeat mode is initiated with a value of 77777, the complement of which is 00000. The accumulative addition of the value in B2 will finally reach the value contained in the A-register +1 which will cause an ANEG skip. With each accumulative addition and execution of the instruction, 1 is subtracted from the repeat value in B7 (77777). The complement of this value is the square root of the number which is stored in the lower portion of the word labeled ANS.

```
ENT · B2 · 2
RPT · 77777 · ADDB
SUB · A · B2 + 1 · ANEG
JP  · ERROR
STR · B7 · CPL (ANS)
```

■ B SKIP

This operation tests the content of a specified B-register. If the value in the B-register is equal to the value in a memory location specified by Y the next operation is skipped and the B-register is cleared to zero. If the value in the B-register is not equal to the operand location, the normal sequence of operations continues and 1 is added to the value in the B-register.

OPERATION	Y	J
BSK · Bn	**note 1**	**not permitted**

n may be 0, 1, 2, 3, 4, 5, 6, or 7 to specify the B-register.

NOTE 1

A read class operand. The form W(x) will refer to the low order 15 bits of the operand location.

Example:

BSK · B2 · L(LABEL+3) 71213 21237

BSK · B2 · B2 71202 00000
(The previous instruction will clear B2 and skip.)

BSK · B0 · L (LABEL + 3) 71010 01237
(A skip will be performed if L (LABEL + 3) in the previous instruction equals zero.)

- B JUMP

This instruction tests the content of a specified B-register. If the value in the register is zero, the normal sequence of operations continues. If the value in the register is not zero, the register is decremented by 1 and a new sequence of operations begins at the address specified by the Y operand.

OPERATION	Y	J
BJP · Bn	**note 1**	**not permitted**

n may be 0, 1, 2, 3, 4, 5, 6, or 7 to specify the B-register.

NOTE 1

A read class operand. The form W(x) will refer to the low order 15 bits of the operand location.

Example:

BJP · B4 · W(LABEL+B2) 72432 01234

6. Subtraction

Subtraction is covered before addition, since addition actually makes use of subtraction. Subtraction in the UNIVAC 490 Central Processor is a binary subtract with an end around borrow. An end around carry means that a borrow generated in stage 29 results in a one being subtracted from stage zero of the difference. This end around borrow subtraction allows subtracting a negative number from a positive number without complementing. The following "4 bit register" example shows the end around borrow employed in subtracting negative 1 from positive 3, resulting in a +4:

```
+ 3 = 0011
- 1 = 1110
------------
      0101
         1 <-- bit from end around borrow
------------
+ 4 = 0100
```

This octal example subtracts −5 from + 24 leaving + 31:

```
+ 24 = 00024
−  5 = 77772
       00032
           1 ←end around borrow
       00031
```

The rules of algebraic subtraction will apply. Thus, if n represents any number:

+ n	− n	+ n	− n
− (− n)	− (+ n)	− (+ n)	− (− n)
+ n	− n	± n	± n

The rules for zero subtraction may be summarized as follows:

+ 0	− 0	− 0
− (− 0)	− (+ 0)	− (− 0)
+ 0	− 0	+ 0

If a number is subtracted from itself the result will be + 0.

When the difference exceeds a value that may be contained in 29 bits, these rules will not apply because of possible overflow into the sign position.

■ SUBTRACT A

This instruction subtracts an operand from the contents of the A-register and retains the difference in the A-register.

OPERATION	Y	J
SUB · A	**read class**	**normal**

Example:

SUB · A · W (LABEL) · ANOT 21530 01234

■ SUBTRACT Q

This instruction subtracts an operand from the contents of the Q-register and retains the difference in the Q-register.

OPERATION	Y	J
SUB · Q	**read class**	**note 1**

NOTE 1

QZERO and QNOT are substituted for AZERO and ANOT.

Example:

SUB · Q · 12D · QNOT 27500 00014

- ENTER Y-Q

This instruction subtracts the contents of the Q-register from the operand and retains the difference that is formed in the A-register. The contents of the Q-register are not disturbed by this instruction.

OPERATION	Y	J
ENT · Y-Q	**read class**	**normal**

Example:

ENT · Y-Q · L (LABEL) 31010 01234

- STORE A-Q

This instruction subtracts the contents of the Q-register from the A-register, retains the difference in the A-register, and stores this difference in a storage location.

OPERATION	Y	J
STR · A-Q	**store class**	**normal**

Example:

STR · A-Q · CPU (LABEL) 33060 01234

- REPLACE A-Y

This instruction subtracts an operand from the contents of the A-register, retains the difference that is formed in the A-register, and stores this sum in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · A-Y	**replace class**	**normal**

Example:

RPL · A-Y · L (LABEL) 25010 01234

- REPLACE Y-Q

This instruction subtracts the contents of the Q-register from an operand, retains the difference that is formed in the A-register, and stores this difference in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · Y-Q	**replace class**	**normal**

Example:

RPL · Y-Q · L (LABEL) 35010 01234

■ REPLACE Y-1

This instruction subtracts a binary 1 from the operand, retains the difference that is formed in the A-register, and stores this difference in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · Y-1	**replace class**	**normal**

Example:

RPL · Y-1 · UX (LABEL) 37060 01234

7. Addition

The instructions involving addition are accomplished by the same process described under SUBTRACTION. The addend is complemented and a subtraction is performed to obtain the sum.

The rules of algebraic addition will apply. Thus, if n represents any number:

− n	+ n	+ n	− n
+ (− n)	+ (+ n)	+ (− n)	+ (+ n)
− n	+ n	± n	± n

The rules for zero addition may be summarized as follows:

+ 0	− 0	+ 0
+ (+ 0)	+ (− 0)	+ (− 0)
+ 0	− 0	+ 0

If a number is added to its complement the sum will be + 0.

When the sum exceeds a value that may be contained in 29 bits, these rules will not apply because of possible overflow into the sign position.

■ ADD A

This instruction adds an operand to the contents of the A-register, and retains the sum that results in the A-register.

OPERATION	Y	J
ADD · A	**read class**	**normal**

Example:

ADD · A · 773 20000 00773

- ADD Q

This instruction adds an operand to the contents of the Q-register and retains the sum that is formed in the Q-register.

OPERATION	Y	J
ADD · Q	**read class**	**note 1**

NOTE 1

QZERO and QNOT are substituted for AZERO and ANOT.

Example:

ADD · Q · U (LABEL) · QNOT 26520 01234

- ENTER Y + Q

This instruction adds an operand to the contents of the Q-register and retains the sum that is formed in the A-register. The contents of the Q-register are undisturbed by this instruction.

OPERATION	Y	J
ENT · Y + Q	**read class**	**normal**

Example:

ENT · Y + Q · W (LABEL) 30030 01234

- STORE A + Q

This instruction adds the contents of the A and Q-registers, retains the sum in the A-register and stores the sum in a storage location.

OPERATION	Y	J
STR · A + Q	**store class**	**normal**

Example:

STR · A + Q · U (LABEL + B3) 32023 01234

- REPLACE A + Y

This instruction adds an operand to the contents of the A-register, retains the sum in the A-register, and stores this sum in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · A + Y	**replace class**	**normal**

Example:

RPL · A + Y · UX (LABEL + 3) 34060 01237

- REPLACE Y + Q

This instruction adds an operand to the contents of the Q-register, retains the sum in the A-register, and stores this sum in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · Y + Q	**replace class**	**normal**

Example:

RPL · Y + Q · W (B7) 34037 00000

- REPLACE Y + 1

This instruction adds a binary 1 to an operand, retains the sum in the A-register, and stores this sum in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · Y + 1	**replace class**	**normal**

Example:

RPL · Y + 1 · UX (LABEL + B3) 36063 01234

8. Multiplication and Division

- MULTIPLY

This instruction multiplies the contents of the Q-register by the operand specified in the instruction. The product is formed in the 60 bit positions of the combined AQ-register.

Multiplication is performed with positive numbers. If a multiplication involves any negative numbers, they are made positive by complementing them prior to performing the multiplication. After the positive product is formed in the AQ-register, the sign of this product is corrected by complementing the contents of the AQ-register, if one, but not both, of the original numbers was negative.

The following rules apply for multiplication:

- If a positive number is multiplied by a positive number or a negative number by a negative number, the product will be positive.
- If a positive number is multiplied by a negative number or a negative number by a positive number, the product will be negative.
- If positive 0 is multiplied by positive 0 or negative 0 by negative 0, the product will be positive 0.
- If positive 0 is multiplied by negative 0 or negative 0 by positive 0, the product will be negative 0.

No product can be generated which will overflow AQ. The maximum positive product is:

17777 77777	00000 00001
A	Q

The maximum negative product is:

60000 00000	77777 77776
A	Q

OPERATION	Y	J
MUL	**read class**	**note 1**

NOTE 1

The skip condition is tested prior to any final sign conversion. The significance of the normal skip condition for a multiply operation may be outlined as follows:

J MNEMONIC	SKIP CONDITION
(absent)	No skip.
SKIP	Skip next instruction.
QPOS	Skip next instruction if Q is + or + 0 prior to final sign correction. If a skip does not occur, a double length product is indicated since there is a significant bit in bit position 29 of the Q-register.
QNEG	Skip next instruction if Q is – or – 0 prior to final sign correction. If a skip occurs, a double length product is indicated since there is a significant bit in bit position 29 of the Q-register.
AZERO	Skip next instruction if A is + 0 prior to final sign correction. If a skip occurs, it indicates that the product has 30 or less significant bits, and that the A-register contains only sign bits. This does not mean the Q-register contains the correct product, since bit position 29 of the Q-register may contain a significant bit of the product, thus making bit position 0 of the A-register the first sign bit. If a skip does not occur, it indicates that significant bits of the product are in the A-register.
ANOT	Skip next instruction if A is not + 0 prior to final sign correction. If a skip occurs, it indicates that significant bits of the product are in the A-register. If a skip does not occur, it indicates the same condition that exists when a skip occurs with AZERO.

APOS	Skip next instruction if A is + or + 0 prior to final sign correction. A skip should always occur since A should always be a positive number prior to final sign correction.
ANEG	Skip next instruction if A is – or – 0 prior to final sign correction. A skip should never occur since A should never be a negative number prior to final sign correction.

Examples:

MUL ' W (LABEL) 22070 01234

The result of operations for various values contained in the Q-register and the A-register (initially the location defined by LABEL) follows:

A		Q		AQ
00000 00012	X	00000 00010	=	00000 00000 00000 00120
77777 77767	X	00000 00012	=	77777 77777 77777 77657
77777 77765	X	77777 77767	=	00000 00000 00000 00120

- DIVIDE

This instruction divides the contents of the combined AQ-register by the operand specified in the instruction and retains the quotient and remainder that are formed in the Q and A-registers respectively.

Division is performed with positive numbers. If a division involves any negative numbers, they are made positive by complementing them prior to performing the division. After the positive quotient and remainder are formed in the Q and A-register respectively, the signs of the quotient and remainder are corrected by complementing the contents of the Q and A-registers if one, but not both, of the original numbers was negative.

The following rules apply for division:

- If a positive number is divided by a positive number or a negative number by a negative number, the quotient and remainder will be positive numbers.
- If a positive number is divided by a negative number or a negative number by a positive number, the quotient and remainder will be negative numbers.

Negative Zero Quotients and Remainders

Division, if handled improperly, may generate a negative 0 quotient or remainder that can have an adverse affect on further calculations. This situation can occur in the following cases:

(1) Even Division Where the Dividend and Divisor have Unlike Signs and are Non-Zero

The result of such a division is that the correct quotient will be in the Q-register and the remainder in the A-register will be a negative 0.

When the interpretation of the j-designator is made, the Q-register will contain the absolute value of the quotient and the A-register will contain a positive 0. For example:

	000000000000000000000000000000	000000000000000001010011100101	(dividend)
		111111111111111110101100011010	(divisor)
At j interpretation		000000000000000000000000000010	(quotient in the Q-register)
		000000000000000000000000000000	(remainder in the A-register)
Final Result		111111111111111111111111111101	(quotient in the Q-register)
		111111111111111111111111111111	(remainder in the A-register)

(2) Division where the Absolute Value of the Divisor is Greater than the Absolute Value of Dividend, the Signs are Unlike, and Both are Non-Zero

When division is performed in this case, the quotient in the Q-register will be a negative 0 and the remainder in the A-register will be the complement of the absolute value of the dividend. At the time the interpretation of the j-designator is made, the Q-register will contain a positive 0 and the A-register will contain the absolute value of the dividend. For example:

	000000000000000000000000000000	000000000000000000000000000011	(dividend)
		111111111111111111111111111010	(divisor)
At j interpretation		000000000000000000000000000000	(quotient in the Q-register)
		000000000000000000000000000011	(remainder in the A-register)
Final Result		111111111111111111111111111111	(quotient in the Q-register)
		111111111111111111111111111100	(remainder in the A-register)

(3) Division by Positive or Negative Zero

The following rules apply in these cases:

- If a positive number is divided by positive 0, the quotient in the Q-register will be a negative 0 and the remainder in the A-register will be the initial contents of the Q-register. At the time interpretation of the j-designator is made, the contents of the Q and A-registers will be the same as the final contents.

- If a negative number is divided by positive 0, the quotient in the Q-register will be a positive 0 and the remainder in the A-register will be the initial contents of the Q-register. At the time interpretation of the j-designator is made, the Q-register will contain a negative 0 and the A-register will contain the complement of the initial contents of the Q-register.
- If a positive number is divided by negative 0, the quotient in the Q-register will be a positive 0 and the remainder in the A-register will be the complement of the initial contents of the Q-register. At the time the j-designator is interpreted, the Q-register will contain a negative 0 and the A-register will contain the initial contents of the Q-register.
- If a negative number is divided by negative 0, the quotient in the Q-register will be a negative 0 and the remainder in the A-register will be the complement of the initial contents of the Q-register. At the time the j-designator is interpreted, the contents of the Q and A-registers will be the same as the final contents.

The following examples illustrate these rules:

A positive number divided by positive 0.

00000000000000000000000000001	00000000000000000000000000001	(dividend)
	00000000000000000000000000000	(divisor)
At j interpretation	11111111111111111111111111111	(quotient in the Q-register)
	00000000000000000000000000001	(remainder in the A-register)
Final Result	11111111111111111111111111111	(quotient in the Q-register)
	00000000000000000000000000001	(remainder in the A-register)

A negative number divided by positive 0.

11111111111111111111111111111	11111111111111111111111111110	(dividend)
	00000000000000000000000000000	(divisor)
At j interpretation	11111111111111111111111111111	(quotient in the Q-register)
	00000000000000000000000000001	(remainder in the A-register)
Final Result	00000000000000000000000000000	(quotient in the Q-register)
	11111111111111111111111111111	(remainder in the A-register)

A positive number divided by negative 0.

01011111111111111111111111111111	11011111111111111111111111111111	(dividend)
	11111111111111111111111111111111	(divisor)
	11111111111111111111111111111111	(quotient in the Q-register)
At j interpretation	11011111111111111111111111111111	(remainder in the A-register)
	00000000000000000000000000000000	(quotient in the Q-register)
Final Result	00100000000000000000000000000000	(remainder in the A-register)

A negative number divided by negative 0.

10011111111111111111111111111111	11011111111111111111111111111111	(dividend)
	11111111111111111111111111111111	(divisor)
	11111111111111111111111111111111	(quotient in the Q-register)
At j interpretation	00100000000000000000000000000000	(remainder in the A register)
	11111111111111111111111111111111	(quotient in the Q-register)
Final Result	00100000000000000000000000000000	(remainder in the A-register)

(4) Division of Positive or Negative Zero by a Non-Zero Divisor with an Unlike Sign

When division is performed in this case, the quotient in the Q-register and the remainder in the A-register will be a negative 0. At the time interpretation of the j-designator is made both the Q and A-register will contain a positive 0. The following examples will illustrate this:

00000000000000000000000000000000	00000000000000000000000000000000	(dividend)
	11111111111111111111111111111110	(divisor)
	00000000000000000000000000000000	(quotient in the Q-register)
At j interpretation	00000000000000000000000000000000	(remainder in the A-register)
	11111111111111111111111111111111	(quotient in the Q-register)
Final Result	11111111111111111111111111111111	(remainder in the A-register)

11111111111111111111111111111111	11111111111111111111111111111111	(dividend)
	00000000000000000000000000000001	(divisor)
	00000000000000000000000000000000	(quotient in the Q-register)
At j interpretation	00000000000000000000000000000000	(remainder in the A-register)
	11111111111111111111111111111111	(quotient in the Q-register)
Final Result	11111111111111111111111111111111	(remainder in the A-register)

Divide Overflow with Non-Zero Divisor and Dividend

In division, the dividend in the AQ-register may have up to 59 significant bits while the divisor may have as few as 1. In these cases, a quotient may be generated that has as many as 59 significant bits. Since the Q-register has a 30-bit capacity, an overflow situation will result when a quotient is generated that has more than 29 significant bits. If overflow does occur, the quotient in the Q-register will be positive 0 if the divisor and dividend have unlike signs, or it will be a negative 0 if the signs were the same. At the time the j-designator is interpreted the Q-register will always contain a negative 0.

The following rules govern the occurrence of a divide overflow:

- If the most significant bit of the divisor is in bit position n, a divide overflow will not occur if the dividend has no significant bits beyond bit position n + 28.
- If the most significant bit of the divisor is in bit position n, a divide overflow will occur if the dividend has a significant bit in bit position n + 30 or beyond.
- If the most significant bit of the divisor is in bit position n, a divide overflow may occur if the most significant bit of the dividend is in bit position n + 29.

The following examples illustrate these rules:

- No overflow.

00000000000000000000000000000000	00000000000000000101100010111100	(dividend)
	00000000000000000000001010011100	(divisor)
	00000000000000000000000000010001	(quotient in the Q-register)
At j interpretation	00000000000000000000000000000000	(remainder in the A-register)
	00000000000000000000000000010001	(quotient in the Q-register)
Final Result	00000000000000000000000000000000	(remainder in the A-register)

- Overflow occurs.

000000000000000100011001100011		010011110111100111100110101011	(dividend)
		000000000000000010000001000000	(divisor)
At j interpretation	{	111111111111111111111111111111	(quotient in the Q-register)
		010011110111100001100111101011	(remainder in the A-register)
Final Result	{	111111111111111111111111111111	(quotient in the Q-register
		010011110111100001100111101011	(remainder in the A-register)

- Overflow may occur.

111111111111111011100101011100		101100001000011010001100111111	(dividend)
		000000000000000010000001000000	(divisor)
At j interpretation	{	111111111111111111111111111111	(quotient in the Q-register)
		010011110111100111110011000000	(remainder in the A-register)
Final Result	{	000000000000000000000000000000	(quotient in the Q-register)
		101100001000011000001100111111	(remainder in the A-register)

In this example, overflow occurs.

000000000000000000000000000001		000000000000000000000000000000	(dividend)
		000000000000000000000000000011	(divisor)
At j interpretation	{	000010001000100010000111101101	(quotient in the Q-register)
		000000000000000000000000000001	(remainder in the A-register)
Final Result	{	000010001000100010000111101101	(quotient in the Q-register)
		000000000000000000000000000001	(remainder in the A-register)

In this example, overflow does not occur.

The remainder in overflow division is difficult to determine and the value of such information, when obtained, is questionable. The rules that are stated below are valid at least in the above examples. They should not, however, be considered universal rules.

(1) If the dividend and divisor are positive numbers, add the dividend and divisor. The remainder in the A-register will be the low-order 30 bits of the sum that is formed. At the time the j-designator is interpreted, the contents of the A-register will be the same as the final contents.

(2) If the dividend and divisor are negative numbers, complement the dividend and divisor, and then add them. The remainder in the A-register will be the low-order 30 bits of the sum that is formed. At the time interpretation of the j-designator is made, the contents of the A-register will be the same as the final contents.

(3) If the dividend is a positive number and the divisor is a negative number, the divisor should be complemented and then added to the dividend. The final remainder in the A-register will be the complement of the low-order 30 bits of the sum that is formed. At the time interpretation of the j-designator is made, the contents of the A-register will be the low-order 30 bits of the sum that is formed.

(4) If the dividend is a negative number and the divisor is a positive number the dividend should be complemented and then added to the divisor. The final remainder in the A-register will be the complement of the low-order 30 bits of the sum that is formed. At the time the j-designator is interpreted, the contents of the A-register will be the low-order 30 bits of the sum that is formed.

OPERATION	Y	J
DIV	read class	note 1

NOTE 1

The interpretation of the skip condition for a divide operation is as follows:

J MNEMONIC	SKIP CONDITION
(absent)	No skip.
SKIP	Skip next instruction.
OF	Skip if there is an overflow condition. The instruction to which the skip is made should be a jump instruction which will direct the program to a routine which provides remedial means of noting and/or correcting the error.
NOOF	Skip if there is no overflow condition. In this case, a correct answer is indicated when a skip occurs.
AZERO	Skip if A is zero.

ANOT	Skip if A is not zero.
APOS	Skip if A is positive.
ANEG	Skip if A is negative.

Examples:

Examples are for normal division, where all results are shown following final sign correction, if correction is required.

(1) 00000 00000 00000 26134 ÷ 00000 01234
quotient in Q = 00000 00021
remainder in A = 00000 00000
no final sign correction on A or Q

(2) 00000 00000 00000 26152 ÷ 00000 01234
quotient in Q = 00000 00021
remainder in A = 00000 00016
no final sign correction on A or Q

(3) 7777777 7777777 7777777 51625 ÷ 77777 76543
quotient in Q = 00000 00021
remainder in A = 00000 00016
no final sign correction on A or Q

(4) 02400 21166 21233 52654 = 54000 16354
quotient in Q = 67777 03046
remainder in A = 54733 20156
A and Q were the complements at j interpretation

(5) 75377 56611 56544 25123 ÷ 23777 61423
quotient in Q = 67777 03046
remainder in A = 54733 20156

(6) 00000 00000 00000 12345 ÷ 77777 65432
quotient in Q = 77777 77776
remainder in A = 77777 77777 (even division)
A and Q were the complements at j interpretation

9. Logical and Selective Instructions

The instructions contained in this group are based upon five operations: logical product, selective set, selective clear, selective complement and selective substitute. The individual bit positions of the operands are involved in these operations. The condition of a specific bit position in the result is determined by the condition of the corresponding bit positions in the operands.

- LOGICAL PRODUCT

Logical product operations are generally used for masking, which causes selected portions of the operand to be reproduced in the result. The remaining portions of the operand are assumed to be zero. This is accomplished by placing a mask in the Q-register consisting of 1 bits in the desired bit positions and 0 in the remaining bit positions. A simple example of all the possible combinations is:

```
1010  (Y)
0110  (Q)
----
0010  (logical product)
```

- SELECTIVE SET

Selective set operations are used to force 1 bits into selected bit positions of the A-register. The bit positions selected to be filled with 1 bits are determined by the condition of the bit positions of the operand. If either or both the corresponding bit positions of the A-register and the operand contain a 1 bit, the result will be a 1 bit. A simple example of all the possible combinations is:

```
1010  (A)
0110  (Y)
----
1110  (selective set of A)
```

- SELECTIVE CLEAR

Selective clear operations are used to clear selective bit positions of the A-register to 0. The bit positions selected to be cleared to 0 are determined by the condition of the bit positions of the operand. If a bit position of the operand contains a 1 bit, the result will be 0. A simple example of all the possible combinations is:

```
1010  (A)
0110  (Y)
----
1000  (selective clear of A)
```

- SELECTIVE COMPLEMENT

Selective complement operations are used to complement the bits in selected bit positions of the A-register. The bit positions to be complemented are determined by the condition of the bit positions of the operand. If a bit position of the operand contains a 1 bit, the corresponding bit position of the A-register will be complemented. A simple example of all the possible combinations is:

```
1010  (A)
0110  (Y)
----
1100  (selective complement of A)
```

■ SELECTIVE SUBSTITUTE

Selective substitute operations are used for replacing bits in selected bit positions of the A-register with bits from the corresponding bit positions of an operand. The bits of the operand that will replace those in the A-register are determined by placing 1 bits in the Q-register.

The first twelve bit positions of the various registers involved in this operation will be used to show how this is accomplished:

(1) The individual bits of the A-register are cleared corresponding to 1 bits in the Q-register.

```
010101010101  (initial contents of A)
000000111111  (Q)
------------
010101000000  (result in A)
```

(2) The logical product of the Y operand and Q are formed in the X-register.

```
111000000111  (Y)
000000111111  (Q)
------------
000000000111  (result in X)
```

(3) The individual bit positions of A are then set to 1 corresponding to one bits in the X-register.

```
010101000000  (A)
000000000111  (X)
------------
010101000111  (final result)
```

■ ENTER LOGICAL PRODUCT

This instruction forms the logical product of the contents of the Q-register and an operand and retains it in the A-register.

OPERATION	Y	J
ENT · LP	**read class**	**note 1**

NOTE 1

QPOS and QNEG are replaced with EVEN or ODD, which causes a skip if the parity (number of 1 bits) of A is respectively even or odd.

Example:

```
ENT · LP · W(LABEL) · EVEN          40230 01234
```

- STORE LOGICAL PRODUCT

This instruction forms the logical product of the contents of the Q-register and the A-register and stores this product in a storage location.

OPERATION	Y	J
STR · LP	**store class**	**normal**

Example:

STR · LP · L(LABEL + B5) 47015 01234

- REPLACE LOGICAL PRODUCT

This instruction forms the logical product of the contents of the Q-register and an operand, retains the logical product in the A-register, and stores this logical product in the storage location from which the operand was obtained.

OPERATION	Y	J
RPL · LP	**replace class**	**note 1**

NOTE 1

QPOS and QNEG are replaced with EVEN or ODD which causes a skip if the parity (number of 1 bits) of A is respectively even or odd.

Example:

RPL · LP · W(LABEL) · ODD 44330 01234

- ADD LOGICAL PRODUCT

This instruction adds the contents of the A-register to the logical product of the contents of the Q-register and an operand. The sum is retained in the A-register.

OPERATION	Y	J
ADD · LP	**read class**	**normal**

Example:

ADD · LP · X77773 41040 77773

- SUBTRACT LOGICAL PRODUCT

This instruction subtracts the logical product of the contents of the Q-register and an operand from the contents of the A-register. The difference is retained in the A-register.

OPERATION	Y	J
SUB · LP	**read class**	**normal**

Example:

SUB · LP · W(LABEL) · QPOS 42230 01234

- REPLACE A + LOGICAL PRODUCT

This instruction forms the logical product of the contents of the Q-register and an operand, then adds this product to the contents of the A-register. The sum is retained in the A-register and is stored in the location from which the operand was obtained.

OPERATION	Y	J
RPL · A+LP	**replace class**	**normal**

Example:

RPL · A + LP · LX(LABEL + B4) 45054 01234

- REPLACE A - LOGICAL PRODUCT

This instruction forms the logical product of the contents of the Q-register and an operand, then subtracts this product from the contents of the A-register. The difference is retained in the A-register and is stored in the location from which the operand was obtained.

OPERATION	Y	J
RPL · A-LP	**replace class**	**normal**

Example:

RPL · A - LP · UX(LABEL + B3) 46063 01234

- SELECTIVE SET

This instruction forces 1 bits into selected bit positions of the A-register. The bit positions that 1 bits are forced into are determined by the corresponding bit positions in the operand. If either or both bit positions contain a 1 bit, the result will be a 1 bit. If both positions contain a zero, the result will be zero.

OPERATION	Y	J
SEL · SET	**read class**	**normal**

Example:

SEL · SET · W(LABEL) 50030 01234

- SELECTIVE COMPLEMENT

This instruction complements the contents of selected bit positions of the A-register. The bit positions that are to be complemented are determined by the presence of 1 bits in the corresponding bit positions of the operand. If a bit position of the operand contains a 1 bit, the corresponding bit position of the A-register will be complemented.

OPERATION	Y	J
SEL · CP	**read class**	**normal**

Example:

SEL · CP · 77777 51000 77777

- SELECTIVE CLEAR

This instruction will clear selected bit positions of the A-register to zero. A 1 bit in the operand bit position clears the corresponding bit position in the A-register.

OPERATION	Y	J
SEL · CL	**read class**	**normal**

Example:

SEL · CL · W(LABEL + B6) 52036 01234

- SELECTIVE SUBSTITUTE

This instruction replaces the contents of selected bit positions of the A-register with the content of corresponding bit positions in an operand. The bit positions selected for replacement are determined by the presence of 1 bits in the corresponding bit positions of the Q-register.

OPERATION	Y	J
SEL · SU	**read class**	**normal**

Example:

SEL · SU · X11000 53040 11000

- REPLACE SELECTIVE SET

This instruction forces 1 bits into selected bit positions of the A-register. The bit positions that 1 bits are forced into are determined by placing 1 bits into the corresponding bit positions of the operand. After the selective set operation is performed, the result is retained in the A-register, and is also stored in the location from which the operand was obtained.

OPERATION	Y	J
RSE · SET	replace class	normal

Example:

RSE · SET · UX(LABEL + B1) 54061 01234

- REPLACE SELECTIVE COMPLEMENT

This instruction complements the contents of selected bit positions of the A-register. The bit positions selected to be complemented are determined by the presence of 1 bits in the corresponding bit positions of the operand. After the selective complement operation is performed, the result is retained in the A-register and is also stored in the location from which the operand was obtained.

OPERATION	Y	J
RSE · CP	replace class	normal

Example:

RSE · CP · U(LABEL) 55020 01234

- REPLACE SELECTIVE CLEAR

This instruction clears selected bit positions of the A-register to zero. The bit positions that are cleared to zero are determined by the presence of 1 bits in the corresponding bit positions of the operand. After the selective clear operation is performed, the result is retained in the A-register and is also stored in the storage location from which the operand was obtained.

OPERATION	Y	J
RSE · CL	replace class	normal

Example:

RSE · CL · L(LABEL) · SKIP 56110 01234

- REPLACE SELECTIVE SUBSTITUTE

This instruction replaces the contents of selected bit positions of the A-register with the contents of selected bit positions of the operand. The bit positions that will be replaced are determined by placing 1 bits in the corresponding bit positions of the Q-register. After the selective substitute operation is performed, the result is retained in the A-register, and is also stored in the location from which the operand was obtained.

OPERATION	Y	J
RSE · SU	**replace class**	**normal**

Example:

RSE · SU · LX(LABEL) 57050 01234

Example:

The uses of the logical and selective instructions are varied. Individual examples become meaningless unless seen in a context. The portion of coding presented below is designed to add two numbers in Fieldata code to produce a sum in Fieldata code.

```
MASK    6060606060
 ADJ    5252525252
        ENT · Q · W(FD1)        (1)
        SUB · Q · W(ADJ)        (2)
        ADD · Q · W(FD2)        (3)
        ENT · LP · W(MASK)      (4)
        LSH · AQ · 30D          (5)
        LSH · Q · 27D           (6)
        SEL · SET · W(MASK)     (7)
        STR · A-Q · W(FDSUM)    (8)
```

Let us assume that the two Fieldata numbers to be added are 12345 and 12345 (FD1 and FD2). The result will be 24690 (FDSUM). The following operations are performed:

(1) The first number is entered in the Q-register.

(2) An adjusting value (ADJ) is subtracted from this number and

(3) The second number is added to the first number as follows:

```
6162636465  (FD1, in Fieldata)
5252525252  (ADJ)
----------
0710111213
6162636465  (FD2, in Fieldata)
----------
7072747700  (Contents of Q-register)
```

(4) The logical product of the contents of the Q-register and the operand MASK is formed and entered in the A-register as follows:

```
111000  111010  111100  111111  000000  (Q)
110000  110000  110000  110000  110000  (MASK)
------------------------------------------
110000  110000  110000  110000  000000  (A)
```

(5) The contents of A and Q are interchanged by a 30 bit left shift of AQ,

(6) and the resulting contents of Q are shifted to account for a carry from one Fieldata digit to the next. The result of these operations appears in Q as follows:

000110 000110 000110 000110 000000

(7) A selective set operation and

(8) a storage of A-Q results in the sum of the two Fieldata numbers as follows:

111000	111010	111100	111111	000000	(A)
110000	110000	110000	110000	110000	(MASK)
111000	111010	111100	111111	110000	(A)
000110	000110	000110	000110	000000	(Q)
110010	110100	110110	111001	110000	(FDSUM)
62	64	66	71	60	(24690 in Fieldata)

10. Miscellaneous Instructions

The following instructions are variations of a basic computer instruction. They are used frequently for such operations as clearing portions of memory and complementing values. For this reason, a separate mnemonic operation code is assigned to each of these operations.

- COMPLEMENT A

This is a variation of the STORE A instruction. The contents of the A-register will be complemented as a result of this operation.

OPERATION	Y	J
CP · A	**none**	**normal**

Example:

CP · A 15070 00000

- COMPLEMENT Q

This is a variation of the STORE Q instruction. The contents of the Q-register will be complemented as a result of this operation.

OPERATION	Y	J
CP · Q	**none**	**normal**

Example:

CP · Q 14000 00000

- CLEAR A

This is a variation of the ENTER A instruction. The contents of the A-register will be cleared to zero by this operation.

OPERATION	Y	J
CL · A	**none**	**normal**

Example:

CL · A 11000 00000

- CLEAR Q

This is a variation of the ENTER Q instruction. The contents of the Q-register will be cleared to zero by this operation.

OPERATION	Y	J
CL · Q	**none**	**normal**

Example:

CL · Q 10000 00000

- CLEAR Bn

This is a variation of the ENTER Bj instruction. The contents of the specified B-register is cleared to zero. One of the B-registers is specified by a value of 0, 1, 2, 3, 4, 5, or 7 for n.

OPERATION	Y	J
CL ·Bn	**none**	**none**

Example:

CL · B6 12600 00000

- CLEAR Y

This is a variation of the STORE Bn instruction. The content of the storage location specified by Y is cleared to zero.

OPERATION	Y	J
CL	**store class**	**none**

Example:

CL · L (LABEL + B5) 16015 01234

CL · CPL (LABEL) 16050 01234

- NO OPERATION

  This instruction is a variation of the ENTER Bn instruction. No operation is performed and the program advances to the next instruction.

OPERATION	Y	J
NO-OP	**none**	**none**

Example:

```
NO-OP                                12000 00000
```

11. Assembler Macro-Operations

A group of instructions are often used to perform some common operation. The SPURT assembler provides a convenient format in which one line of SPURT coding will generate all the instructions required to complete the operation. Operations of this type will be called macro-operations. The line of coding will consist of a standard label, the appropriate macro-operation mnemonic, and a variable number of operands to complete and modify the individual computer instructions that are generated.

If reference to a macro-operation is required, the label will refer to the first computer instruction that is generated.

If a macro-operation is preceded by an instruction with a skip condition, a next instruction skip will skip only the first computer instruction that has been generated.

- ENTRY and EXIT

  The ENTRY and EXIT mnemonics provide a standard means of starting and exiting from all subroutines. The ENTRY mnemonic is the first line of the subroutine and should therefore have a label which gives the subroutine a name. The entrance to the subroutine will be by a return jump instruction. The return address will be placed in the lower portion of the computer instruction generated by the ENTRY mnemonic.

  An EXIT mnemonic should appear at any point where an exit from the subroutine is desired. More than one EXIT may be used if there are alternate paths in the subroutine. The computer instruction generated will refer back to the return address that was placed in the computer instruction associated with the ENTRY mnemonic.

  Example:

```
                                      Computer        Computer
1                  w                  Address         Instruction

CHKINPUT           ENTRY              02244           61000 00000

   (subroutine instructions)

                   EXIT               02277           61010 02244
```

The subroutine may be entered by a return jump instruction which will appear as follows:

```
RJP · CHKINPUT      04322      65000 02244
(return address)    04323
```

After the return jump is executed coding for the ENTRY line will appear as follows:

```
                    02244      61000 04323
```

- CLEAR

This operation clears a specified number of words in computer memory.

w	v_0	v_1
CLEAR	number of words (read class)	starting address (store class)

v_0 Specifies the number of words to be cleared. This is a read class operand. If a value of zero is specified, the operation will accomplish nothing except a time delay.

v_1 specifies the first address to be cleared. This may be a label with either or both an increment and B-register modification.

Example:

```
CLEAR · 36 · LABEL      70100 00036
                        16030 01234
```

Timing considerations are involved in the coding for this operation.

- RESERVE

This operation reserves a block of memory word locations in the final object program. The uses for such an area include:

- setting aside a specific area for the storage of parameters
- the provision of an area for working storage
- the reservation of space for program expansion.

w	v_0
RESERVE	number of words

v_0 specifies the number of memory words to be reserved. Only a constant may be used in this operand location.

Example:

l	w	v_0
TABLE	RESERVE	25D

The reservation of space begins at the location following that of the last previously generated instruction. If the reserved space is 26 words or less, all words will be zero filled; if greater than 26 words, only the first word may be assumed to be zero.

The label assigned to the RESERVE mnemonic will reference the first word of the area. Other words in the area may be referred to by an increment, such as TABLE + 5, which refers to the sixth word of the area.

- PUT

The PUT operation places a single word in a designated storage address through the Q register.

w	v_0	v_1
PUT	word	destination address

v_0 A read class operand.

v_1 A store class operand.

Example:

```
PUT · 1 · L(LABEL)          10000 00001
                            14010 01234
```

- MOVE

The MOVE operation moves a block of data from one area to another. The contents of the Q-register and B7 will be affected by the instruction.

w	v_0	v_1	v_2
MOVE	number of words	from address	to address

v_0 a read class operand which specifies the number of words to be moved.

v_1 a read class operand which specifies the initial address of the area from which data will be moved. The form W(x) will refer to the lower portion of the word.

v_2 a read class operand which specifies the initial address of the area to which data will be moved. The form W(x) will refer to the lower portion of the word.

Example:

SPURT INSTRUCTION	COMPUTER ADDRESS	COMPUTER INSTRUCTION
MOVE · L(B4+12) · B1 · B5	05300	10001 00000
	05301	14010 05306
	05302	10005 00000
	05303	14010 05307
	05304	12714 00012
	05305	61000 05310
	05306	10037 00000
	05307	14037 00000
	05310	72700 05306

■ U-TAG

The U-TAG operation provides the programmer with a means by which the value represented by a label may be placed in the upper portion of a word. A label may also be placed in the lower portion of the word. This operation is useful for such purposes as the preparation of jump tables and the specification of upper and lower buffering limits.

w	v_0	v_1
U-TAG	upper label	lower label

v_0 may be any valid label. A constant is not permitted.

v_1 may be any valid label. If no label is desired, this must be 0 or some other constant not exceeding 77777.

Example:

Assume that the label ENDFILE has been assigned the computer address 00227. The label LABEL will have its usual assignment of 01234.

U-TAG · ENDFILE · LABEL 00227 01234

- FD (Fieldata)

  The FD operation will enter an alphanumeric literal into a program in Fieldata code. This operation is particularly useful in forming lists or tables of alphanumeric information.

w	v_0	v_1
FD	number of words occupied	literal

  v_0 specifies the number of computer words that the literal should occupy. The literal will be justified left in all cases. An octal or decimal value may be used. If the stated value is zero, the literal will automatically occupy computer words at the rate of five characters per word. If the stated number of words is greater than the number the literal would normally occupy, the remainder of the character positions will be packed with Master space codes (00_8). This permits variable size literals to each occupy the same amount of space to facilitate the searching of lists or tables.

  v_1 contains the literal which may consist of alphanumeric Fieldata characters. The literal will be placed in the program following the last word generated by the previous operation. Each character will occupy six binary bits with a maximum of five characters per word. A numeric zero and the letter "O" will both produce the letter "O" (24_8). Spaces between words will generate Fieldata spaces (05_8). See APPENDIX B, Table 1 for clarification of the use of special characters. An end of statement symbol is critical following the last character to avoid extraneous space fill.

  Examples:

```
(1) FD · 3 · ENDΔOFΔFILE        12231 10524
                                13051 31621
                                16000 00000

(2) FD · 3 · NEWΔYORK           23123 40536
                                24272 00000
                                00000 00000
```

- The following operations will generate more than one instruction word. They are associated with a particular context and will be fully described in that context:

  FORM-TEXT

  CKSTAT

  TYPEC

  TYPET

12. Basic Input/Output Instructions

When standard peripheral channels are shared between programs in an environment controlled by the Real-Time Executive Routine (REX), input/output operations must be requested by use of the mnemonic statements described in Section 6.

Non-standard peripheral input/output operations, and standard peripheral input/output operations for programs not controlled by REX, must be requested by using the basic input/output instructions that are here described.

a. Input/Output Instruction Word

The format for the input/output instruction word is shown in Figure 5-5.

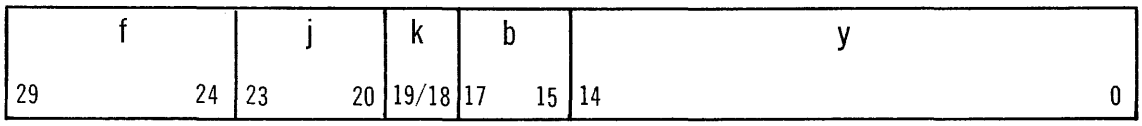


Figure 5-5. Input/Output Instruction Word

f The function code designator, f, is a 6-bit code that specifies the operation to be performed.

y The operand designator, y, is a 15-bit code that represents either the operand or the operand address.

j The channel designator, j, is a 4-bit code that specifies the input or output channel that the instruction refers to.

k The operand-interpretation designator, k, is a 2-bit code that controls where the operand is procured from or where it is stored, or both.

b The operand address modification designator, b, a 3-bit code, specifies the B-register containing a quantity that is added to y.

j k COMBINATIONS

As shown in Figure 5-5, j and k together occupy the same bit positions as j and k in all other instructions; however, j and k are 4 bits and 2 bits as opposed to 3 bits and 3 bits for j and k in other operations. When input/output instructions are written, the 6-bits that represent the j k combination appear as two octal digits as do the 6-bits representing j k in all other instructions, these octal digits are considered as a unit that represents a specific j k combination rather than having one digit represent j and one represent k. The octal digits that represent the j k combination are shown at the intersections of the j value rows and k value columns in Figure 5-6.

For example, assume that an input/output instruction is to be written with the requirement that j = 5 and k = 3. An examination of the intersection of the j = 5 row and the k = 3 column in the diagram will show that 27 is the 2-digit octal combination that meets this requirement.

	k = 0	k = 1	k = 2	k = 3
j = 0	00	01	02	03
j = 1	04	05	06	07
j = 2	10	11	12	13
j = 3	14	15	16	17
j = 4	20	21	22	23
j = 5	24	25	26	27
j = 6	30	31	32	33
j = 7	34	35	36	37
$8_D = 10_8$	40	41	42	43
$9_D = 11_8$	44	45	46	47
$10_D = 12_8$	50	51	52	53
$11_D = 13_8$	54	55	56	57
$12_D = 14_8$	60	61	62	63
$13_D = 15_8$	64	65	66	67

Figure 5-6. Combination for Input/Output Instructions

b. Input/Output Buffers

An input buffer is a block of consecutive storage locations into which a peripheral subsystem, connected to an input channel, places data. Conversely, an output buffer is a block of consecutive storage locations from which a peripheral subsystem, connected to an output channel, receives data. The assignment of the buffers is made by the buffer instructions (73, 74, 75, and 76). These instructions activate a buffer and place a control word in the appropriate buffer-control register. The control word contains two addresses that define the size and location of the buffer. Figure 5-7 shows the format of the control word.

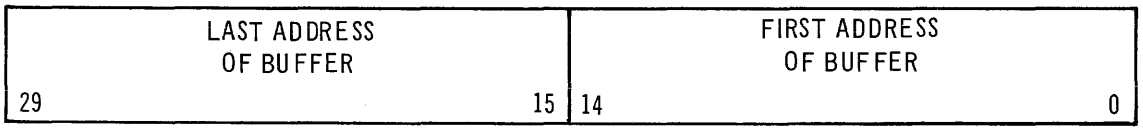


Figure 5-7. Control Word

As shown in the diagram, the high order 15-bits of the control word contain the address of the last word in the buffer and the low order 15-bits contain the address of the first word.

There are 14 input channels numbered 0 through 15 (octal) and 14 output channels numbered 0 through 15 (octal). Channels 0 and 1 are reserved for computer to computer communication. For each of these channels a fixed storage location is designated as a buffer control register. The input buffer control registers are located at octal addresses 00100 through 00115 and the output buffer control registers at octal addresses 00120 through 00135. The buffer control register for a particular channel is determined by the j designator in the buffer instruction word and, since it may range from 0 to 15_8, the input buffer control register for channel j is at address 00100 + j and the output buffer control register is at address 00120 + j.

At the time a buffer is activated, the lower half of the appropriate buffer control register contains the first address of the buffer and the upper half contains the last address. For example, if data is transferred on input channel 12 to a five word input buffer located at addresses 01000, 01001, 01002, 01003, and 01004, the input buffer control register, for this channel, located at address 00112, contains 0100401000.

As the operation is performed, 30-bit data words are transferred to or from the consecutive addresses in the buffer. The first transfer is made to or from the storage location whose address is contained in the lower half of the buffer control register. When the first transfer is completed, the lower half of the buffer control register is incremented by 1 to contain the address to which or from which the next data word will be transferred. These operations continue to transfer and increment until the buffer is filled or emptied. At this point the operation is terminated and the lower half of the buffer control register contains the address of the first storage location beyond the buffer; that is, the last address of the buffer + 1.

For example, data is being transferred from a peripheral subsystem on input channel 12 to a 5-word input buffer located at addresses 01000, 01001, 01002, 01003, and 01004. In this case, the buffer operation would be as follows:

(1) The initial contents of the input buffer control register at address 00112 will be 0100401000.

(2) Thirty-bit data words will be transferred to addresses 01000 through 01004 and the buffer control register will be incremented following each transfer.

(3) When the buffer mode is terminated, the input buffer control register will contain 0100401005.

c. Buffering Operations

- ACTIVATE INPUT BUFFER

  This instruction activates an input buffer on a specified input channel and sets up the appropriate input buffer control register by placing a control word that defines the size and location of the buffer at address 00100+n.

W	V_0	V_1
IN . Cn	Buffer Control Word	Absent
		MONITOR

n specifies the buffer control register that will be activated and modified. The actual location of the buffer control word is 100+n. The value of n must be in the range from 0 to 15_8.

v_o may be specified in any of the following forms:

L(x)	The low order portion of the operand (x) replaces the low order portion of the buffer control word thus establishing the first address of the buffer. The upper half of the buffer control word remains unchanged. If the upper address is zero, no data is transferred.
W(x)	The entire contents of the buffer control word are replaced by the operand (x).
A Constant	If the constant is equal to or less than five octal digits (77777) the value will replace the low order portion of the buffer control word; if more than five octal digits the entire buffer control word will be replaced by the value of the constant.

v_1 If this operand is absent, the buffer control word is not monitored.

Monitoring is specified by placing MONITOR in this operand position.

If a buffer operation is monitored, the main program is interrupted and control is transferred to 00040 + n when the buffer operation is terminated by an equal upper and lower address in location 00100 + n. The upper address is incremented following this comparison.

Examples:

```
IN . C5  . 52367                    73240 52367
IN . C13 . W(LABEL) . MONITOR       75570 12345
```

- ACTIVATE OUTPUT BUFFER

This instruction activates an output buffer on a specified output channel and sets up the appropriate output buffer control register by placing a control word that defines the size and location of the buffer at address 00120+n.

w	v_o	v_1
OUT . Cn	Buffer Control Word	Absent
		MONITOR

n specifies the buffer control register that will be activated and modified. The actual location of the buffer control word is 120+n. The value of n must be in the range from 0 to 15_8.

v_o may be specified in any of the following forms:

L(x)	The low order portion of the operand (x) replaces the low order portion of the buffer control word thus establishing the first address of the buffer. The upper half of the buffer control word remains unchanged.
W(x)	The entire contents of the buffer control word is replaced by the operand (x).
A Constant	If the constant is equal to or less than five octal digits (77777), the value will replace the low order portion of the buffer control word; if more than five octal digits the entire buffer control word will be replaced by the value of the constant.

v_1 If this operand is absent, the buffer control word is not monitored.

Monitoring is specified by placing MONITOR in this operand position.

A buffer operation is monitored if the main program is interrupted and control is transferred to 00060+n when the buffer operation is terminated by an equal upper and lower address in location 00120 + n. The upper address is incremented following this comparison.

Examples:

```
OUT . C12 . 33377                    74500 33377
OUT . C4  . W(LABEL) . MONITOR       76230 12345
```

- JUMP IF BUFFER ACTIVE

This instruction tests to determine whether or not an input or output buffer is active. If the buffer is active, control will be transferred to a specified address; if the buffer is not active, the instruction immediately following the jump instruction is executed.

W	v_o	v_1
JP	Jump Operand	Cn . ACTIVEIN
		Cn . ACTIVEOUT

v_o The jump operand, a Read Class operand, may be specified in any of the following forms:

U(x)	Jump to the address contained in the upper portion of the word represented by x.
L(x) or W(x)	Jump to the address contained in the lower portion of the word represented by x.
x	Jump directly to an address represented by x.

v_1 If Cn . ACTIVEIN, the input buffer is tested.

If Cn . ACTIVEOUT, the output buffer is tested.

The input or output buffer to be tested is specified by n, which may be a value from 0 to 15_8.

Examples:

```
JP . L(LABEL) . C13 . ACTIVEIN      62550 12345
JP . U(LABEL) . C5  . ACTIVEOUT     63260 12345
```

- TERMINATE BUFFER

This instruction terminates the input or output buffer on a specified channel. The transfer currently in process is completed and no further transfers are made to the buffer. If the buffer was activated with monitor, the internal interrupt that would normally follow after the buffer was filled will not occur. The last buffer address to or from which a transfer was made can be determined by examining the lower half of the buffer control word at address 00100+n for input, or address 00120+n for output.

W	v_o	v_1
TERM	Cn	INPUT
		OUTPUT

v_o Specifies the channel to be terminated. The value of n must be in the range from 0 to 15_8.

v_1 INPUT terminates input buffer.
OUTPUT terminates output buffer.

Examples:

```
TERM . C5  . INPUT      6624000000
TERM . C15 . OUTPUT     6764000000
```

d. External Control

Peripheral equipment is controlled by the use of function words. Function words are sent from the computer to the controlling elements of the peripheral equipment through the output data lines. A function word is preceded in the program definition by an External Function Instruction which distinguishes the function word from data. The function word is analyzed and a unique operation is performed. The peripheral equipment communicates with the central processor by an interrupt (See f. External Interrupt). An interrupt may result from the normal termination of an operation or from other causes, such as an error condition, or upon reaching the end of a tape or a drum. In addition, status words are sent to the computer by the peripheral equipment on the input data lines. Status words are the means by which the external equipment informs the central processor of the reason for the interrupt.

When the external interrupt control lines are activated, computer program control will be transferred to an address associated with the channel. An instruction placed in this address may begin a subroutine which will analyze the status word and take appropriate action.

The mode of operation described briefly in the above paragraphs permits the external equipment to perform operations without reference to the central processor logic. Central processor action is required only at the initiation and termination of an operation.

- EXTERNAL FUNCTION

  This instruction will cause a function word to be sent to the control elements of a peripheral subsystem on a specified channel.

W	V_0	V_1
EX-FCT	Cn	function word

  V_0 Specifies the channel (2 through 15_8) through which the function word is transferred. This may be defined by a channel label that was previously defined in a MEANS statement (see PROGRAM PREPARATION).

  V_1 May be an operand address in the form W(x). The function word may also be written as a ten octal digit constant. The format of the actual function words related to the type of subsystem may be found in APPENDIX D.

  Examples:

```
EX-FCT . C2 . W(LABEL)          13130 12345
EX-FCT . C4 . 6243052437        13230 70333
```

  In the last example, it is assumed that the ten digit function word was allocated to address 70333.

- TEST CHANNEL

  This instruction will test to determine if the input buffer for the other computer to computer channel (0 or 1) is active. If the buffer is active, the instruction immediately following the test instruction is skipped; if the buffer is inactive, the instruction is executed.

W	V_0
TEST	C0
	C1

v_o Specifies the channel to be tested. C0 specifies channel 0; C1 specifies channel 1.

Examples:

TEST . C0	13000 00000
TEST . C1	13040 00000

- STORE CHANNEL

This instruction will store into a specified memory location the status word that is generated following an external interrupt.

w	v_o
STR . Cn	storage location

n may be a value from 2 to 15_8 to specify the channel from which the word will be stored. stored.

v_o Specifies the storage location into which the status word will be stored. Only the form W(x) may be used.

e. Internal Interrupt

An internal interrupt occurs when a buffer that was activated by an IN or OUT instruction with MONITOR is filled or emptied. At this point, the program is interrupted, further interrupts are prevented from occurring (locked out), and instead of executing the next instruction in the program, control is transferred to an instruction contained in the appropriate internal interrupt entrance register. The input internal interrupt entrance registers are located at addresses 00040 through 00055, and the output internal interrupt entrance registers are located at addresses 00060 through 00075.

If n is a value equal to the channel number then the appropriate internal interrupt entrance register is determined as follows:

INPUT	40+n
OUTPUT	60+n

These registers should contain an instruction that will:

(1) Transfer program control to a routine that will process the interrupt.

(2) Capture the present contents of the program address register (P register) which contains the address of the next instruction at the time the interrupt occurred.

These functions will be performed by a Set Interrupt Lockout Return Jump statement. The captured value of the program address register incremented by 1 (P + 1) will be stored in the lower portion of the word specified by the v_0 operand. Control will be transferred to the address immediately following the address specified by the v_0 operand (v_0 operand + 1).

w	v_o
SILRJP	note 1

After the interrupt has been processed, the interrupt lockout may be released and control may be returned to the interrupted program by a Remove Interrupt Lockout and Jump statement in the following format:

w	v_o
RILJP	note 1

NOTE 1

This operand may be written in any of the forms specified for the Jump Instruction (JP).

f. External Interrupt

An external interrupt is a program interrupt initiated by an external device. A peripheral subsystem connected to any one of the input channels (0 through 15_8) may interrupt the current program by sending an external interrupt to the central computer. When this occurs, the current program is interrupted and instead of executing the next instruction in the current program an instruction contained in the appropriate external interrupt entrance register is executed. The external interrupt registers are located at addresses 00020 through 00035.

If n is a value equal to the channel number (0 through 15_8), then the appropriate external interrupt entrance register is at address 20+n.

The procedures for processing external interrupts are the same as those outlined for internal interrupts with the exception that a status word is generated by the peripheral subsystem. The status word may be removed from the input data lines and stored in a memory location for analysis with a STORE CHANNEL instruction (See paragraph d. of this section). This status word must be removed from the input data line if further input activity on the interrupted channel is expected.

A listing of the status words related to the type of subsystem may be found in APPENDIX D.

g. Interrupt Control

When it is desirable to prevent interrupts from occurring, this may be accomplished by a Set Interrupt Lockout statement in the following format:

w
SIL

Program control will proceed to the next instruction following the SIL statement. The lockout may be released to permit interrupts by a Release Interrupt Lockout statement in the following format:

w
RIL

Program control will proceed to the next instruction following the RIL statement.

Examples:

SIL	64000 000000
RIL	60000 000000

h. Examples of Basic Input/Output Sequence

The following examples pertain to the UNISERVO IIIC Subsystem to illustrate the sequencing of input/output instructions. Instructions for other subsystems are sequenced in a similar manner.

(1) A simple function such as the positioning of tape requires only an External Function instruction, which activates the subsystem to accept the appropriate Function Word. The following instruction and function word will rewind unit 7 on channel 12 with interrupt:

EX-FCT • C14 • W(LABEL) 13630 12345

Note that the underlined portion of the instruction would appear in binary format and be interpreted as follows:

j	k
6	3
1 1 0 0	1 1

channel
12 (14_8)

A value of 3 for k is required when the function code is 13.

A function word at address 12345 (LABEL) would appear as follows:

30100 00200

301 designates the subsystem function (rewind with interlock). The underlined portion specifies unit 7 as follows:

OCTAL:		0			2			0			0	
BINARY:	0	0	0	0	1	0	0	0	0	0	0	0
LOGICAL UNIT:	11	10	9	8	7	6	5	4	3	2	1	0

(2) Writing requires a total of five words, three instruction words addressed in sequence, a function word, and a buffer control word. In this example, 777_8 words are written in binary at high density, on logical unit 11, on channel 5, with interrupt. The coding would appear as follows:

EXT-FCT • C5 • W(FCT3)	13270	01010
NO-OP	12000	00000
OUT • C5 • W(BUF5)	74270	01020

The Function Word located at 01010 (FCT3) would appear as follows:

20776 20000

A NO-OP instruction must be placed between consecutive input/output instructions to provide the necessary spacing in all sequences.

(3) Reading from tape requires a total of five instructions, three instructions addressed in sequence, a buffer control word and a function word. Reading is similar to writing, but the order is reversed; the function is initiated after the buffer limits have been established. In this example, 377_8 words are read in binary high density from logical unit 6 on channel 6, with interrupt (assuming the end of file is not reached before the 377_8 words are read). In this example, as in the write example, the buffer area begins at location 20000.

IN • C6 • W(BUF6)	73330	01022
NO-OP	12000	00000
EX-FCT • C6 • W(FCT6)	13330	01016

The Buffer Control Word located at 01022 (BUF6) would appear as follows:

20376 20000

The Function Word located at 01016 (FCT6) would appear as follows:

52000 00100

(4) Searching for a specific block of data (and the subsequent automatic reading) requires a total of eight instruction words, five instruction words addressed in sequence, one buffer control word, one function word, and one identifier word. Search is similar to the read function, but in addition an identifier word must be supplied to which the first word of each block is compared. In this example, the file on logical unit 2 of channel 2 is searched until a "find" occurs. At this time, 377_8 words are read in binary coded decimal at low density (if that many words remain before end of file). The transfer to memory starts at address 10000. The coding would appear as follows:

IN • C2 • W(BUF2)	73130 01044
NO-OP	12000 00000
EX-FCT • C2 • W(FCT2)	13130 01045
NO-OP	12000 00000
EX-FCT • C2 • W(IDENT2)	13130 01046

The first instruction establishes the buffer with a Buffer Control Word that appears at location 01044 (BUF2) as follows:

10376 10000

The next instruction is a NO-OP for spacing.

The third instruction sends a function word to the subsystem which appears at location 01045 (FCT2) as follows:

57200 0004

Another NO-OP instruction is required between consecutive input/output instructions.

The fifth instruction provides a word for the search comparison. It may be in any format and would be contained in location 01046 (IDENT2).

Figure 5-8 shows the sequence of consecutive instructions that are required for each type of operation for the various subsystems.

TYPE OF OPERATION	PERIPHERAL SUBSYSTEM						
	UNISERVO IIA	UNISERVO IIIC	UNISERVO IIIA	FLYING HEAD 880 MAGNETIC DRUM	FASTRAND	CARD	PRINTER
WRITE (Punch or Print)	(1) EX-FCT (2) NO-OP (3) OUT						
READ	(1) IN (2) NO-OP (3) EX-FCT						
SEARCH READ	(1) IN (2) NO-OP (3) EX-FCT (4) NO-OP (5) EX-FCT						
MASKED SEARCH			(1) IN (2) NO-OP (3) EX-FCT (4) NO-OP (5) OUT				
SEARCH				(1) EX-FCT (2) NO-OP (3) EX-FCT			
The operations that are listed under the respective subsystems in the columns to the right are not involved with the movement of data to or from the central processor; consequently, only one basic input/output instruction (EX-FCT) is needed to cause the subsystem to execute them. In the these cases the EX-FCT instruction supplies the particular subsystem with a function word which defines the operation.	TERMINATE						
	REWIND				POSITION	SET INPUT MODE (Translation Binary, or Binary coded Decimal) SELECT STACKER 1 SELECT STACKER 2	
		ERASE WRITE END OF FILE	CONTINGENCY WRITE WRITE FILE SEPARATOR				

Figure 5-8 Basic Input/Output Instructions Sequence Requirements

NOT APPLICABLE

6. SPURT INPUT/OUTPUT UNDER EXECUTIVE CONTROL

A major consideration in the design of the UNIVAC 490 Real-Time System is the supervision required for the orderly and efficient processing of input/output requests. The overall supervision is provided by the Real-Time Executive Routine (REX), which gives priority to the input-output requests of the real-time program. A request for the use of input/output facilities is presented to REX by coding in the object program. The object program coding is generated from SPURT instructions within the source program.

The use of the appropriate SPURT mnemonics will assure the proper linkage to REX. REX will then execute the request if the required facilities are not in use. If the required facilities are in use, the request will be listed by REX for later initiation. Batch program requests will be listed in the order in which they are received. Indicators that are contained within areas associated with each program will record the present status of each input/output request. These areas are created when a program is loaded.

A. STATUS CHECKING

When it becomes necessary to determine the status of an input/output request, a CKSTAT line is written in the source program coding. The CKSTAT line is a SPURT mnemonic which is linked to a particular input/output request through the label assigned to the input/output request. The main purpose of the CKSTAT line is to provide an orderly method of declaring return points for completed input/output requests.

The CKSTAT line may be written with the following options:

- Control will be returned to the program following the CKSTAT line upon satisfactory completion of the input/output request. Should an error occur the program will be suspended.

- Control will be returned as in the first case, but in addition an error address will be provided to which control will proceed if an error condition occurs.

- An address may be specified to which program control will immediately pass. When the input/output request is completed the appropriate return point will be eligible for control. This return point for successful completion or error may be any of the three conditions outlined previously.

The text which follows will describe the required source program coding to initiate input/output requests for each of the standard pheripheral input/output subsystems. The CKSTAT line is written in a similar format to determine the status of any input/output request, except Console Typewriter input-output requests, for which a CKSTAT line is not used.

The general format for a CKSTAT line is:

i	w	v_0	v_1	v_2
	CKSTAT	request label	error address	alternate address specifier

v_0 the label of the input/output request that is being interrogated.

v_1 the error address to which control will pass if an error condition exists. The omission of this operand or the insertion of STOPRUN as an operand will cause the program to be suspended (see STOPRUN under VOLUNTARY RELEASE OF CONTROL).

v_2 an address to which control will immediately pass. This may be the label of another input/output request.

1. Error Analysis

As explained previously, the status of an input/output request is determined by execution of object code generated by a CKSTAT operator.

When an interlock error occurs, REX records the error, analyzes it, and relays an appropriate message via the Console Printer. Non-interlock errors may be referred to the input/output routine for attempted recovery. If recovery is made, no indication is given to the user program.

If the error is a non-recoverable error, it may cause either program suspension or a transfer to an error routine that was written by the REX user. The following chart shows the information that is available in the various registers at the time a non-recoverable error is detected.

<table>
<tr><th>REGISTER</th><th>CONTENTS</th></tr>
<tr><td>A</td><td>A status word that indicates the type of error.</td></tr>
<tr><td>Q</td><td>
<table>
<tr><th>Upper</th><th>Lower</th></tr>
<tr><td>Address of input/output request</td><td>address of SPURT instruction following CKSTAT</td></tr>
</table>
</td></tr>
<tr><td>B7</td><td>An address, aaaaa, which is the first address of a three-word area that contains the contents of the B1 through B6 registers at the time the CKSTAT was performed. The contents of these registers are stored as follows:
<table>
<tr><th></th><th>Upper</th><th>Lower</th></tr>
<tr><td>aaaaa</td><td>B1</td><td>B2</td></tr>
<tr><td>aaaaa + 1</td><td>B3</td><td>B4</td></tr>
<tr><td>aaaaa + 2</td><td>B5</td><td>B6</td></tr>
</table>
</td></tr>
</table>

2. Examples of CKSTAT

Let us assume for the following examples that an input/output request has been submitted to REX which has been labelled as follows:

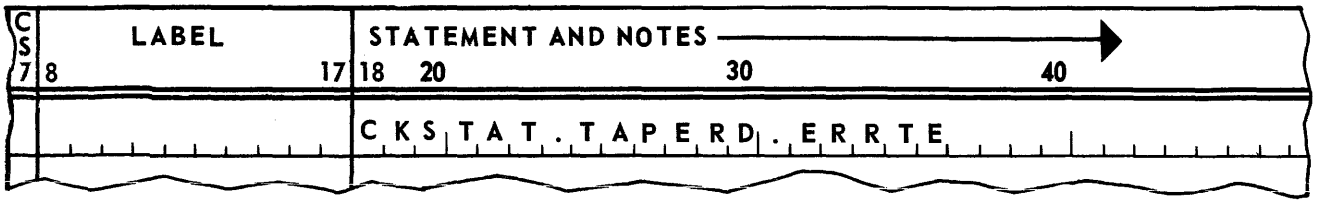
```
CS 7 | 8        LABEL        17 | 18  20        STATEMENT AND NOTES ------>        30        40
     | TAPERD                    | any input/output instruction
```

If the user does not desire to receive control of the program until the input/output request is completed, and if a STOPRUN halt is desired in the event of an error condition, the CKSTAT line must simply refer to the input-output request label as follows:

```
CS 7 | 8        LABEL        17 | 18  20        STATEMENT AND NOTES ------>        30        40
     |                           | CKSTAT.TAPERD
```

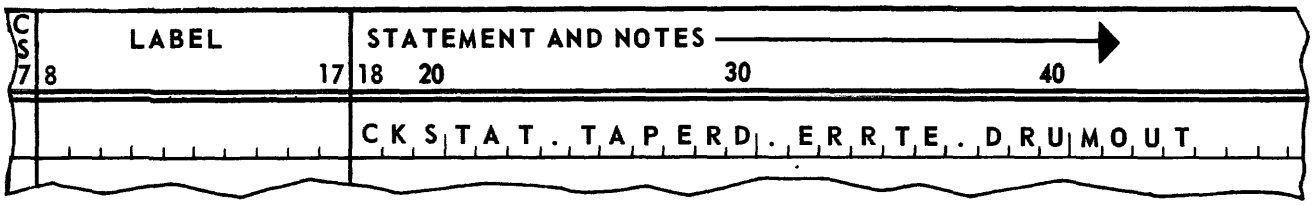
(control transferred to next SPURT instruction upon completion of request)

The user may wish to enter his own routine in the event of an error condition. If the error routine is labelled ERRTE, the CKSTAT line will appear as:



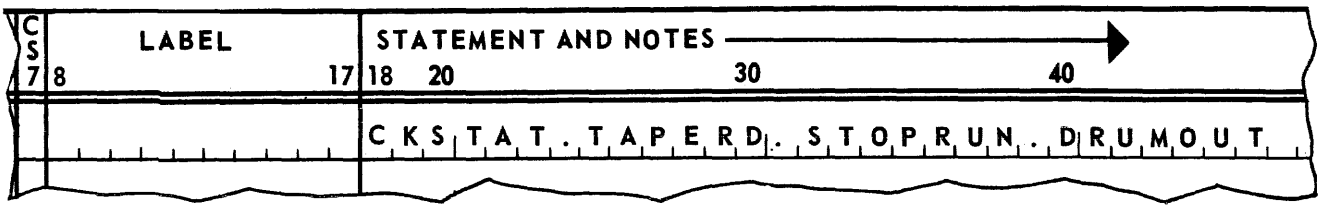
(control transferred to next SPURT instruction upon completion of request)

The time required to process the input/output request may be overlapped by further processing or by another input/output request. Once an input/output request is initiated, the transfer into or out of the computer is accomplished as an independent operation. REX makes note of the completion of the request and control will be returned following the CKSTAT. If the user wishes to perform some other task while waiting for the completion of the input/output request, and if this alternate processing has an entrance label of DRUMOUT, the CKSTAT line will appear as follows:



(control transferred to next SPURT instruction upon completion of request)

If a STOPRUN halt is desired, instead of a transfer to ERRTE, STOPRUN must be inserted as an error address to maintain the relative positions of the operands. The CKSTAT line would then appear as follows:



(control transferred to next SPURT instruction upon completion of request)

Control must be returned to REX by a TAKEOVER release at the end of the input/output completion path as explained under VOLUNTARY RELEASE OF CONTROL.

The text which follows will describe the SPURT programming that is required to initiate input/output operations for each of the peripheral units. The description of the CKSTAT line and its use applies to all the following input/output instructions. Following the description of the programming is a list of the status indicators that are placed in the A register when control is returned to a program.

B. UNISERVO IIA MNEMONICS

UNISERVO IIA input/output and positioning operations are requested by the submission of the MTAPE operator. The general format for this operator is as follows:

l	w	v_0	v_1	v_2	v_3
	MTAPE	operation	unit name	buffer control word or block count	identifier word

v_0 names the individual tape operation to be performed. See section 6.B.1, UNISERVO IIA Tape Operations.

v_1 defines the name of the unit upon which the tape action will occur. See PROGRAM PREPARATION.

v_2 defines the address of the buffer control word. The lower portion of the buffer control word specifies the first address used to hold input or output data. For the Move Forward or Move Backward operations, this operand indicates either the number of blocks tape is to be moved or is a label of the location in memory where such a value may be found. In this case only the least significant 15 bits are used as a block count.

v_3 is the search identifier used for search operations. This may be an operand which defines a location where the identifier can be found.

1. UNISERVO IIA Tape Operations

OPERATION	TYPE OF OPERATION	MNEMONIC	V-OPERANDS USED
Rewind	Positioning	REWIND	0, 1
Rewind with Interlock	Positioning	RWI	0, 1
Locate Backward	Positioning	LOCATEB	0, 1, 2, 3
Locate Forward	Positioning	LOCATEF	0, 1, 2, 3
Search Read Backward	Input/Output	SEARCHB	0, 1, 2, 3
Search Read Forward	Input/Output	SEARCHF	0, 1, 2, 3
Write at 125 characters per inch	Input/Output	PWRITE	0, 1, 2
Write at 250 characters per inch	Input/Output	WRITE	0, 1, 2
Read Forward	Input/Output	READF	0, 1, 2
Read Backward	Input/Output	READB	0, 1, 2
Move Forward	Positioning	MOVEF	0, 1, 2
Move Backward	Positioning	MOVEB	0, 1, 2

- Rewind and Rewind with Interlock

  Rewind and rewind with interlock will cause the tape to be rewound. Another operation cannot be given a unit that is rewound with interlock until the interlock is manually released.

- Locate Backward

  Same as Search Read Backward operation except that data are not read into memory.

- Locate Forward

  Same as Search Read Forward operation except that data are not read into memory.

- Search Read Backward

  The tape will be searched backward toward the beginning of tape. A comparison will be made between the first encountered word of each block and the identifier word. A block upon which an equal comparison is made will be read into the area of memory specified by the buffer control word. The order of placement in the input buffer is reversed.

  If the contents of the buffer control word are equal to zero, this will indicate a search to position only (Locate Backward).

  If the identifier word is not discovered, indication will be given via the end of file status word that the block could not be found.

- Search Read Forward

  This operation will search forward, toward the end of the tape. A comparison will be made between the first encountered word of each block and the identifier word. A block upon which an equal comparison is made will be read into the area of memory specified by the buffer control word.

  If the contents of the buffer word are equal to zero, this will indicate a search to position only (Locate Forward).

  If the identifier word is not discovered, indication will be given via the end of file in the status word that the block could not be found.

- Write (125 characters per inch)

  This operation will write one block on magnetic tape at a density of 125 characters per inch.

- Write (250 characters per inch)

  This operation will write one block on magnetic tape at a density of 250 characters per inch.

- Read Forward

  This operation will read one block in a forward direction.

- Read Backward

  This operation will read one block in a backward direction. The order of placement in the input buffer is reversed.

- Move Forward

  This operation will move the tape forward the number of blocks specified in the block count operand. If the number of blocks specified is greater than the number of blocks that can be traversed from the point of initiation, the number of blocks actually passed will be indicated in the high order 15 bit positions of the status word.

- Move Backward

  This operation will move the tape backward the number of blocks specified in the block count operand. If the number of blocks specified is greater than the number of blocks that can be traversed from the point of initiation, the number of blocks actually passed will be indicated in the high order 15 bit positions of the status word.

2. UNISERVO IIA Status Words

The status of every operation that is initiated will be indicated by means of a status code placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be completed successfully because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

In some instances supplementary information will be contained in the upper 15 bit positions of the A-Register along with the status code; that is, for all successful Read and Search-Read operations, the number of words (w) read from the block to the buffer will be expressed; and for Move operations, the number of blocks (b) actually traversed will be indicated.

a. Status Indications

CODE	MEANS	ERROR MESSAGE
wwwww 00001	Operation completed	
bbbbb 00004	End of file	
bbbbb 00005	End of tape	
00000 00006	Unit interlocked	MSG 200 Pxx y
00000 00007	Illegal Procedure	MSG 201 Pxx y
00000 00010	Read or Write Error	MSG 201 Pxx y
00000 00011	Sequence Error	MSG 201 Pxx y
00000 00012	Incorrect Parameter	MSG 202 Pxx

P = Program identification number

xx = Unit number

y = Last computer generated interrupt word. For operator and service use.

b. Explanation of Status Indications

01 Operation successfully completed.

04 This indication will result when an end of file gap is detected.

05 The end of tape warning marker was detected during a Write operation.

06 The unit for which the operation is intended is non-operable because of an interlock condition.

07 An error indication resulting from:

- An invalid computer function.
- A meaningless interrupt code.
- An invalid unit number.

10 This status indication is returned when parity errors, character count errors, or the like are deemed non-recoverable by REX following standard recovery procedures on:

- A Read operation, or the read portion of a Search-Read operation. The tape will be positioned under the read-write head at the point at which the operation started.
- A Write operation. The tape will be restored so it is positioned under the read-write head directly in front of the badly written block.

11 Indicates a discrepancy in the number of characters that should have been read from a block of data. Because this particular error can result in a 2-block read, the position of the tape is unknown.

12 Indicates an incorrect parameter such as:

- An invalid operation code.
- A unit number greater than 12.
- A block count of zero.
- Lower half of buffer control word greater in value than upper half.

C. UNISERVO IIIC MNEMONICS

UNISERVO IIIC initializing, positioning, and input/output operations are requested by the submission of the CTAPE operator.

The initializing operations do not cause any tape action. They are used to specify the tape density and mode for the input/output operations that will follow on a given unit. If an initializing operation is not performed, the results of ensuing input/output operations such as Read or Write will be unpredictable. The tape density and mode for a given unit remains in effect until redefined by another initializing operation.

The general format of the CTAPE operator when an initializing operation is requested is as follows:

I	W	V_0	V_1	V_n
	CTAPE	operation	unit name	unit name

V_0 names the initializing operation to be performed. See section 6.C.1., UNISERVO IIIC Tape Operations.

V_1 defines the names of the units that are initialized as indicated by V_0. See PROGRAM PREPARATION.

.
.
.

V_n

The input/output operations effect the transfer of data between the Central Processor and a given unit and the positioning operations control the positioning of the tape on that unit. The general format of a CTAPE operator when an input/output or positioning operation is requested is as follows:

I	W	V_0	V_1	V_2	V_3
	CTAPE	operation	unit name	buffer control word or block count	identifier word

V_0 names the operation to be performed. See section 6.C.1., UNISERVO IIIC Tape Operations

V_1 defines the name of the unit upon which the input/output or positioning operation will be performed. See PROGRAM PREPARATION.

v_2 defines the address of the buffer control word. The lower portion of the buffer control word specifies the first address used to hold input or output data and the upper portion specifies the last address.

For Move Forward or Move Backward operations, this operand indicates the number of blocks the tape is to be moved or it is the label of the location in memory that contains such a value.

v_3 is the search identifier used for search operations. This may be the label of a location in memory where the identifier may be found.

1. UNISERVO IIIC Tape Operations.

OPERATION	TYPE OF OPERATION	MNEMONIC	V-OPERANDS USED
Select Binary Coded Decimal at high density (556 characters per inch)	Initializing	HIBCD	0, 1, through n
Select Binary Coded Decimal at low density (200 characters per inch)	Initializing	LOBCD	0, 1, through n
Select Binary at high density (556 characters per inch)	Initializing	HIBIN	0, 1, through n
Select Binary at low density (200 characters per inch)	Initializing	LOBIN	0, 1, through n
Rewind	Positioning	REWIND	0, 1
Rewind with interlock	Positioning	RWI	0, 1
Search Read	Input/Output	SEARCH	0, 1, 2, 3
Write	Input/Output	WRITE	0, 1, 2
Write end of file block	Input/Output	ENDFILE	0, 1
Read	Input/Output	READ	0, 1, 2
Move Forward	Positioning	MOVEF	0, 1, 2
Move Backward	Positioning	MOVEB	0, 1, 2
Move Backward to End Of File or Load Point	Positioning	BACKFILE	0, 1

- Select Binary Coded Decimal At High Density

  Indicates that the tape density for a given unit or units will be 556 characters per inch and that the mode will be Binary Coded Decimal.

- Select Binary Coded Decimal At Low Density

  Indicates that the tape density for a given unit or units will be 200 characters per inch and that the mode will be Binary Coded Decimal.

- Select Binary At High Density

  Indicates that the tape density for a given unit or units will be 556 characters per inch and that the mode will be binary.

- Select Binary At Low Density

  Indicates that the tape density for a given unit or units will be 200 characters per inch and that the mode will be binary.

- Rewind

  Causes the tape to be rewound.

- Rewind With Interlock

  Same as the Rewind operation with the exception that subsequent operations cannot be initiated on the specified unit until the interlock is manually released.

- Search Read

  This operation will search forward, toward the end of tape. A comparison will be made between the first encountered word of each block and the identifier word. A block upon which an equal comparison is made will read into the area of memory specified by the buffer control word.

  If the identifier word is not discovered, indication will be given via an end of file status word that the block could not be found.

- Write

  This operation will write one block on tape.

- Write End Of File

  This operation will cause a special two-character end of file block to be recorded on the tape specified. When this block is sensed during a Read, Search Read, or Move operation an end of file condition will be reported to the program.

- Read

  This operation will read one block in the forward direction.

- Move Forward

  This operation will move the tape forward the number of blocks specified by the block count operand. If the number of blocks specified is greater than the number of blocks that are contained forward of the point of initiation, the number of blocks actually passed will be indicated in the upper 15 bit positions of the status word.

- Move Backward

  This operation will move the tape backward the number of blocks specified by the block count operand. If the number of blocks specified is greater than the number of blocks recorded to this point the number of blocks actually passed will be indicated in the upper 15 bit positions of the status word.

- Backspace File

  This operation moves the tape backward until either an end of file block or the load point is encountered.

2. UNISERVO IIIC Status Words

The status of every operation that is initiated will be indicated by means of a status code placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be completed successfully because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

In some instances supplementary information will be contained in the upper 15 bit positions of the A-Register along with the status code; that is, for all successful Read and Search-Read operations, the number of words (w) read from the block to the buffer will be expressed; and for Move operations the number of blocks (b) actually traversed will be indicated.

a. Status Indications

CODE	MEANS	ERROR MESSAGE
wwwww 00001	Operation complete	
bbbbb 00004	End of file	
bbbbb 00005	Tape limit detected	
00000 00006	Unit interlocked	MSG 220 Pxx y
wwwww 00007	Subsystem error	MSG 221 Pxx y
00000 00010	Incorrect parameter	MSG 222 Pxx

P = Program identification number

xx = Unit number

y = Last computer generated interrupt word. For operator and service use.

b. Explanation of Status Indications

01 The operation has been completed. An abnormal character condition incurred during a Read operation will be considered normal. In this case, bit position 29 of the status word will be set to binary one to indicate this particular status.

04 This indication will result when:

- The special two-character end of file is detected during a Read, Search-Read, or Move Forward operation.
- A block of four characters or less is detected during Backspace File or Move Backward operation.

05 This indication will result when:

- A Write operation has been successfully completed beyond the tape limit mark.
- The load point has been encountered during a Move Backward or Backspace File operation.

06 The unit for which the operation is intended is nonoperable because of an interlock condition.

07 A subsystem error such as:

- An invalid computer function
- A meaningless interrupt code
- An invalid unit number
- Non-recoverable equipment errors. If a non-recoverable situation occurs the number of words read will be indicated in the status word.

10 An incorrect parameter such as:

- An invalid operation code.
- A unit number greater than 12.
- A block count of zero.
- The lower half of the buffer control word is greater in value than the upper half.

D. UNISERVO IIIA MNEMONICS

UNISERVO IIIA input/output and positioning operations are requested by the submission of the MTAPE operator. The general format for this operator is as follows:

I	W	V_0	V_1	V_2	V_3	V_4
	MTAPE	operation	unit name	buffer control word or block count	identifier word	search mask

V_0 names the operation to be performed. See section 6.D.1., UNISERVO IIIA Tape Operations.

V_1 defines name of the unit upon which the operation will be performed. See PROGRAM PREPARATION.

V_2 defines the address of the buffer control word. The lower portion of the buffer control word specifies the first address to hold input or output data and the upper portion specifies the last address.

For the Move Forward or Move Backward operations, this operand indicates the number of blocks tape is to be moved or it is the label of the location in memory whose lower portion contains such a value.

This operand is not required for the Locate Forward or Locate Backward operations.

V_3 is the search identifier used for search operations. This may be the label of a location in memory where the identifier can be found.

V_4 is the search mask associated with Search operations. This may be the label of a location in memory where the search mask can be found.

The effect of the search mask is to select the portion of the first word of each block upon which a comparison will be made. The portion of the first word of each block that is selected is determined by the presence of binary ones in specific positions in the search mask; that is, only those bit positions in the first word of each block that correspond to the bit positions in the mask will be carried down and be used as basis for comparison. For example if the bit configuration of the identifier word specified in the V_3 operand is

000000000000000000000000001100

and the user desires that the comparison between the first word of each block and the identifier word to be based upon the contents of the last four bit positions, the following mask would be specified:

000000000000000000000000001111

If the bit configuration of the first word of a particular block was

00000000000000001100111000 1100

the following operation would be performed prior to the comparison:

00000000000000001100111000 1100 (first word of block)

00000000000000000000000000 1111 (mask)

00000000000000000000000000 1100 (masked operand)

The resulting masked operand is then compared with the identifier word. In this case equality would result and the search would be terminated.

1. UNISERVO IIIA Tape Operations

OPERATION	TYPE OF OPERATION	MNEMONIC	V-OPERANDS USED
Rewind	Positioning	REWIND	0, 1
Rewind with interlock	Positioning	RWI	0, 1
Locate Backward	Positioning	LOCATEB	0, 1, 3, 4
Locate Forward	Positioning	LOCATEF	0, 1, 3, 4
Search Read Backward	Input/Output	SEARCHB	0, 1, 2, 3, 4
Search Read Forward	Input/Output	SEARCHF	0, 1, 2, 3, 4
Write	Input/Output	WRITE	0, 1, 2
Overwrite	Input/Output	WRITEO	0, 1
Write End of File	Input/Output	ENDFILE	0, 1
Read Backward	Input/Output	READB	0, 1, 2
Read Forward	Input/Output	READF	0, 1, 2
Move Backward	Positioning	MOVEB	0, 1, 2
Move Forward	Positioning	MOVEF	0, 1, 2

- Rewind

  Causes the tape to be rewound.

- Rewind With Interlock

  Same as the Rewind operation with the exception that subsequent operations cannot be initiated on the specified unit until the interlock is manually released.

- Locate Backward

  This operation is the same as the Search Read Backward operation except that data are not read into memory. A buffer control word is not required with this operation. For this operation the search identifier will be in the v_2 operand position and, if required, the search mask will be in the v_3 operand position.

- Locate Forward

  This operation is the same as the Search Read Forward operation except data are not read into memory. A buffer control word is not required with this operation. For this operation the search identifier will be in the v_2 operand position and, if required, the search mask will be in the v_3 operand position.

- Search Read Backward

  The tape will be searched backward toward the beginning of tape. A comparison will be made between the first encountered word of each block and the identifier word. A block upon which an equal comparison is made will be read into the area of memory specified by the buffer control word. The order of placement in the input buffer is reversed.

  If the contents of the buffer word are equal to zero, this will indicate a search to position only (Locate Backward).

  If the identifier word is not discovered, indication will be given via an end of file status word that the block could not be found.

- Search Read Forward

  This operation will search forward toward the end of tape. A comparison will be made between the first encountered word of each block and the identifier word. A block upon which an equal comparison is made will be read into the area of memory specified by the buffer control word. The tape will be positioned beyond the block that was read in.

  If the contents of the buffer word are equal to zero, this will indicate a search to position only (Locate Forward).

  If the identifier word is not discovered, indication will be given via an end of file status word that the block could not be found.

- Write

  This operation writes one block on tape.

- Overwrite

  This operation is used in those instances when it is necessary or desirable to write over previously recorded data. When this operation is requested it causes a particular pattern to be written until erased tape is detected by the read-write head. This pattern is ignored during a Read or Search-Read operation.

- Write End of File

  This operation will cause the specified unit to create an end of file gap on tape; that is, to erase approximately 2-½ inches of tape while moving forward. When this gap is sensed during any operation other than Rewind, an end of file condition will be reported to the program.

- Read Backward

  This operation will read one block in a backward direction. The order of placement in the input buffer is reversed.

- Read Forward

  This operation will read one block in a forward direction.

- Move Backward

  This operation will move the tape backward the number of blocks specified by the block count word. If the number of blocks specified is greater than the number of blocks recorded to this point, the number of blocks actually passed will be indicated in the upper 15 bit positions of the status word.

- Move Forward

  This operation will move the tape forward the number of blocks specified by the block count word. If the number of blocks specified is greater than the number of blocks that are contained forward of the point of initiation, an "end of file" will result. The number of blocks actually passed will be indicated in the upper 15 bit positions of the status word.

2. UNISERVO IIIA Status Words

The status of every operation that is initiated will be indicated by means of a status code placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be completed successfully because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

In some instances supplementary information will be contained in the upper 15 bit positions of the A-Register along with the status code; that is, for all successful Read and Search-Read operations, the number of words (w) read from the block to the buffer will be expressed; and for Move operations the number of blocks (b) actually traversed will be indicated.

a. Status Indications

CODE	MEANS	ERROR MESSAGE
wwwww 00001	Operation completed	
bbbbb 00004	End of file	
bbbbb 00005	Tape limit detected	
00000 00006	Unit interlocked	MSG 210 Pxx y
00000 00007	Subsystem error	MSG 211 Pxx y
00000 00010	Incorrect parameter	MSG 212 Pxx

P = Program identification number

xx = Unit number

y = Last computer generated interrupt word. For operator and service use.

b. Explanation of Status Indications

01 Operation successfully completed.

04 This indication will result when the end of file gap is detected.

05 This indication will result when:

- A Write operation has been successfully completed beyond the tape limit mark.
- The beginning of the tape is encountered during a Read,Search Read, or Move Backward operation.

06 The unit for which the operation is intended is non-operable because of an interlock condition.

07 A subsystem error such as:

- An invalid computer function.
- A meaningless interrupt code.
- An invalid unit number.

10 An incorrect parameter such as:

- An invalid operation code.
- A unit number greater than 16.
- The lower half of the buffer control word is greater in value than the upper half.

E. FLYING HEAD-880 DRUM MNEMONICS

Flying Head-880 Drum input/output and positioning operations are requested by the submission of the DRUM operator. The general format of this operator is as follows:

I	W	V_0	V_1	V_2	V_3	V_4
	DRUM	drum system name	operation	buffer control word	drum address	identifier word

V_0 defines the name of the drum system upon which the operation will be performed. If an installation has all its drums on one channel, this operand is unnecessary. See PROGRAM PREPARATION.

V_1 names the operation to be performed. See section 6.E.1., Flying Head-880 Drum Operations.

V_2 defines the address of the buffer control word. The lower portion of the buffer control word specifies the first address used to hold input or output data and the upper portion specifies the last address.

This operand is not required for the Block Locate or Locate operations.

V_3 defines the first drum address that is associated with the input/output operation. This may be the label of a location in memory that contains a drum address.

V_4 is the search identifier used for search operations. This may be the label of a location in memory where the identifier may be found.

1. Flying Head-880 Drum Operations

OPERATION	TYPE OF OPERATION	MNEMONIC	V-OPERANDS USED
Continuous Write	Input/Output	WRITE	0, 1, 2, 3
Continuous Read	Input/Output	READ	0, 1, 2, 3
Locate	Positioning	LOCATE	0, 1, 3, 4
Search Read	Input/Output	SEARCH READ	0, 1, 2, 3, 4
Block Locate	Positioning	BLOCATE	0, 1, 3, 4
Block Search Read	Input/Output	BSEARCH	0, 1, 2, 3, 4
Block Read	Input/Output	BREAD	0, 1, 2, 3

- Continuous Write

  Starting with the drum address specified, data is written on the drum in consecutive locations until the output buffer is exhausted or the end of file (the last address in the drum system) is reached.

- Continuous Read

  Starting with the drum address specified, data is read from the drum into the input buffer until the buffer is filled or the end of file is reached.

- Locate

  Starting with the drum address specified, a search is made through consecutive locations for a word identical to the identifier word until discovery is made or end of file is reached. The address of the identical word will be placed in the status word.

  For this operation the drum address and the identifier word will be in the v_2 and v_3 operand positions respectively.

- Search Read

  Starting with the drum address specified, a search is made through consecutive locations for a word identical to the identifier word until discovery is made or end of file is reached. A Continuous Read operation commences with the identical word.

- Block Locate

  This operation is the same as the Locate operation except that the search ceases when an end of block word is detected. If discovery is made, the address of the find word will be placed in the status word.

  If an end of block (a word of all one's) is detected before discovery is made, the overflow address (the contents of the word immediately following the end of block) will be placed in the status word.

  For this operation the drum address and the identifier word will be in the v_2 and v_3 operand positions respectively.

- Block Search Read

  This operation is the same as the Search Read operation except that search ceases when an end of block word is detected. If discovery is made, a Block Read operation commences with the identical word.

  When an end of block word is detected the overflow address will be placed in the status word.

- Block Read

  Starting with the drum address specified, data is read from the drum into the input buffer until the end of block word is detected. At that time the input/output routine will replace the original buffer control word with the current buffer control word. The overflow address will be placed in the status word.

2. Flying Head-880 Drum Status Words

The status of every operation that is initiated will be indicated by means of a status code placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be successfully completed because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

In some instances supplementary information will be contained in the upper 15 or upper 23 bit positions of the A-Register along with the status code. This information will be the number of words (w) read or written (placed in the upper 15 bit positions), or an address (a) that represents either the overflow address or the address of a find word (placed in the upper 23 bit positions).

a. Status Indications

CODE	MEANS	ERROR MESSAGE
aaaaa aaa01	Operation completed	
00000 00002	End of file without find	
aaaaa aaa03	End of block without find	
00000 00004	Buffer filled before end of block	
wwwww 00005	End of file before complete transfer	
00000 00007	Illegal error	MSG 251 Pxx y
00000 00010	Illegal function	MSG 251 Pxx y
00000 00011	Illegal address	MSG 251 Pxx y
wwwww 00012	Read error	MSG 251 Pxx y
wwwww 00013	Write error	MSG 251 Pxx y
aaaaa aaa14	Overflow error after read	MSG 251 Pxx y
aaaaa aaa15	Overflow error before read	MSG 251 Pxx y
00000 00016	Illegal parameter	MSG 252 Pxx
00000 00017	Drum down	MSG 251 Pxx y

P = Program identification number

xx = Unit number

y = Last computer generated interrupt word. For operator and service use:

b. Explanation of Status Indications

01 Operation successfully completed.

02 This indication will result during Locate, Block Locate, Search Read, or Block Search Read operations when end of file is detected before a find is made.

03 This indication will result during Block Locate or Block Search Read operations when the end of block is reached without a find. The address of the overflow word will be placed in the upper 23 bits of the status word.

04 This indication will result during Block Read or Block Search Read operations if the input buffer is filled before end of block is reached.

05 This indication will result if end of file is detected during:

- A Continuous Write, Continous Read, or Search Read operation before the specified buffer filled or exhausted. The number of words read or written up to this point will be placed in the upper 15 bit positions of the status word.
- A Block Read or Block Search Read operation after reading has begun. The number of words read up to this point will be placed in the upper 15 bit positions of the status word.

06 There is a mechanical fault on the drum.

07 Illegal error.

10 The input/output routine has determined that the operation is not legal.

11 The input/output routine has determined that the drum address is invalid.

12 This indication will result if any equipment errors occur, including Channel Synchronizer-Control Unit or parity errors, during non-write operations. If a data transfer has occurred, the number of words read to this point will be placed in the upper 15 bit positions of the status word.

13 This indication will result if any equipment error occurs, including Channel Synchronizer-Control Unit or parity errors, during a Continuous Write operation.

14 This indication will result if an overflow word error occurs during a Block Search Read operation after reading has begun. The address of the overflow word that was in error will be placed in the upper 23 bits of the status word.

15 This indication will result if an overflow word error occurs during a Block Search Read operation where the identifier word was not in the block. The address of the overflow word that was in error will be placed in the upper 23 bits of the status word.

16 This indication will result if either an invalid operation code is detected or the beginning drum address in the buffer control word is greater than the ending address.

17 The drum unit requested by an operation is inoperative.

F. HIGH-SPEED PRINTER MNEMONICS

The user program can request two types of High-Speed Printer operations – initializing operations and output operations. An initializing operation is requested by the submission of the PIN operator and an output operation is requested by the submission of the PRINT operator.

An initializing operation does not cause any Printer action. It is used to specify the page format for the output operations that will follow on a given unit; that is, the total number of lines per page, the number of lines to be left blank at the top of the page (the top margin), and the number of lines to be left blank at the bottom of the page (the bottom margin). A page format will remain in effect until it is redefined by another initializing operator. The general format of the PIN operator is as follows:

I	w	v_0	v_1	v_2	v_3
	PIN	Printer name	number of lines per page	number of lines of top margin	number of lines of bottom margin

v_0 defines the name of the Printer for which initialization is desired. This operand is required if there is more than one Printer at an installation; that is, where there is a Printer on more than one input/output channel. See PROGRAM PREPARATION.

v_1 specifies the total number of lines per page. This number may be expressed in decimal or octal. The total number of lines per page may not exceed 511 decimal or 777 octal.

v_2 specifies the number of lines to be left blank at the top of each page. This number may be expressed in decimal or octal. The number may not exceed 63 decimal or 77 octal.

v_3 specifies the number of lines to be left blank at the bottom of each page. This number may be expressed in decimal or octal. The number may not exceed 63 decimal or 77 octal.

An output operation effects the transfer of data from the Central Processor to a given Printer, spaces the paper the number of lines specified, and causes one line to be printed. The general format of the PRINT operator is as follows:

I	w	v_0	v_1	v_2
	PRINT	Printer name	base address of print buffer	number of lines to advance

v_0 defines the name of the Printer on which print action is desired. This operand is required if there is more than one Printer at an installation; that is, where there is a Printer on more than one input/output channel. See PROGRAM PREPARATION.

v_1 specifies the base address of the print buffer.

v_2 specifies the number of lines to be advanced before printing occurs. This operand may be expressed in either decimal or octal. The acceptable values are 0 through 63 decimal or 0 through 77 octal. If 0 is specified, the paper will not be advanced and the printing will overlay the previous line. If NP is specified, the paper will be advanced to the first line of the next page before printing.

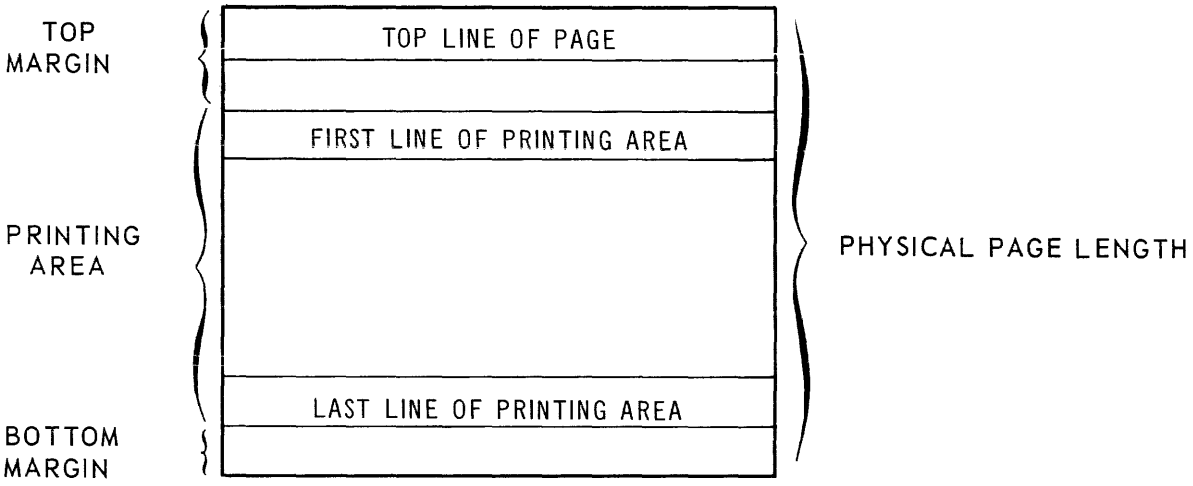
1. High-Speed Printer Operations

OPERATION	TYPE OF OPERATION	MNEMONIC	V-OPERANDS USED
Initialize Printer	Initializing	PIN	0, 1, 2, 3
Print One Line	Output	PRINT	0, 1, 2

- Initialize Printer

Before this operation can be requested, the paper must be positioned so that the top line of the page is in printing position; that is, aligned with the scribe mark on the Printer carriage.

This operation establishes the page format for the PRINT operations that follow on a given unit. It specifies the number of lines per page (physical page length), the number of lines to be left blank at the top of the page (top margin), and the number of lines to be left blank at the bottom of the page (bottom margin).



If no special page format is desired, an initializing operation should be requested in which the physical page length, the top margin, and the bottom margin each have a value of zero. If this is done, all subsequent data will be printed without top or bottom margins.

It should be noted that if an initializing operation is requested, it will remain in effect until it is redefined by another initializing operation.

- Print

This operation spaces the paper the number of lines specified and then prints one, 128 character line. The data to be printed must be in Fieldata code. If the amount of data to be printed on one line is less than 128 characters, the user program should place a stop character (a character of all ones) in the character position in the print buffer immediately following the last valid character. This will cause all of the remaining data in the buffer to be ignored when the line is printed.

If an initializing operation has been requested, the Print operation is affected as follows:

- In the case of the first Print operation following initialization, the line is printed on the first line of the printing area regardless of the spacing that is specified in the Print operation.

- If the Print operation specifies that the base address of the print buffer is zero, no printing will occur, spacing will be ignored, and the paper will be moved to the top line of the next page. The next operation may be either Print or Initialize Printer. If the next operation is a Print operation, the previously defined top margin will be honored and the line will be printed on the first line of the printing area. The primary purpose of this option is to permit a program to position the paper without operator intervention so that a new initializing operation may be requested.

- As lines are printed the input/output routine remains cognizant of the paper position relative to initialization. When it is determined that the line about to be printed will fall into the area reserved for the bottom margin, spacing will be ignored, the paper will be moved to the next page, and the line will be printed on the first line of the printing area.

- If "next page" (NP) is specified in the Print operation, the paper is moved to the next page and the line is printed on the first line of the printing area.

If an initializing operation has been requested in which the physical page length, the top margin, and the bottom margin each have a value of zero; the only spacing performed is that specified in the Print operation.

2. High-Speed Printer Status Words

The status of every operation that is initiated will be indicated by means of a status code placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be completed successfully because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

a. Status Indications

CODE	MEANS	ERROR MESSAGE
00000 00001	Operation completed	
00000 00006	Interlock fault	MSG 230 P
00000 00007	Illegal error	MSG 231 P
00000 00010	Illegal function	MSG 231 P
00000 00011	Illegal unit select	MSG 231 P
00000 00012	Print error	MSG 231 P

P = Program identification number

b. Explanation of Status Indications

01	Operation successfully completed.
06	A mechanical fault in the Printer.
07	A meaningless interrupt code was received following an operation.
10	The control unit has received an invalid function code.
11	An operation has been requested on a unit that is not available.
12	This indication will result if any equipment errors occur, including a Channel Synchronizer sequence or character count error, during a Print operation.

G. CARD MNEMONICS

Card initializing and input/output operations are requested by the submission of the CARD operator.

The initializing operations do not cause any Card Reader or Card Punch action. They are used to specify the mode for the input operations that will follow on the Card Reader and the output operations that will follow on the Card Punch. The mode for the Card Reader or the Card Punch will remain in effect until it is redefined by another initializing operation. The general format of the CARD operator when an initializing operation is requested is as follows:

I	w	v_0	v_1
	CARD	initialized unit	input or output mode

v_0 indicates the unit to be initialized. See section 6.G.1, Card Operations.

v_1 specifies the input or output mode.

The input/output operations activate the Card Reader or Card Punch and effect the transfer of data between these units and the Central Processor. The general format of the CARD operator when an input/output operation is requested is as follows:

I	w	v_0	v_1	v_2
	CARD	operation	buffer control word or number of cards	buffer control word

v_0 names the input/output operation to be performed. See section 6.G.1., Card Operations.

v_1 defines the address of the buffer control word or is a constant value equal to the buffer control word. The lower portion of the buffer control word specifies the first address used to hold input or output data and the upper specifies the last address.

For the Multiple Read operation, this operand indicates the number of cards to be read. The maximum number of cards that may be specified is 63 decimal or 77 octal.

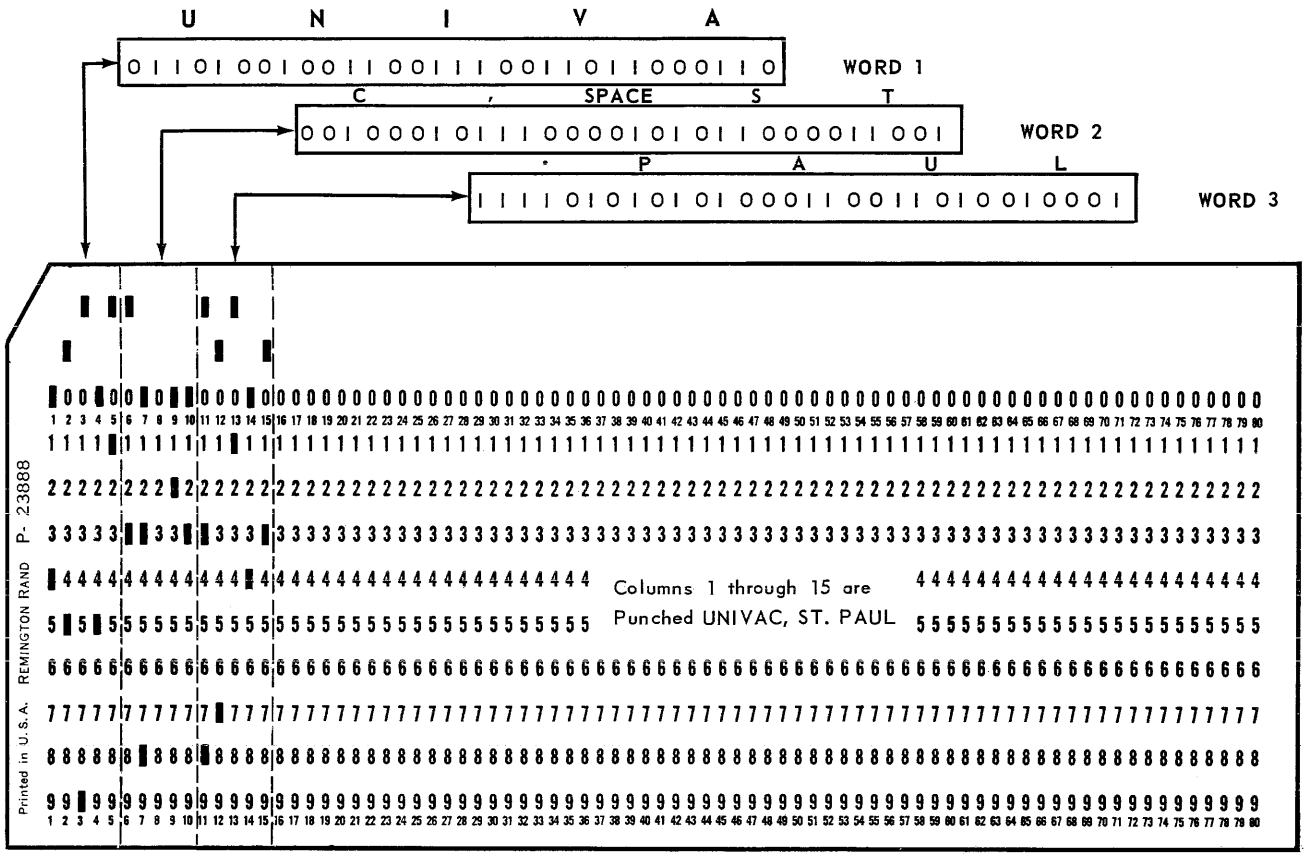
v_2 this operand is used only with Multiple Read operation. It defines the address of the buffer control word in the same manner as the v_1 operand does for other operations.

1. Card Operations

OPERATION	TYPE OF OPERATION	MNEMONIC(S)	V-OPERANDS USED
Set Input Mode Translation	Initializing, Card Reader	RMODE . FD	0, 1
Set Input Mode Column Binary	Initializing, Card Reader	RMODE . CBIN	0, 1
Set Input Mode Row Binary	Initializing, Card Reader	RMODE . RBIN	0, 1
Set Output Mode Translation	Initializing, Card Punch	PMODE . FD	0, 1
Set Output Mode Column Binary	Initializing, Card Punch	PMODE . CBIN	0, 1
Set Output Mode Row Binary	Initializing Card Punch	PMODE . RBIN	0, 1
Select Stacker 1	Input	STACK 1	0
Select Stacker 2	Input	STACK 2	0
Read Without Transfer	Input	READNT	0
Read With Transfer	Input	READ	0, 1
Transfer	Input	TRANS	0, 1
Multiple Read	Input	MREAD	0, 1, 2
Punch	Output	PUNCH	0, 1
Punch Select	Output	PUNCHS	0, 1

- Set Input Mode Translation

This operation establishes a mode which translates card data, during input transfer, into Fieldata code. Each card column will be translated into a 6-bit code; consequently, five columns will be packed into one computer word. This concept is illustrated in the following diagram.



The data transfer to the Central Processor in this mode requires a minimum buffer area of 16 words for each card.

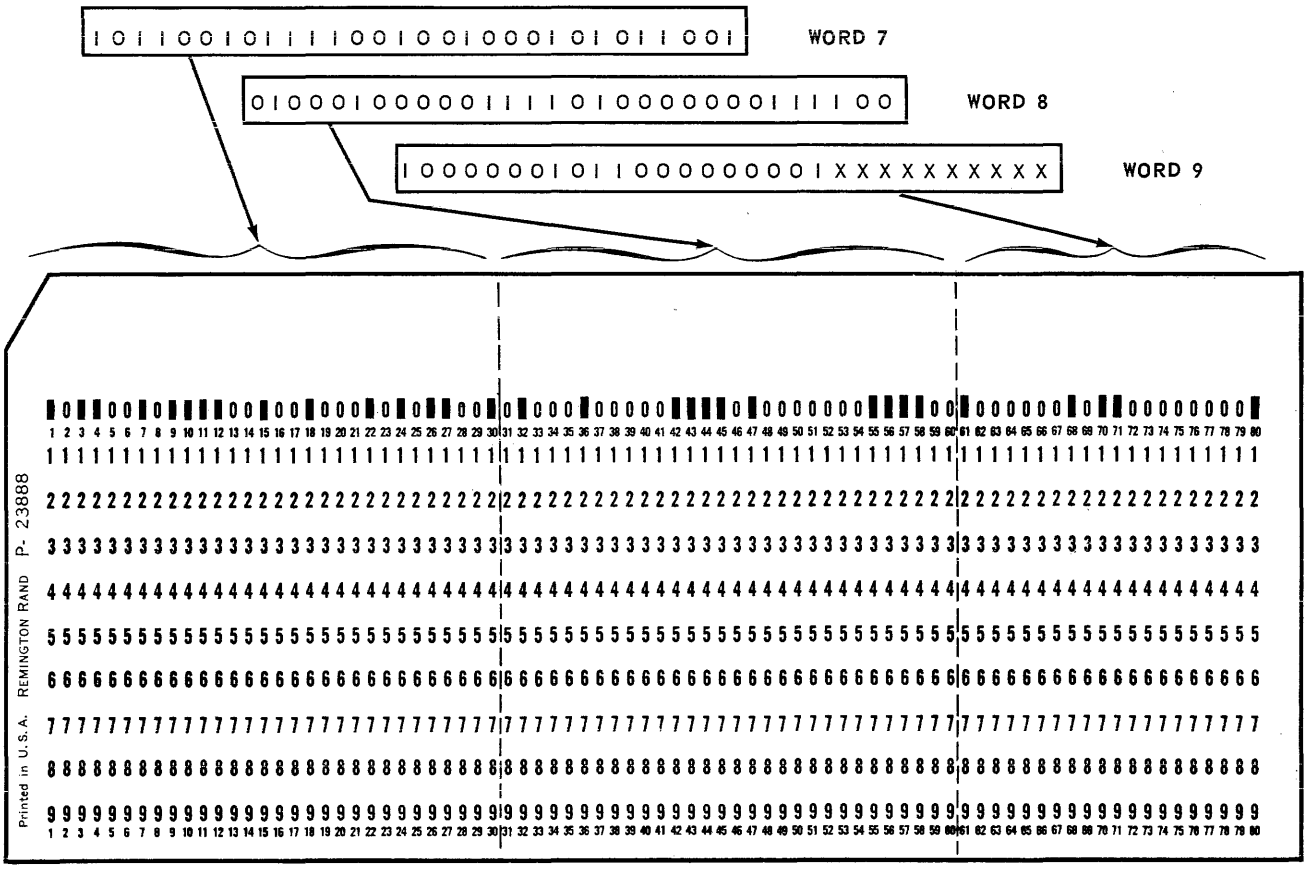
- Set Input Mode Column Binary

  This operation establishes a mode that will enable card data to be read in column binary image for all subsequent input transfers. As shown in the diagram that follows, data from 2-1/2 columns are packed into one computer word.

When data are transferred to the Central Processor in this mode, a minimum buffer area of 32 words is required for each card.

- Set Input Mode Row Binary

  This operation establishes a mode that will enable card data to be read in row binary image for all subsequent input transfers. As shown in the diagram that follows, each row of a card (80 bits) will be contained in three computer words. Columns 1 through 30 will be contained in one word, columns 31 through 60 will be contained in one word, and columns 61 through 80 will be contained in the 20 most significant bit positions of the third word (the 10 least significant bit positions of this word are hash filled).



- Set Output Mode Translation

  This operation establishes a mode whereby Fieldata is translated to card coding during output transfer. When this mode is used during card punching operations, a buffer area of 16 words is required for each card. The output data will be punched on the card as shown in the diagram that is included in the Set Input Mode Translation operation description.

- Set Output Mode Column Binary

  This operation establishes a mode which enables cards to be punched in column binary image for all subsequent punch operations. When this mode is used during card punching operations, a buffer area of 32 words is required for each card. The output data contained in one computer word will be punched on 2-1/2 columns as shown in the diagram that is included in the Set Input Mode Column Binary operation description.

- Set Output Mode Row Binary

  This operation establishes a mode which enables cards to be punched in row binary image for all subsequent punch operations. When this mode is used during card punching operations, a buffer area of 36 words is required for each card. The output data that is contained in three computer words, with the exception of the data in the 10 least significant bit positions of the third word, will be punched on the card in one row as shown in the diagram that is included in the Set Input Mode Row Binary operation description.

- Select Stacker 1

  This operation causes the card, that has just passed the second read station in the Card Reader and from which data has been transferred to the Central Processor, to be routed to stacker 1.

- Select Stacker 2

  This operation is the same as the Select Stacker 1 operation except that the card is routed to stacker 2.

- Read Without Transfer

  This operation causes a card to feed and the data from the card to be placed in card memory. Data are not transferred to the Central Processor as in the case where a Read With Transfer, Multiple Read, or Transfer operation is requested. If a Read Without Transfer operation is requested when the card memory is full, the card feed will be inhibited and an Inappropriate Operation indication will be placed in the status word. See section 6.G.1.a., Card Reader Action.

- Read With Transfer

  This operation will transfer, from card memory to the specified buffer area in the Central Processor, the data from one card and then will read as many cards as the card memory can store. See section 6.G.1.a., Card Reader Action.

- Multiple Read

  This operation will read a specified number of cards and transfer the data from these cards to the specified buffer area in the Central Processor.

- Transfer

  This operation will transfer, from card memory to the specified buffer area in the Central Processor, the data from one card. During this operation the card feed is inactive; consequently, no additional cards are read into card memory.

  If this operation is requested when card memory is clear, an Inappropriate Operation indication will be placed in the status word. See section 6.G.1.a., Card Reader Action.

- Punch

  This operation causes a card to be punched with data from the specified buffer area in the Central Processor. The card will be routed to stacker 0.

- Punch Select

  This operation is the same as the Punch operation except that the card is routed to stacker 1.

  a. Card Reader Action

     The following chart indicates the Card Reader action that will result from requesting a card operation when the various initial conditions exist in the card memory.

OPERATION	INITIAL CARD MEMORY CONDITION			
	Clear	With Data Store From Card 1	With Data Store From Card 1 and Card 2	With Data Store From Card 1 Card 2, and Card 3
READ WITHOUT TRANSFER	1. Feed one card. 2. Store data from the card in card memory.	1. Feed one card. 2. Advance card 1 data in sequence. 3. Store data from card 2 in card memory.	1. Feed one card. 2. Advance card 1 and card 2 data in sequence. 3. Store data from card 3 in card memory.	Inappropriate operation.
READ WITH TRANSFER	1. Feed four cards. 2. Transfer data from card 1 to the Central Processor. 3. Store data from cards 2, 3, and 4 in the card memory.	1. Transfer data from card 1 to the Central Processor. 2. Feed three cards. 3. Store data from cards 2, 3, and 4 in card memory.	1. Transfer data from card 1 to the Central Processor. 2. Advance card 2 data in sequence. 3. Feed two cards. 4. Store data from cards 3 and 4 in card memory.	1. Transfer data from card 1 to the Central Processor. 2. Advance card 2 and card 3 data in sequence. 3. Feed one card. 4. Store data from card 4 in card memory.
TRANSFER	Inappropriate operation	Transfer data from card 1 to the Central Processor and clear card memory.	1. Transfer data from card 1 to the Central Processor. 2. Advance card 2 data in sequence.	1. Transfer data from card 1 to the Central Processor. 2. Advance card 2 and card 3 data in sequence.

2. Card Status Words

The status of every operation that is initiated will be indicated by means of a status code placed in bit positions 0-5 of the A-Register.

If the requested operation cannot be completed successfully because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

In some instances supplementary information is placed in bit positions 15-20 of the A-Register along with the status code; that is, when an interruption occurs during a Multiple Read operation, the number of cards (c) from which data has been transferred to the Central Processor will be indicated.

a. Status Indications

CODE	MEANS	ERROR MESSAGE
00000 00001	Operation completed	
00000 00006	Interlock fault	MSG 260/270 P
00000 00007	Verify error	MSG 263/273 P
000cc 00010	Illegal character	
000cc 00011	Illegal function	MSG 261/271 P
00000 00012	Inappropriate operation	MSG 264 PSS[1]
00000 00013	Incorrect parameter	MSG 262/272 P

P = Program identification number

[1] *Only on Late Stacker Selection. SS is Stacker Number.*

b. Explanation of Status Indications

01 Operation completed.

06 This indication will result when any of the following conditions occur:

- Full card stackers
- Card jam.
- Empty hopper
- Card misfeed.
- Abnormal switch action.
- Parity error.
- Reader drive motor off.
- An attempt is made to punch more than 240 holes in a card.

07 This indication will result when:

- The card data most recently transferred to the Central Processor did not read properly.
- A card is not punched correctly; that is, the data punched into the card does not agree with the data presented to be punched. If the automatic recovery switch is on and a card is punched in error, that card, when it is detected at the verification station will be routed to stacker 1 along with the two following cards. At this point, the operation is reinitiated and an attempt is made to punch the data correctly on a new card. If successful, this card is routed to the proper stacker. In the event that the data cannot be punched correctly on this second try, the Card Punch unit will stop.

10 This indication will result when:

- An illegal character code has been detected within the translated data most recently transferred to the Central Processor. If this indication occurs during a Multiple Read operation, the operation will be terminated at the completion of the current transfer.
- An illegal character code was detected during the translation of the output data most recently punched.

11 This indication will result when the control unit attempts to perform an invalid function.

12 This indication will result if:

- A Select Stacker 1 or Select Stacker 2 operation is initiated too late; that is, the operation is presented more than 80 milliseconds after the card, to which the operation applies, has passed the second read station. In this case the card will be routed to Stacker 0.
- A Read Without Transfer operation is initiated when card memory is full.
- A Transfer operation is initiated when card memory is clear.

13 An incorrect parameter such as:

- An invalid operation code.
- The buffer area is not sufficient.

H. FASTRAND MNEMONICS

FASTRAND input/output and positioning operations are requested by the submission of the FAST operator. The general format of this operator is as follows:

I	W	V_0	V_1	V_2	V_3	V_4	V_5
	FAST	subsystem name	operation	buffer control word	sector address	search length indicator	identifier word

V_0 defines the name of the FASTRAND subsystem upon which the operation will be performed. If an installation has a FASTRAND subsystem on one channel only, this operand is unnecessary. See Section 9, PROGRAM PREPARATION.

V_1 names the operation to be performed. See Section 6.H.1., FASTRAND Operations.

V_2 defines the address of the buffer control word. The lower portion of the buffer control word specifies the first address to hold input or output data and the upper portion specifies the last address.

This operand is not required for Position operations.

V_3 defines the first address of the sector (an area that can hold 33 computer words) that is associated with the operation. This may be the label of a location in memory that contains a sector address.

V_4 is the search length indicator used for Search operations. This may be the label of a location in memory that contains the search length indicator.

If the search indicator is zero, the search will commence at a specified sector within a position range (an area that contains 4096 sectors) and continue from this point through each succeeding sector until discovery is made or the end of the position range is reached.

If the search indicator is unequal to zero, the search will commence at a specified sector within a track range (an area that contains 64 sectors) and continue through each succeeding sector until discovery is made or the end of the track range is reached.

This operand is not required for Read, Write, or Position operations.

V_5 is the search identifier used for Search operations. This may be the label of a location in memory where the identifier may be found.

This operand is not required for Read, Write, or Position operations.

1. FASTRAND Operations

OPERATION	TYPE OF OPERATION	MNEMONICS	V-OPERANDS USED
Continuous Write	Input/Output	WRITE	0, 1, 2, 3
Continuous Read	Input/Output	READ	0, 1, 2, 3
Search Read Sector	Input/Output	SEARCH	0, 1, 2, 3, 4, 5
Search Read Words	Input/Output	WSEARCH	0, 1, 2, 3, 4, 5
Position	Positioning	POSITION	0, 1, 3

- Continuous Write

  Starting with a specified sector, data is written on a FASTRAND unit in consecutive locations until the output buffer is exhausted.

- Continuous Read

  Starting with a specified sector, data is read from a FASTRAND unit until the input buffer is filled.

- Search Read Sector

  Starting with a specified sector, this operation causes the first word of each sector to be searched for a word that is identical to the identifier word until discovery is made or the search is terminated.

  If discovery is made, a Continuous Read operation commences with the find word.

  If discovery is not made, the search will be terminated at the point determined by the setting of the search length indicator; that is, when the end of the position range or the end of the track range is reached.

- Search Read Words

  Starting with a specified sector, this operation causes each word within a sector to be searched for a word that is identical to the identifier word until discovery is made or the search is terminated.

  If discovery is made, a Continuous Read operation commences with the find word.

  If discovery is not made, the search will be terminated at the point determined by the setting of the search length indicator; that is, when the end of the position range or the end of the track range is reached.

- Position

  This operation causes the read-write head on a FASTRAND unit to be moved to a specified position. For this operation the sector address will be in the v_2 operand position.

2. FASTRAND Status Words

The status of every operation that is initiated will be indicated by means of a status code that is placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be successfully completed because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

In some instances supplementary information will be contained in the upper 23 bit positions of the A-Register along with the status code. This information will be the address (a) of the sector from which data was last transferred or at which the last comparison was made.

a. Status Indicators

CODE	MEANS	ERROR MESSAGE
aaaaa aaa01	Operation completed	
aaaaa aaa02	Unsuccessful search	
00000 00006	Unit interlocked	MSG 320 Pxx y
00000 00007	Subsystem error	MSG 321 Pxx y
00000 00010	Incorrect parameter	MSG 322 Pxx y

P = Program identification number

xx = Unit number

y = Last computer generated interrupt word. For operator and service use.

b. Explanation of Status Indications

01 Operation successfully completed.

02 This indication will follow a Search Read Sector or Search Read Words operation when the identifier word cannot be found. The sector address at which the last comparison was made will be placed in the upper 23 bit positions of the status word.

06 The unit for which the operation is intended is nonoperable because of an interlock condition.

07 A subsystem error such as:

- A malfunction of the control unit.
- The recurrence of data transfer errors.

10 An incorrect parameter such as:

- An invalid operation code.
- The lower half of the buffer control word is greater in value than the upper half.

3. FASTRAND Addressing

The actual sector addresses (see Appendix D) are continuous only within the individual FASTRAND units; however, these addresses have been modified so that when the FASTRAND mnemonics are used, the sector addresses are continuous throughout the subsystem. A listing of these modified addresses is provided in the table that follows:

UNIT	BEGINNING ADDRESS	ENDING ADDRESS
0	0000 0000	0137 7777
1	0140 0000	0277 7777
2	0300 0000	0437 7777
3	0440 0000	0577 7777
4	0600 0000	0737 7777
5	0740 0000	1077 7777
6	1100 0000	1237 7777
7	1240 0000	1377 7777

I. PAPER TAPE MNEMONICS

Input/output operations for 5,6,7, or 8 channel paper tape are requested by the submission of the PTAPE operator.

The general format of the PTAPE operator is as follows:

I	w	v_0	v_1	v_2
	PTAPE	subsystem name	operation	buffer control word

v_0 defines the name of the Paper Tape subsystem upon which the operation will be performed. If an installation has a Paper Tape subsystem on one channel only, this operand is unnecessary. See Section 9, PROGRAM PREPARATION.

v_1 names the operation to be performed. See Section 6.I.1., Paper Tape Operations.

v_2 defines the address of the buffer control word. The lower portion of the buffer control word specifies the first address used to hold input or output data and the upper portion specifies the last address.

1. Paper Tape Operations

OPERATION	TYPE OF OPERATION	MNEMONICS	V-OPERANDS USED
Punch Tape	Input/Output	PUNCH	0, 1, 2
Read Tape Forward	Input/Output	READF	0, 1, 2
Read Tape Backward	Input/Output	READB	0, 1, 2

- Punch Tape

  This operation causes the paper tape to be punched with data from the specified area in the Central Processor. The transfer of data is made on the basis of one character per output buffer word; that is, the bit configuration that is contained in the low order bit positions of one output buffer word will be punched into the channels of one frame of paper tape.

- Read Tape Forward

  This operation will transfer the data contained on paper tape to be transferred to the specified buffer area in the Central Processor. The transfer of data is made on the basis of one character per input buffer word; that is, the bit configuration that represents the combination of holes in the channels of one frame of paper tape will be placed in the low order bit positions of one input buffer word.

- Read Tape Backward

  This operation is the same as the Read Tape Forward operation with the exception that the order of the placement of data in the input buffer is reversed.

2. Paper Tape Status Words

The status of every operation that is initiated will be indicated by means of a status code that is placed in bit positions 0-5 of the A-Register when program control is returned to the user program.

If the requested operation cannot be successfully completed because of an error condition, the applicable status code will be placed in the A-Register and an error message will be relayed via the Console Printer.

a. Status Indications

CODE	MEANS	ERROR MESSAGE*
00000 00001	Operation completed	
00000 00006	Unit interlocked	MSG 300/310 P
00000 00007	Subsystem error	MSG 301/311 P
00000 00010	Incorrect parameter	MSG 302/312 P

P = Program identification number

b. Explanation of Status Indications

01 Operation successfully completed.

06 This indication will result when the Paper Tape Reader or Punch is out of tape.

07 A subsystem error such as:

- A punch compare error.
- Excessive tape skew.

10 An incorrect parameter such as:

- The lower half of the buffer control word is greater in value than the upper half.
- The buffer control word contains all zeros.

* *MSG 307 P will appear if an interrupt that was not anticipated occurs on the paper tape input/output channel during the execution of an operation.*

7. CONSOLE PRINTER CONTROL

During program execution or upon termination, it may be necessary for a program to print short informative messages or the contents of specified registers and locations on the Console Printer. In addition, it may also be necessary to communicate with the running program via the Console Keyboard. The sections that follow describe the means by which console input and output is effected.

A. CONSOLE OUTPUT OPERATIONS

Output operations are used at various points in the program to relay information such as notification that an error has occurred or instructions for the operator, or they are used to print the contents of specified registers and locations. In many cases the program will require that several messages be printed contiguously. To effect this in a complex environment where several programs are competing for the use of the Console Printer, it is necessary to utilize operations that will reserve the Console Printer for the exclusive use of the requesting program and then release it for other use when the message sequence is completed.

■ CONSOLE HOLD

This operation reserves the Console Printer for the exclusive use of the requesting program. If the requesting program desires to submit several messages and have them contiguous, it is necessary that a Console Hold operation be initiated before the message sequence is requested.

The Console Hold operation is initiated as follows:

W	V_0
CONSOLE	HOLD

■ TYPE TEXT

This operation causes a specified message to be printed by the Console Printer. In addition to printing the message, the Printer will also execute any printer commands that have been included with the message. The Type Text operation is requested as follows:

W	V_0
TYPET	message and printer commands

v_0 specifies the message to be printed and the printer commands that are to be executed. A message may contain a maximum of 70 characters. Each printer command that is included in a message is counted as one character. These commands and their symbols are as follows:

Carriage Return and Line Feed	|CR|
Line Feed	|LF|
Space	|SP| or Δ

When these commands are used with the Type Text operation they are separated from the text by means of a vertical bar, |, that is placed before and after each command. The exception to this rule is when the symbol Δ is used to indicate that a space is desired. For example:

TYPET . text Δ text |CR| |LF| |LF| text |SP| text

It should also be noted that a tab symbol should be used to indicate the end of the message to be printed. This symbol should be placed after the last valid character in the message.

TYPE

This operation is the same as the Type Text operation with the exception that the message to be printed is not included in the operation request. Instead, the user specifies the number of characters in the message and the initial address of an area where the message (in Fieldata code) has been stored.

The Type operation is requested as follows:

W	V_0	V_1
TYPE	number of characters	initial address of message area

v_0 specifies the number of characters that are contained in the message. A message may contain a maximum of 70 characters. Each printer command that is included in a message is counted as one character. Since the number of characters contained in the message is specified, a tab symbol does not have to be used to indicate the end of the message.

v_1 specifies the initial address of the area where the message is stored.

- TYPE CONTENTS

This operation causes the octal coded contents of specified registers and/or storage locations to be printed by the Console Printer. In addition to printing the requested information, the printer will also execute any printer commands that have been included in the request. The Type Contents operation is requested as follows:

W	V_O
TYPEC	information to be printed and printer commands

V_O specifies the information to be printed and the printer commands that are to be executed. The information to be printed may be the contents of the upper half, the lower half, or the whole word in the specified storage location. It may also be the contents of the A, Q, or B registers. Each operand that specifies information to be printed and each printer command must be preceded by a point separator. For example:

```
TYPEC . W(ALPHA) . |SP| . U(BETA + B3 - 6) . |CR|
      . A . |SP| . Q . |SP| . L(GAMMA) . |CR| . |LF|
      . W(B2)
```

- CONSOLE RELEASE

This operation is used to terminate an existing hold mode when it is no longer needed. The Console Release operation is requested as follows:

W	V_O
CONSOLE	RELEASE

Examples of Console Output Operations:

For each independent console output request REX attaches a double line feed, a program identification number (Pxx), and one space to precede the supplied data. If the following independent output requests were submitted, where COUNTER is the label of a location whose upper half contains the number of cards processed and the lower half contains the actual number of cards,

```
TYPET . STARTΔOFΔJOB |CR| NR.10576Δ
TYPET . PROGRAMΔCARDΔCOUNT
TYPEC . U(COUNTER)
TYPET . CARDΔCOUNT
TYPEC . L(COUNTER)
TYPET . JOBΔ10575ΔCOMPLETE
```

the console output would appear as

```
Pxx START OF JOB
    NR. 10576

Pxx PROGRAM CARD COUNT
```

```
Pxx nnnnn
Pxx CARD COUNT
Pxx nnnnn
Pxx JOB 10575 COMPLETE
```

Since the line feed and program number are automatically supplied for each independent request, a |CR| as the first character is ignored. All other |CR| are honored (as shown) with a carriage return, line feed, and a four space indentation.

If a hold mode is established for the purpose of printing several contiguous messages, all of the console output requests in the message string will be considered as one independent request. If the output requests used in the previous example were submitted as follows:

```
CONSOLE . HOLD
TYPET . STARTΔOFΔJOB |CR| NR.10576Δ
TYPET . PROGRAMΔCARDΔCOUNT
TYPEC . U(COUNTER)
TYPET . CARDΔCOUNT
TYPEC . L(COUNTER). |SP|
TYPET . JOBΔ10575ΔCOMPLETE
```

the console output would appear as

```
Pxx START OF JOB
    NR.10576 PROGRAM CARD COUNTnnnnnCARD COUNTnnnnn JOB 10575 COMPLETE
```

B. CONSOLE INPUT OPERATIONS

Console input operations are used to condition REX to expect Console Keyboard input. This input is usually a response to a message that has been printed on the Console Printer.

■ ACCEPT CONSOLE KEYBOARD INPUT

The Accept Console Keyboard Input operation is requested as follows:

W	V_0	V_1	V_2
ACCEPT	number of characters	first address of buffer	alternate address specifier

V_0 specifies the maximum number of characters that will be accepted. A count of the number of characters that are typed is maintained. If this count becomes greater than the number of characters specified, the entry will be rejected with an explanatory message. If this occurs, the operator may attempt entry again. All characters that are typed, with the exception of the carriage return and backspace characters, are entered into the input buffer. Each character will be checked to see if it is either a stop or backspace character. If a stop character is detected, it will be counted and stored in the buffer and the accept mode will be terminated. If a backspace character is detected, the previous character that was typed will be erased from the buffer. If three consecutive backspace characters are typed, the entire entry will be erased.

V_1 specifies the first address of the input buffer.

V_2 an address to which control will immediately pass. Since some period of time may elapse while a response is being composed, this option provides a means whereby REX can retain control after an Accept mode is established.

If an alternate address is not specified, the program will be suspended until the entry is completed. At that point control will be transferred to the address of the next SPURT instruction.

If an alternate address is specified, REX will immediately transfer control to this address. When the response to the Accept operation has been completed, control will be transferred to the address of the next SPURT instruction.

As can be readily seen, the acceptance of input via the Console Keyboard operates in the same manner as a request for other standard peripheral input/output with the exception that in this case the functions of the input/output request and the CKSTAT are combined.

It should also be noted that the establishment of the Accept mode will terminate a previously imposed hold mode.

Examples Of Console Input Operations:

Normally, a console input request is preceded by one or more output requests that describe the entry to be made. REX assigns a delay number (Dxx) to each input request. At the time a request is made, this number will be printed on the Console Printer. When the operator responds to the request, this number will be included in the response. Whether or not an input request is an independent request or part of a hold mode determines how it will appear on the Console Printer.

If an input request is an independent request the delay number will have a program identification number assigned to it. For example, if the following independent requests were submitted.

```
TYPET . MOUNTΔTHEΔREELΔLABELEDΔ1A |CR| ONΔUNITΔ2ΔCHANNELΔ13Δ
TYPET . TYPEΔYESΔWHENΔREELΔISΔMOUNTED
ACCEPT . 4 . REELCHANGE . WAIT1
```

the console output would appear as

```
Pxx MOUNT THE REEL LABELED 1A
    ON UNIT 2 CHANNEL 13
Pxx TYPE YES WHEN REEL IS MOUNTED
Pxx Dxx
```

If an input request is included within a hold mode, the delay number will not have a program identification number assigned to it. For example, if the same requests used in the previous example were submitted as follows

```
CONSOLE . HOLD
TYPET . MOUNTΔTHEΔREELΔLABELEDΔ1A |CR| ONΔUNITΔ2ΔCHANNELΔ13Δ
TYPET . TYPEΔYESΔWHENΔREELΔISΔMOUNTED
ACCEPT . 4 . REELCHANGE . WAIT1
```

the console output would appear as

```
Pxx MOUNT THE REEL LABELED 1A
    ON UNIT 2 CHANNEL 13 TYPE YES WHEN REEL IS MOUNTED Dxx
```

For both examples the operator response would be

Dxx ☐ YES Ⓢ

8. PROGRAM DEFINED MACRO OPERATIONS

SPURT provides the user with the means whereby a single line of SPURT coding can be utilized to incorporate a previously defined operation within a program. Operations of this type are called macro operations and will usually contain several basic instructions that perform a common task.

Each macro operation consists of a unique label and the basic instructions necessary to define the operation. Any portion of the instructions of which the macro operation is composed may be made variable in order that the operation may be modified or linked to other instructions in the program.

The line of SPURT coding that will incorporate a macro operation in the program is called a macro call line. This line consists of a standard label, the label of the macro operation in the operator position, and a series of operands that will replace the variable portions of the associated macro operation when it is called for by the program.

During assembly, drum storage is utilized to hold the macro operations until such time as they are called for by the program. Prior to this time the macro operations will not produce any object coding. When a macro operation is called for by the program, the instructions that are contained therein will be incorporated into the program at the point of call.

A. DEFINING A MACRO OPERATION

Except for the ability to insert variable operands, a macro operation is defined in the same manner as any other operation that is contained within a program. The coding defining a macro operation is preceded by a MACRO statement and terminated by an ENDMAC statement. These statements cause the enclosed coding to be set aside as a macro operation.

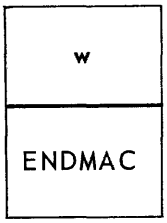
1. The MACRO Statement

l	w	v_o
macro name	MACRO	notation

l the label that is assigned to the MACRO statement will become the name of the macro operation. This label will be placed in the operator position of a call statement which brings the macro coding into the program.

v_o the operand position of this statement is ignored. Notation may be included at this point describing the variable parameters that will be defined in the call line.

2. The ENDMAC Statement



The ENDMAC statement follows the last line of coding in the macro operation.

3. Variable Parameters

Those portions of the coding in a macro definition that are to be substituted when the defined macro operation is called are represented by a number (n) enclosed in commas (,n,). The enclosed number specifies which parameter listed on the macro call line is to be substituted when the macro coding is placed in the program. The parameters on the macro call line are numbered from left to right starting with the number 1.

4. Examples of MACRO Definition

Presented below are examples of two macros which could be used to move a memory area. The notation in the macro line indicates the order in which the parameters should appear on the macro call line.

a.

```
l          w                                    NOTES
MOVE1      MACRO. (number of words, pick up address, deposit address)
           ENT . B7 . ,1,
           BJP . B7 . ALPHA
ALPHA      ENT . Q.. W (,2, + B7)
           STR . Q . W (,3, + B7)
           BJP . B7 . ALPHA
           ENDMAC
```

b.

```
l          w
MOVE2      MACRO . (register, index, number words, from address, to address, label)
           ENT    . ,2, . ,3,
           BJP    . ,2, . BETA
,6,        ENT    . ,1, . W (,4, + ,2,)
           STR    . ,1, . W (,5, + ,2,)
           BJP    . ,2, . ,6,
           ENDMAC
```

B. CALLING A MACRO

1. The Call Line

A previously defined macro operation may be incorporated into a program by using the macro name in the operator position. This statement is referred to as a call line and is written as follows:

l	w	$v_0 \ldots v_n$
	macro name	parameter 1, parameter 2, . . . parameter n

w is the macro name that was used in the label position of the MACRO statement. The use of the macro name in this position causes the coding that was previously defined to be called into the program.

v_0 the variable parameters that were previously provided for in the coding are here defined. Parameter 1 corresponds to ,1,, parameter 2 corresponds to ,2,; parameter n corresponds to ,n,. The parameters are separated from the macro name by a standard point separation symbol. Parameters are separated from each other by a comma.

The absolute maximum number of parameters that may be defined is 24. However, if the size of a number of the operands is exceptionally large, the area provided for the storage of parameters could be exhausted below this limit. A reasonably safe maximum would be 20.

When a macro operation is inserted in the coding of a program, each instruction is examined for a number enclosed in commas in any of the positions of the statement. If such a number is encountered, the parameter in the corresponding position of the call line is inserted by character substitution. The resulting coding must not violate any of the rules for standard SPURT coding, and if the operator field is replaced it must be a legitimate operator consisting of a maximum of ten characters which cannot be a macro name. The call line must be terminated by a standard end statement character.

If a label is used on a macro call line it will be assigned to the first instruction of the inserted coding.

2. Examples of Macro Call Line

The examples of call lines presented below refer to the examples of macro definitions presented previously. The call line on the left and the resultant coding that will be inserted in the program is shown on the right. This is not a representation of a High Speed Printer listing. Only the object coding that results from the insertion of the macro coding and the call line itself will appear on a printed listing.

MACRO CALL		RESULTANT CODING	
l	w		
PHI	MOVE1 . 10, CAT, DOG	PHI	ENT . B7 . 10
			BJP . B7 . ALPHA
		ALPHA	ENT . Q . W (CAT + B7)
			STR . Q . W (DOG + B7)
			BJP . B7 . ALPHA
	MOVE2.. Q, B1, 4, THETA, DOG, BETA		ENT . B1 . 4
			BJP . B1 . BETA
		BETA	ENT . Q . W (THETA + B1)
			STR . Q . W (DOG + B1)
			BJP . B1 . BETA

C. NESTING OF MACROS

A macro may be defined that uses one or more macros that have been previously defined. This is called nesting. The nesting procedure is handled similar to that of a simple macro definition. A simple nest may be represented as follows:

```
l       w
MAC1    MACRO
          ≀
        ENDMAC
```

```
l       w
MAC2    MACRO
          ≀
        ENDMAC
```

```
l       w
MAC3    MACRO
          ≀
        MAC2 . OPA, OPB, OPC
          ≀
        MAC1 . OPD, /4, OPE, /5
        ENDMAC
```

When MAC3 is called, MAC2 and MAC1 will also be called and incorporated into the coding. The parameters OPA, OPB, OPC, OPD, and OPE are simple parameters as previously described for a call line. The slash (/) followed by a number (n), will cause the n th parameter in the call line which calls the macro being defined to be included in the coding of the nested macro. For example, let us assume MAC3 is called by the following statement:

```
      W
MAC3 . OPF, OPG, OPH, OPI, OPJ
```

The parameter /4 in turn refers to the fourth parameter of the MAC3 call line (OPI), and /5 will refer to the fifth parameter (OPJ).

D. LABEL REFERENCE WITHIN MACROS

Consideration should be given to the following rules when using labels and referencing labels within macros.

- A macro may refer to any label that exists in the main line of coding.
- A macro may refer to any label within the macro itself.
- The main line of coding can reference a label within a macro only if that label is unique throughout the program. If a macro definition is used more than once, a label within a macro cannot be unique unless it is replaced by a parameter label from the referencing call line (/n option).
- A nested macro may refer to a label within the same macro, or a label within a macro within the same nest of macros, or a label in the main line of coding. Where a label is duplicated within a nest of macros, a reference to that label will result in a reference to the label within that macro that is closest to the macro currently being processed.

9. PROGRAM PREPARATION

The individual operations that were described in the previous sections are organized into a logical statement or a series of logical statements which is called a program. In addition to the individual operations, various statements are included in the program definition to control and direct the assembly process, or to communicate with the Real-Time Executive Routine.

This section provides the necessary information to prepare a program for submission to the SPURT Assembly System. The required statements for operation of a program under control of the Real-Time Executive Routine are also described.

A. SPURT REQUIREMENTS

1. The Program Header

A SPURT Program must be preceded by a PROGRAM header in the following format:

l	w	v_0	v_1
program name	PROGRAM	programmer name	date

The first line of coding following the header line *must* be labelled. Program headers other than the first may be used as a convenience to mark points within the program. These additional program headers will appear in printed listings of the source program but will have no effect in the assembly process.

2. Allocation

Allocation is a process which assigns numeric values to labels. Labels that appear within a program will be automatically assigned by the assembler as instructions are generated and storage areas are reserved. This assignment is relative to a base value that is assigned by allocation.

a. Automatic Allocation

The first instruction of a program will automatically be allocated to a base of 00000, if no allocation is made by the user.

b. Required Allocation

Allocation by the user is required:

- if a program in absolute format is desired. An absolute program is loaded at a particular address in computer memory. Therefore, the first instruction of the program must be allocated to the base address desired.
- if a program refers to another program or subroutine that has been or is to be loaded into a particular portion of memory.
- when referencing storage areas that are not defined within the program.

c. Types of Allocation

- Direct Allocation

Direct allocation assigns a given numeric value to a label. Direct allocation input consists of a header statement to initiate the allocation process and to identify the type of allocation. The header statement is written as follows:

1	w	v_o	v_1
program name	ALLOCATION	programmer name	date

Following the header are allocation statements which are written as follows:

1	
allocated label	value

Example:

```
                I
-------------------------------------------
        DRUMRTE     ALLOCATION . JONES . 15 DEC 63
        RDINPUT     22600
        CONSTLIST   23546
        ERRORRTE    24677
```

- Relative Allocation

Relative allocation provides a technique for changing program allocation by entry of a base value when requested on the console printer. Allocation is made as described above for direct allocation except that the mnemonic REL-ALLOC is substituted for ALLOCATION in the header statement. The values allocated may be considered increments to the base value that is entered.

Example:

I	
TAPERTE	REL-ALLOC . JONES . 15 DEC 63
LABELRTE	33000
READERR	37200
WRITERR	37300

If a value of 20000 were entered when requested, the actual value allocated to LABELRTE would be 53000.

- Indirect Allocation

Indirect allocation is used in conjunction with direct allocation and the U-TAG operation to create a general table of entrances to subroutines. This allows several different programs to access the same subroutine, or another program, if programs are to be combined when executed. The address of the subroutine may be changed without requiring changes in the programs that refer to it.

As an example, let us assume that we have two subroutines for which we intend to make an indirect allocation. The subroutines are ENDFILE located at 32500, and ENDREEL located at 56455. The necessary operations are:

(1) The following entries are made as allocation input:

I	
TBLADR	04556
ENDFILE	32560
ENDREEL	56455

(2) A U-TAG operation is incorporated in the program as follows:

I	W	V_0	V_1
TBLADR	U-TAG	ENDFILE	ENDRELL

This establishes table entries which would appear in computer code as follows:

LOCATION	(2)	(1)
04556	32560	56455

(3) An indirect allocation reference is established as follows:

l	w
TAPRTNE	INDR-ALLOC . SMITH . 15 JAN 64
ENDFILE	204556
ENDREEL	104556

Indirect allocation is made under a header containing INDR-ALLOC in the w position. The higher order digit of the allocation value may be 1 or 2, referring respectively to the lower or upper portion of the word in the jump table.

Assume that a return jump instruction appears in the program as follows:

w
RJP . ENDFILE

This refers the program to the upper portion of word 04556, which contains the entrance address to the subroutine (32560). This entrance address may be changed by a reassembly of the original ALLOCATION tape with a different value for ENDFILE. No change is required in the program definition.

Only a jump (JP) or return jump (RJP) instruction may refer to an address defined by indirect allocation.

d. The EQUALS Statement

Allocation may be made within a program by the use of the EQUALS statement. By this means a label may be equated with previously defined labels and/or an increment or decrement. The format of the EQUALS statement is as follows:

l	w	v_o
label to be equated	EQUALS	label ± label ± constant value

l specifies the label to be equated.

v_o the equating operand may consist of:

- a previously defined label
- a label ± a constant
- a label + another label
- a label ± another label ± constant
- a constant alone

Labels referring to values which represent locations within a complex relative program will be modified relative to the base address at which the program is loaded. Labels referring to constant values will not be modified.

Any label set equal to a constant or to a label that has previously been set equal to a constant with an EQUALS statement will be taken as an absolute value and will not be modified when the program is loaded. If two labels are used, the equated label will assume the characteristics of the last label in the equating expression.

Example:

l	w	v_o	
CONSTANT	EQUALS . 40000		(1)
PLABEL	ENT . A . 0		(2)
COMB	EQUALS . PLABEL + CONSTANT + 1		(3)
ROMB	EQUALS . CONSTANT + PLABEL + 1		(4)

(1) The label represents a constant value of 40000.

(2) A standard program label.

(3) COMB will be treated as a constant.

(4) ROMB will be referenced as a program label and will be modified relative to the base address.

Note: The EQUALS statement may not be used to allocate a program label in a complex relative format. This would conflict with the assignment of labels that are made relative to a base address of 00000.

Example of allocation of program label:

l	w	v_o
SETA	EQUALS . 20000	
≀		
SETA	ENT . A . X77777	

B. REX REQUIREMENTS

1. The Executive Information Region

In order to facilitate exchange of information between worker programs and REX, the first five locations relative to the initial address of a worker program are assigned specific uses. These locations will be referred to as the Executive Information Region. Special information is required of the real-time program. This information is made available to REX via an Initialization Table which will be explained under REAL-TIME PROGRAMMING CONSIDERATIONS.

EXECUTIVE INFORMATION REGION

WORD	UPPER HALF	LOWER HALF
0	Label of program starting address	Entrance address of fault recovery routine. (note 1)
1	Address of relocatable area bounding address. (note 2)	No. of Addendum Storage Elements to be provided. (note 4)
2	(note 5)	
3	Address of unsolicited operator entry indicator word. For use during operation of the program.	Entrance address of addendum overflow recovery routine. (note 1)
4	Starting Address of parameter storage area. (note 3)	P-register value at time of fault/number of parameter words entered. For REX use.

NOTES:

1. Optional. If zero, REX will automatically suspend the program pending operator intervention.
2. Inserted by REX. Bounding core and bounding relocatable drum area addresses are preserved within a program's addendum (a storage area that is provided when a program is loaded). They will remain there until the program performs a console typeout of more than 50 characters.
3. Optional. It is required only if optional parameters are to be conveyed to the program at load time. The number of words loaded will be stored in the lower half of word 4.
4. An Addendum Storage Element is a 10-word temporary storage within a program's executive addendum (a storage area provided when a program is loaded). One element is placed into use each time a program performs one of the following operations:
   - Submits an input/output request
   - Uses ACCEPT
   - Requests a subroutine load
   - Requests a Batch Load
   - Calls a segment
   - Requests a core or drum memory dump
   - Has a time table routine started.

   An element is released for reuse each time the result of the operation which originally caused its use is reported to the program. That is, whenever control appears at the completion or error address associated with the operation.

Consider, for example, a typical input/output sequence. When the request is submitted a vacant storage element is located. B-register values and bookkeeping information are saved therein. When the request is processed the status word generated by the routine is saved within the same element. When the status of the request is interrogated, the status word is reported to the program and the element is freed. (Note, however, that if CKSTAT is used to mark a return point, the element is not freed but will remain in use until request status is subsequentially reported in response to program use of TAKEOVER).

It can readily be seen that if each time a program requested an operation it chose to wait until that operation was complete before requesting another, only one storage element would be required. If the program was more complex, additional elements would be required.

5. If the high order bit of this word is 0, there is normal treatment of tape errors; if 1, a logical lockout is imposed on tape errors. This option is described in more detail under CONTINGENCY CONTROL.

6. Unused fields should be set to zero.

2. Declaration of Facility Requirements

An operating program makes use of various facilities such as computer memory, drum area, and input/output channels with their associated peripheral units. When several programs are operating concurrently, under control of the Real-Time Executive Routine, the routine allocates facilities to assure their availability when a program is loaded.

Declarative statements, which generate no computer instructions, must be included in the SPURT coding to inform the executive routine of the facility requirements of the program. Facilities may be requested as fixed or relocatable. If fixed, specific assignments of areas, channels and units are made; if relocatable, the executive routine makes assignments from the available facilities.

Facility Statements must appear after the Executive Information Region and before any instructions that generate coding.

- ASSIGN

The ASSIGN statement provides a method of assigning several similar units to a channel. An example is magnetic tape units of which a maximum of twelve units may be connected to the same channel. The individual units may be defined as either fixed or relocatable. These mnemonic unit names may be referred to in the input/output mnemonic operations such as MTAPE. By declaring channel grouping in one statement, the user need not repeat the channel name in subsequent input/output mnemonic operations. The mnemonic names used must begin with an alphabetic character excluding O, and they may consist of a maximum of ten alphanumeric characters. The ASSIGN statement has the following format:

l	w	v_0
channel name	ASSIGN	unit name(s)

l specifies the mnemonic channel name to which all similar units are assigned. This must be the same channel name used in other declaratives, such as MEANS and FACIL, which give other information about the peripheral equipment connected to a given channel.

v_0 specifies the mnemonic name given to each unit or the names given to several units. A maximum of twelve unit names may be given for a single channel.

- Relocatable Units

  If the units are relocatable they will be given logical unit numbers by position. The first unit specified will be designated 0, the second unit will be designated 1, etc... This assignment may or may not exist when the object program is loaded depending upon the availability and utilization of existing facilities.

- Fixed Units

  If a fixed assignment of unit numbers is desired, the assignment is made by placing the fixed unit number within parentheses following the mnemonic unit designation. For example, if the mnemonic name SERVOA is specified for a unit, and if a fixed assignment to unit 5 is desired, the assignment will be written as follows:

  SERVOA (5)

  For a given channel, the units must be specified as either fixed or relocatable. It is not permissible to mix both fixed and relocatable units for a single ASSIGN statement.

■ MEANS

The MEANS statement provides for the assignment of a mnemonic name to either a fixed or relocatable channel. This mnemonic channel name is used in the ASSIGN and FACIL statements.

This operation generates no instructions in the final object program. An actual channel number is substituted for the mnemonic channel name.

If more than one MEANS statement is present with the same label, the first statement takes precedence.

The format for this statement is as follows:

!	w	v_0
channel name	MEANS	C. channel nbr
		F. channel nbr

1 a mnemonic name which will be replaced by the actual channel number. This is a standard label.

v_0 specifies the actual channel number that should be substituted wherever a mnemonic channel name is used. The channel number must in all cases be an octal number from the set 0-7, 10-15.

– Relocatable Channels

If the operand consists of a channel number preceded by the letter C, the channel number may be reassigned when the program is loaded.

– Fixed Channels

If the operand consists of a channel number preceded by the letter F, the channel number is fixed.

- FACIL

The facility statement provides necessary information to the Real-Time Executive Routine concerning the channels, peripheral units, and the number of units required for each program. In addition, provision is also made for the declaration of additional units, core storage, or drum storage, if a program is designed so that it can make use of additional facilities. There must be one facility statement for each channel required by a program. These declarative statements are mandatory for programs that will be controlled by the Real-Time Executive Routine. A complex relative program format is required when a program operates under control of the Real-Time Executive Routine. A facility record, which is part of the complex relative format, is created from the information contained in the facility statement. The format of the facility statement is as follows:

!	w	v_0	v_1	v_2	v_3
channel name	FACIL	facility	type of assignment	number of units or area definition	maximum number of units or area

1 channel name is the mnemonic name given to the channel. It is the same name used with the ASSIGN and MEANS operation. It is required for all statements except where the hardware name is CORE.

v_0 the facility name is a fixed name describing the type of peripheral equipment connected to the channel. The possible facility names are listed below:

NAME	TYPE
CORE	Core area
DRUM	Drum area
DISC	Disc area
FAST	Fastrand area
MTAPE	UNISERVO IIA units
UTAPE	UNISERVO IIIA units
CTAPE	UNISERVO IIIC units
CARDR	Card Reader
CARDP	Card Punch
PRINT	Printer
PTAPER	Paper Tape Reader
PTAPEP	Paper Tape Punch
COMM	Communications

v_1 this operand specifies whether the units are fixed or relocatable when the program is loaded.

F indicates fixed assignment at load time.

R indicates relocatable assignment at load time.

v_2 this operand specifies the number of units required on a channel for all facility types except CORE and DRUM.

For the facility name CORE, this operand specifies the number of relocatable core memory locations that could be used in addition to the total assigned to the program by the assembler.

For the facility name DRUM, the total relocatable storage area required is specified by this operand. This would be the sum of the relocatable area declared for a drum system by the DRUM-AREA statement. The use of the DRUM-AREA statement is explained in this section.

v_3 this operand specifies the maximum number of units for all facility names except DRUM. The Maximum relocatable area is specified if the facility name is DRUM.

This operand is not used for the facility name CORE. Additional core memory requirements are requested as described for the v_2 operand.

- Example of facility declaration for a tape system.

(1) Relocatable

```
!          w
TAPESYS1   MEANS . C . 12
TAPESYS1   ASSIGN . S1 . S2 . S3 . S4
TAPESYS1   FACIL . MTAPE . R . 4
```

(2) Fixed

```
!          w
TAPESYS1   MEANS . F . 12
TAPESYS1   ASSIGN . S1 (0) . S2 (1) . S3 (2) . S4 (3)
TAPESYS1   FACIL . MTAPE . F . 4
```

3. Drum and Fastrand Statement

- DRUM-AREA

The DRUM-AREA statement reserves a relative drum area for the use of the program within which it is defined. There may be many drum areas reserved with each drum system. The actual location of the drum area will be assigned at load time and will depend upon what area is available. The Real-Time Executive Routine will recognize from the label that the program requires a relocatable drum area. All areas defined for a drum system will be assigned according to the drum base address. The format for the DRUM-AREA statement is as follows:

w	v_0	v_1	v_2
DRUM-AREA	drum system name	area name	area length for DRUM
			number of sectors for FAST

v_0 specifies the name of the drum system as used in the ASSIGN statement. This operand may be omitted if an installation has all its drums on one channel. It is always required for Fastrand.

v_1 a standard alphanumeric label used as a reference for this particular drum area. Drum area names should never be used as an operand in the DRUM input/output operation. The actual input/output drum labels to be used in the DRUM operation are defined in the D-TAG statement.

v_2 specifies the number of locations to be reserved in the area.

- D-TAG

The D-TAG statement labels a point within a drum area or Fastrand area that has been defined by a DRUM-AREA statement. Drum area is assigned when the program is loaded. The D-TAG assignment is made relative to the assignment of the drum area. The following format is used for this statement:

l	w	v_0
drum area label	D-TAG	drum area name + increment

l defines a location within the area defined by a DRUM-AREA statement.

v_0 names a drum area defined by a DRUM-AREA statement. The increment locates the position relative to the first location of the drum area or Fastrand area to which the label of this statement refers. The increment may be any number not exceeding the length of the previously defined DRUM-AREA.

Example of facility declaration for a drum system with DRUM-AREA and D-TAG statements.

```
l            w
DS1          MEANS . C.5
DS1          FACIL . DRUM . R . 2000
BLKONE       D-TAG . INPTAREA + 100
BLKTWO       D-TAG . INPTAREA + 199
BLKTHREE     D-TAG . INPTAREA + 299
```

4. Text Statement

- FORM-TEXT

The FORM-TEXT statement will convert an alphanumeric text into Fieldata code. The text is placed into a specified position within a previously defined print area. This prepares a line of printing for a subsequent PRINT operation which will cause the line to be printed on the High-Speed Printer.

w	v_0	v_1	v_2
FORM-TEXT	print area name	initial character position	text to be printed

v_0 specifies the area in which the text is to be stored. This is a 26 word area that must have been previously defined. A RESERVE statement may be used to define this area.

v_1 specifies the initial character expressed as octal or decimal (1-200 or 1D-128D) at which the text is to be positioned in the print area. Allowing 128 characters to one line of a High Speed Printer page, this specifies where the first character of the text will be placed. This statement will not destroy any data which may precede or follow the present text.

v_2 contains the alphanumeric text to be stored for printing. Words may be separated by normal spacing. If a partial line is desired the symbol |STOP| (vertical bar, STOP, vertical bar) will generate a Fieldata code of 77. This code when encountered by the print command will cause spaces to appear in the remaining positions of the printed line.

5. Program Segmentation

During the execution of programs of exceptional length, the memory requirements of the program may be reduced through segmentation. A segmented program consists of a controlling segment which calls secondary segments into memory to be executed as required. In addition, tables which modify addresses within the secondary segment relative to the base address of the segment are created to permit communication between the controlling segment and the secondary segment. The loading and modification of segments are performed under control of the Real-Time Executive Routine. A complex relative program format is assumed. The declarative statements which are here described will produce no instructions in the final object program. A segment description record which precedes the object program coding will be created from information supplied by these declarative statements.

- SEGMENT LABEL

The first operation of a segment must be labelled. A segment is defined by its label. If a segmented program is on one source tape or card deck, the only PROGRAM header required is that which occurs at the first line of coding. All other PROGRAM headers are superfluous but may be used as a convenience in marking segment points.

- DEFINING SEGMENTS

Segments are defined within a program by the SEGMENT statement. The format of this statement is as follows:

w	$v_0 \ldots v_n$
SEGMENT	segment labels

v_0 the label of the first operation of each segment is specified. This operation must generate instructions in the final object program. All instructions up to the specification of the next segment label are included in the defined segment.

- JUMP MODIFICATION

Secondary segments are loaded relative to the controlling segment. The controlling segment is itself loaded in a relative position by the Real-Time Executive Routine. When it is necessary for the controlling segment to access portions of a loaded secondary segment by a jump instruction, the address to which the jump is made must be modified to account for this relative placement. The points in the secondary segment to which a jump is to be made are specified in a series of S-TAG statements. A jump table will be created as a result of these statements, which will be placed at the end of the controlling segment.

The S-TAG statements follow the SEGMENT statement. The format for the S-TAG list is as follows:

I	W	V_0	V_1
SEG1	S-TAG	jump address	. jump address
	S-TAG	jump address	. jump address
		≀	
SEG2	S-TAG	jump address	. jump address
		≀	
SEGn	S-TAG	jump address	. 0
	ENDSEG		

1 A fixed label is assigned to each segment for which jump addresses are specified. The S-TAG label number corresponds to the position of the first label specified in the SEGMENT statement.

V_0 & V_1 the label of the jump address or addresses in the appropriate segment is specified as V_0 and V_1. If more than two addresses are specified, the required number of S-TAG statements are used. If an odd number of jump addresses are specified, the last v operand should contain a zero. The jump address entries may not be incremented.

The list is terminated by an entry of ENDSEG in the operation position following the last S-TAG entry.

- LOADING SEGMENTS

A segment is loaded into memory in response to the LOAD statement. The format of the LOAD statement is as follows:

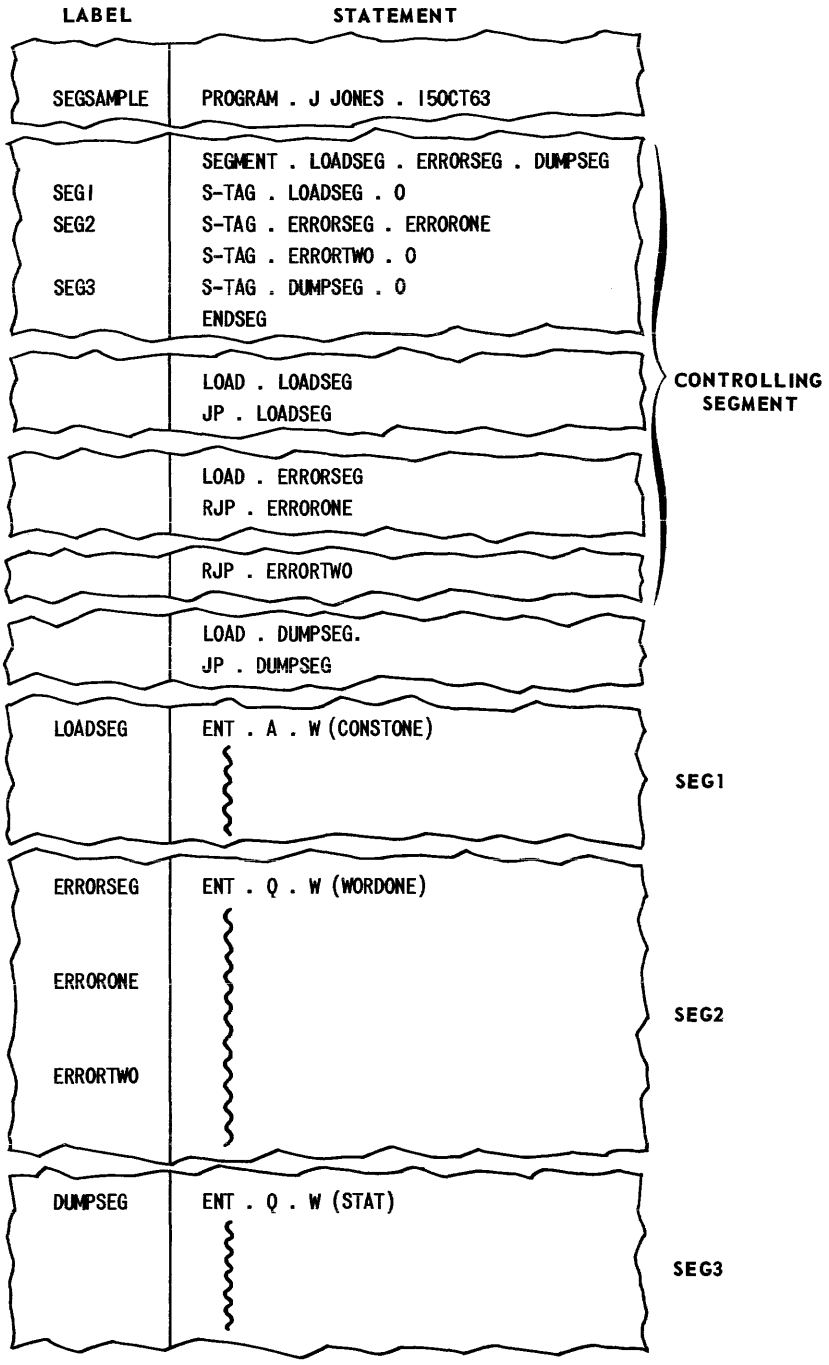
W	V_0
LOAD	segment label

V_0 specifies the first label of the segment to be loaded.

The area reserved for secondary segments will be equal to the area required for the largest segment. After a segment is loaded by the controlling segment, control may be transferred to the segment by a jump or return jump instruction.

- Example of Segmentation:

  The program may be represented as follows:



C. COMMENTS

The COMMENT statement provides for the inclusion of notes or comments at convenient points within a program. These notes will appear on some of the printed listings of the source program. This statement has no effect upon the final object program. The length of the notation entered in the operand position must be within the maximum for a statement for the input format. (See SPURT INPUT FORMATS, APPENDIX B.) Comments are written in the following format:

w	v_o
COMMENT	notation

Example:

```
      w
COMMENT . ERROR PATH FOR WRONG TAPE
```

10. PROGRAM TESTING AND CORRECTION

Before a program can be utilized it must be tested to see if it contains any errors. To aid in this process, SPURT provides the user with the ability to request outputs in various forms, the ability to include program testing routines in the program, and the ability to make corrections.

A. SPURT OUTPUTS

Figure 10-1 shows the outputs that are available from an assembly on Paper Tape, Magnetic Tape, and the High Speed Printer.

TYPE OF OUTPUT	PAPER TAPE		HIGH SPEED PRINTER		HIGH SPEED PRINTER BLOCKS ON MAGNETIC TAPE		MAGNETIC TAPE	
	NUMBER	USE	NUMBER	USE	NUMBER	USE	NUMBER	USE
INPUT LANGUAGE	1	SPURT INPUT OR EDIT	101[1]	EDIT	201	EDIT	301[1]	SPURT INPUT
SELECTIVE DUMP OF INPUT LANGUAGE	2	SPURT INPUT OR EDIT					302	SPURT INPUT
LABELS AND ADDRESSES	3	SPURT INPUT OR EDIT	103	EDIT	203	EDIT	303	SPURT INPUT
RELATIVE LABELS AND ADDRESSES	4	SPURT INPUT OR EDIT					304	SPURT INPUT
EDITED INPUT LANGUAGE AND COMPUTER BINARY	10	LOAD OR EDIT	110[2,3]	EDIT	210	EDIT		
ALPHABETIC SORT OF LABELS	11	SPURT INPUT OR EDIT	111	EDIT	211	EDIT		
NUMERIC SORT OF LABELS	12	SPURT INPUT OR EDIT	112	EDIT	212	EDIT		
ABSOLUTE	20	LOAD					320[2]	LOAD
SIMPLE RELATIVE	21	LOAD					321[3]	LOAD
COMPLEX RELATIVE							322	LOAD

1 If Output Number 501 is requested, concurrent Output Numbers 101 and 301 will be produced.

2 If Output Number 520 is requested, concurrent Output Numbers 110 and 320 will be produced.

3 If Output Number 521 is requested, concurrent Output Numbers 110 and 321 will be produced.

Figure 10-1. SPURT OUTPUTS

Each of the outputs that are listed in Figure 10-1 can be requested by including the following statement in the source program coding:

w	v_0	v_n
OUTPUTS	output number	output number

v_0 specifies the number of the output that is desired. A maximum of seven outputs may be requested. It should be noted that if SPURT Output Number 320, 321, 322, 520 or 521 (magnetic tape outputs) is requested, SPURT will assign a provisional call number (a five-digit number that identifies a program on tape) to this output. The provisional call numbers that will be assigned are 00320, 00321, and 00322 respectively (00320 or 00321 will be assigned to the magnetic tape output that results from a request for SPURT Output Number 520 or 521). If the user desires he can assign a unique call number to the outputs mentioned by prefixing the requested output number with an octal number that ranges from 1 through 77. If this is done, SPURT will assign a combination of the octal number and the number of the desired output as the call number. For example, assume that SPURT Output Number 322 is prefixed by the octal number 10; that is, the output number is 10322. In this case, SPURT would assign 10322 as the call number for this particular output.

.
.
.

v_n

SPURT outputs can also be requested by entering the numbers of the desired outputs when they are requested during assembly. A maximum of seven outputs can be requested. In the sections that follow the individual outputs are discussed within the category that applies.

1. Paper Tape Output

   The following outputs are available on paper tape:

   - Output Number 1, Input Language

     This output consists of a complete dump of all the program operations, including notes, on paper tape. If the L1 Corrector (See Section 10.C., Correction Procedures) is used to correct the original program, this output provides a corrected program tape that is acceptable as input to the assembler.

   - Output Number 2, Selective Dump of Input Language

     This output is used to obtain a dump on paper tape of the program operations, including notes, that are contained within a particular area(s) of an assembled program. The programmer can request this output during a correction run. The program area(s) to be dumped is selected by means of the L1 identifiers (sequence numbers that are assigned to the program operations by SPURT during assembly). When this output is requested, the programmer specifies to the operator the initial and final L1 identifiers of the area(s) to be dumped. This tape is acceptable as input to the assembler.

- Output Number 3, Labels and Addresses

  This output consists of a listing on paper tape of all program labels (labels within Macro operations are not listed) including labels introduced on allocation input to the assembler and their corresponding addresses as they appear in the assembled program. This tape is acceptable as allocation input.

- Output Number 4, Relative Labels and Addresses

  This output consists of a listing on paper tape of all program labels (labels within Macro operations are not listed) and their corresponding addresses made relative to zero (See Section 9. PROGRAM PREPARATION). A base address assigned by the operator at load time will be used to modify the label addresses. This tape is acceptable as allocation input.

- Output Number 11, Alphabetic Sort of Labels

  This output consists of a listing on paper tape, of all program labels (labels within Macro operations are not listed) and their corresponding addresses, in alphabetical order of the labels, as they appear in the assembled program. This tape is acceptable as allocation input.

- Output Number 12, Numeric Sort of Labels

  This output consists of a listing on paper tape of all program labels (labels within Macro operations are not listed) and their corresponding addresses, in numerical order of the addresses, as they appear in the assembled program. This tape is acceptable as allocation input.

- Output Number 10, Edited Input Language and Computer Binary

  This output consists of the L1 identifiers, the program operations including notes, coded error output if any errors are present, and the machine code instructions and their addresses. This tape is acceptable as program input to the computer.

- Output Number 20, Absolute

  This output consists of absolute-addressed machine code instructions. This tape is acceptable as program input to the computer.

- Output Number 21, Simple Relative

  This output consists of relatively-addressed machine code instructions. This tape is acceptable as program input to the computer. It is loaded relative to the starting address specified by the operator.

2. High Speed Printer Output

The following outputs are available on the High Speed Printer:

- Output Number 101, Input Language

  This output consists of a complete dump of all program operations, their associated card numbers (if punched cards were used as input to the assembler), the L1 identifiers, and notes. It provides a listing of the program that is useful for editing and making corrections.

Example of SPURT Output Number 101:

SPURT OUTPUT NO. 101

CARDS	L1 ID	LABEL	STATEMENT		NOTES
0000.00	00000	FORMB	PROGRAM KLEINHAUS*26JULY62		
0000.20	00001	FORMB	JP	0	
0000.30	00002		STR	B3*L(FORMB4)	SAVE B-REGISTERS
0000.40	00003		STR	B4*L(FORMB4+1)	
0000.50	00004		STR	B5*L(FORMB4+2)	
0000.60	00005		ENT	A*B7	BUFFER BASE ADDRESS

- Output Number 103, Labels and Addresses

  This output consists of a complete listing of all the program labels (labels within Macro operations are not listed) and their corresponding addresses as they appear in the assembled program.

  Example of SPURT Output Number 103:

SPURT OUTPUT NO. 103

FORMB KLEINHAUS*26JULY62

LABEL	LOC	LABEL	LOC	LABEL	LOC
FORMB	00000	FORMB1	00032	FORMB2	00034
FORMB3	00035	FORMB4	00042		

- Output Number 111, Alphabetic Sort of Labels

  This output is same as Output Number 103 with the exception that the labels will appear in alphabetic order.

- Output Number 112, Numeric Sort of Labels

  This output is the same as Output Number 103 with the exception that the labels will appear in numerical order; that is, in order of their respective addresses.

- Output Number 110, Edited Input Language and Computer Binary

  This output consists of a complete listing of card numbers (if punched cards were used as input to the assembler), the L1 identifiers, coded error output if any errors are present, the program operations, the machine code instructions and their addresses, and notes.

Example of Output Number 110:

SPURT OUTPUT NO. 110

CARDS	L1 ID	LABEL	STATEMENT	LOC	F	JKB	Y	NOTES
0000.00	00000	FORMB	PROGRAM KLEINHAUS*26JULY62					
0000.20	00001	FORMB	JP O	00000	61	000	00000	
0000.30	00002		STR B3*L(FORMB4)	00001	16	310	00042	SAVE B-REGISTERS
0000.40	00003		STR B4*L(FORMB4+1)	00002	16	410	00043	
0000.50	00004		STR B4*L(FORMB4+2)	00003	16	510	00044	
0000.60	00005		ENT A*B7	00004	11	007	00000	BUFFER BASE ADDRESS

It should be noted that when SPURT Output Number 110 is requested for a segmented program, each segment is identified by its own header on the printed copy in the form

xxxxxxxxxx -SEGMENT SPURT OUTPUT NO. 110

where xxxxxxxxxx is the segment label. In addition, when the % symbol or the : symbol appear on the printed copy in the first column to the right of the Y portion of a machine coded instruction, this means that the upper half of the instruction will be modified at load time. If either of these symbols appear in the second column to the right, this means that the lower half of the instruction will be modified at load time. The % symbol indicates modification relative to the current segment base and the : symbol indicates modification relative to the control segment base.

3. High Speed Printer Output on Magnetic Tape

SPURT Outputs Number 201, 203, 210, 211, and 212 are the same as outputs number 101, 103, 110, 111, and 112 with the exception that the information is stored on magnetic tape instead of being printed.

4. Magnetic Tape Output

The following outputs are available on magnetic tape:

- Output Number 301, Input Language

  This output consists of a complete dump of all the program operations, including notes, on magnetic tape. This tape is acceptable as input to the assembler.

- Output Number 302, Selective Dump of Input Language

  This output consists of a dump on magnetic tape of the program operations, including notes, that are contained within a particular area(s) of an assembled program. The programmer can request this output during a correction run. The program area(s) to be dumped is selected by means of the L1 identifiers. When this output is requested, the programmer specifies to the operator the initial and final L1 identifiers of the area(s) to be dumped. This tape is acceptable as input to the assembler.

- Output Number 303, Labels and Addresses

  This output consists of a listing on magnetic tape of all program labels (labels within Macro operations are not listed) including those labels introduced on allocation input to the assembler and their corresponding addresses as they appear in the assembled program. This tape is acceptable as allocation input.

- Output Number 304, Relative Labels and Addresses

  This output consists of a listing on magnetic tape of all program labels (labels within Macro operations are not listed) and their corresponding addresses made relative to zero. A base address assigned by the operator at load time will be used to modify the label addresses. This tape is acceptable as allocation input.

- Output Number 320, Absolute

  This output consists of absolute-addressed machine code instructions. It is acceptable as program input to the computer.

- Output Number 321, Simple Relative

  This output consists of relatively-addressed machine code instructions. This tape is acceptable as program input to the computer. It is loaded relative to the starting address specified by the operator.

- Output Number 322, Complex Relative

  This output consists of machine-coded instructions that are preceded by blocks that describe the facility requirements of the program. At load time REX will analyze these requirements. It will then place the program in an available core storage area and assign the required peripheral units on the basis of the information that is supplied. If the program is segmented, this is the only SPURT output that is acceptable as program input to the computer.

  It should be noted that Output Numbers 101, 103, 110, 201, 203, 210, and 301 are the only outputs that may be requested in conjunction with Output Number 322.

5. Concurrent High Speed Printer and Magnetic Tape Output

   The following concurrent outputs are available:

   - Output Number 501, Concurrent Output Numbers 101 and 301.
   - Output Number 520, Concurrent Output Numbers 110 and 320.
   - Output Number 521, Concurrent Output Numbers 110 and 321.

B. PROGRAM TESTING ROUTINES

SPURT provides routines that can be utilized by the programmer to aid in testing a program. The operations that can be requested and the various methods of utilizing these routines are discussed in the sections that follow.

1. Program Testing Operations

Program testing operations may be used to provide output on the High Speed Printer at various points during the running of the program. This output consists of the contents of the various registers, the contents of a specified core storage area, or a listing of changes that have occurred in a specified core storage area.

- Define Area

This operation is used to define the area to which program testing operations will apply. The format of this operation is as follows:

w	v_0	v_1	v_2
DEF-AREA	area name	initial area address	number of words

v_0 defines the name of the area to which program testing operations will apply.

v_1 specifies the initial address of the area.

v_2 specifies the number of words in the area.

- Establish Core Image

This operation will establish an area in core storage that will initially duplicate a program area that was previously defined by a Define Area operation. The format for this operation is as follows:

w	v_0	v_1	v_2
CORE-IMAGE	area name	initial image address	key setting

v_0 specifies the area, previously defined by a Define Area operation, that is to be imaged.

v_1 specifies the initial core address for the image. The length of the image is determined by the length specified in the Define Area operation that is associated with the program area to be imaged.

v_2 is optional. The allowable entries in this position are KEY 1, KEY 2, or KEY 3. These entries are associated with Console Switches 1, 2, and 3 respectively. If an entry is made, the operation will not be performed unless the associated console switch is set. It should also be noted that if programs are to be run concurrently, care must be taken to insure that a key setting for one program will not affect any of the other programs during processing.

- Establish Drum Image

This operation establishes an area in drum storage that will initially duplicate a program area that was previously defined by a Define Area operation. The format for this operation is as follows:

w	v_0	v_1	v_2
DRUM-IMAGE	area name	initial image address	key setting

v_0 specifies the area, previously defined by a Define Area operation, that is to be imaged.

v_1 specifies the address of the core storage location that contains the initial drum address for the image. The length of the image is determined by the length specified in the Define Area operation that is associated with the program area to be imaged.

v_2 is optional. The allowable entries in this position are KEY 1, KEY 2, or KEY 3. These entries are associated with Console Switches 1, 2, and 3 respectively. If an entry is made, the operation will not be performed unless the associated console switch is set. It should also be noted that if programs are to be run concurrently, care must be taken to insure that a key setting for one program will not affect any of the other programs during processing.

- Test Image

This operation compares a program area, previously defined by a Define Area operation, against the corresponding core or drum image established for that area. The image and the program area are compared word by word and any non-comparing words will be printed on the High Speed Printer. When the operation is completed, the contents of the area that was tested will become the new image. It should also be noted that a listing of the contents of the A, Q, and B1 through B7 registers will always precede the listing of the non-comparing words when this operation is executed. The format of the Test Image operation is as follows:

w	v_0	v_{n-1}	v_n
TEST-IMAGE	area name	area name	key setting

v_0 specifies the area, previously defined by a Define Area operation, that is to be tested.

.
.
.

v_{n-1}

v_n is optional. The allowable entries in this position are KEY 1, KEY 2, or KEY 3. These entries are associated with Console Switches 1, 2, and 3 respectively. If an entry is made, the operation will not be performed unless the associated switch is set. It should also be noted that if programs are to be run concurrently, care must be taken to insure that a key setting for one program will not affect any of the other programs during processing.

Example of Test Image Operation Listing:

```
                        ENTRANCE ADDRESS 30036

A  12345 67012                    Q  76543  21076

B1 00001   B2 00022   B3 00333   B4 04444   B5 055555   B6 60606   B7 07070

                        ENTRANCE ADDRESS 300037   TEST10 30066 THRU 30105

LOC  IMAGE       AREA       LOC  IMAGE       AREA
30066 00010 20304  00001 20304  30070 12131 41516  12130 41516
30073 31323 33435  31323 45565  30076 50515 25354  02024 66642
```

■ Dump Registers

This operation causes the contents of the A, Q, and B1 through B7 registers to be listed on the High Speed Printer. The format of the Dump Registers operation is as follows:

w	v_0
DUMP-REG	key setting

v_0 is optional. The allowable entries in this position are KEY 1, KEY 2, or KEY 3. These entries are associated with Console Switches 1, 2, and 3 respectively. If an entry is made, the operation will not be performed unless the associated switch is set. It should also be noted that if programs are to be run concurrently, care must be taken to insure that a key setting for one program will not affect any of the other programs during processing.

Example of Dump Registers Operation Listing

ENTRANCE ADDRESS 25076

A 02530 33330 Q 00000 22746

B1 00010 B2 07774 B3 00022 B4 00040 B5 02040 B6 00000 B7 00012

- Dump Area

This operation causes all non-zero words in an area, previously defined by a Define Area operation, to be listed on the High Speed Printer. The format of a Dump Area operation is as follows:

w	v_0	v_{n-1}	v_n
DUMP-AREA	area name	area name	key setting

v_0 specifies the area, previously defined by a Define Area operation, that is to be dumped.

.
.

v_{n-1}

v_n is optional. The allowable entries in this position are KEY 1, KEY 2, and KEY 3. These entries are associated with Console Switches 1, 2, and 3 respectively. If an entry is made, the operation will not be performed unless the associated switch is set. It should also be noted that if programs are to be run concurrently, care must be taken to insure that a key setting for one program will not affect any of the other programs during processing.

Example of Dump Area Operation Listing:

LOC	F JKB Y	LOC	F JKB Y	LOC	F JKB Y	LOC	F JKB Y
30066	00002 20304	30067	05060 71011	30070	12127 41516	30071	17202 12223
30072	24252 62730	30073	31323 57714	30074	36374 04142	30075	43444 54647
30076	33323 77756	30077	55565 76061	30100	62636 46566	30101	67707 17273
30102	74757 67710	30103	01011 02103	30104	10410 51061	30105	07110 00000

2. Utilizing Program Testing Routines

The Program Testing Routines, when utilized, may be considered as a subroutine of the program that is being tested. The Program Testing Routines have facility requirements that must be added to the facility requirements of the program being tested. Specifically, if Drum Image operations are to be used, declarative operations associated with Drum input/output must be present in the program coding. A Printer will be required in all cases to print out the pertinent information that is requested by, or which results from the testing operations. Since a Printer is required, declarative operations associated with Printer input/output must also be included in the program coding.

Input/output operations within the Program Testing Routines refer to the Printer channel by the mnemonic name DEBUG790 and to the Drum channel by the mnemonic name DEBUG780. This implies that the user must at least provide a MEANS statement to define the specific channel for the Printer, and optionally, the Drum if it is used. These statements would appear in the program coding as follows:

```
l          w                NOTES

DEBUG790  MEANS  .  Cxx  (Printer)

DEBUG780  MEANS  .  Cxx  (Drum, if used.)
```

xx is the octal channel number that is assigned

Other declaritive statements (ASSIGN, FACIL) may be required in the program coding depending upon the environment in which the program being tested and the Program Testing Routines will operate. Furthermore, it is possible that the program being tested may require the use of a Printer subsystem and/or Drum. In this case some modification to either the program being tested or the Program Testing Routines may be required to assure a uniform channel assignment to the Printer subsystem and the Drum if used.

If Core Image operations are used, the core area utilized for the image must be provided by the program being tested. The core area can be provided by the use of allocation or by reserving an area within the program.

The four methods by which the user can utilize the Program Testing Routines are discussed in the text that follows:

- Method 1

  This method requires that the Program Testing Routines header, DEBUG-AIDS, be placed in the source program coding as follows:

```
l          w                          NOTES
-------------------------------------------------------------
           DEBUG-AIDS                 PROGRAM TESTING ROUTINES HEADER
TEST       PROGRAM . JONES . 8 JULY 63  PROGRAM HEADER
             ≀
             ≀
           Program Coding
             ≀
             ≀
```

  During assembly the Program Testing Routines package will be called from the SPURT library and will be placed at the end of the program coding. Linkage to the package will be generated by the program testing operations that have been included in the program.

  If the program is segmented or if it is to be assembled in complex relative format, this is the only method by which the program Testing Routines can be utilized.

  The linkage can be eliminated by removing the Program Testing Routines header from the original source program coding and reassembling.

If Input Language output is requested, the Program Testing Routines header will not be included in this output; however, the Program Testing Routines package will be placed at the end of the program coding. If this output is used as input to a future assembly the Program Testing Routines header will have to be loaded at assembly time. This can be accomplished via punched cards, paper tape, or by using Method 4.

- Method 2

This method differs from Method 1 in that it allows linkage to the Program Testing Routines package to be generated without actually calling the package from the SPURT library. To accomplish this, it is necessary to change the coding shown for Method 1 as follows:

I	W	NOTES
	DEBUG-AIDS	PROGRAM TESTING ROUTINES HEADER
TEST	PROGRAM . JONES . 8 JULY 63	PROGRAM HEADER
	IGNORE . DEBUG-AIDS	
	≀	
	Program Coding	
	≀	

During assembly the Program Testing Routines package will not be called from the SPURT library. Instead, the program testing operations that have been included in the program coding will generate linkage to the package at address 36000.

When Method 2 is used, the Program Testing Routines package will have to be loaded in absolute format at address 36000 at the time the program is run. The linkage can be eliminated by removing the Program Testing Routines header from the source program coding and reassembling.

- Method 3

This method is the same as Method 2 with the exception that it allows the user to specify the address where the Program Testing Routines package will be loaded when the program is run. To provide this option, it is necessary to change the coding shown for Method 2 as follows:

I	W	NOTES
	DEBUG-AIDS	PROGRAM TESTING ROUTINES HEADER
TEST	ALLOCATION . JONES . 8 JULY 63	ALLOCATION HEADER
DEBUG	aaaaa	ALLOCATION OF PROGRAM TESTING ROUTINES PACKAGE
TEST	PROGRAM . JONES . 8 JULY 63	PROGRAM HEADER
	IGNORE . DEBUG-AIDS	
	≀	
	Program Coding	
	≀	

This result is the same with this method as with Method 2 with the exception that the program testing operations will generate linkage to the Program Testing Routines package at address aaaaa instead of address 36000. When Method 3 is used, the Program Testing Routines package will have to be loaded in absolute form at address aaaaa at the time the program is run.

The linkage can be eliminated by removing the Program Testing Routines header and reassembling.

It should also be noted that an Equals statement could be used to allocate DEBUG.

- Method 4

This method produces the same results as Method 2. It differs from Method 2 in that it is not implemented by the inclusion of the Program Testing Routines header, DEBUG-AIDS, and the IGNORE . DEBUG-AIDS operation in the source program coding. Instead, it is implemented by entering 10000 when outputs are requested during assembly.

As with Method 2 the Program Testing Routines package is not called from the SPURT library during assembly. Instead, the program testing operations that are in the program coding will generate linkage at address 36000. When Method 4 is used, the Program Testing Routines package will have to be loaded in absolute form at address 36000 at the time the program is run.

If the Program Testing Routines package has already been incorporated as part of the program coding, Method 4 will cause the Program Testing Routines header to be loaded and linkage to the package will be generated as described for Method 1.

C. CORRECTION PROCEDURES

Following an assembly run it is often necessary to delete or replace existing program operations, or to add new operations to the source program. The procedures for effecting deletions, replacements, and additions are described in the text that follows.

1. Card Correction Procedures

If the source program is on punched cards, deletions are made by removing cards from the program deck, replacements are made by removing the original cards and inserting new ones, and new operations are added by inserting cards. The operations that are to be deleted or replaced and the points where new operations are to be added are located by means of the card numbers. These numbers can be determined by examining a High Speed Printer Output Number 101 or 110 of the source program.

For example, assume that a High Speed Printer Output Number 101 shows that a portion of the source program is

SPURT OUTPUT NO. 101

CARDS	L1 ID	LABEL	STATEMENT		NOTES
0000.00	00000	FORMB	PROGRAM	KLEINHAUS*26JULY62	
0000.20	00001	FORMB	JP	O	
0000.30	00002		STR	B3*L(FORMB4)	SAVE B-REGISTERS
0000.40	00003		STR	B4*L(FORMB4+1)	
0000.50	00004		STR	B5*L(FORMB4+2)	
0000.60	00005		ENT	A*B7	BUFFER BASE ADDRESS

and that the following changes are to be made:

- The statement JP. O is to be deleted.
- The statement STR.B6 . L(FORMB5) is to be added to the program between the statements STR . B5 . L(FORMB4+2) and ENT . A . B7 .
- The statement STR . B4.L(FORMB+1) is to be replaced with the statement STR . B2.U(FORMB+1).

These changes are effected as follows:

- The statement JP . O is deleted by removing card number 0000.20 from the program deck.
- The statement STR . B6 . L(FORMB5) is added to the program by preparing a card that has a card number that is greater than the card number assigned to the statement STR . B5 . L(FORMB4+2) but less than the card number assigned to the statement ENT . A . B7. When such a card has been prepared, it is then inserted into the program deck between card number 0000.50 and card number 0000.60.
- The statement STR . B4 . L(FORMB+1) is replaced by the statement STR . B2 . U(FORMB+1) by preparing a card that contains the latter statement and has the same card number as the former statement. The card containing the original statement (card number 0000.40) is then removed from the program deck and the new card is inserted in its place.

2. The L1 Corrector

The L1 Corrector is a routine within the assembler that enables the user to make corrections to a source program which is on paper tape (SPURT Output Number 1) or magnetic tape (SPURT Output Number 301). A maximum of 92 (decimal) changes (deletions, replacements, and additions) are made by preparing a paper correction tape or card deck that begins with a header that has the following format:

I	W	V_0	V_1
program name	CORRECT-L1	programmer's name	date

This header is followed by coding that will effect the required changes to the source program. The operations that are to be deleted or replaced and the points where new operations are to be added are located by means of the L1 identifiers that were assigned during the initial assembly. The identifiers can be determined by examining a High Speed Printer Output Number 101 or 110 of the source program.

To illustrate, assume that a High Speed Printer Output Number 101 shows that a portion of the source program is

SPURT OUTPUT NO. 101

```
CARDS   L1 ID   LABEL    STATEMENT                    NOTES
        00000   TESTGA   PROGRAM      JONES*30JULY63
        00001   TEST1    RSH  Q*1*AZERO
        00002            RSH  A*2
        00003            RSH  AQ*3
        00004            COM  A*W(CAT)*YLESS
        00005            COM  Q*L(CAT)*YMORE
```

and that the following changes are to be made:

- The statement RSH . Q.1 . AZERO is to be deleted.
- The statement STR . B3 . L(DOG) is to be added to the program between the statements RSH . AQ . 3 and COM . A. W(CAT) . YLESS.
- The statement COM . Q . L(CAT) . YMORE is to be replaced by the statement COM . Q . U(CAT+2) . YMORE.

These changes are effected by preparing a paper correction tape or card deck that has the following format:

```
I                w
TESTGA      CORRECT-L1 . JONES . 31 JULY 63
00001.000
            DELETE
00003.001
            STR . B3 . L.(DOG)
00005.000
            COM . Q . U(CAT + 2) . YMORE
```

As shown above each change is located by an L1 identifier and a three-digit octal insertion number. (The sequence of the insertion numbers is .000 through .007, .010 through .017, and so on.) Since the first change is a deletion the insertion number is .000. In the case of the second change the insertion number is .001 because this statement is to be inserted between two existing statements; that is, the L1 identifier and the insertion number of the statement to be inserted has to be greater than that of the preceding statement but less than that of the following statement. In the case of the third change the insertion number is .000 because this statement is replacing an existing statement.

Two cards will be needed for each correction if card input is used. This card pair will consist of one card that contains the L1 identifier and insertion number and one card that contains the word DELETE (for a deletion), a statement that is to be added, or a statement that is to replace an existing statement. These cards will be punched in the standard SPURT card input format with one exception. This exception is that the L1 identifier and insertion number must be placed in columns 8 through 16.

It should also be noted that it is possible to delete a series of consecutive statements from the source program by including on the correction tape or card deck coding in the following format:

```
I              w
iiiii.000
            DELETE . n
```

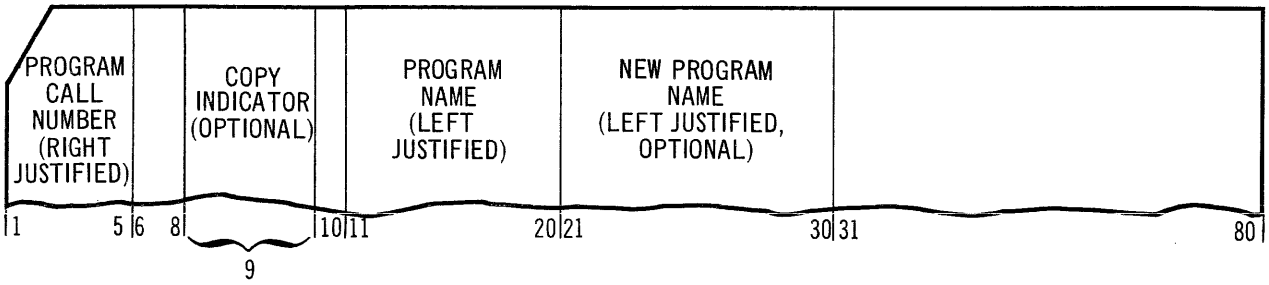
iiiii = L1 identifier of the first statement in the series to be deleted.

n = the number (octal or decimal) of statements to be deleted.

3. The Card Image Corrector Routine – CIMCO

CIMCO is a utility routine that operates under the control of REX. It utilizes punched cards to make corrections to a source program that is on magnetic tape in input language format (SPURT Output Number 301). Since CIMCO is not a part of the assembler, the user can save considerable time by utilizing it instead of the L1 Corrector whenever corrections have to be made to a source program on magnetic tape.

Changes to a source program (deletions, replacements, and additions) are made by preparing a card deck that begins with a header card that has the following format:



The header card is followed by one or more correction cards that contain the word DELETE (for a deletion), a statement that is to be added, or a statement that is to replace an existing statement. The operations that are involved and the points where new operations are to be added are located by means of the L1 identifiers that were assigned during the initial assembly. The identifiers can be determined by examining a High Speed Printer Output Number 101 or 110 of the source program. The correction cards are punched in the standard SPURT input card format with the exception that the L1 identifier of the statement that is involved is placed in columns 73-77 (leading zeros need not be punched) and nd the insertion number is placed in columns 78-80. Since columns 73-80 are never used in processing, this format allows the correction cards to be used for updating a source program deck. When a correction deck is prepared, the correction cards must be placed in the deck in ascending L1 identifier order; that is, the correction card with the lowest L1 identifier must be the first card after the header card, the card with next highest L1 identifier must be the second card, and so on. The use of card numbers on correction cards is optional; however, if card numbers are placed on correction cards it is the user's responsibility to see that these numbers are in the proper sequence.

The final correction card for a given program is followed by an end of corrections cards that contains slashes in columns 1-5.

If there is only one program to be corrected, the end of corrections card is followed by a card that has the word END in columns 1-3.

If another program on the same tape is to be corrected, the end of corrections card is followed by a new header card.

If more than one program on the same tape is to be corrected, the card with the word END will follow the end of corrections card in the last group of corrections.

To illustrate the use of CIMCO, assume that a High Speed Printer Output Number 101 shows that a portion of the source program TESTGA is

SPURT OUTPUT NO. 101

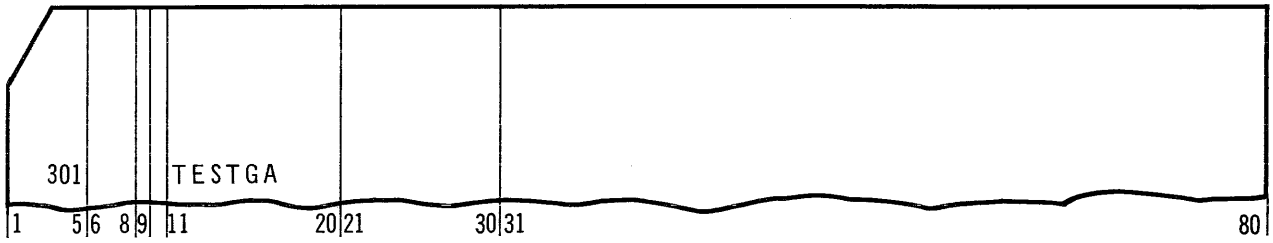
CARDS	L1 ID	LABEL	STATEMENT	NOTES
	00032		SUB A*LX(CAT+1)	
	00033		SUB Q*UX(CAT+2)*QNEG	
	00034		SUB LP*W(CAT)*ANOT	
	00035		MUL W(CAT)	
	00036		MUL 123456	
	00037		DIV W(CAT)	

and that the following changes are to be made:

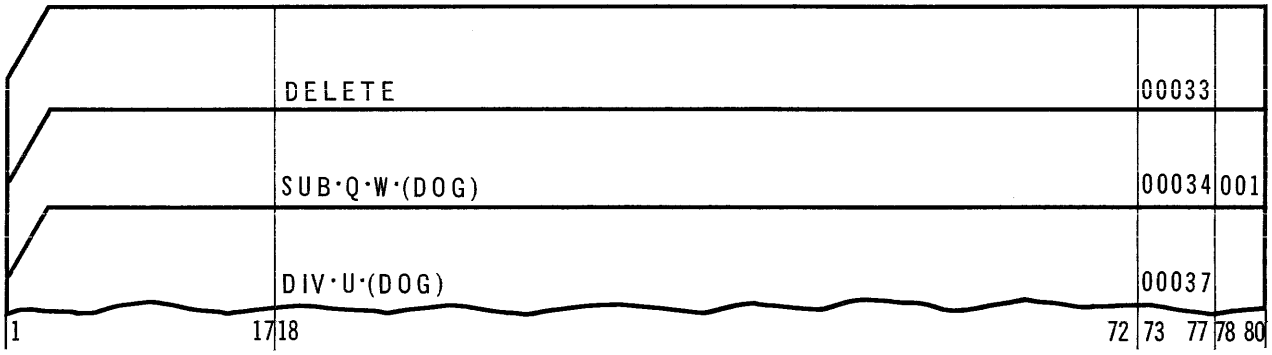
- The statement SUB . Q . UX(CAT+2) . QNEG is to be deleted.
- The statement SUB . Q . W(DOG) is to be added to the program between the statement SUB . LP . W(CAT+2) . ANOT and MUL . W(CAT).
- The statement DIV . W(CAT) is to be replaced by the statement DIV . U(DOG).

These changes are effected by preparing a card deck that has the following format:

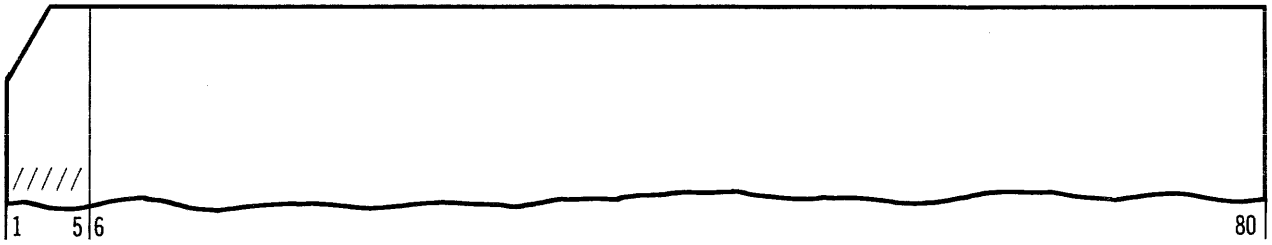
HEADER CARD



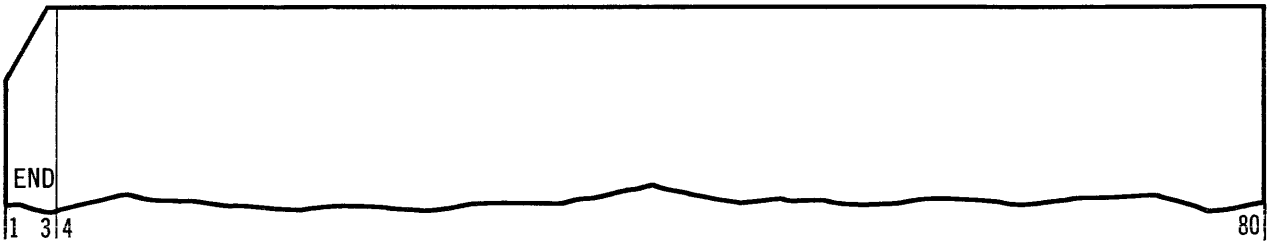
CORRECTION CARDS



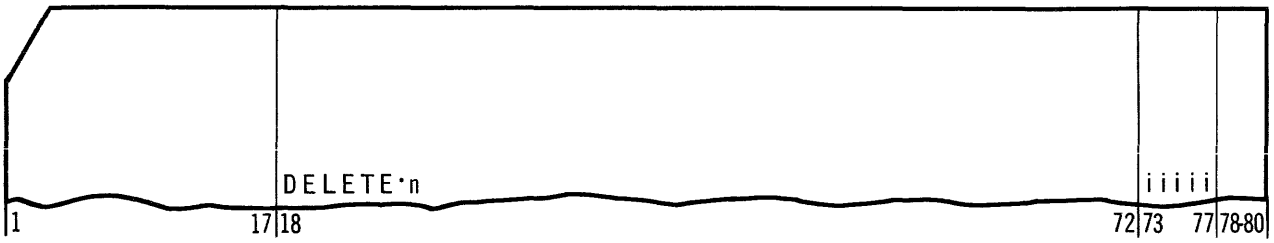
END OF CORRECTION CARD



END CARD



It is also possible to delete a series of consecutive statements from the source program by including a card that has the following format in the correction deck:



iiiii = L1 identifier of the first statement in the series to be deleted.

n = the number of statements to be deleted. This number cannot exceed 777 octal.

It should be noted that the following options are also available:

- Copy

  If the letter C is placed in column 9 of the first header card in the correction deck, the input tape will be copied onto an output tape until the first program to be corrected is found. The specified program is then corrected and copying continues until the end of file or the next program to be corrected is reached.

- Multiple Reel Magnetic Tape Input

  If an end of file is encountered before the END card in the correction deck is read, the current input tape will be rewound. When this occurs, the user can terminate the correction run or mount a new input tape and continue.

- Selective Copy

  A particular program(s) can be extracted from a tape containing several programs and be copied onto an output tape. This is accomplished by preparing a card deck that contains a header card and an end of corrections card for each program that is to be copied.

D. CODED ERROR OUTPUT DURING ASSEMBLY

SPURT provides an automatic High Speed Printer listing (SPURT Output Number 777) of any translation or allocation errors that occur during assembly. This output consists of the program operations that were in error and associated error codes, their card numbers (if punched cards were used as input to the assembler), their L1 identifiers, and notes.

If a High Speed Printer is not available, the error output will be written on magnetic tape in High Speed Printer edited format.

Example:

SPURT OUTPUT NO. 777

CARDS	L1 ID	LABEL	TA	STATEMENT		NOTES
	00017		O	CTAPE	LOBCD*T7*T11*T13*T10*T22*T12	
	00024		O	CTAPE	L1BIN*T1	
	00025		S	CTAPE	SEARCH*T12*BCW	
	00026	SID	O	CTAPE	READ*T22*B4	
	00030		O	CTAPE	MOVEF*3400D	
	00034		O	CTAPE	HIBCE*T17	
	00037		O	CTAPE	HVBCT*T7	
	00040					
	00045		R	COM	A*W(CAT)*YLESS	
	00355	CON	D	CONSOLE	HOLD	

As shown in the above example, two columns labeled T and A appear to the left of the STATEMENT column. These columns serve to indicate the type of error. If the error code appears in column T, it is a translation error; if it appears in column A, it is an allocation error. The table that follows shows the various error codes that can appear during assembly:

CODE	TYPE OF ERROR	MEANING
C	Translation	Underfined channel.
I	Translation	Illegal operator.
N	Translation	A decimal number not followed by a D or a non-numeric character within a probable number.
O	Translation	Illegal format or an illegal B, J, or K designator.
S	Translation	Statement too long or incomplete.
D	Allocation	Duplicate label.
E	Allocation	EQUALS used incorrectly.
F	Allocation	First instruction does not have a label. (Instruction will be allocated to zero.)
R	Allocation	Reference to a duplicate label.
U	Allocation	Unallocated label. (Label will allocated to zero.)

If errors occur during assembly, the message

Pxx program name ERRORS FOUND n

xx = Program identification number.

n = The number (octal) of errors that were found

will be relayed via the Console Printer prior to the error output.

It should also be noted that if a SPURT Output Number, number 10 or 110 has been requested and errors occur during assembly, the error codes that appear on the error listing will also appear on the SPURT output listing.

14. MISCELLANEOUS ROUTINES

This section contains reference information for routines that will perform various subsidiary functions in a UNIVAC 490 Real-Time System. Most of these routines are designed to operate under control of the REX (Real-Time) Executive Routine which permits their use in a multiprogram environment. Some include an option which provides for independent operation.

A. RMOPL II

RMOPL II (Routine for Maintaining an Object Program Library) is used to create and maintain magnetic tape files of final object programs produced by the SPURT assembler. These files may serve as a final object code source for MITAR (Master Instruction Tape Assembly Routine), which creates a program file containing scheduling and operational information for operation of programs under control of REX (Real-Time Executive Routine). The library may also be the source for direct program input to the system.

1. System Requirements

Minimum equipment requirements consist of:

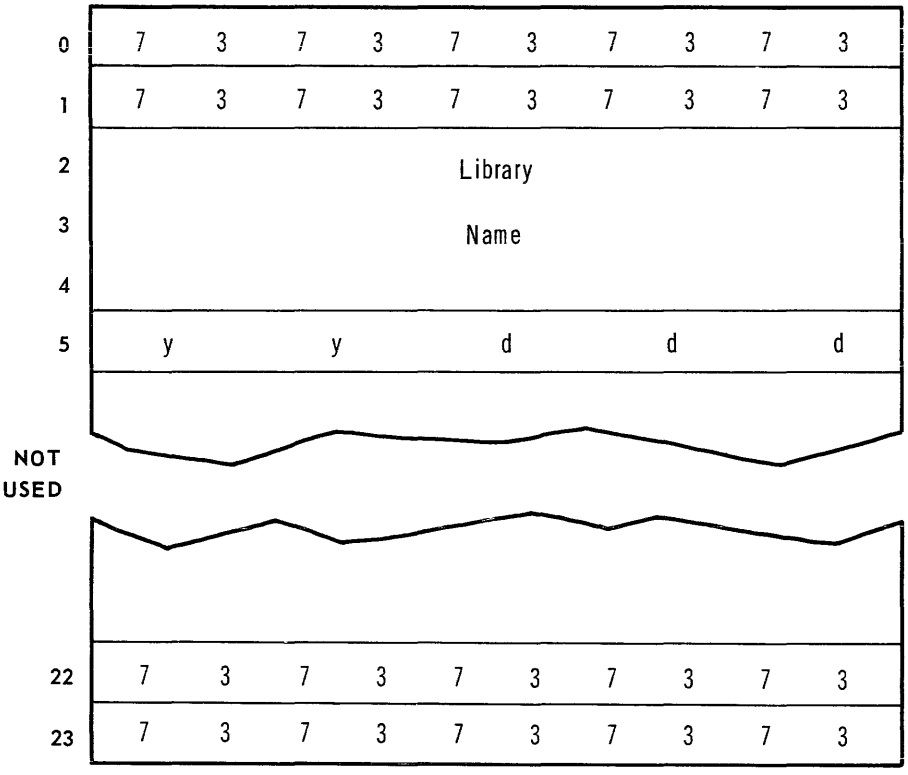
- $2{,}300_8$ words of core storage
- 3 UNISERVO units
- a paper tape or card reader
- a High Speed Printer and/or a Card Punch.

REX is utilized for program loading, input/output requests, and the entry of parameters which define variables in the program. A maximum of 50 parameter cards may be entered

2. Tape Composition

A library tape is composed of:

- A label record in the following format:



yyddd is year and day

- Any number of SPURT object programs as defined in APPENDIX E. The words in the identification record containing the library number will be modified to reflect the number assigned by RMOPL II.
- Two standard end of file sentinels.
- A tape mark if the UNISERVO IIIC was used for output.

3. Operations Performed

RMOPL II will perform the following operations:

- create a new library
- update an existing library
- coincident with either of the above functions, or as a separate pass, print and/or punch a library directory.

Any SPURT object code output may be included in a library. As part of the placement process, a 5 octal digit number is assigned each program as a library number. Programs within a library are arranged in ascending sequence by library number.

4. Parameter Cards

- Function Card

A function card is used to specify the function to be performed. There are two functions:

BLD Form the input programs into a new library.

UPD Operate upon an input library to add and/or delete programs producing an updated library.

COLUMNS	CONTENTS
1	C
2, 3	05
11 - 13	BLD or UPD
16 - 30	(The name of the library being built or updated)

- Program Card

BLD Function

One or more program cards may follow the BLD card to identify specific programs from the input tape which are to be placed in the library.

Absence of program cards implicitly specifies that all object code from the input tape is to be placed in the library. In this case RMOPL II will automatically assign library numbers commencing with 10_8. Successive assignments will be made by adding 10_8 to the preceding number.

UPD Function

One or more program cards may follow the UPD card to identify specific programs from the input tape which are to be added to the library and/or programs which are to be deleted from the library being updated.

Absence of program cards implies that all object code from the input tape is to be added to the library. In this case, RMOPL II will automatically assign library numbers commencing with the highest number on the old library plus 10_8. Successive assignments will be made by adding 10_8 to the preceding number.

COLUMNS	CONTENTS
1	C
2, 3	05
11	D if delete, otherwise blank
16 – 25	(Program name)
26 – 30	(SPURT output number or old library number – 5 digit octal value. Library number is required if the program is being deleted. If there is more than one SPURT object code output of this program on the input tape, the desired output must be specified. Otherwise, the field may be left blank.)
31 – 35	(Assigned library number-5 digit octal value. If the field is left blank, RMOPL II will assign a library number.)

- Directory Cards

  A directory card or cards may be included with or may constitute the input deck.

COLUMNS	CONTENT
1	C
2, 3	05
11 – 13	PRT (Print a library directory) PCH (Punch a library directory)

- New Input Tape Card

  If more than one input tape is to be processed, the second and subsequent tapes must be identified so that RMOPL II may direct tape mounting.

COLUMNS	CONTENT
1	C
2, 3	05
11 - 13	NIT
16 - 30	(The name to be used in directing tape mounting)

- Order of the Parameter Deck

  (1) Directory card(s)

  (2) Function card

  (3) Program card(s) for program on initial input tape

  (4) New Input Tape Card

  (5) Program Card(s) for programs on last-named input tape

  (6) Four or more blank cards which signify the end of the parameter deck

  A maximum of 50 parameter cards is allowed. Only one function card is permitted for any given run.

B. MITAR II

UNIVAC 490 Real-Time System procedures provide for the concurrent operation of two or more batch programs. Programs may be submitted for selection within sets on a Master Instruction Tape (MIT) which contains scheduling information, facility statements, object code, and optionally, parameters for programs. A directory of the programs contained on the library is also provided.

From the information contained on the MIT, the Real Time Executive Routine, through its selection and loading operations, may select programs to achieve the optimum utilization of core memory and peripherals. An MIT is created by the use of MITAR II (Master Instruction Tape Assembly Routine).

1. System Requirements

Minimum system requirements consist of:

- 3237_8 words of core storage
- three UNISERVO units
- a paper tape or a card reader
- if a directory is to be printed, High Speed Printer

REX is utilized for program loading, input/output requests, and parameter entry.

2. Execution Sequence

Constraints governing the execution sequence of programs comprising an MIT may be imposed when the MIT is created. The following controls are provided:

- PRIORITY

  Programs within a low number priority group will be initiated before those of a higher numbered group.

- STRING

  A string is a set of programs within a priority classification which must be executed serially.

- LOCK

  A program or string may beplaced in a lock condition. Programs so designated will not be executed unless the lock is removed by the computer operator during the run of an MIT.

3. Tape Composition

An MIT may contain a maximum of 64 individual programs. It contains contiguous groups of records in the order listed below.

(1) A label record.

(2) One to eight index records.

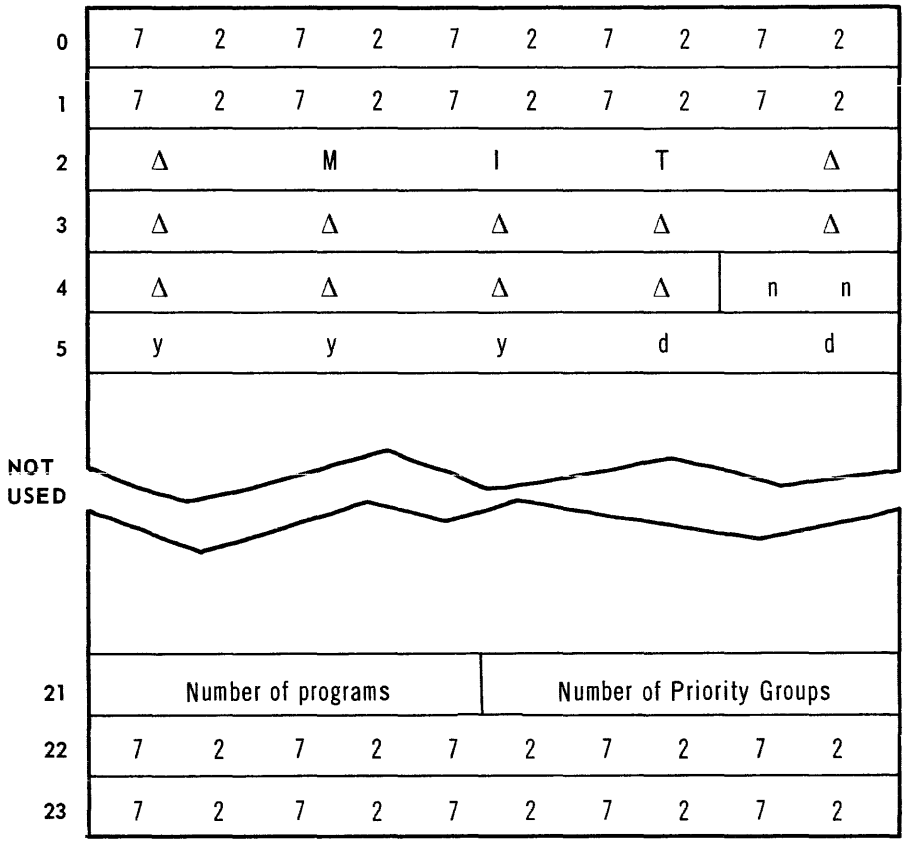
(3) One to eight program facility summary records.

(4) One to 64 programs in complex relative format together with operational parameters.

(5) Two standard end of file sentinels.

(6) Tape mark if UNISERVO IIIC revision was used.

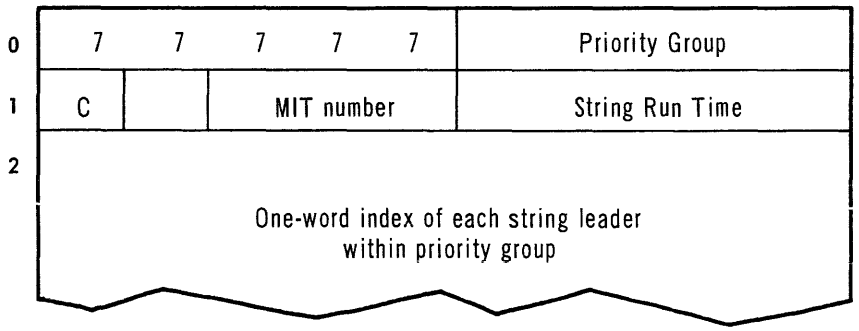
- LABEL RECORD



nn is an octal identifier assigned at time of creation. yyddd is year and day.

- INDEX RECORD

There is one index record for each priority group. A maximum of eight priority groups may be contained on the MIT, and a priority group may contain any number of programs. The MIT, however, can contain a maximum of 64 programs.



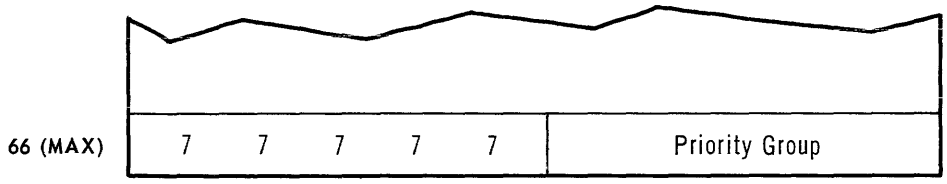
C is lock condition indicator in bit positions 27-29

1 string lock

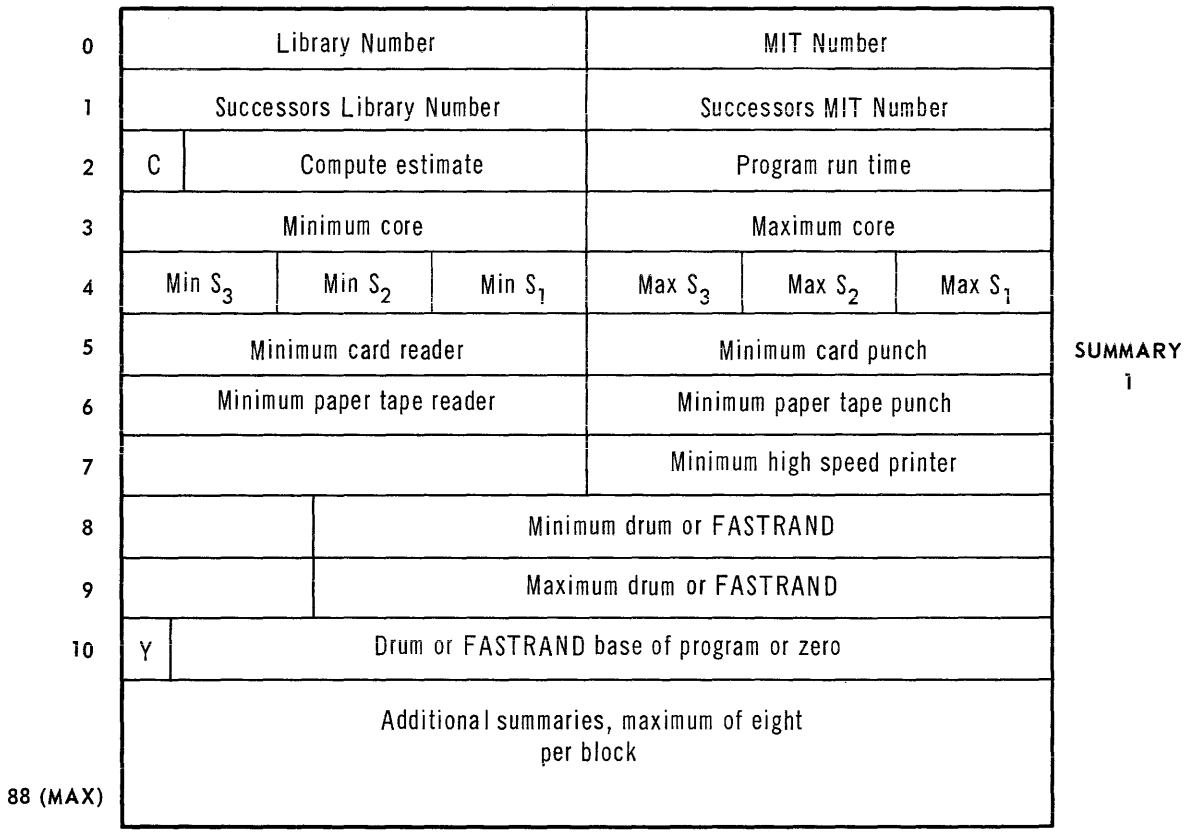
2 program lock

6 no lock

MIT number is that of string leader.



■ PROGRAM FACILITY SUMMARY RECORD



C is lock condition indicator in bit positions 27-29 as in index record

S_3 indicates UNIVAC IIIC servos used by program

S_2 indicates UNIVAC IIA servos used by program

S_1 indicates UNIVAC IIIA servos used by program

Y indicates the presence of operational parameters if bit 29 is set.

Drum base used for drum stored programs includes drum channel normalized right in bit position 28-24.

- PROGRAM FORMAT

Program format is complex relative. Block descriptors are modified to include MIT number as the most significant part of the descriptor word during MIT creation.

Identification record

0	MIT Number	0 0 0 0 0
1	Program Name	
2	(10 characters)	
3	Programmer	
4	(10 characters)	
5		
6	Y	drum base of program or zero
7		3
8	MIT Number	0 0 0 0 0

Y parameter indicator as in Program Facility summary record

MIT Number	0 0 0 0 1

(Facility record)

MIT Number	0 0 0 0 2

(Segment description record)

MIT Number	0 0 0 0 3

(File description record)

MIT Number	0 0 0 0 4

(Control Segment record)

MIT Number	Segment Number

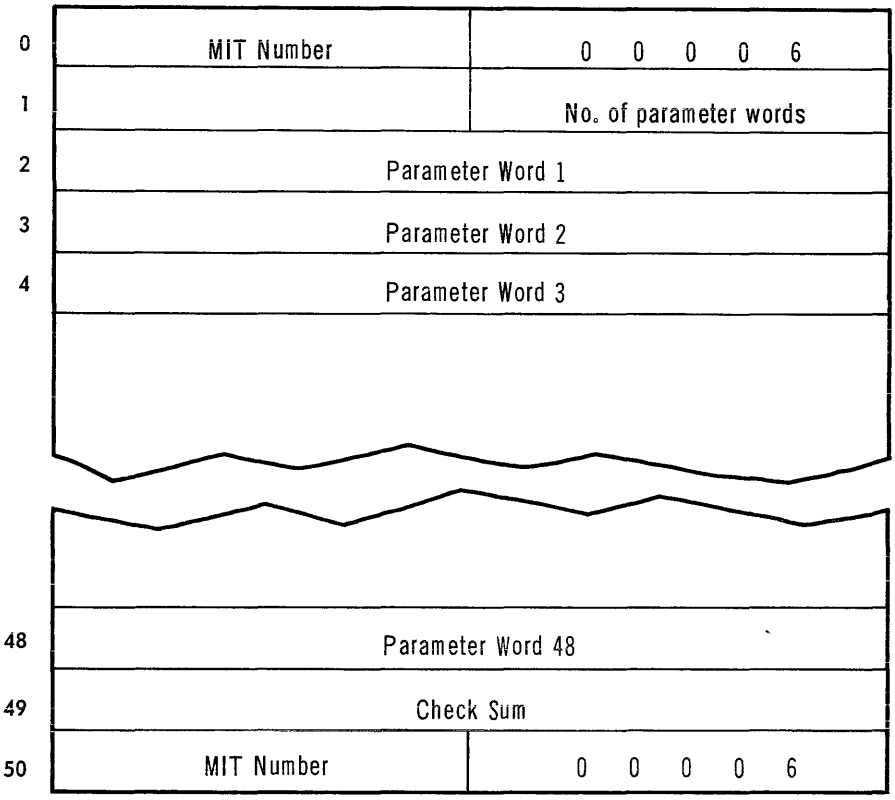
(Secondary Segment record)

Only the first block of each record bears descriptors as shown. Subsequent blocks bear descriptors of binary zeros. This also applies to parameter records.

When programs are drum stored, only the identification record, parameter records, and end of program sentinel appear on MIT.

- OPERATIONAL PARAMETERS

Operational parameters may be inserted within a program at the time an MIT is created. The parameter record generated immediately precedes the end of program sentinel of the program. Parameters are loaded automatically when the program is selected and loaded.



(51 word block)

4. MITAR card parameters

MITAR II accepts parameters through the REX parameter entry mechanism.

- MIT CARD

A MIT Card is used to assign a number to the Master Instruction Tape that is being created.

COLUMNS	CONTENTS
1	C
2, 3	02
11 – 13	MIT
16 – 17	(Octal number to be assigned)

- CONTROL CARDS

Control cards define priority classifications, strings, and lock conditions.

COLUMNS	CONTENTS
1	C
2, 3	02
11 – 20	PRIORITYx (All programs grouped under this header will be scheduled in the specified priority classification and x must be 0-7.) STRING (Program cards which follow this header define a string. Order of execution will correspond to card sequence.) LOCK (If this card precedes a program card, the program will be placed in a lock condition. If it precedes a string card, the string will be placed in a lock condition.)

- PROGRAM CARDS

Program cards specify programs to be placed on the MIT. In addition to specifying programs from the object code library, program cards may be used to schedule programs which will be on the drum at execution time.

COLUMNS	CONTENTS
1	B
2, 3	03
21 – 25	(Library number right justified)
26 – 28	(an octal estimate of program run time relative to those of other programs – optional)
29 – 30	(An estimate of the ratio of computer time to basic processing cycle time expressed in tenths – 0 to 12_8 – optional.)
31 – 40	(A ten octal digit drum or FASTRAND address of the program. Two high order digits indicate channel. If program is not to be loaded from the drum, this field must be left blank.)

- OPERATIONAL PARAMETER CARDS

Parameter cards are prepared in standard REX format and are grouped by program. Each group is prefixed with a BEGIN PRAM card, suffixed with an END PRAM Card, and incorporated into the input deck immediately following the program card to which it relates.

COLUMNS	CONTENTS
1	C
2, 3	02
11 - 20	BEGIN PRAM END PRAM

C. CATUT

CATUT (Card to Magnetic Tape Utility Routine) is a generalized program designed to read punched cards and form the resultant images into a tape file for which block length and item size are specified by variable parameters. The routine is designed for operation in a multi-program environment. The need for off-line card to tape equipment may be eliminated by the use of this routine.

1. System Requirements

Minimum system requirements consist of:

- $2{,}045_8$ words of core storage (includes buffers)
- a card reader
- a tape unit

REX is used to control input/output requests.

2. Program Structure

CATUT consists of an Initialization Routine, an End File Routine, and several major subroutines, all joined by a relatively simple control thread. This structure facilitates modification of the program, if desired. A brief description of the major routines and subroutines with entry and exit labels and other pertinent information is presented at the end of this section. These descriptions will further facilitate program modification. Provision is made for optional inclusion of own code. The structure of CATUT allows own code to assume any degree of responsibility desired. Any of the routines of which the package consists may or may not be used by the own code and may or may not exit to END FILE.

The presence of own code is indicated by placing the first address in the lower half of location NCODE. Entry is by indirect jump.

3. Card Input Composition and Arrangement

a. Types of Cards

- CONTROL CARD

  The control card is a Hollerith card that will be translated to Fieldata code when read. Columns 30 and 34-35 are pertinent only if input mode is translation. The format of the card is as follows:

COLUMNS	CONTENTS
1 – 5	CCCCC
10	Defines input mode as follows: (blank) – Translation 1 – Column Binary 2 – Row Binary
14 – 15	The number of computer words comprising an input item – a right justified octal number.
18 – 20	The number of items per tape block – a right justified octal number.
25	Type of run: (blank) – Normal Run 1 – Restart – type 1 ① 2 – Restart – type 2 ①
30	Action to take in event of illegal character error: (blank) – Stop to allow operator correction card. 1 – Leave image as read. Set B_2 non-zero to indicate error. 2 – Overlay image with that of next card. 3 – Terminate run.
34 – 35	Illegal character error diagnostic option: (blank) – Perform no typeout nn – The number of card columns to be typed starting with column 1. A maximum of 70 is permitted. The number is octal and right justified.

① See description of INITIALIZATION ROUTINE for more detailed discussion.

- LABEL CARD

One or more label cards must follow the control card. A label card is a Hollerith card that is translated to Fieldata code when read. A standard tape label will be constructed on the output tape based on this information. The format of the label card is as follows:

COLUMNS	CONTENTS
1 – 10	Ten semicolons (a label identifier).
11 – 25	Alphanumeric file identifier
26 – 30	Alphanumeric date.
34 – 35	Governs use of data from columns 40-80. (blank) – ignore columns 40-80. nn – place the contents of columns 40-80 into supplementory label words starting with the word specified by nn (a right justified octal number).
39	(blank) – last label card. 1 – another label card follows.
40 – 80	Column 40 must be A if the field is alphanumeric, blank if octal. *Alphanumeric* Each card column (including blanks) represents one Fieldata character. Characters from columns 41-80 are stored into eight consecutive words (five characters per word) starting with the label word specified by columns 34-35. *Octal* Each card column represents an octal digit. Ten digits comprise an octal word, which is the smallest unit of data that will be stored. Storage is as specified by columns 34-35 (columns 41-50 go to nn, 51-60 go to nn + 1, etc.). A non-octal character within a word causes that word and any remaining words to be ignored.

11-30 are moved to the proper label words without alteration.

For second and subsequent label cards, only 34-80 will be considered.

- DATA CARD

A data card may be in any format which will not result in its being mistakenly identified as a sentinel.

- SENTINEL CARD

A sentinel card must follow the last data card, and in turn must be followed by a minimum of three blank cards to allow for a possible change in input modes between files. Conventions for end of file sentinel blocks for the three input modes follow.

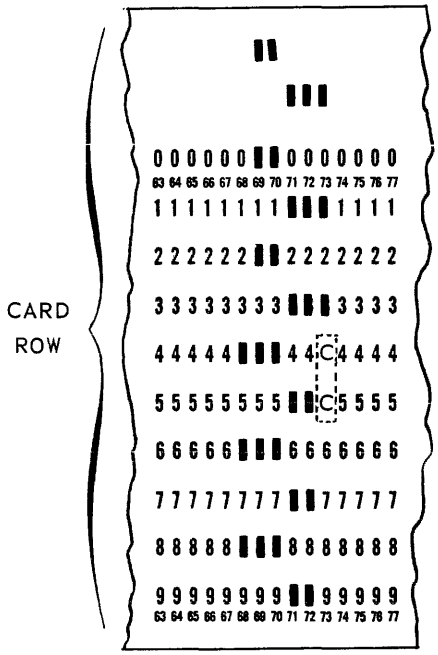
(1) Input Mode Translation

Each card read is tested to see if it is the end of file sentinel described below:

COLUMNS	CONTENTS
1 - 10	(Ten Fieldata "special characters" (76 code) (blank) – Terminate CATUT 1 – Another file, new reel 2 – Another file, same reel

(2) Input Mode Column Binary

Each card read is tested to see if it is the end of file sentinel shown:



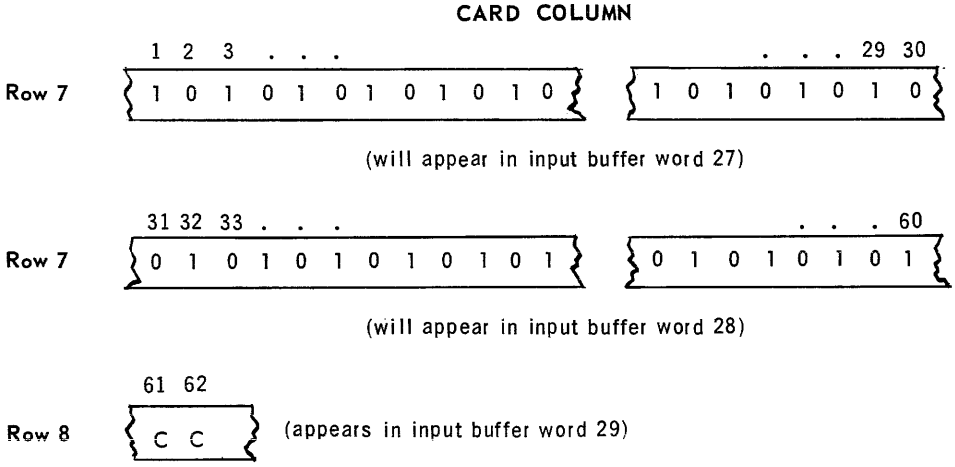
Rows 4-9 of column 68 and all rows of columns 69 and 70 appear in input buffer word 27. All rows of column 71, 72 and rows 12-3 of column 73 appear in input buffer word 28. The two bits indicated by CC in rows 4-5 of column 73 appear in the high order bits of input buffer word 29. CC is interpreted as follows:

CC	
0 0	Terminate CATUT
0 1	Another file, new reel
1 0	Another File, same reel

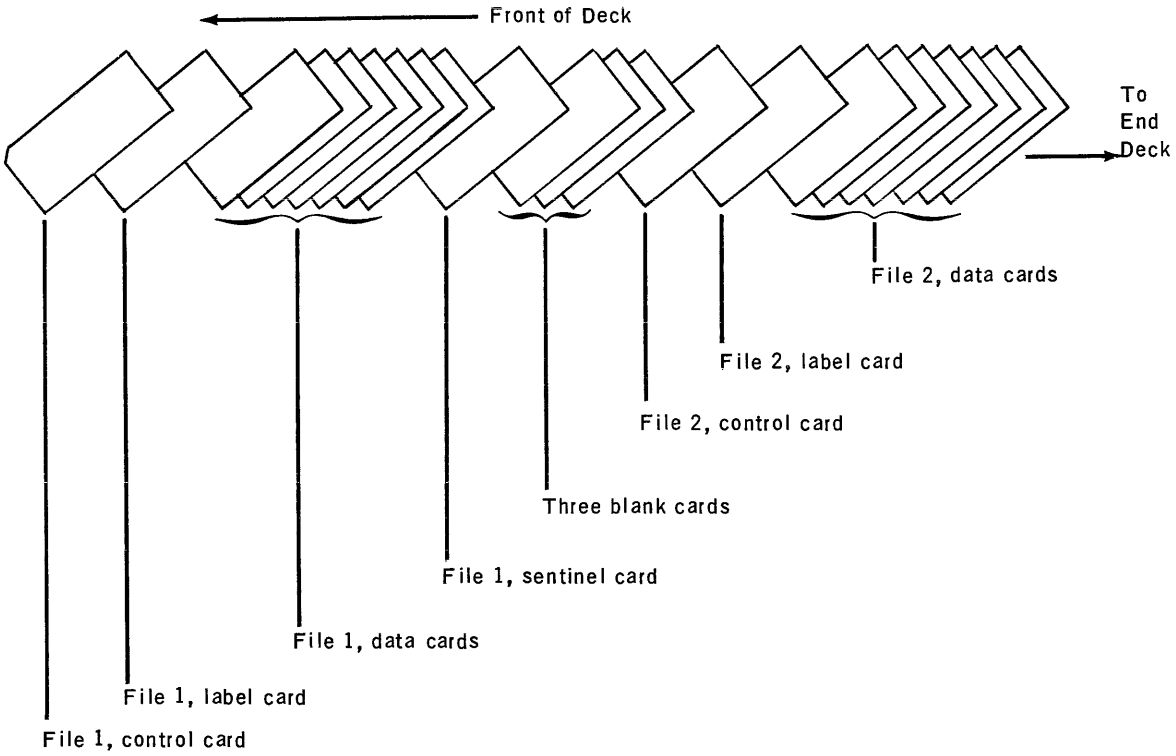
(3) Input Mode Row Binary

Each card read is tested to see if it is an end of file sentinel. An end of input sentinel in the row binary mode of input consists of alternate punches and blanks in card columns 1-60 of row 7, as shown in the figure below. Columns 61 and 62 of row 7 contain CC, which is interpreted as follows:

00	Terminate CATUT
01	Another file, new reel
10	Another file, same reel



where 1 = punch
0 = blank



- OUTPUT TAPE COMPOSITION

Each input deck processed results in creation of a magnetic tape file comprised of a label block, one or more data blocks and two end of file sentinels, in that order.

(1) Label Block

The label written is a conventinal data file label unless modified by Supplementary Label Data.

(2) Data Blocks

Data is recorded in conventional nonsearchable data block format.

(3) Sentinel Blocks

End of reel and end of file sentinels are of conventional format. Words 3 through 21 of the sentinel card image will be copied into words 3 through 21 of the end of file sentinel.

Appropriate own code can, of course, create any output tape format desired.

4. Description of Routines and Subroutines in CATUT

The following description of the routines and subroutines of which CATUT is composed are presented to clarify the operations of CATUT and as an aid to determine possibilities for program modification to make CATUT conform more closely to the requirements of individual installations. Actual program modification would require the use of Technical Documentation including flow charts and program coding.

- INITIALIZATION ROUTINE

This routine clears card memory and commences moving one card at a time through card memory into core until a control card is sensed. Input mode choice, input item length, number of items per block, illegal character error option, the associated diagnostic option, and the restart field will be recorded for future reference.

Label card is read and a label is assembled in consecutive words starting at LABEL. Reel number is set to one.

The restart field is tested to see if this is a restart. If it is, a search is initiated for the specified file. When the file has been located, the reel number is used to replace that in the LABEL assembly area, and the output tape is positioned so that writing may commence. Restart type controls tape positioning. If type 1, tape is positioned so that writing commences immediately following the last recorded block. If type 2, tape is positioned so that the last recorded block is overwritten.

Once the output tape has been positioned, control reverts to the normal flow exactly as if restart had not been specified. The next action is to set the proper card input mode and test for presence of own code. If own code is present, control is transferred thereto. At this point the output tape label has been assembled and the restart field has been set to show the nature of the run. If own code is not present, the label is written and processing started. If this is a restart, label write is bypassed.

- READ SUBROUTINE

This subroutine controls the card reader, acquiring card images on demand.

READ tests each card read to see if it is an end-of-input sentinel. When an end of input sentinel is detected, the end of input flag is set and the continuation indicator stored.

Subsystem errors will be corrected if possible. Detection of an illegal character invokes action as specified by the control card. In event of noncorrectable error, the subroutine exits to L(ERROR) with the U(ERROR) set to one.

Entry:	READ	
Entry Parameters:	None	(This routine does *not disturb* B1.)
Exit Parameters:	U(NPUTEND)	77777 if sentinel detected, otherwise zero. Continuation indicator in L(NPUTEND)
	B2	Set non-zero to signal illegal character (option 1), otherwise zero.
Output:	Card image in consecutive words starting with location NPUTB	
EXIT:	L(READ)	

- ITEM ASSEMBLY SUBROUTINE

This subroutine builds an input item in consecutive words starting at the location specified by B1. An item is built word by word using card image words in order starting with the first. If item size is not an integral multiple of card image length, excess image words will be ignored. The READ subroutine is used to obtain card images.

The Item Assembly Subroutine will recognize the end of input flag set by READ and exit. As on normal exit, the count of words assembled will be right-justified in Q. The end of input flag remains set.

Entry:	ASTEM	
Entry Parameters:	B1	Starting location for item assembly
Exit Parameters:	B1	Same as at entry
	Q	Count of words assembled
EXIT:	L(ASTEM)	

- WRITE SUBROUTINE

This subroutine writes the area of core memory specified by the entry parameters onto the output servo. A count of words written since the tape was first mounted is maintained and used to calculate end of tape. When end of tape is reached, two conventional end of tape sentinels are written, the tape rewound, and operator intervention solicited. When a new reel has been mounted, the subroutine duplicates the label last written, updating reel number.

In event of tape error, the subroutine exits to L(ERROR) with the U(ERROR) set to 2.

Entry:	WRTE	
Entry Parameters:	B1	Starting address of output buffer
	Q	Number of words to output
Exit Parameters:	None	
Exit:	L(WRTE)	

- ERROR ROUTINE

The primary function of the error routine is to make available to the operator information necessary to restart after the condition precipitating error has been rectified.

In order to make restart as simple as possible, any partially assembled items or blocks will be disregarded. The last block successfully written on tape represents the last successfully processed card or cards. Before diagnostic typeout, the error routine reads this block into the output buffer so that it may be dumped using the REX core dump feature. Termination of the run is left to operator discretion.

In order to resume processing, the operator must ascertain the first unprocessed card, and place the control card (modified to contain the appropriate restart punch) and the label card or cards in front of the unprocessed deck. CATUT may then be started at its beginning address. These same procedures apply even though the run is terminated at time of error and resumed another day.

Entry:	L(ERROR)		
Entry Parameters:	U(ERROR)	00001	(A type 1 restart will be required.)
		00002	(A type 2 restart will be required.)
Exit Parameters:	None		
Output:	Diagnostic		
Exit:	REX.STOPRUN		

- ENDFILE ROUTINE

This routine writes two end of file sentinels on the output tape and performs the action appropriate for the continuation indicator contained in L(NPUTEND).

The contents of BLKCT are used as the count of tape blocks (including labels and sentinels) already written.

Entry:	ENDFILE	
Entry Parameters:	L(NPUTEND)	00000 (terminate run)
		20000 (another file, new reel)
		40000 (another file, same reel)
Exit Parameters:	None	
Exit:	CATUTO or REX.TERMRUN	

SUMMARY OF CATUT LABELS, VARIABLES, AND INDICATORS IMPORTANT TO OWN CODE

- LABELS

READ	Card Read Subroutine
ASTEM	Item Assembly Subroutine
ASBLK	Block Assembly Subroutine
WRTE	Write Tape Subroutine
ENDFILE	End of File Procedure
NPUTB	36-word card input buffer
LABEL	24-word label assembly area starting address
TPUTB	256-word output buffer
ERROR	Error Routine
NCODE	Own Code Entry

- VARIABLES

W(BLKCT)	Output Block Count
L(MANDT)	Type of run
U(MANDT)	Input mode
L(TEMBLK)	Items/block
U(TEMBLK)	Input item length
L(COPT)	Illegal character error option
U(COPT)	Illegal character error diagnostic option

- INDICATORS

U(NPUTEND)	End of input flag
L(NPUTEND)	Continuation Indicator
U(CATUTICT)	Block Assembly Status

For symbolic addresses other then those listed above, CATUT uses only the CATUTXX form.

D. PRINTAPE

PRINTAPE (Magnetic Tape to High Speed Printer Utility Routine) is designed to read a magnetic tape that has been edited for printing and to print the records contained on the tape on a High Speed Printer. The tape to be printed must contain a label block which contains the necessary parameters to set up spacing, margin, page numbering, and other controls, on the printer. Some of the more important features of the PRINTAPE Routine are:

- printing of a test pattern to allow the operator to adjust the paper position as desired.
- determination of both vertical and horizontal margins.
- page numbering
- provision for spacing defined by parameters and the print function word.
- extensive error recovery capability

1. System Requirements

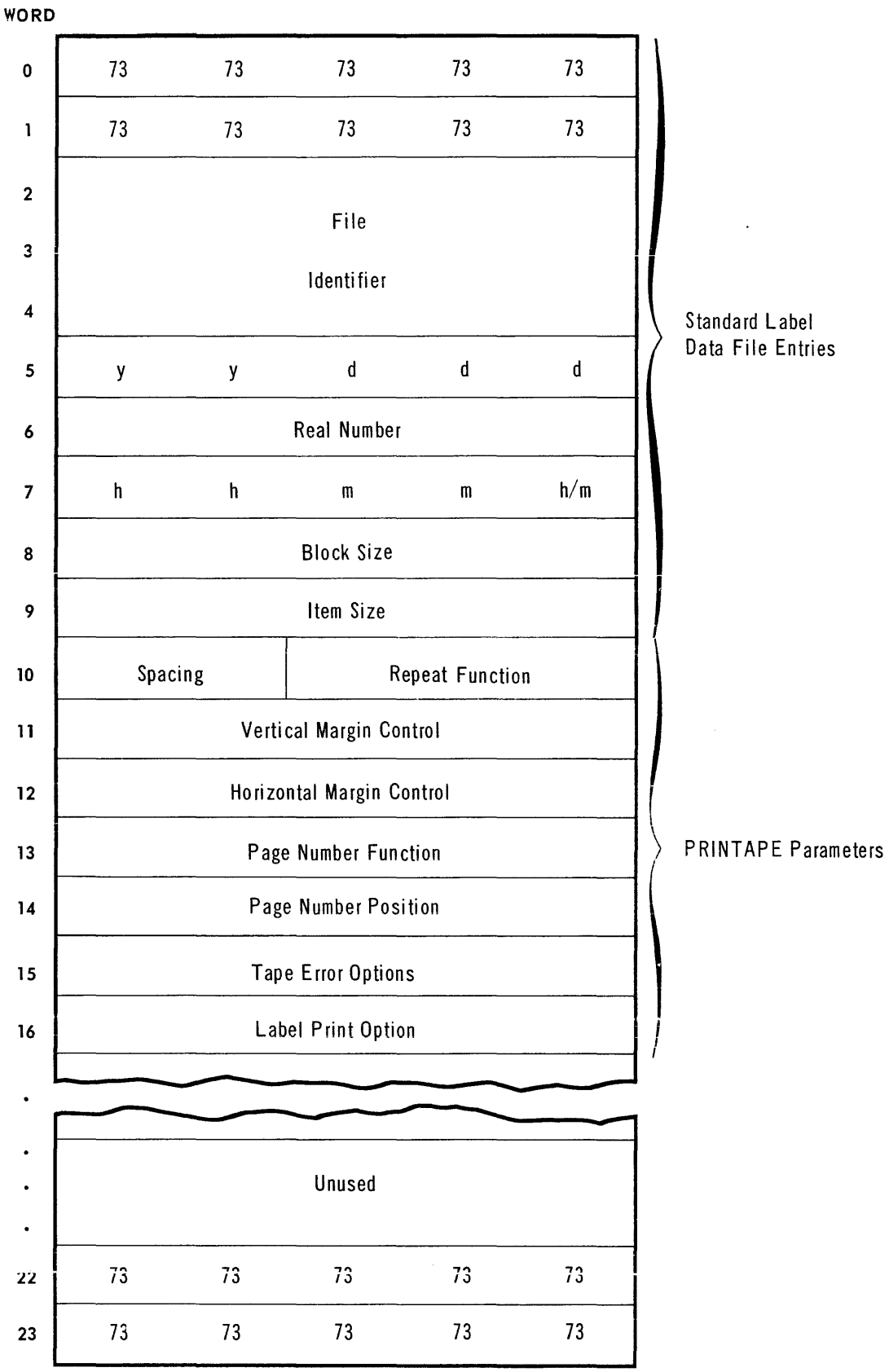
Minimum equipment requirements consist of:

- 1560_8 words of core storage
- a UNISERVO tape unit
- a High Speed Printer

REX is utilized for program loading and for input/output requests.

- LABEL BLOCK

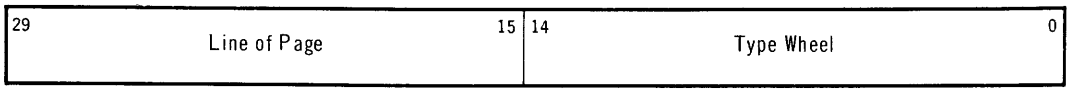
  The label block identifies the file to be processed and supplies the parameters necessary for determining spacing, vertical and horizontal margin control, page numbering, and tape error options. The format is:


WORD
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
22
23
73 73 73 73 73
73 73 73 73 73
File
Identifier
y y d d d
Real Number
h h m m h/m
Block Size
Item Size
Spacing
Repeat Function
Vertical Margin Control
Horizontal Margin Control
Page Number Function
Page Number Position
Tape Error Options
Label Print Option
Unused
73 73 73 73 73
73 73 73 73 73
Standard Label
Data File Entries
PRINTAPE Parameters

WORD	DESCRIPTION
0,1,22,23	Each contains five Fieldata ";" characters (73_8) used for block identification.
2-4	Fifteen Fieldata characters used to identify the data file to be processed.
5	Year and day of the year.
6	Reel number, expressed as a right-justified octal number.
7	Day Clock reading in hours and minutes, in Fieldata code. Half-minutes are indicated by 0 and 30 seconds, + denotes the half-minute between 30 and 60 seconds.
8	Block size expressed as a right-justified octal number showing the maximum number of computer words contained in a data word, including descriptors.
10	Repeat Function (0-14). If equal to 0, no test pattern is to be printed. If not equal to 0, a test pattern of (123456789012.....) is to be printed until terminated by the operator. Spacing (15-29). A right-justified octal value giving the number of lines to be spaced between each two printed items. If 0, own-spacing is indicated, and number of lines of spacing will be obtained from the last word of each item.
11	Vertical margin control format is: 29 Zero 21 | 20 T.M. 15 | 14 N.L.P. 6 | 5 B.M. 0 T.M. – Number of lines in top margin. B.M. – Number of lines in bottom margin. N.L.P. – Number of lines to be printed per page. The sum of T.M., B.M. and N.L.P. equals the physical length of the page. If physical page length is zero, PRINTAPE cannot perform page numbering and skipping functions.
12	In right-justified octal, the number of spaces required for the left horizontal margin of the page.
13	Page number function, in the following format: 29 K 15 | 14 Base Number 0

<table>
<tr><th>WORD</th><th>DESCRIPTION</th></tr>
<tr><td></td><td>If K is not equal to 0, K expresses in octal the increment to be added to the base number the second and each subsequent time a page number is printed.

If K = 0, no page numbering is required. The base number is the right-justified octal equivalent of the decimal number to be printed on the first page following the test pattern. Page numbers are always printed in decimal.</td></tr>
<tr><td>14</td><td>If page numbering was requested (Word 13), then this word gives the position in which the right-most digit of the page number is to appear. Its format is as follows:

Line — A right-justified octal expression for the number of lines down from the top line of the page, to the line on which page number is to be printed.

Type Wheel — A right-justified octal expression for the number of type wheels, counting from left to right, from which the right-most digit of the page number is to be printed.</td></tr>
<tr><td>15</td><td>A right-justified octal value specifying the action to be taken if a tape error has occurred:

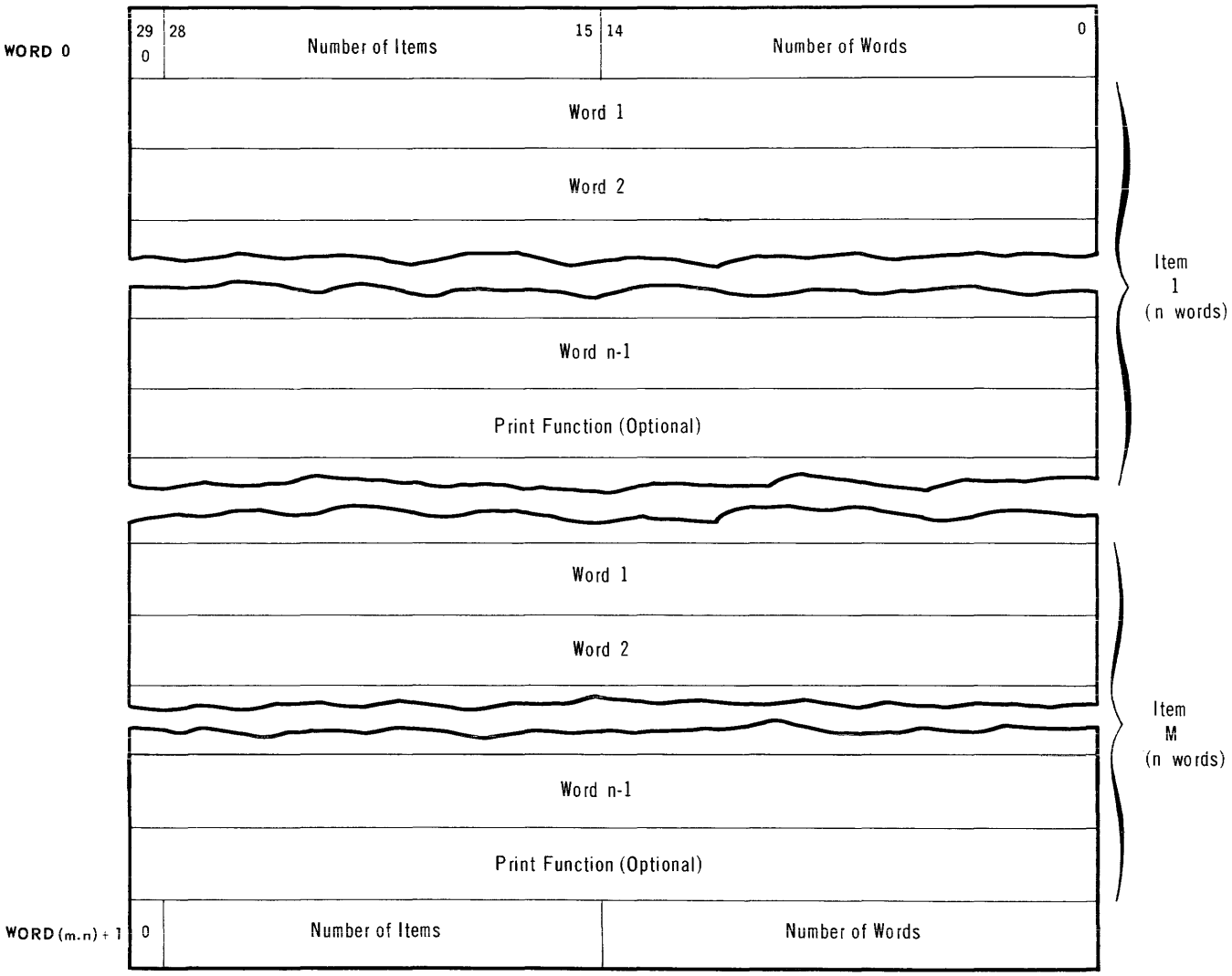
0 Terminate processing

1 Move forward one block and continue processing</td></tr>
<tr><td>16</td><td>If this word is zero, the file label will be printed preceding the test pattern. If this word is not zero, label printing will be bypassed.</td></tr>
</table>

- DATA BLOCK

Each data block is made up of one or more items to be printed. Maximum length of any one data block is 272 computer words, the first and last of which are descriptors. Any item within a datablock may have a maximum length of 26 computer words (27 words if the own space option is used). Words within an item must be in Fieldata code, except for own space function word entries.

A data block is composed of one or more items to be printed, as follows:



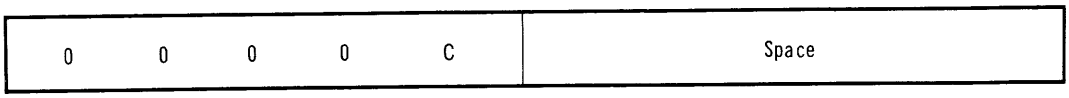
Each item represents one line of data to be printed. The number of words within each item is indicated by item length (see Label Block, Word 9).

(1) Block Descriptors
Words 0 and (m.n)+1 contain:

Number of Items (15-28)	A right-justified octal expression for the number of items within the data block.
Number of Words (0-14)	A right-justified octal expression for the number of computer words within the data block.

(2) Print Function Word

The last word of an item if own-spacing is required. It specifies the number of line-spaces between items.



Space A right-justified octal value for the number of lines to space before beginning to print.

C A three-bit value specifying the spacing option to be taken:

0 Normal. Use spacing indicated, and print. If overflow occurs, print on first printline of next page.

1 Supress overflow. Use spacing indicated, and print. Disregard any overflow which may result until the next normal or skip item is printed.

2 Skip to next page and print first printline. Ignore indicated spacing.

- SENTINELS

(1) Bypass Sentinel (74 74 74 74 74)

When a Bypass sentinel is sensed, printing is stopped and the routine initiates a search for the next Bypass sentinel. Printing resumes with the record immediately following this second Bypass sentinel.

(2) End of Reel Sentinel (75 75 75 75 75)

When an end of reel sentinel occurs, the routine initiates a rewind of the tape just processed, and informs the operator:

END OF REEL xxxxx
MOUNT NEXT REEL Css Sxx
WHEN MOUNTED ENTER S (S)

Processing resumes when an (S) answer is received.

(3) End of File Sentinel (76 76 76 76 76)

Detection of an End of File sentinel causes the routine to type out:

END OF FILE
INSTRUCT

Response is as described under program initiation. See OPERATING PROCEDURES, Utility Routines.

E. TRACE IV

TRACE IV monitors and records on the High Speed Printer the results of instructions that are executed by the program that is placed under its control. TRACE IV can be run under the control of REX (Real-Time Executive Routine) or it can be run as an independent program.

1. System Requirements

Minimum requirements consist of:

- 2670_8 words of core storage
- a High Speed Printer

2. Output Format

As an instruction is executed, TRACE IV produces a High Speed Printer listing of the contents of the following registers:

REGISTER	EXPLANATION
P	Core storage address of the executed instruction
A	Contents of the A register
Q	Contents of the Q register
U	Contents of the U register (the executed instruction as modified by the specified B register)
OP	The operand as determined by the specified k designator
B1 through B7	Contents of the individual B registers

3. Operations Performed

TRACE IV will perform the following operations:

- monitor all the instructions within a program.
- monitor all instructions within a specified area of a program.
- monitor all instructions that modify the contents of a specified address.

In addition to the instructions in the program, the instructions in the input/output routines, in REX, and in the interrupt routines will be monitored when control is transferred to them.

Example of TRACE IV High Speed Printer listing:

```
P 01705 A 0C00000002 Q 2050000000 U 1501001675 OP 0000000002 B1 50000 B2 50031 B3 00000 B4 00000 B5 00000
                                                                                B6                B7 03205
P 01706 A 0C00000000 Q 2050000000 U 1100000000 OP 0000000000 B1 50000 B2 50031 B3 00000 B4 00000 B5 00000
                                                                                B6                B7 03205
P 01707 A 0C00000004 Q 1200000000 U 0700000004 OP 0000000004 B1 50000 B2 50031 B3 00000 B4 00000 B5 00000
                                                                                B6                B7 03205
P 01710 A 0C00000004 Q 1200000000 U 1502001755 OP 0000000004 B1 50000 B2 50031 B3 00000 B4 00000 B5 00000
                                                                                B6                B7 03205
P 01711 A 0C00000004 Q 1200000000 U 1277000000 OP 0000000004 B1 50000 B2 50031 B3 00000 B4 00000 B5 00000
                                                                                B6                B7 00004
P 01712 A 0C00000004 Q 1200000000 U 0000701160 OP FAULT       B1 50000 B2 50031 B3 00000 B4 00000 B5 00000
                                                                                B6                B7 00004
```

4. Parameters

If TRACE IV is to be run under the control of REX, it must be loaded as errata of the program that is to be monitored. In this case, the parameters will be entered via the Console Keyboard by use of the register entry option associated with Program Start (PS).

If TRACE IV is to be run independently, the parameters will be entered via the maintenance panel.

- Control Parameters

The following parameters must be placed in the indicated registers before initiating TRACE IV:

PARAMETER	REGISTER
Base address of TRACE IV	P
First address of the area in the program that is to be monitored.	B1
Last address of the area in the program that is to be monitored.	B2
Executive control specification. If set to 0, TRACE IV is to be run independently. If set to 1, TRACE IV is to be run under the control of REX.	B3
Specified address (optional, see listing option)	B4
Listing option. If set to 0, a High Speed Printer listing will follow the execution of each instruction within the area specified. If set to 1, a listing will follow the execution of all instructions within the area specified with the exception of those that are in subroutines. If set to 2, a listing will follow the execution of Jump and Return Jump instructions only. If set to 3, a listing will follow the execution of each instruction that modifies the contents of the specified address.	B5
Starting address of the program that is to be monitored.	B6

It should be noted that if the B1 and B2 registers are set to 0, every instruction in the program (depending upon the listing option that is selected) will be followed by a High Speed Printer listing.

- Monitored Program Parameters

  After TRACE IV has stored the control parameters, it will relay a message via the Console Printer requesting the parameters for the program that is to be monitored. When this occurs, the program parameters may be entered into the index and operational registers via the Console Keyboard or via the maintenance panel.

F. RMASL

RMASL (Routine for Maintaining a Source Language Library) is used to create and maintain magnetic tape files of source language that may be used as input to the SPURT assembler.

1. System Requirements

Minimum requirements consist of:

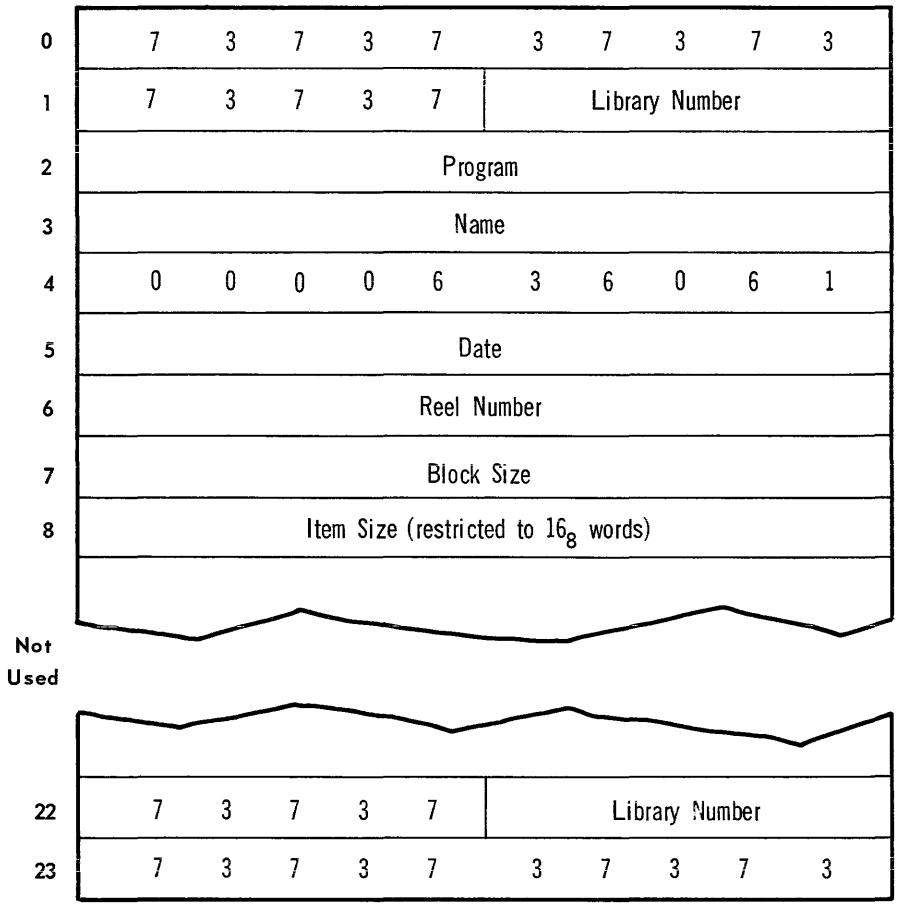
- $2{,}300_8$ words of core storage
- 3 UNISERVO Units
- a Paper Tape or Card Reader
- a High Speed Printer and/or a Card Punch

REX (Real-Time Executive Routine) is utilized for program loading, input/output requests, and the entry of parameters which define variables in the program. A maximum of 50 parameter cards may be entered.

2. Input Format

RMASL will accept source language as formed by SPURT (Output Number 301) or as formed by the card to magnetic tape routine, paper tape to magnetic tape routine, or any routine producing magnetic tape output in standard format.

- a label block

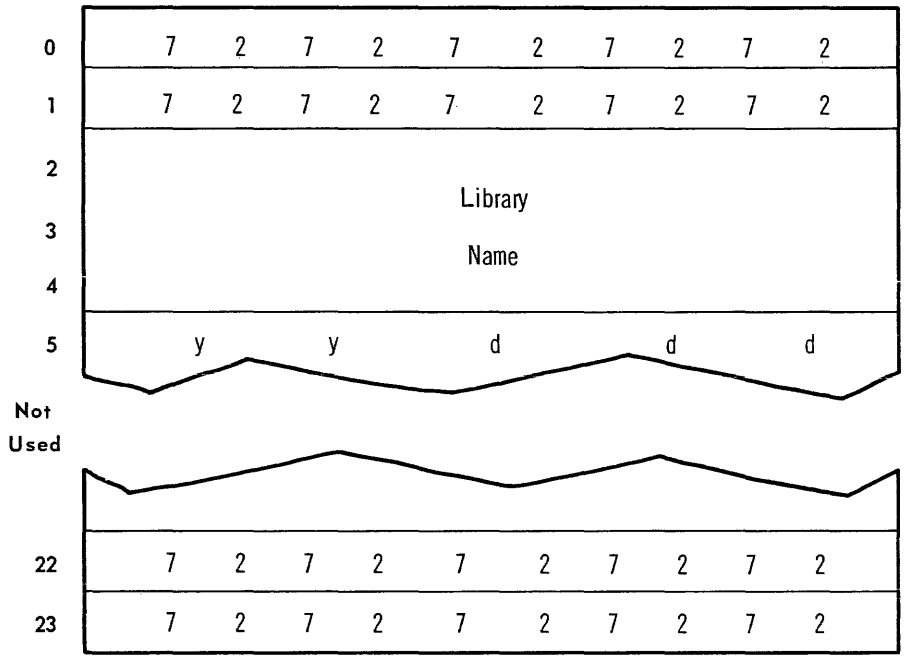


- one or more data blocks. The data blocks are restricted to 17_8 items per block with the exception of the last data block in the program. This block may contain less than 17_8 items. Each item consists of one line of SPURT coding in Fieldata code which utilizes 16_8 computer words. Each data block contains standard block descriptor words that give the number of items within the block and the length of the block.
- an end of file sentinel block.

3. Library Tape Format

A library tape is composed of:

- a label record



- any number of SPURT source language programs and/or data files.
- two standard end of file sentinel blocks.
- a tape mark if UNISERVO IIIC tape units were used for output.

4. Operations Performed

RMASL will perform the following operations:

- create a library.
- update an existing library.

Coincident with either of the above functions, or as a separate pass, to print and/or punch a library directory.

Any source language program in the prescribed format or any data file that is in standard data file format may be included in a library. As part of the placement process, a five-digit octal number is assigned to each program or data file. The source language programs and data files within a library will be arranged in ascending sequence by library number. RMASL also modifies the label blocks of these programs and files so that they may be printed by using the PRINTAPE routine (see PRINTAPE Routine, this section).

5. Parameter Cards

- Function Card

  A function card is used to specify the function to be performed. There are two functions:

  BLD – Form the Source Language programs into a new library.

  UPD – Operate upon an existing library to add and/or delete programs and produce an updated library.

COLUMNS	CONTENTS
1	C
2, 3	05
11 – 13	BLD or UPD
16 – 30	(The name of the library being built or updated.)

- Program Cards

  BLD Function

  One or more program cards may follow the BLD card to identify specific programs from the input tape which are to be placed in the library.

  Absence of program cards implicitly specifies all programs from the input tape are to be placed in the library. In this case, RMASL will automatically assign library numbers commencing with 10_8. Successive assignments will be made by adding 10_8 to the preceding number.

  UPD Function

  One or more program cards may follow the UPD card to identify specific programs from the input tape which are to be added to the library and/or programs which are to be deleted from the library being updated. Absence of program cards implies that all programs from the input tape are to be added to the library. In this case, RMASL will automatically assign library numbers commencing with the highest number on the old library plus 10_8. Successive assignments will be made by adding 10_8 to the preceding number.

COLUMN	CONTENTS
1	C
2, 3	05
11	D if delete, otherwise blank.
16 - 25	(Program Name)
26 - 30	(Old library number – five-digit octal value. Library number is required if the program is being deleted. Otherwise field must be left blank.)
31 - 35	(Assigned library number – five-digit octal value. If the field is left blank, RMASL will assign a library number.)

- Directory Cards

A directory card or cards may be included with or may constitute the input deck.

COLUMN	CONTENTS
1	C
2, 3	05
11 - 13	PRT (Print a library directory) or PCH (Punch a library directory).

- New Input Tape Card

If more than one input tape is to be processed, the second and subsequent tape must be identified so that RMASL may direct tape mounting.

COLUMN	CONTENTS
1	C
2, 3	05
11 - 13	NIT
16 - 30	(The name to be used in directing tape mounting.)

- Order of the Parameter Deck

  (1) Directory Card(s)

  (2) Function Card

  (3) Program card(s) for program on initial input tape

  (4) New Input Tape Card

  (5) Program card(s) for program on last-named input tape

  (6) Four or more blank cards to signify the end of the parameter deck

  A maximum of 50 parameter cards is allowed. Only one function card is permitted for any given run.

G. CIMCO

CIMCO (Card Image Corrector Routine) is used to make corrections to a source program that is on magnetic tape in input language format (SPURT Output Number 301). Changes to a source program (deletions, replacements, and additions) are effected by preparing a card deck.

1. System Requirements

Minimum requirements consist of:

- 1750_8 words of core storage.
- 2 UNISERVO units.
- a Card Reader.

REX (Real-Time Executive Routine) is utilized for program loading and input/output requests.

2. Operations Performed

CIMCO will perform the following operations:

- delete a statement from the program.
- add a statement to the program.
- replace one statement in the program with another.
- copy an input tape onto an output tape until the program to be corrected is found, make the required changes to that program, and continue copying until end of file or the next program to be corrected is reached.
- selectively copy a program or programs from a tape containing several programs onto an output tape.

It should also be noted that if an end of file is encountered before the card that indicates the end of the correction run is read, the current input tape will be rewound. When this occurs, the user can terminate the correction run or mount a new tape and continue.

3. Correction Deck Format

The correction deck consists of a header card followed by one or more correction cards that contain the word DELETE (for a deletion), the statement DELETE.n (for a series of deletions- n is the number of deletions), a statement that is to be added, or a statement that is to replace an existing statement. The operations that are involved and the points where new operations are to be added, are located by means of the L1 identifiers that were assigned during the initial assembly. The identifiers can be determined by examining a High Speed Printer Output Number 101 or 110 of the source program. The Correction cards are punched in the standard SPURT input card format with the exception that the L1 identifier of the statement that is involved is placed in columns 73-77 and the insertion number (if needed) is placed in columns 78-80. Since columns 73-80 are never used in processing, this format allows the correction cards to be used for updating a source program deck. When a correction deck is prepared, the correction cards must be placed in the correction deck in ascending L1 identifier order; that is, the correction card with the lowest L1 identifier must be the first card after the header card, the card with the next highest L1 identifier must be the second card, and so on. The use of card numbers on correction is optional; however, if card numbers are placed on correction cards it is the user's responsibility to see that these numbers are in the proper sequence. The final correction card for a given program is followed by an end of corrections card.

If there is only one program to be corrected, the end of corrections card is followed by an end card.

If another program on the same tape is to be corrected, the end of corrections card is followed by a new header card.

If there is more than one program to be corrected on the same tape, the end card will follow the end of corrections card in the last group of corrections.

The formats of the header, correction, end of corrections, and end cards are as follows:

- Header Card

  The header card is used to specify the program that is to be corrected.

COLUMNS	CONTENTS
1 – 5	301 (right justified)
9	(Copy Indicator – optional. If the letter C is placed in this column, the input tape will be copied onto an output tape until the program to be corrected is found. The specified program is then corrected and copying continues until the end of file or the next program to be corrected is reached.)
11 – 20	(Program Name – left justified)
21 – 30	(New Program Name – left justified, optional)

It is also possible to selectively copy one or more programs from a tape containing several programs onto an output tape. This is accomplished by preparing a card deck that contains a header card and an end of corrections card for each program that is to be copied.

- Correction Card

  A correction may consist of up to three cards. The continuation symbol (col. 7) is used if the statement exceeds one card. **DELETE** and **DELETE · n** entries must be left justified beginning in column 27. Unused columns must be blank.

COLUMNS	CONTENTS
18 - 72	(1) DELETE (to delete a single statement.) (2) DELETE · n (to delete a continuous group of statements) where n – maximum 777_8 – specifies the number of statements.) (3) (A statement to be added.) (4) (A statement to replace an existing statement.)
73 - 77	(The L1 identifier of the statement that is to be added, deleted, or replaced. Leading zeros need not be punched. In the case where a series of statements are to be deleted, the L1 identifier of the first statement in the series is placed here.)

- End Of Corrections Card

The end of corrections card is used to indicate that there are no further changes to be made to a given program.

COLUMNS	CONTENTS
1 - 5	/////

- End Card

The end card is used to indicate the termination of the correction run.

COLUMNS	CONTENTS
1 - 3	END

H. CONVERSION, EDITING, AND ARITHMETIC ROUTINES

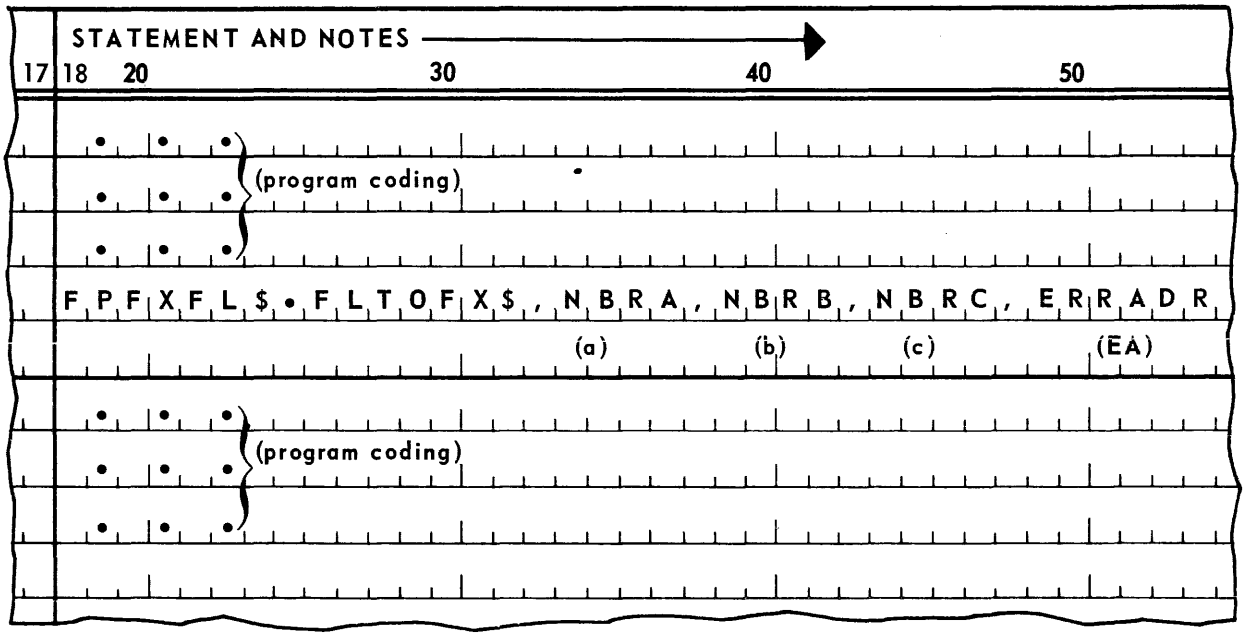
The routines here presented are part of a SPURT library of routines. They will perform the following functions:

- CONVERSION ROUTINES
  - Convert Fieldata code to binary.
  - Convert binary to Fieldata code.
  - Code conversion
  - Convert double-precision binary to Fieldata code.
  - Convert Fieldata code to double-precision binary.
- EDITING ROUTINES
  - Suppress leading zeros of a value contained in a storage location.
  - Place a dollar sign in front of the most significant digit of a value contained in a storage location.
  - Place a Fieldata asterisk code in blank spaces of a numeric field for check protection.
  - Place + or - sign in front of the most significant digit of a value contained in a storage location.
  - Merge characters contained in the upper half words of a storage location with characters in the lower half.
- FLOATING POINT ARITHMETIC ROUTINES
  - Perform floating point addition or subtraction.
  - Perform floating point multiplication or division.
  - Convert fixed point to floating point or floating point to fixed point format.
  - Evaluate floating point sine or cosine.
  - Evaluate floating point arctangent.
  - Calculate floating point square root.
  - Calculate the exponent to the base e of a number in floating point format.
  - Calculate the logarithm to the base e of a number in floating point format.
  - Load numbers in decimal format and store after conversion to floating point format.
  - Punch numbers in decimal format that have been converted from floating point format.
  - Type numbers in decimal format that have been converted from floating point format.
  - Set the number of digits after the decimal point for succeeding outputs.
- DOUBLE PRECISION ARITHMETIC ROUTINES
  - Perform double-precision addition and subtraction with and without sign check.
  - Perform double-precision multiplication and division with sign check.
- FIELDATA ARITHMETIC ROUTINES
  - Perform Fieldata addition.
  - Perform Fieldata subtraction.

Two options are provided for entry and execution of these routines, a MACRO call option and an own code option. Using the Floating Point to Fixed Point Routine (FLTOFX$) as an example, the general formats are shown in the samples which follow.

- MACRO Call Option

  For a general description of macros and the macro call line, see Section 8, PROGRAM DEFINED MACRO OPERATIONS.



FPFXFL$ will cause the entire routine package to be incorporated in the coding.

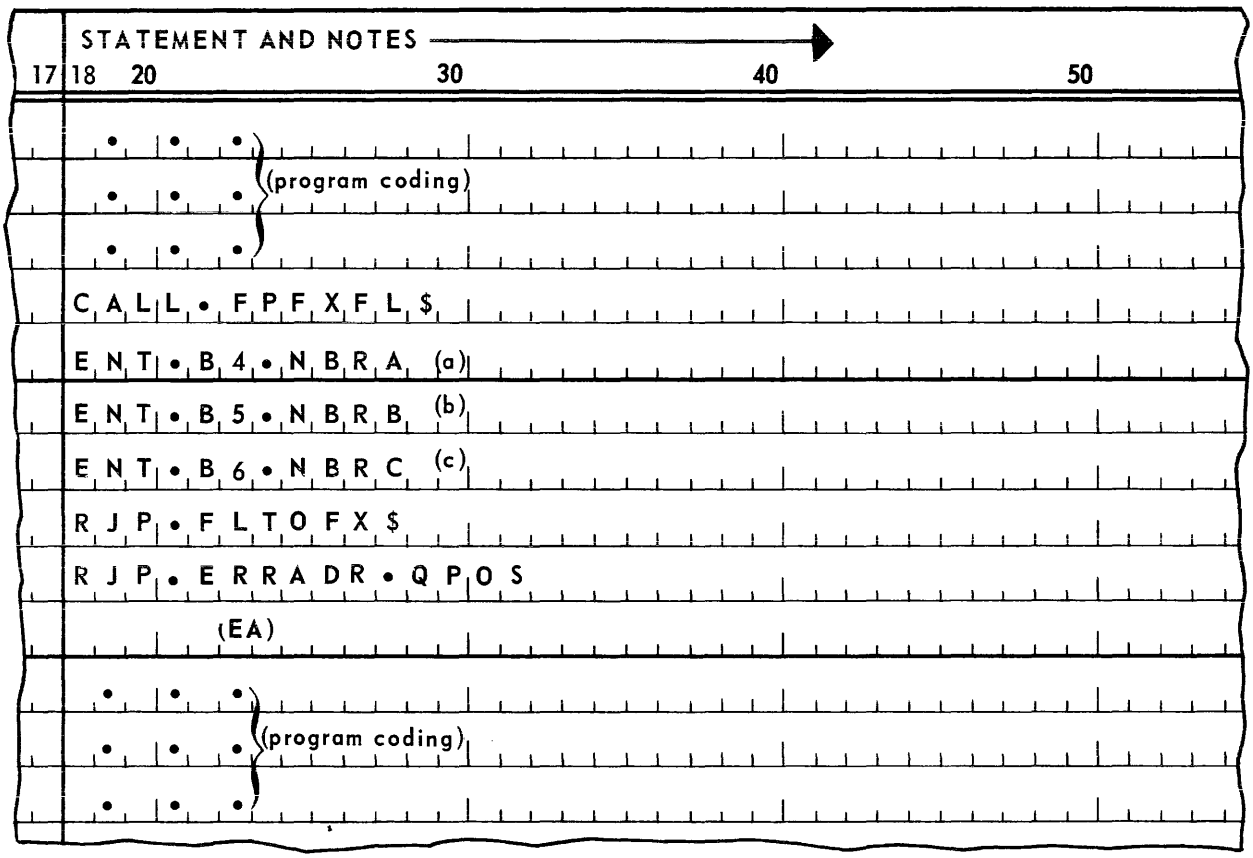
FLTOFX$ specifies the particular function to be performed. Where a routine performs only one function, such specification will be unnecessary .

a, b, c are the variable operands to be specified by the user. These will be defined for each MACRO call. The number of operands required will vary depending upon the routine that is called.

EA is an error address specified by the user, if an error address is appropriate.

- OWN CODE OPTION

  The own code option implements the library CALL or EXECUTE function. The same operations as are defined for the MACRO call shown above would appear as follows:



In this example, the entire package of routines are called into the program. The specific floating-point to fixed-point routine is entered by a return jump to FLTOFX$. This form of entry is required only where more than one function is performed by the inserted coding. The error address, EA, is entered by a similar return jump entry, where it is required.

The operands are entered into B registers where they function as parameters to the routine.

The formats for both options, and for all the various routines, are listed for quick reference at the end of this subsection.

1. Conversion Routines

The UNIVAC 490 conversion routines are designed to provide a method for conversion from one data format to another. These routines are coded in the SPURT mnemonic language and may be called from the library during SPURT assembly operation. The Conversion family consists of two basic types; namely, code conversion and numeric conversion routines.

The code conversion routine provides the means for translation between code types which are in a six bit format, for example Binary Coded Decimal to Fieldata. Provision is also made for adding user-defined translation capabilities.

The numeric conversion routines provide a means for conversion between a machine oriented binary format and a given data code format for example, binary to BCD, or Fieldata to double-precision binary.

– Parameters

The conversion routines are directed by a series of parameters provided by the worker program. In general, these parameters specify the size and location of operand fields and are provided to the routines in various index registers.

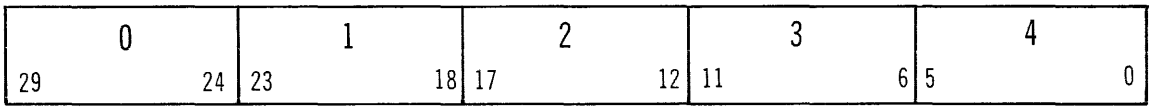
Parameter settings will destroy the contents of registers used for this purpose; consequently, the worker program must save its own settings if required. Unless otherwise noted, the contents of the A and Q registers will always be destroyed by the routine to be implemented.

Two options are available to the worker program for specifying parameters, namely the macro call and own code options. The macro call option allows the programmer to utilize the macro definition statement contained in the routine for presetting parameters. In this case parameters are specified in the form of a series of macro operands (See Section 4 for a discussion of the macro call line). The own code option allows the programmer to code his own parameter setting instructions.

– Character Position

An explanation of normal character positions is necessary for the proper application of the routines contained within the conversion family, since these routines accept and/or provide data in a standard character position format.

Five six bit characters may be contained wholly within one computer word. Normal character positions are illustrated below.



Legend:

Character Position	Bit Positions
0	29 - 24
1	23 - 18
2	17 - 12
3	11 - 6
4	5 - 0

■ FIELDATA TO BINARY (FDTOBIN$)

This routine converts up to eight characters of a two-word Fieldata storage to a one-word binary storage. The Fieldata storage may be signed although this is optional. If a Fieldata sign is present, the binary result will be given the same sign. If a sign character is not present, the value is assumed positive.

Provision is made in FDTOBIN$ for testing to pickup field size error conditions. When an error is detected, control is returned to the worker program at a preset error address. An error indication is put in the Q register for possible testing by the worker program. If no error is detected, the exit address is incremented by one and control is returned to the worker program upon completion of the conversion.

a. Parameters

Four parameters are required for the proper execution of a conversion. These parameters must be specified in various index registers as shown below.

Entrance Registers:

B5 First word address of the two-word Fieldata storage.

B6 Address of the one-word binary storage.

B7 Number of characters to be converted. This parameter may not be greater than eight.

[y] Error subroutine address. This parameter specifies the entrance of a closed subroutine provided by the worker program for handling an error condition. The error address is not specified by register, but will become the y-address of a return jump instruction.

Exit Registers:

B5 First word address of the two-word Fieldata storage.

B6 Address of the one-word binary storage.

B7 Zero.

b. Error Indication

Upon detecting an error condition, FDTOBIN$ returns control to the main program at the next instruction address. This address should provide a return jump instruction to a closed subroutine for handling the error conditions which may exist. An indicator is provided in bit positions 0-2 of the Q register for error analysis. Bit positions 3-29 are set to binary zeros. Error indications are as follows:

Indicator (binary value)	Meaning
1	The parameter specifying the number of characters to be converted is zero. The conversion process has not begun. The deposit field is cleared to binary zeros.
2	The parameter specifying the number of characters to be converted is greater than eight. The conversion process has not begun. The deposit field is cleared to binary zeros. Index register B7 contains the number of characters specified.

The error subroutine to be provided by the worker program is entered by a return jump instruction and should, therefore, be a closed subroutine containing ENTRY and EXIT operators. For example:

```
[error address] → ENTRY
                  x
                  x   } worker program coding
                  x
                → EXIT
```

c. Data Format

– Input

FDTOBIN$ accepts as input a two-word storage containing up to eight Fieldata characters plus sign. The characters may occupy character positions 2-4 of word zero and positions 0-4 of word one (see sample below). Characters must be right justified. The sign character must be a Fieldata plus (42) or minus (41) sign and will appear in character position zero of the first word of the storage. The sign character is optional and if it is not present the value in the storage is assumed positive. Non-significant leading characters must be either Fieldata zeros (60) or binary zeros (00). Characters in the storage, but not included in the operand as defined by the number-of-characters parameter, will be ignored; consequently their value is immaterial.

The following is an example of a Fieldata storage containing the value – 69172378.

word	0	1	2	3	4
0	–	u	6	9	1
1	7	2	3	7	8

Legend:

u Character position one of word zero is unused and will be ignored by the routine.

– Output

FDTOBIN$ provides as output a one-word binary storage in the standard UNIVAC 490 binary arithmetic format. A negative number will appear as the ones complement of a positive number of the same magnitude. For example, the number 1234_8 will appear in positive and negative format as shown below.

positive format

0	0	0	0	0	0	1	2	3	4

negative format

7	7	7	7	7	7	6	5	4	3

d. Routine Requirements

FDTOBIN$ requires approximately 65 locations in memory plus 5 memory locations each time the macro is used.

Execution Time

FDTOBIN$ requires approximately 120 μs per character plus 366 μs each time the routine is entered from the worker program. The macro requires 31 μs for each use.

■ BINARY TO FIELDATA (BINTOFD$)

BINTOFD$ accepts as input a one-word binary storage having either positive or negative value. It produces as output a signed, two-word Fieldata storage. Fieldata characters will be right-justified within the storage. Non-significant leading characters will be set to Fieldata zeros (60).

a. Parameters

Two parameters must be provided by the worker program for the proper execution of a conversion. These parameters are specified in the registers shown below.

Entrance Registers:

B4 Word address of the one-word binary storage.

B5 First word address of the two-word Fieldata storage.

B7 No parameter but used internally by the routine.

Exit Registers:

B4 Word address of the one-word binary storage

B5 First word address of the two-word Fieldata storage.

B7 Sign character indicator for deposit field

0 sign is plus

1 sign is minus.

b. Data Format

– Input

BINTOFD$ accepts as input a one-word (30 bit) binary value in UNIVAC 490 arithmetic format. Values may be either positive or negative numbers. The input format for positive and negative numbers is illustrated below.

Example 1: The value+3217615_8 would appear as follows:

0	0	0	3	2	1	7	6	1	5

Example 2: The value-3217615_8 would appear in the following manner:

7	7	7	4	5	6	0	1	6	2

– Output

BINTOFD$ provides as output a two-word storage containing eight, right-justified Fieldata characters, plus a Fieldata sign character in the six most significant bit positions of the first word (see example). Non-significant leading characters will be set to Fieldata zeros. All characters will occupy normal character positions. The following is an example of a Fieldata storage containing the converted value from Example 2 above.

word					
0	–	u	0	0	8
1	6	0	0	4	5

Legend:

u Character position one of word zero is unused and will be set to six binary zeros.

c. Routine Requirements

BINTOFD$ requires approximately 35 memory locations plus three locations for each use of the macro.

Timing requirements are 195 μs for entry to the routine (220 μs if macro option is used) plus approximately 141 μs for each character converted to Fieldata.

■ CODE CONVERSION (CODECON$)

– Structure

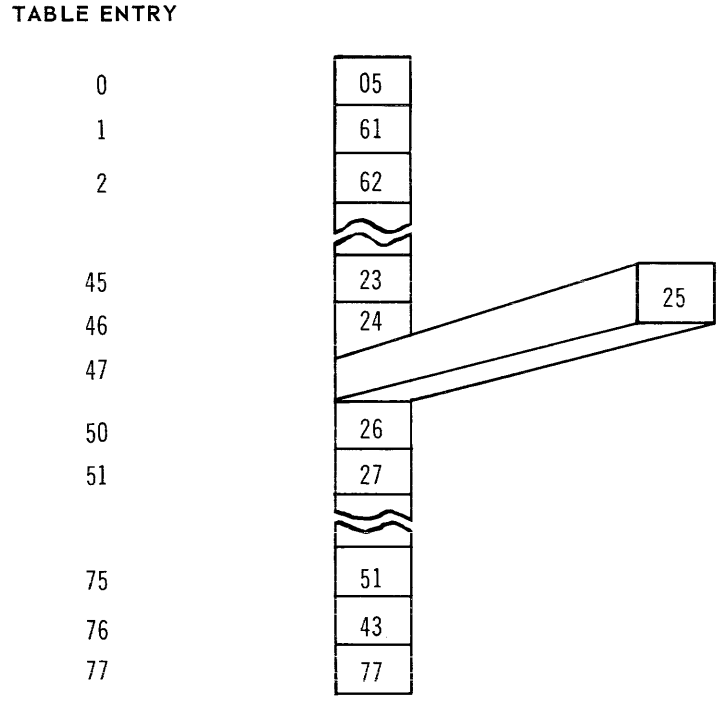
CODECON$ is divided into two sections, the control subroutine and the translation table. The control subroutine directs the translation process using parameters supplied by the worker program. The design of the control subroutine makes it possible to operate with any appropriately designed translation table. The translation table is a 64-word constant area which contains the translated characters at the appropriate entries. This table may be provided by the user. See USER PROVIDED TRANSLATION TABLES.

– Method of Operation

In any code system having six bits, there are 64 different combinations. Each combination has a unique binary value regardless of the coding system that character represents. This property may be used to determine any entry within a table where the translated equivalent of the original character is located. For example, suppose the conversion routine is given a six bit BCD (Binary Coded Decimal) character for translation to its equivalent in Fieldata code. If, for illustrative purposes, the character is the BCD letter P, then its bit configuration is

100111

and its octal value is 47. The Fieldata equivalent would thus be located at entry 47 in the translation table as illustrated.



The Fieldata letter P, whose octal value is 25, may then be retrieved directly from table entry 47 and be substituted for the BCD letter P. This technique is called "table look up" and it is utilized in this case because it holds translation time to a minimum.

The translation table, FBXTAB$, provides the basic mechanism for conversion between the several code systems. Tables H-1 and H-2 show the equivalent codes.

Translation Table Design

Translation Table Entries

A translation table consists of 64 (100 octal) computer words. Each translation table may contain up to five character tables. A character table consists of the translated characters for a given type or mode of translation, for example, BCD to Fieldata. Character tables are numbered 0 through 4. The relationship between translation table and character table is illustrated.

FIELDATA CHARACTER	OCTAL REPRESENTATION		BCD CHARACTER
	FD	BCD	
NP	00	14	NP
NP	01	20	NP
NP	02	20	NP
#	03	13	#
NP	04	57	Δ
Space	05	20	¢
A	06	61	A
B	07	62	B
C	10	63	C
D	11	64	D
E	12	65	E
F	13	66	F
G	14	67	G
H	15	70	H
I	16	71	I
J	17	41	J
K	20	42	K
L	21	43	L
M	22	44	M
N	23	45	N
O	24	46	O
P	25	47	P
Q	26	50	Q
R	27	51	R
S	30	22	S
T	31	23	T
U	32	24	U
V	33	25	V
W	34	26	W
X	35	27	X
Y	36	30	Y
Z	37	31	Z

FIELDATA CHARACTER	OCTAL REPRESENTATION		BCD CHARACTER
	FD	BCD	
)	40	55	)
–	41	40	–
+	42	60	+
NP	43	76	<
NP	44	35	=
NP	45	16	>
&	46	72	&
$	47	53	$
*	50	54	*
(	51	75	(
%	52	34	%
:	53	15	:
NP	54	37	"
NP	55	52	!
Comma	56	33	Comma
NP	57	32	÷
0	60	12	00
1	61	01	01
2	62	02	02
3	63	03	03
4	64	04	04
5	65	05	05
6	66	06	06
7	67	07	07
8	70	10	08
9	71	11	09
Apost.	72	36	Apost.
;	73	56	;
/	74	21	/
.	75	73	.
NP	76	74	□
Stop	77	77	≒

NP Non-printing character

÷ Record Mark

¢ Blank Position or Tape with Even Parity

≒ Group Mark

Table H-1. FD/BCD Code Equivalents

FIELDATA CHARACTER	OCTAL REPRESENTATION		XS-3 CHARACTER
	FD	XS-3	
NP	00	56	@
NP	01	17	[
NP	02	01	]
#	03	35	#
NP	04	57	Δ
Space	05	00	Space
A	06	24	A
B	07	25	B
C	10	26	C
D	11	27	D
E	12	30	E
F	13	31	F
G	14	32	G
H	15	32	H
I	16	34	I
J	17	44	J
K	20	45	K
L	21	46	L
M	22	47	M
N	23	50	N
O	24	51	O
P	25	52	P
Q	26	53	Q
R	27	54	R
S	30	65	S
T	31	66	T
U	32	67	U
V	33	70	V
W	34	71	W
X	35	72	X
Y	36	73	Y
Z	37	74	Z

FIELDATA CHARACTER	OCTAL REPRESENTATION		XS-3 CHARACTER
	FD	XS-3	
)	40	77	)
–	41	02	–
+	42	63	+
NP	43	36	<
NP	44	37	=
NP	45	76	>
&	46	20	&
$	47	42	$
*	50	41	*
(	51	75	(
%	52	61	%
:	53	21	:
NP	54	23	?
NP	55	43	!
Comma	56	62	Comma
NP	57	15	\
0	60	03	0
1	61	04	1
2	62	05	2
3	63	06	3
4	64	07	4
5	65	10	5
6	66	11	6
7	67	12	7
8	70	13	8
9	71	14	9
Apost.	72	40	Apost.
;	73	16	;
/	74	64	/
.	75	22	.
NP	76	75	□
Stop	77	60	≠

NP Non-printing character

Table H-2. FD/XS-3 Code Equivalents

An entry in the translation table may contain entries of up to five different character tables. For example, the following is an entry taken from the translation table now in the SPURT LIBRARY.

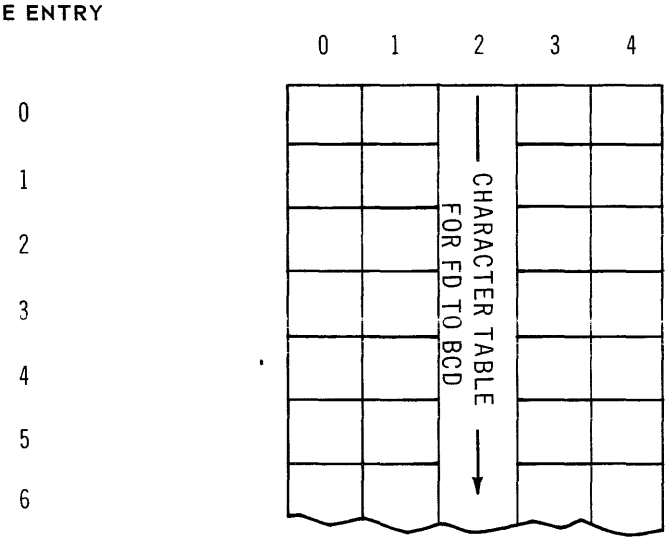
Table Entry	0	1	2	3	4
34	52	16	26	71	00

Legend:

Character Table	
0	Referenced during BCD to Fieldata Translation
1	Referenced during XS-3 to Fieldata Translation
2	Referenced during Fieldata to BCD Translation
3	Referenced during Fieldata to XS-3 Translation
4	Character Table 4 is not currently used.

Character Table Entries

A character table consists of 64 six bit characters. Each character within a table occupies a preassigned character position in a computer word. Only one character position in a translation table entry may be utilized by a given character table. For example, the characters in the table used for Fieldata (FD) to BCD translation are always located in character position 2 of each computer word in the translation table.



Character table entries for a given code type are assigned in the following manner. First, the octal value of each character in the code type is determined. The octal value then becomes the table entry where the translated equivalent is entered. There is no duplication of table entries since each character has a unique octal value. The translated character must be positioned in the computer word so that it is contained wholly within one character position. Character positions are defined as follows:

0	1	2	3	4
29 24	23 18	17 12	11 6	5 0

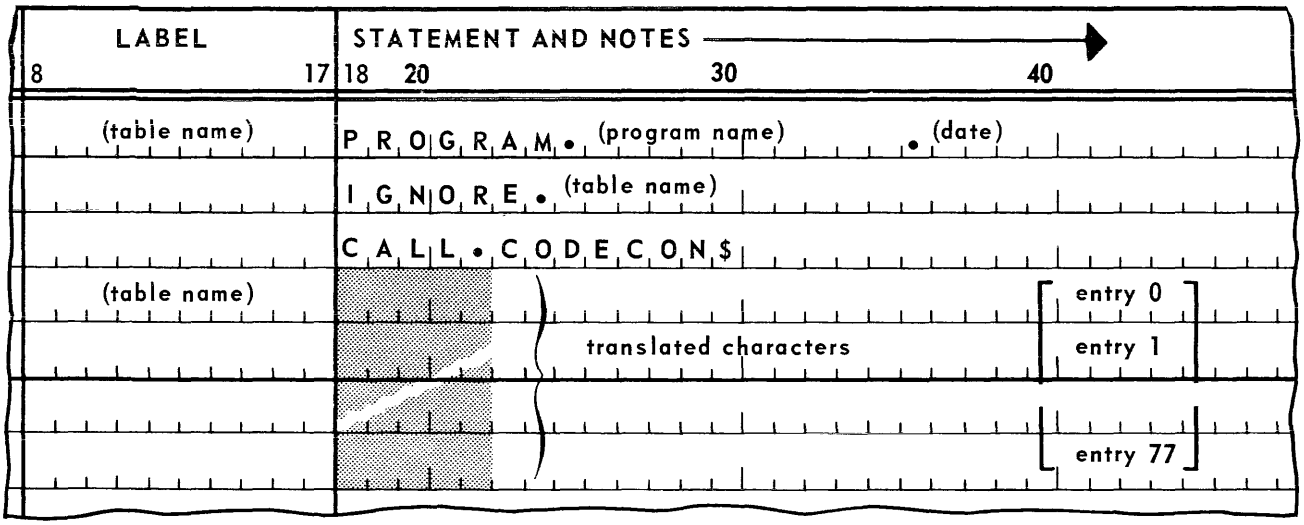
Legend:

Character Position	Bit Positions
0	29-24
1	23-18
2	17-12
3	11-6
4	5-0

– *User Provided Translation Tables*

Translation tables provided by the user for operation with CODECON$ must be designed in the format prescribed in the preceding section (Translation Table Design).

When a table has been prepared it must receive a unique mnemonic label, and it is recommended that a library call for CODECON$ be added (See programming example below). The table must be assembled, using the SPURT assembly system, and then placed in the Library as an independent program. The following coding shows an acceptable format for the program containing a translation table.



a. Parameters

Five parameters direct the execution of the translation routine. These parameters are supplied to the control routine in various registers. The worker program must assume the responsibility for saving the contents of these registers if it is necessary to do so.

Entrance Registers:

B4 First word address of the pickup field.

B5 First word address of the deposit field. Although it is not necessary, the FWA (first word address) of the deposit field may be the same as the FWA of the pickup field. In this event the original field will be destroyed.

B6 Number of words to be translated. This refers to the number of computer words in the pickup and deposit fields. The worker program must allocate areas of equal size for these two fields. Only whole words are translated.

B7 Translation mode. This parameter specifies the character table to be used during the translation process. Character table should not be confused with translation table. See TRANSLATION TABLE DESIGN. The "mode" parameter must be numeric in the range 0 to 4 and is defined as follows:

Mode	Action
0	Translate BCD to Fieldata
1	Translate XS-3 to Fieldata
2	Translate Fieldata to BCD
3	Translate Fieldata to XS-3
4	Not currently used. If specified all characters will be translated to binary zeros.

Translation mode is determined by the number of the character table to be referenced during a given translation. For example, the parameter for a Fieldata to XS-3 translation table would be 3, since character table 3 contains the translated characters for this mode of conversion.

A First word address of the translation table. This parameter provides the FWA of the table to the control routine.

Exit Registers:

B4 Destroyed

B5 Destroyed

B6 Zero

B7 Translation mode

A Destroyed

b. Data Format

CODECON$ is designed to translate six bit code characters which occupy normal character positions as defined under GENERAL INFORMATION. The translated characters in a deposit field will occupy the same character positions that the original characters do in the pickup field. For example:

Word	Pickup Field				
0	U	N	I	T	
1	P	R	I	C	E
2		$	2	.	5
3	5			T	0
4	T	A	L		$
5	7	9	.	0	5

Binary Coded Decimal

Word	Deposit Field				
0	U	N	I	T	
1	P	R	I	C	E
2		$	2	.	5
3	5			T	0
4	T	A	L		$
5	7	9	.	0	5

Fieldata

c. Routine Requirements

Control Routine	22 words
Translation Table	*64* words
Total	86 words

In addition the macro requires 6 words for each use.

Execution Time

Approximately 288 μs per word plus 85 μs each time the routine is entered from the worker program. The macro requires an additional 32 μs.

d. Minor Modifications

If it should become necessary to change the translation value of a given character, this may be accomplished in the following manner. For example, suppose it becomes desirable to translate the XS-3 asterisk sign, whose octal value is 41, to a Fieldata dollar sign, whose octal value is 47. The Fieldata asterisk sign has an octal value of 50. It is therefore, necessary to change entry 41 in the XS-3 to Fieldata character table from an octal 50 to an octal 47; i.e.,

Now Reads		change to	
Table Entry		Table Entry	
00041	50	00041	47

For a discussion of character table formats, See TRANSLATION TABLE DESIGN.

Minor modifications may be made in one of the following four ways:

1. If the modification is to be a permanent one, the necessary corrections should be made to the Library card deck. The program should then be reassembled and replaced in the Library.

2. If the modification is to be a permanent one, but applicable only to a given program, then the modification might be made by main program coding.

3. If the modification is to be a temporary one, the two following methods are possible at load time:

   - Use of the Change Core function of REX.
   - Use of the errata card load function of REX.

- DOUBLE PRECISION BINARY TO FIELDATA (DPBINTOFD$)

This routine accepts as input a signed, two-word binary working storage and provides as output a Fieldata field of up to 16 contiguous characters plus sign (See DATA FORMAT). The sign is optional and may be selected or inhibited by parameter.

DPBINTOFD$ tests for various errors during routine operation. Upon detecting an error condition, it returns control to the main program at a preset error address. An error indicator is made available in the Q register for possible testing by the error subroutine. If no error is found, the exit address is incremented by one and control is returned to the main program upon completion of the conversion.

a. Parameters

Six parameters must be provided by the worker program for the proper execution of a conversion. These parameters are specified in various registers as shown below.

Entrance Registers:

B4 First word address of the pickup field. This parameter specifies the address of the most significant word of the two-word binary storage.

B5 First word address of the deposit field. This parameter specifies the address of the word in which the most significant character of the field appears. If the sign option is selected it is the address of the word containing the sign.

B6 First character position of the deposit field. This parameter specifies the character position of the most significant character in the deposit field (See GENERAL INFORMATION for a discussion of character position).

B7 Number of characters. This parameter specifies the number of characters in the deposit field including sign, if the sign option is selected.

[y] Error subroutine address. This parameter specifies the entrance of a closed subroutine provided by a worker program for handling the various error conditions detected by DPBINTOFD$. The error address is not specified by register, but will become the y address of a return jump instruction.

A Sign option indicator. This parameter directs the conversion routine to inhibit or to allow sign insertion in the most significant character position of the deposit field. It may be one of the following:

0 Inhibit sign character insertion

1 Allow sign character insertion

Exit Registers:

B4 First word address of pickup field.

B5 First word address of deposit field.

B6 Character position of next character to be inserted.

B7 Zero.

[y] Error subroutine address.

A Destroyed.

b. Error Indications

Upon detecting an error condition, DPBINTOFD$ returns control to the main program at a preset error address. An error indicator is provided by DPBINTOFD$ in bit positions 0-2 of Q register. Bit positions 3-29 are set to binary zeros. Error indicators are defined as follows:

Indicator (binary value)	Meaning
1	The parameter specifying the number of characters to be converted is greater than 16, exclusive of sign. The conversion process has not begun and the deposit field is unchanged.
2	The parameter specifying the number of characters to be converted is zero after sign option adjustment. The conversion process has not begun and the deposit field is unchanged.
3	The value in the two word binary storage exceeds the specified size of the deposit field. The most significant characters in the result have been truncated.

The error subroutine to be provided by the main program is entered via a return jump instruction and should, therefore, be a closed subroutine containing ENTRY and EXIT operators. For example:

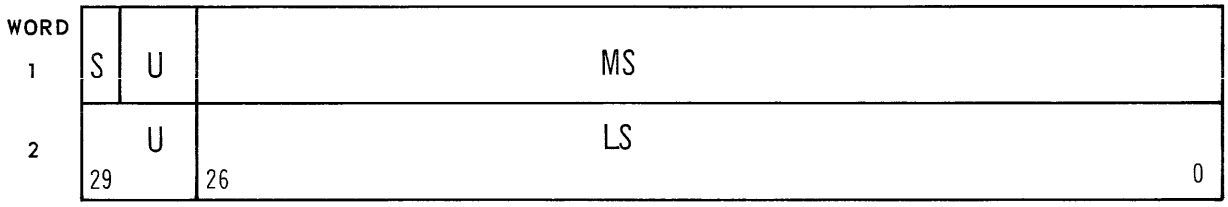
[error address] → ENTRY → error subroutine

x
x } error subroutine coding
x

→ EXIT

c. Data Format

The double-precision arithmetic number is a signed, absolute, binary value represented in two consecutive computer words. Each computer word can represent a decimal number not greater than 10^8 -1. The maximum total value of the two-word field is 10^{16} -1. The format for the binary number is shown below.



Legend:

MS Most significant part of the binary number. Occupies bit positions 0-26 of word 0.

LS Least significant part of the binary number. Occupies bit positions 0-26 of word 1.

S Sign indicator. Occupies bit position 29 of word 0. The sign indicator may be one of the following:

0 The binary number is positive.

1 The binary number is negative.

U Unused. Bit positions 27-28 of word 0 and bit positions 27-29 of word 1 will be ignored.

– Input

DPBINTOFD$ accepts as input a two-word binary number in the format described above.

– Output

DPBINTOFD$ provides as output a field of up to 16 contiguous Fieldata characters plus sign. The Fieldata characters will occupy normal character positions as defined under GENERAL INFORMATION. Two sample fields are shown below.

0	1	2	3	4
	+	1	9	5
1	7	2	1	6
9	1	2	8	7
3	4	6		

16-Character, Signed Deposit Field

0	1	2	3	4
			1	7
1	2	4	6	1
3	9	8	2	3

12-Character, Unsigned Deposit Field

When the sign option is selected, a Fieldata plus (42) or minus (41) sign will appear as the most significant character of the deposit field. The sign is determined by testing bit position 29 of word 0 in the two-word binary storage.

d. Routine Requirements

Approximately 90 memory locations plus 7 memory locations each time the macro is used.

Execution Time

Approximately 195 μs per character plus 300 μs each time the routine is entered from the main program. If the sign option is selected, an addition 110 μs are required. The macro requires 46 μs for each use.

■ FIELDATA TO DOUBLE-PRECISION BINARY (DPFDTOBIN$)

This routine accepts as input a Fieldata field of up to 16 contiguous characters and converts it to a signed, two-word, binary working storage suited for use with the double-precision arithmetic routines. The Fieldata field may be signed or unsigned (See DATA FORMAT). If it is unsigned it is assumed positive and the sign position of the two-word binary storage is set accordingly.

DPFDTOBIN$ tests for various errors during routine operation. Upon detecting an error condition, it returns control to the worker program at a preset error address. An error indicator is made available in the Q register for possible testing by the error subroutine. If no error is found the exit address is incremented by one and control is returned to the main program upon completion of the conversion.

a. Parameters

Five parameters must be provided by the worker program for the proper execution of a conversion. These parameters are specified in various registers as shown below.

Entrance Registers:

B4 First word address of the pickup field. This parameter specifies the address of the most significant word in the Fieldata field. If the most significant character of the field is a sign character, then it is the address of the word containing the sign.

B5 First word address of the pickup field. This parameter specifies the address of the most significant word of the two-word binary storage.

B6 First character position of the pickup field. This parameter specifies the character position of the most significant character in the pickup field.

B7 Number of characters. This parameter specifies the number of characters in the pickup field including sign, if the sign option is selected.

[y] Error subroutine address. This parameter specifies the entrance to a closed subroutine provided by the worker program for handling the various error conditions detected by DPFDTOBIN$. The error address is not specified by register, but will become the y address of a return jump instruction.

Exit Registers:

B4 First word address of pickup field.

B5 First word address of deposit field.

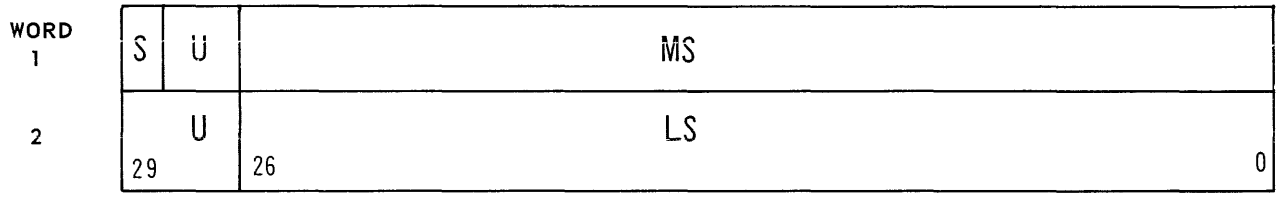
B6 Character position of next character to be extracted.

B7 Zero.

[y] Error subroutine address.

c. Data Format

The double-precision arithmetic number is a signed, absolute, binary value represented in two consecutive computer words. Each computer word can represent a decimal number not greater than 10^8 -1. The maximum total value of the two-word field is 10^{16} -1. The format for the binary number is shown below.



Legend:

MS Most significant part of the binary number. Occupies bit positions 0-26 of word 0.

LS Least significant part of the binary number. Occupies bit positions 0-26 of word 1.

S Sign indicator. Occupies bit position 29 of word 0. The sign indicator may be one of the following:

0 The binary number is positive.

1 The binary number is negative.

U Unused. Bit positions 27-28 of word 0 and bit positions 27-29 of word 1 will be ignored.

b. Error Indications

Upon detecting an error condition, DPFDTOBIN$ returns control to the main program at a preset error address. An error indicator is provided by DPFDTOBIN$ in bit positions 0-2 of the Q register. Bit positions 3-29 are set to binary zeros. Error indicators are defined as follows:

Indicator (binary value)	Meaning
1	The parameter specifying number of characters to be converted is greater than 16 exclusive of sign. The conversion process has not begun and the deposit field has been cleared to binary zeros.
2	The parameter specifying number of characters to convert is zero after sign adjustment. The conversion process has not begun and the deposit field has been cleared to binary zeros.

The error subroutine to be provided by the worker program is entered by a return jump instruction and should, therefore, be a closed subroutine containing ENTRY and EXIT operators. For example:

```
[error address] → ENTRY → error subroutine
                    x
                    x   } error subroutine coding
                    x
                → EXIT
```

– Input

DPFDTOBIN$ accepts as input a field of up to 16 contiguous Fieldata characters plus sign. If the field is signed, the sign must be a Fieldata plus (42) or minus (41) sign and must appear as the most significant character of the field. All characters in the field must occupy normal character positions as defined under GENERAL INFORMATION. Two sample fields are shown below.

0	1	2	3	4
			+	7
3	1	4	9	6
2	8	1	1	2

11-Character Signed Field

0	1	2	3	4
	9	1	2	3
8	7	3	2	1
2	3	8	1	2
9	4			

16-Character Unsigned Field

– Output

DPFDTOBIN$ provides as output a two-word binary number in double-precision format.

d. Routine Requirements

Approximately 75 memory locations plus 6 locations for each macro use.

Execution Time

Approximately 132 μs per character plus 328 μs each time the routine is entered from the worker program. An additional 64 μs are required if the Fieldata field is signed. The macro execution time is 37 μs.

2. Editing Routines

The UNIVAC 490 editing routines are part of the SPURT library and provide a simple method of inserting symbols, numbers and letters into the body of a data field. These routines are coded in the SPURT mnemonic language and may be called from the SPURT library at the time of program assembly. The routines are designed to work with many of the service routines incorporated in the SPURT library. Typical editing operations are zero suppression, check protection and dollar sign floatation.

– Internal Working Storage Layout

				±
n				
n+ 1				
n+ 2				
n+ 3				
n+ 4				
n+ 5				
n+ 6				
n+ 7				
n+ 8				
n+ 9				
n+10				
n+11				
n+12				
n+13				
n+14				
n+15				
n+16				
n+17				
n+18				
n+19				

UPPER SIX BITS

LOWER SIX BITS

The data that is processed by the editing routines is contained in a 20 word working storage as shown under Internal Working Storage Layout. Six bit Fieldata characters occupy the six least significant bit positions of the lower and upper portions of each word. This storage is utilized by the editing routines with the following assumptions:

(1) the lower six bits of the lower half of each word contain the data pertinent to the storage. The sign position is located in word n and the least significant character is located in word n + 19.

(2) the lower six bits of the upper half of each word will contain insertion symbols for those operations in which they are required.

(3) Spaces precede the first significant character in an alphanumeric field.

(4) Zeros precede the first significant character in a numeric field.

The address of the first word, n, is the key parameter necessary for the operation of editing routines. Each routine restores the parameter upon exit for continuous working efficiency. Other parameters required for entrance to the editing routines include such information as character count and sign type.

The size of the working storage can be increased or decreased as illustrated in each routine. If the size is altered, the same modifications should be made uniform in the family of editing routines, as well as the family of associated service routines. This will be illustrated when applicable.

- ZERO SUPPRESSION (ZSEDIT$)

ZSEDIT$ suppresses (clears to blanks) the leading words containing Fieldata zero codes in the lowest six bits of the working storage. The routine initially checks the entrance parameter containing the number of characters to zero suppress. Starting at word n + 1 the routine tests each character, the lowest six bits of each word, in the working storage for a zero code. If all characters are zero and the zero suppression parameter is zero, the entire working storage is cleared to blanks. Otherwise, the routine will zero suppress the number of characters, starting at word n + 1, as indicated by the zero suppression parameter. As soon as a non-zero code is found, zero suppression is terminated.

a. Parameters

Entrance:

B6 Number of characters (words) to zero suppress.

B7 Address of the first word, n, in the 20 word working storage.

Exit:

Same as entrance

b. Error Indications

No error exit is provided in this routine. The routine will test until a non-zero code is found. Once this has occured, action indicated by the B6 parameter setting will take place.

c. Examples

In order to use this method of zero suppression the numeric field must be in the 20 word working storage. For purposes of illustration it is assumed that a numeric field has been moved into the 20 word working storage with a sign character placed in word n and zero codes placed in the lowest six bits of all words preceding the most significant character. The following examples illustrate the before and after condition of the working storage when ZSEDIT$ is used.

LABEL B7=ZEROWORK	EXAMPLE 1 B6 = 0		Word	EXAMPLE 2 B6 = 8		Word	EXAMPLE 3 B6 = 11	
	Before	After		Before	After		Before	After
ZEROWORK	+	B	n	+	+	n	+	+
	0	L	n+ 1	0	B	n+ 1	0	B
	0	A	n+ 2	0	L	n+ 2	0	L
	0	N	n+ 3	0	A	n+ 3	0	A
	0	K	n+ 4	0	N	n+ 4	0	N
	0	S	n+ 5	0	K	n+ 5	0	K
	0		n+ 6	0	S	n+ 6	0	S
	0		n+ 7	0		n+ 7	0	
	0		n+ 8	0		n+ 8	0	
	0		n+ 9	0	0	n+ 9	0	
	0		n+10	0	0	n+10	0	
	0		n+11	7	7	n+11	7	7
	0		n+12	4	4	n+12	4	4
	0		n+13	6	6	n+13	6	6
	0	B	n+14	9	9	n+14	9	9
	0	L	n+15	8	8	n+15	8	8
	0	A	n+16	1	1	n+16	1	1
	0	N	n+17	5	5	n+17	5	5
	0	K	n+18	3	3	n+18	3	3
	0	S	n+19	2	2	n+19	2	2

Example 1:

The entire field will be blanked to spaces when B6 = 0 and the working storage field contains all zero codes.

Example 2:

The number of leading zeros in the working storage is greater than the number of zeros to suppress in B6. In this case the total number suppressed will equal the parameter in B6.

Example 3:

The number of leading zeros in the working storage is less than the number of zeros to suppress in B6. In this case only the leading zeros found in the field are suppressed.

d. Routine Requirements

26 words of storage. The macro section of coding, when used, generates 3 computer words.

This routine will be executed in approximately 322 μs if ten zero codes are suppressed. The entrance timing to set parameters consumes 25 μs.

e. Modification

The amount of word storage may be increased or decreased by inserting in the coding a different constant at the following mnemonic memory locations.

ZCHECK$ + 2 contains RPT. 19D . ADV

ZCHECK$ + 6 contains ENT. A. 19D

ZSEND$ + 2 contains ENT. B7. 19D

For example, if the working storage was being expanded to 30 words, the three locations would appear as follows:

RPT.29D.ADV

ENT.A.29D

ENT.B7.29D

- FLOAT DOLLAR SIGN (FLEDIT$)

FLDEDIT$ places a Fieldata dollar sign ($) code into the lower six bits of the least significant blank word of a working storage. The routine performs two checks before placing a dollar sign character into the storage. Starting with word n + 1, the routine examines each word for a blank and/or checks word 1 for a non-zero character. If all words are blank, or if word n + 1 contains a non-zero character, the routine exits. If the most significant character is located other than in word n + 1, a Fieldata dollar sign ($) code is inserted into the lower six bits of the preceding blank word.

a. Parameters

Entrance:

B7 Address of the first word, n, in the 20 word working storage.

Exit:

Same as entrance.

b. Error Indications

No error exit is provided in this routine. The routine will test until the least significant blank word is located. However, if all words tested are blank, the dollar sign is not floated, but and error indication is provided.

c. Examples

In order to take full advantage of this routine the numeric field in the 20-word storage should first be zero-suppressed (ZSEDIT$), if leading zeros are present. Leading zeros are considered significant characters. After leading zeros have been blanked, routine FLDEDIT$ floats the dollar sign into the least significant blank word. The following examples illustrate the before and after structure of the working storage when FLDEDIT$ is used:

LABEL B7=ZEROWORK	EXAMPLE 1		Word	EXAMPLE 2		Word	EXAMPLE 3	
	Before	After		Before	After		Before	After
ZEROWORK	+	+	n	+	+	n	+	+
	B	B	n+ 1	0	0	n+ 1	0	0
	L	L	n+ 2	0	0	n+ 2	0	0
	A	A	n+ 3	0	0	n+ 3	0	0
	N	N	n+ 4	0	0	n+ 4	0	0
	K	K	n+55	0	0	n+ 5	0	0
	S	S	n+ 6	0	0	n+ 6	0	0
	↓	↓	n+ 7	0	0	n+ 7	0	0
			n+ 8	0	0	n+ 8	0	0
			n+ 9	0	0	n+ 9	0	0
	↓	$	n+10	0	0	n+10	0	0
	7	7	n+11	0	0	n+11	7	7
	4	4	n+12	0	0	n+12	4	4
	6	6	n+13	0	0	n+13	6	6
	9	9	n+14	0	0	n+14	9	9
	8	8	n+15	0	0	n+15	8	8
	1	1	n+16	0	0	n+16	1	1
	5	5	n+17	0	0	n+17	5	5
	3	3	n+18	3	3	n+18	3	3
	2	2	n+19	0	0	n+19	2	2

Example 1:

The dollar code is floated into the lowest six bits of the least significant blank word, in this case word 10.

Example:2:

The entire field is packed with zero codes. Thus the dollar sign is not floated.

Example 3:

The leading zeros, preceding the first significant non-zero code in word n + 11, have not been blanked. Thus the dollar sign is not floated.

d. Routine Requirements

16 words of storage. When the macro call is employed, or if own entrance coding is applied, two additional computer words are used.

This routine will be executed in approximately 182 μs, if the dollar sign is floated in word n + 10 of the 20 word storage. An additional 19 μs is consumed by the macro call.

e. Modification

The size of the 20 word working storage may be increased or decreased by inserting in the coding a different constant at the following mnemonic memory locations:

FLDEDIT$ + 4 contains RPT. 19D . ADV

FLDEDIT$ + 7 contains ENT. A . 18D

For example, if the working storage was being expanded to 30 words, the two memory locations above would read as follows:

RPT.29D.ADV

ENT.A.28D

When the size of the storage is changed and other editing routines in the editing family are being used, these routines must be changed in a similar manner.

If a character other than a dollar sign, for example a Fieldata number sign (#), is required to precede the most significant number in the 20 word storage, the instruction located at FLDEXIT$-2 may be modified as follows:

ENT.Q.03

03 is the Fieldata code for the number sign. Some other Fieldata code may be inserted if desired.

- CHECK PROTECTION (CHKEDIT$)

CHKEDIT$ places a Fieldata asterisk code (*) into the lowest six bits of each blank word preceding the first word containing a significant character in a 20 word working storage. Starting at word n + 1, the routine checks the lowest six bits of each word for a blank. As each blank is found, a Fieldata asterisk code is inserted into the lower six bits of the word. This procedure is continued until a non-blank code is located, at which time the routine examines the lower six bits of the next word. If the proceeding word is blank an asterisk is inserted and the routine continues. If, however, the proceeding word is another non-blank code, the routine terminates. Once two non-blank words are found in succession, the routine exits.

a. Parameters

Entrance:

B7 Address of the first word, n, in the 20 word working storage.

Exit:

Same as entrance.

b. Error Indications

No error exit is provided for this routine. The routine inserts asterisks over leading blank words. If no blank words precede the most significant character, no asterisk is inserted. If the entire working storage is blank, an asterisk code will be placed into the lowest six bits of all words in the storage.

c. Examples

In order to use this method of check protection, the numeric field must be placed in a 20 word working storage. The Fieldata asterisk code will be inserted into the lowest six bits of each blank word preceding the word containing the most significant character. Leading zeros are considered as significant characters. These must be replaced with blanks if asterisks are to be inserted. The following examples illustrate the before and after status of a 20 word storage when routine FLDEDIT$ is executed.

LABEL B7=ZEROWORK	EXAMPLE 1		Word	EXAMPLE 2		Word	EXAMPLE 3	
	Before	After		Before	After		Before	After
ZEROWORK	+	+	n	+	+	n	+	+
	B	*	n+ 1	B	*	n+ 1	$	$
	L	*	n+ 2	L	*	n+ 2	B	*
	A	*	n+ 3	A	*	n+ 3	L	*
	N	*	n+ 4	N	*	n+ 4	A	*
	K	*	n+ 5	K	*	n+ 5	N	*
	S	*	n+ 6	S	*	n+ 6	K	*
	↓	*	n+ 7		*	n+ 7	S	*
		*	n+ 8	$	$	n+ 8	↓	*
		*	n+ 9		*	n+ 9		*
		*	n+10		*	n+10		*
	7	7	n+11	7	7	n+11	7	7
	4	4	n+12	4	4	n+12	4	4
	6	6	n+13	6	6	n+13	6	6
	9	9	n+14	9	9	n+14	9	9
	8	8	n+15	8	8	n+15	8	8
	1	1	n+16	1	1	n+16	1	1
	5	5	n+17	5	5	n+17	5	5
	3	3	n+18	3	3	n+18	3	3
	2	2	n+19	2	2	n+19	2	2

Example 1:

The asterisks will be inserted into the lower six bits of each blank word preceding the most significant non-blank word in the working storage.

Example 2:

The asterisks will be inserted into the lower six bits of each word in the working storage preceding the dollar sign, and then into each blank word before the first significant character.

Example 3:

The asterisk code will be inserted into the lower six bits of each blank word even though word 1 holds a significant character. Since word 2 does not contain a significant character, asterisks are inserted until the next significant non-blank word is found, in this case word n + 11.

d. Routine Requirements

17 words of storage. Two additional words are necessary for parameter settings.

If ten asterisk codes are inserted by the routine, the execution time will be approximately 633 μs. This includes the parameter settings preceding the routine.

e. Modification

The size of the working storage may be increased or decreased by inserting into the coding a different constant at the following mnemonic memory locations:

CHKEXIT$-2 contains BSK.B7. 19D

For example, if the working storage was being expanded to a 30 word storage, the instruction above would appear as follows:

BSK.B7.29D

The decimal entry is one less than the total word storage. If the size of the field is changed, and other routines utilize the 20 word storage, these routines must be changed in a similar manner. If a character other than an asterisk is desired, the designated Fieldata code can be inserted into the location CHKEDIT$+6 as follows:

ENT . Q. xx

xx = desired code

■ FLOAT PLUS OR MINUS SIGN (FLPMEDIT$)

FLPMEDIT$ checks the sign character in the first word of a 20 word working storage and places a Fieldata plus or minus code into the lower six bits of the least significant blank word. The routine starts by examining the lower six bits of word n, the sign position, to determine if a sign code is present. If the word is blank, a Fieldata plus code is placed into the lower six bits of word n. If the word is not blank, a test is made to find out if a plus or minus Fieldata code is present. After the sign determination, the routine searches the lower six bits of each word until the first non-blank character is located. The sign code is then inserted into the lower six bits of the preceding blank word. After the insertion, the routine exits. The sign code will not be floated when the entire storage is blank or when the sign code of the working storage is plus and a minus sign is being floated.

a. Parameters

Entrance:

B6 Code to indicate floating plus or minus code. Where 0 equals plus and 1 equals minus code.

B7 Address of the first word, word n, in the 20 word working storage.

Exit:

Same as entrance.

b. Error Indications

No error exit is provided in this routine. However, under certain circumstances no sign character is floated and the routine exits. Such cases are the following:

(1) The entire working storage is blank;

(2) The entire working storage is filled with non-blank characters

(3) A minus sign is being floated in a working storage signed by a plus code.

c. Examples

In order to utilize this type of sign insertion, the numeric field must be in the 20 word working storage. The field must also be zero suppressed, as described in routine ZSEDIT$, in order to replace the lower six bits of the least significant blank word with the sign code being floated. The following examples illustrate the before and after condition of the working storage when FLPMEDIT$ is used.

LABEL B7=ZEROWORK	EXAMPLE 1 B6 = 0		Word	EXAMPLE 2 B6 = 1		Word	EXAMPLE 3 B6 = 0	
	Before	After		Before	After		Before	After
ZEROWORK	+	+	n	+	+	n	+	+
	B	B	n+ 1	B	B	n+ 1	0	0
	L	L	n+ 2	L	L	n+ 2	0	0
	A	A	n+ 3	A	A	n+ 3	0	0
	N	N	n+ 4	N	N	n+ 4	0	0
	K	K	n+ 5	K	K	n+ 5	0	0
	S	S	n+ 6	S	S	n+ 6	0	0
	↓	↓	n+ 7	↓	↓	n+ 7	0	0
	↓	↓	n+ 8	↓	↓	n+ 8	0	0
	↓	↓	n+ 9	↓	↓	n+ 9	0	0
	↓	+	n+10	↓	↓	n+10	0	0
	7	7	n+11	7	7	n+11	7	7
	4	4	n+12	4	4	n+12	4	4
	6	3	n+13	6	6	n+13	6	6
	9	9	n+14	9	9	n+14	9	9
	8	8	n+15	8	8	n+15	8	8
	1	1	n+16	1	1	n+16	1	1
	5	5	n+17	5	5	n+17	5	5
	3	3	n+18	3	3	n+18	3	3
	2	2	n+19	2	2	n+19	2	2

Example 1:

The plus sign code is floated into the lower six bits of the least significant blank word preceding the most significant non-blank word.

Example 2:

The working storage is identical to that as shown in example 1. However, the parameter setting containing the type of code to float indicates a minus sign. No sign is floated since word n, which contains the sign code for the working storage, holds a plus sign.

Example 3:

The leading zeros preceding the most significant non-blank word prevent the sign from being floated. In this case the sign position in word n remains unaltered.

d. Routine Requirements

22 words of storage. The entrance to the routine uses three computer words.

This routine will be executed in approximately 230 μs if a plus sign is floated into word n + 10 as illustrated in example 1. The entrance time using the macro call or own code techniques takes approximately 25 μs.

e. Modification

The amount of word storage may be increased or decreased by replacing in the coding a different constant at the following mnemonic locations.

FLPMEDIT$ + 11D contains RPT. 19D . ADV

FLPMEXIT$ - 5 contains ADD . A . 19D

For example, if the working storage was being expanded to 30 words, the two locations above would appear as follows:

RPT . 29D . ADV

ADD . A . 29D

The decimal entry is always one less then the total number of words in the storage. If the size of the work area is changed, other routines using this same working storage configuration must also be changed in a similar manner.

- MERGE CHARACTERS (MERGEDIT$)

MERGEDIT$ utilizes the upper and lower portions of the 20 word working storage. A skeletal format composed of commas, a dollar sign, DB, CR, or any Fieldata characters is placed in the lowest six bits of the upper portion of the storage location. The Fieldata characters located in the lowest six bits of the lower portion of the storage location are merged and distributed into the blank positions contained in the upper word portion. The merged characters are then moved to the lower portion of the word and the upper portion is cleared to blanks.

The routine initially checks the lower portion of the last word in a 20 word storage, word n + 19, for a blank character. The entire storage is cleared to blanks, if this character is blank. If the character in the lower half of word n + 19 is not blank, the character in the upper half word of word n + 19 is tested. If blank, the non-blank character from the lower half word in word n + 19 is replaced over the blank six bits of the upper half word in word n + 19. If, however, the upper half word contains a significant character, the preceding half word, n + 18, in the upper half of storage is checked. When the next blank half word is found in the upper section, the character from the lower half word is inserted. This process of character insertion is continued sequentially until all the significant characters from the lower half have been merged into the blank half words in the upper half of the storage. When this insertion sequence is completed the remainder of the upper half words are cleared of any zero or comma codes that may have been stored previously. Other characters such as dollar signs or dashes are not cleared to blank. After this check, the entire upper half of the storage is moved into the lower half of the storage. The upper half of the storage is then cleared to blanks and the routine terminates.

a. Parameters

Entrance:

B7 Address of the first word, n, in the 20 word working storage.

Exit:

Same as entrance.

b. Error Indications

No error exit is provided in this routine. Since character insertion can be determined by the worker program, the only check that can be made is on the lowest six bits of the last word, n + 19, in the working storage. If this character is blank, the routine clears the entire 20 word working storage and exits.

The total number of editing characters in the upper half words plus the total number of stored characters in the lower half word should not normally exceed 20 characters. When all 20 words in the upper half of the storage have been filled, any remaining characters in the lower half of the storage that may have been inserted will be destroyed when the routine replaces the original lower half words with the newly merged upper half words.

c. Examples

Any type of character can be positioned into the lower six bits of any upper half word or words in the 20 word storage. Depending on the nature of the data in the lower six bits of all lower half words in the storage, the format design required by the main program can be inserted into proper upper half word locations in the Fieldata storage. The following examples illustrate some of the various editing symbols that can be merged when routine MERGEDIT$ is used.

LABEL B7=ZEROWORK	EXAMPLE 1			EXAMPLE 2			EXAMPLE 3	
	Before	After		Before	After		Before	After
ZEROWORK	+	+	n	+	+	n	+	+
		B	n+ 1			n+ 1		
	B	L	n+ 2			n+ 2		B
	, L	A	n+ 3	$ B	$	n+ 3	B	L
	A	N	n+ 4	L		n+ 4	L	A
	N	K	n+ 5	A		n+ 5	A	N
	K	B 7	n+ 6	N	B	n+ 6	N	B K
	,	,	n+ 7	, K		n+ 7	K	
		L 4	n+ 8		L	n+ 8	#	L #
		6	n+ 9		6	n+ 9		7
		A 9	n+10		A 9	n+10		A 4
	, 7	,	n+11	,	,	n+11	7	6
	4	N 8	n+12		N 8	n+12	– 4	N –
	6	1	n+13	6	1	n+13	6	9
	9	K 5	n+14	9	K 5	n+14	9	K 8
	8	.	n+15	. 8	.	n+15	8	1
	1	3	n+16	1	3	n+16	– 1	–
	5	2	n+17	5	2	n+17	5	5
	C 3	C	n+18	D 3	D	n+18	3	3
	R 2	R	n+19	B 2	B	n+19	2	2

Example 1:

This illustrates a method whereby certain symbols such as commas, the abbreviated symbols for credit and a decimal point are merged with the continuous numeric field to make the format suitable for a specific type of program usage; for example, printing. The extra comma in the upper half of word 3 is not merged with the numeric field, as the size of the number is not large enough to merit the insertion. This extra comma has been cleared.

Example 2:

This illustration is similar to example 1 except that the dollar sign is merged with the numeric field. The comma has been cleared but the dollar sign is considered an essential editing character. The only characters that are erased from the upper half storage are commas and zeros.

Example 3:

This illustration demonstrates another type of character insertion dependent upon the type of data being edited. Any type of symbol can be positioned in the upper half storage to prepare the lower half storage for a particular type of format.

d. Routine Requirements

44 words of storage. The macro or own code entrance requires 2 instructions.

This routine will be executed in approximately 1843 μs when nine significant characters from the lower half-words of the storage are merged with five editing characters from the upper half-words of the storage as illustrated in example 1. The entrance parameter settings require approximately 19 μs.

e. Modification

The amount of word storage may be increased or decreased depending upon program format requirements. The coding at the mnemonic locations listed below must be modified.

MERGEDIT$ + 9D contains ENT . A . L (B7 + 18D) . AZERO

MERGEDIT$ + 11D contains RPT . 20D . ADV

MERGEDIT$ + 16D contains ENT . B7 . 18D

MERGEDIT$ + 17D contains ENT . B6 . 19D

MERGEDIT$ + 30D contains ENT . A . B7 + 19D

The values contained within the brackets must be modified to reflect the change. For example, if the working storage was being expanded to 30 words, memory at the above locations would appear as follows:

ENT . A . L (B7 + 28D) . AZERO

RPT . 30D . ADV

ENT . B7 . 28D

ENT . B6 . 29D

ENT . A . B7 + 29D

The decimal entries must be similarly changed in other routines utilizing the working storage.

The two instructions that cause zeros or commas to be erased from the remaining area, after all significant characters from the lower half words of the storage have been inserted into the blank upper half words, are shown below.

(1) MERGEDIT$ + 39D contains ENT . Y-Q . 60 . AZERO

(2) MERGEDIT$ + 40D contains ENT. Y-Q . 56 . ANOT

Instruction (1) checks for a Fieldata zero code, while instruction (2) tests for a Fieldata comma code. In normal operation, if either code is present, the code is blanked. If neither code is present, the word remains unchanged and the next upper half word is checked. However, if codes other than a zero or comma are to be erased, or if no code tests are to be made, the above instructions would have to be modified by coding in the main program.

3. Floating Point Routines

The UNIVAC 490 Floating Point Library contains routines that perform some special operation, such as multiplication. The performance of this routine may require the use of a subordinate routine such as a conversion routine. The subordinate routine is placed in the library so that it may be utilized as an individual routine.

These routines may be divided into classes as listed below. The library is designed to permit the addition of other routines with minimum difficulty.

Arithmetic Routines

Add

Subtract

Multiply

Divide

Conversion Routines

Fixed-Point to Floating Point

Floating-Point to Fixed Point

Special Function Evaluations

Sine

Cosine

Arctangent

Square Root

Exponent (Base e)

Logarithm (Base e)

Input-Output Routines

Start (Load) and Convert

Convert and Punch

Convert and Type

Set Output Length

Auxiliary Routines

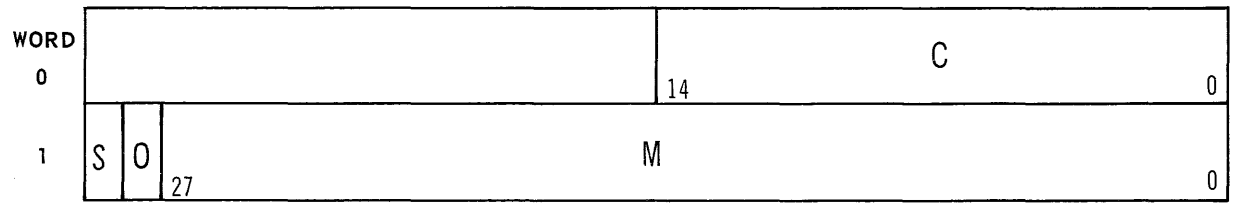
Scale (Normalize)

Convert to Decimal

– Floating Point Format

The UNIVAC 490 floating point routines require as standard format a pair of ordered numbers (C, M) in two consecutive 490 computer words.

The floating-point binary number is based on a two word format consisting of a 15 bit exponent part with a shift bias of 40000 octal, one sign bit, one overflow bit and a 28 bit fixed-point part illustrated as follows:



C exponent part – 15 bits

S sign bit – 1 bit

O overflow bit – 1 bit

M fixed point part – 28 bits

In any floating point operation, the binary floating-point operand is assumed to be in a "normalized" condition. This is to infer that the fixed-point part has, at some time, been shifted to the left until bit position 27 is set to "1", with the exponent part modified accordingly whenever this adjustment occurs. Hence, the radix point has been adjusted with the final value of the fixed-point part within the range $1/2 \leq M < 1$ with the radix point between bit positions 27 and 28.

– Number-Representation

The floating-point number representation may be expressed by the equation

$$N = M \cdot b^{E}$$

where:

N number represented.

M fixed point part (mantissa).

E exponent part (characteristic).

b base of the number system (floating-point radix) which must be a positive integer.

A floating point number is said to be in standard form (normalized) if M lies within some prescribed range, so chosen that any given number N can be represented by only a pair of numbers, M and C.

UNIVAC 490 hardware considerations dictate that the base number b shall be 2, subject to a common standard range for the fixed point part $1/2 \leq M < 1$. Thus, the modified equation for one 490 floating-point number is

$$N = M \cdot 2^{(C-40000_8)}$$

where:

M a normalized 28-bit binary value with the radix point left of the 28th bit.

C a 15 bit binary value biased by 40000_8. Therefore, the true exponent is $(C-40000_8)$ and subject to the following restrictions:

(1) If N = O then C = O, M = O.

(2) $1/2 \leq |M| < 1$.

(3) If E $>77777_8$, an overflow condition is indicated.

(4) If $M \leq O$, N is set to zero.

(5) M is always rounded to 28 bits after any floating-point operation.

Although the floating point package contains routines that perform computations in the above form, conversion and input-output routines are provided for dealing with numbers in decimal form or numbers with fixed binary points.

Decimal numbers can be loaded or read out in the form:

$$D = \pm X_0 . X_1 \ldots\ldots X_n \, EXP_{10} \, (\pm X_8 X_9)$$

where:

$10^{-40} \leq D \leq 10^{40}$

$1 \leq X_0 < 10$

$2 \leq n \leq 7$

The format required for loading numbers into the computer via paper tape is:

$$\leftarrow \uparrow N_1 N_2 N_3 N_4 N_5 \left\{\rightarrow\right\} \pm X_0 . X_1 \ldots . X_n \, \text{b} \pm X_8 \, X_9 \downarrow . \; . \leftarrow$$

and the format for punching or typing out results is:

$$\pm X_0 . X_1 \ldots X_n \, \text{b} \pm X_8 \, X_9 \left\{\begin{matrix}\leftarrow \\ \rightarrow\end{matrix}\right\}$$

where:

N_i Refers to the octal digits of the address of the number.

X_i Refers to the decimal digits of the number and the exponent.

n Specifies the number of decimal places.

$\leftarrow$ Flexcode for a carriage return.

$\rightarrow$ Flexcode for a tab.

b Flexcode for a space.

$\uparrow$ Flexcode for upshift.

$\downarrow$ Flexcode for downshift.

The use of the carriage return, tab or space codes are optional as indicated by use of $\{\ \}$ within the aforementioned input-output formats. Where applicable the + option of the $\pm$ notation in the enter format is customarily omitted.

The index "n" may vary as indicated according to requirements of the number desired. In the load (FPSTART$) operation, anything over seven decimal places is ignored.

When a string of numbers is loaded via paper tape only the address of the first storage location is required; however, the numbers are separated by carriage returns ( ) with the last carriage return in the string followed by the *end of information* code of two periods (..).

There are two floating point conversion routines for dealing with numbers with fixed binary points in the form

$$N = X_{29}X_{28}\cdots X_{s}.X_{s-1}\cdots X_{0}$$

where s specifies the location of a fixed point. One routine performs fixed-point to floating-point conversion; the other, floating-point to fixed-point. Both conversions are normally made between storage locations.

– Routine Library Organization

Since the Floating Point Routines were organized to provide maximum efficiency and utilization of memory area, it was necessary to group together related routines as "composite" groups or classes wherever possible. The "composite" routines were thus included under one program name indicated as follows:

Floating Point Functions	Routine Name	Composite Program or MACRO Name	Class
ADD	FPADD$	FPAS$	ARITHMETIC
SUBTRACT	FPSUB$		ARITHMETIC
MULTIPLY	FPMUL$	FPMD$	ARITHMETIC
DIVIDE	FPDIV$		ARITHMETIC
FIXED TO FLOAT	FXTOFL$	FPFXFL$	CONVERSION
FLOAT TO FIXED	FLTOFX$		CONVERSION
SINE	FPSIN$	FPSNCS$	SPECIAL FUNCTION
COSINE	FPCOS$		SPECIAL FUNCTION

Therefore, in order to perform any one of the above functions, the entry must be made through the composite name.

All other floating point routines are considered simple (single) and may be entered via the assigned mnemonic name as indicated below:

Floating Point Function	Program or MACRO Name	Class
Arctangent	FPATAN$	SPECIAL FUNCTION
Square Root	FPSQR$	SPECIAL FUNCTION
Exponent	FPEXP$	SPECIAL FUNCTION
Logarithm	FPLOGE$	SPECIAL FUNCTION
Start & Convert	FPSTART$	INPUT-OUTPUT
Convert & Punch	FPPUNCH$	INPUT-OUTPUT
Convert & Type	FPTYPE$	INPUT-OUTPUT
Set Output Length	FPSET$	INPUT-OUTPUT
Scale (Normalize)	FPSCL$	AUXILIARY
Convert to Decimal	FPCONV$	AUXILIARY

- FLOATING POINT ADDITION OR SUBTRACTION (FPAS$)

FPAS$ is designed to perform either floating point addition (FPADD$) or floating point subtraction (FPSUB$). The desired operation is specified by the coding which calls this routine. This routine is thereby directed to select either the FPADD$ or FPSUB$ subroutine.

a. Parameters

B4 Address of the first word of a two-word floating point operand (augend or minuend).

B5 Address of the first word of a two-word floating point operand (addend or subtrahend).

B6 Address of the first word of a two-word floating point result (sum or difference).

b. Error Indications

Overflow (Exponent > 77777_8) indicated by setting Q positive.

c. Routine Requirements

Both operations require 76 words of storage. Time requirements are:

Addition 111μs minimum to 340.8μs maximum.

Subtraction 195μs minimum to 424.8μs maximum.

- FLOATING POINT MULTIPLICATION AND DIVISION (FPMD$)

FPMD$ is designed to perform either floating point multiplication (FPMUL$) or floating point division (FPDIV$). The desired operation is specified by the coding which calls this routine. This routine is thereby directed to select either the FPMUL$ or FPDIV$ subroutine.

a. Parameters

B4 Address of the first word of a two-word floating point operand, the multiplicand or dividend.

B5 Address of the first word of a two-word floating point operand, the multiplier or divisor.

B6 Address of the first word of a two-word floating point result, the product or quotient.

b. Error Indications

Overflow (Exponent > 77777_8) indicated by setting Q positive.

c. Routine Requirements

Both operations require 65 words of storage.

Time Requirements are:

Addition 217.6μs minimum to 450μs maximum.

Divide 328.8μs minimum to 561.8μs maximum.

- FLOATING POINT CONVERSION (FPFXFL$)

FPFXFL$ is designed to perform either fixed point to floating point (FXTOFL$) or floating point to fixed point (FLTOFX$) conversions.

The desired conversion is specified by the coding which calls this routine.

a. Parameters

B4 Position of the fixed point.

B5 Address of the first word of the operand.

B6 Address of the first word of the result.

b. Error Indications

Fixed to Floating

Both result words are cleared if exponent > 77777_8.

Also if overflow occurs in scaling routine (FPSCL$)

Q is set positive.

Floating to Fixed

Q is set positive if exponent > 77777_8.

c. Routine Requirements

Both operations require 77 words of storage.

Time requirements are:

Fixed to floating 158μs minimum to 380.4μs maximum.

Floating to fixed approximately 110μs.

- FLOATING POINT SINE OR COSINE EVALUATION (FPSNCS$)

FPSNCS$ is designed to calculate the value of the sine (FPSIN$) or (FPCOS$) where the angle is represented in proper floating point notation. The desired operation is specified by the coding which calls this routine. The accuracy attained is six digits.

a. Parameters

B4 Address of the first word of a two word floating point operand.

B6 Address of the first word of a two word floating point result, the sine or cosine.

b. Error Indication

Q set positive if the exponent of the floating point operand > 40034_8.

c. Routine Requirements

Both evaluations require 103 words of storage.

Time requirements are:

Sine 85.2μs minimum to 932μs maximum.

Cosine 121.2μs minimum to 949μs maximum.

- FLOATING POINT ARCTANGENT EVALUATION (FPATAN$)

FPATAN$ is designed to calculate the arctangent (in radians) of a number in floating point notation. The method of computing is based on Hasting's Approximation. The accuracy attained is six digits.

a. Parameters

B4 Address of the first word of a two word floating point operand.

B6 Address of the first word of the two word result, the arctangent (in radians).

b. Error Indication

Value of the argument > 1.

c. Routine Requirements

132 words of storage.

Time requirement is 9.2μs minimum to 794μs maximum.

- FLOATING POINT SQUARE ROOT (FPSQR$)

FPSQR$ is designed to calculate the positive square root of a positive number in floating point notation. The routine uses the Newton Raphson iteration technique of successive approximations.

a. Parameters

B4 Address of the first word of a two word floating point radicand.

B6 Address of the first word of a two word floating point square root.

b. Error Indication

A register is set negative, if the fixed point part is negative.

c. Routine Requirements

43 words of storage.

Time requirements are:

30μs when error exists.

350.4μs when one iteration yields the result.

212.4μs for each additional iteration.

- FLOATING POINT EXPONENT (FPEXP$)

FPEXP$ is designed to calculate the exponent to the base e of a number in floating point format. The resulting exponent is also in floating point format.

a. Parameters

B4 Address of the first word of a two word floating point operand.

B6 Address of the first word of a two word floating point result.

b. Error Indication

Q register set positive, if the exponent part of the floating point operand exceeds 40034_8.

c. Routine Requirements

63 words of storage.

Time requirement is 695.2μs maximum.

- FLOATING POINT NATURAL LOGARITHM (FPLOGE$)

FPLOGE$ is designed to calculate the logarithm to the base e of a number which is in floating point notation. The resultant logarithm is also in floating point notation.

a. Parameters

B4 Address of the first word of a two word floating point operand.

B6 Address of the first word of the two word floating point logarithm.

b. Error Indication

A register set non-zero if

(1) fixed point part of operand is > 1 or < 1/2.

(2) fixed point part of logarithm is zero.

c. Routine Requirements

121 words of storage.

Time requirement is 208.8μs minimum to 1130.2 maximum.

■ FLOATING POINT LOAD AND CONVERT (FPSTART$)

FPSTART$ is a routine designed to load paper tape containing decimal numbers in the specified input format (see NUMBER REPRESENTATION, this section), convert them to the two word floating point format, and store the floating point numbers in the preset locations given on the paper tape at input time.

a. Parameters

B6 Contains the address for storing the converted floating point number.

The original contents of index registers B4, B5, B6, B7 are restored prior to the exit from FPSTART$.

b. Error Indications

FPSTART$ recognizes certain types of errors. Detection of an error results in a console typewriter printout giving pertinent information about the error. The different error printouts and descriptions are as follows:

PRINTOUT	DESCRIPTION
NOT OCT	Address on the input tape is not octal.
NO TAB	No tab code following the address on the input tape.
NOT DEC	A non-decimal code is present in the power or fixed part of the number on the input tape.
NO DEC PT	The decimal point is missing or out of place in the input number.
RANGE ERR	The power of the number is out of range, i.e., Power > 40.
END CODE	Ending code error. The tape does not end with a double period.
SCALE ERR	Normalizing or scaling error from scale routine, FPSCL$.
MULT ERROR	Overflow error from multiply routine, FPMUL$.
DIV ERROR	Overflow or zero division error from divide routine, FPDIV$.

After a console error printout, FPSTART$ is temporarily suspended by a REX STOPRUN instruction. The program may be resumed by initiating the appropriate program start (PS) request via the console.

c. Routine Requirements

354 words of storage.

Timing requirements vary with the value of the input number and the speed of the input equipment.

- FLOATING POINT CONVERT AND PUNCH OUTPUT (FPPUNCH$)

FPPUNCH$ is an output routine designed to convert a number, in floating point notation, to its decimal equivalent and punch it out on paper tape. It also punches out the specified flexcode (as indicated by input parameter B5) following the number.

a. Parameters

B4 Address of the first word of a two word floating point number to be converted and punched out.

B5 Flexcode for the desired end-of-line action.

B6 Contains the number of decimal equivalents to convert and punch.

b. Error Indications

FPPUNCH$ recognizes certain types of errors through the conversion routine, FPCONV$. Detection of an error results in a console typewriter printout giving pertinent information about the error. The different error printouts and descriptions are as follows:

PRINTOUT	DESCRIPTION
DIVIDE ROUTINE ERROR FPCONV$	Overflow or zero divisor error from the divide routine, FPDIV$.
MULTIPLY ROUTINE ERROR FPCONV$	Overflow error from the multiply routine, FPMUL$.
OVERFLOW ERROR	Converted decimal value > 10^{40}.

c. Routine requirements

314 words of storage.

Timing requirements vary with the value of the numbers to be converted and the speed of the output equipment.

- FLOATING POINT CONVERT AND TYPE (FPTYPE$)

FPTYPE$ is an output routine designed to convert a number, in floating point notation, to its decimal equivalent and type the number on the console typewriter.

a. Parameters

B4 Address of the first word of a two word floating point number to be converted and typed out.

B6 Contains the number of decimal equivalents to convert and type.

b. Error Indications

FPTYPE$ recognizes certain types of errors through the conversion routine, FPCONV$. Detection of an error results in a console typewriter printout giving pertinent information about the error. The different error printouts and descriptions are as follows:

PRINTOUT	DESCRIPTION
DIVIDE ROUTINE	Overflow or zero divisor error from the divide routine, FPDIV$.
MULTIPLY ROUTINE ERROR FPCONV$	Overflow error from the multiply routine, FPMUL$.
OVERFLOW ERROR	Converted decimal value $> 10^{40}$.

c. Routine Requirements

301 words of storage.

Timing requirements vary with the value of the numbers to be converted and the speed of the output equipment.

■ FLOATING POINT SET OUTPUT LENGTH (FPSET$)

FPSET$ is an output routine designed to set the number of digits (n) after the decimal point such that $2 \leq n \leq 7$. All following outputs, either punched on paper tape or printed on the console typewriter, will have the number of digits specified by the previous instruction providing linkage to FPSET$.

a. Parameters

B4 Contains the value which sets the number of digits in the output number.

b. Error Indications

If the number of digits specified is less than 2 or greater than 7 the following message is printed:

NUMBER SIZE ERROR

c. Routine Requirements

236 words of storage.

Timing requirement is 75μs.

– SUBROUTINES

These routines are used internally in other floating point routines.

■ FLOATING POINT SCALE ROUTINE (FPSCL$)

FPSCL$ is used to maintain the floating point operands in the correct format with a 15 bit exponent biased at 40000_8, and a 28 bit normalized fixed point part.

a. Parameters

A Signed value of the fixed point part.

B6 Address of the first word of the two word floating point operand before normalizing.

b. Error Indications

Overflow (exponent $>77777_8$) indicated by setting Q positive.

c. Routine Requirements

39 words of storage.

Timing requirement 156μs minimum to 496.8 maximum.

d. Routine Entrance

This routine may be incorporated into a program and entered by the following coding:

```
ENT.B6.a
ENT.A.W(B6+1)
{[EXECUTE.FPSCL$]
        or
 [CALL.FPSCL$]
 [RJP.FPSCL$]}
RJP.EA.QPOS
```

a	Address of the first word of the two word floating point operand
EA	Error Address

■ FLOATING POINT DECIMAL CONVERSION (FPCONV$)

FPCONV$ is used to convert notation to its decimal equivalent as required for a subsequent operation, such as punching or typing a converted number.

a. Parameters

B4 Address of the first word of the two word floating point number (used internally in the conversion process).

The three decimal equivalent after the conversion process will be contained in a three word output storage (FPTYPE1$, FPTYPE2$, FPTYPE3$).

The original contents of index registers B4, B5, B6 are restored prior to the exit from FPCONV$.

b. Error Indications

FPCONV$ recognizes certain types of errors. Detection of an error results in a console typewriter printout giving pertinent information about the error. The different error printouts are as follows:

PRINTOUT	DESCRIPTION
DIVIDE ROUTINE	Overflow or zero divisor error from the divide routine, FPDIV$.
MULTIPLY ROUTINE ERROR FPCONV$	Overflow error from the multiply routine, FPMUL$.
OVERFLOW ERROR	Converted decimal value > 10^{40}.

c. Routine Requirements

214 words of storage.

The execution time varies with the value of the numbers to be converted.

d. Routine Entrance

This routine may be incorporated into a program and entered by the following coding:

```
 ENT.B4.a
[EXECUTE.FPCONV$]

        or

 CALL.FPCONV$
 RJP.FPCONV$
```

a	Address of the first word of the two word floating point operand.

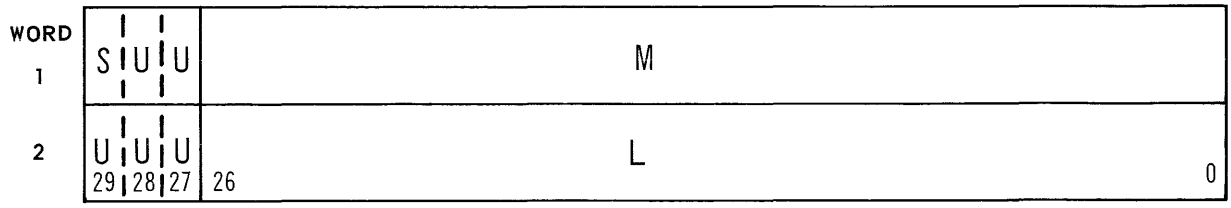
4. Double Precision Arithmetic Routines

The UNIVAC 490 Double Precision Arithmetic Routines are part of the SPURT library and provide the basic arithmetic routines, add, subtract, multiply and divide.

The double precision system is designed to permit inclusion of other routines with minimum difficulty.

The practical limit of numbers in single precision representation is eight decimal digits in one word. The double precision system extends the range of numbers to $\pm 10^{16} - 1$. The double precision routines operate in the binary mode. Double precision calculations are more efficiently performed in this mode than in a character oriented mode.

– Double Precision Format



Legend

M Most significant part of the binary number.

L Least significant part of the binary number.

S Sign indicator having the following meaning:

0 The two word number is positive.

1 The two word number is negative.

U Unused. Bits 27 and 28 of word 1 and bits 27, 28 and 29 of word 2 are ignored.

The UNIVAC 490 double precision routines require as standard format a binary number represented in two consecutive computer words consisting of a sign-bit part and a fixed-point part. The largest number which can be represented in a computer word is $2^{30}-1$, which is inadequate to accommodate the largest double precision number of sixteen decimal digits equivalent to $10^{16}-1$.

The double precision arithmetic number is a signed, absolute binary value represented in two consecutive computer words. Each computer word can represent a decimal number whose absolute value is not greater than $10^{8}-1$ (note that the upper three bits are not magnitude bits in either word). Therefore, the maximum total absolute value of the two word field is $10^{16}-1$.

The operand number will be considered an integer since the radix point is assumed at the right of L. The radix point is, in fact, disregarded in any arithmetic operation; therefore, the worker program must supply techniques for radix point manipulation, if necessary. For example, in the Add and Subtract operations, point alignment is assumed.

It is important to note that no complementing of operands occurs during any arithmetic operation. The operands are considered positive integers and the sign bit settings provide the data necessary for algebraic manipulation of the operands. For example, if unlike signed operands were involved in an initial add request, the operation actually performed would be a subtract.

– Number Representation

The maximum size operand is $10^{16}-1$ and may be produced directly from a six-bit Fieldata character numeric field by either performing the conversion routine DPFDTOBIN$ (see Conversion Routine), or by the application of any other technique within the worker program prior to execution of an arithmetic routine. However, the two word, double precision binary format must always be maintained.

On input, the maximum field size of the numeric operands is equivalent to 16 decimal digits. For example, if a 16-digit decimal number is to be converted into the proper binary format, the procedure must involve a separation of the number into segments, such that each segment shall not exceed eight decimal digits.

The following example illustrates a 16-digit number represented in segments of eight digits, such that each segment is less than or equal to 10^8-1 which occupies 27 bits when represented as a binary number. The representation is

$$\underbrace{S\ X_{15}\ X_{14}\ X_{13}\ X_{12}\ X_{11}\ X_{10}\ X_9\ X_8}_{M}\ \underbrace{X_7\ X_6\ X_5\ X_4\ X_3\ X_2\ X_1\ X_0}_{L}$$

where S is the sign bit and M and L represent the fixed point part of the binary operand as follows:

S	M
	L

In the conversion process, the eight least significant digits are converted into a binary equivalent and placed in L; the eight most significant digits are converted into a binary equivalent and placed in the word M. If a sign character is present, the appropriate bit setting is represented in the sign bit position of the most significant word M.

Thus the value of the binary arithmetic number is given by

$$N = \pm (M \cdot 10^8 + L \cdot 10^0)$$ where N is

now considered to be some number to the base 10^8 with the following restrictions:

1. $0 \leq M < 10^8$
2. $0 \leq L < 10^8$
3. If $N \geq 10^{16}$, an overflow condition is indicated for all arithmetic results except in the multiply operation where the product $\leq (10^{16}-1)^2$.
4. Decimal point alignment is assumed in Add and Subtract operations.
5. Housekeeping activities relative to the decimal point must be maintained within the main program.
6. A sign indicator code must be present in bit 29 of M.

■ DOUBLE PRECISION ADDITION SIGN CHECK (DPADD$)

DPADD$ is a composite routine designed to perform algebraic double precision addition of binary operands which are in the specified double precision notation.

The signs of the two operands (augend and addend) are inspected to determine if an add (DPAD$) or subtract (DPSB$) routine is to be actually executed. A numeric indicator with a value of 0, 1, 2, or 3 is set by the sign check routine prior to executing the Add or Subtract routine. This numeric indicator corresponds to a specific combination of operand sign values. It is subsequently used in DPAD$ or DPSB$ to determine the sign value of the final arithmetic result.

The operand sign combinations, numeric indicators, and related arithmetic routines are indicated in the following table:

Augend Sign	Addend Sign	Numeric Indicator	Arithmetic Routine
+	+	0	DPAD$
+	–	1	DPSB$
–	–	2	DPAD$
–	+	3	DPSB$

Since DPADD$ automatically executes DPAD$ or DPSB$ on the basis of sign comparisons, the macro call line used to initiate action consists only of the augend address, addend address, receiving field address and error address, as the first, second, third and fourth operands, respectively. There is no routine name operand.

Error conditions exist in the double-precision routines. Therefore an error indication operand is provided in the macro call line; however, the coding for this error procedure must be included within the worker program as a closed subroutine.

In summary, the main functions of DPADD$ are to perform a sign comparison and execute either an add (DPAD$) or subtract (DPSB$) operation to obtain the algebraic sum. If the sign values of the binary arithmetic operands are known, the implementation of this routine is unnecessary. In this case, DPAD$ or DPSB$ may be used directly, provided that the required parameter and indicator presettings are made.

More complete explanations of the individual routines, DPAD$ and DPSB$, are given in separate sections.

a. Parameters

Entrance:

B4 Address of the first word of the two word double precision operand, the augend.

B5 Address of the first word of the two word double precision operand, the addend.

B6 Address of the first word of the two word double precision algebraic sum.

Exit:

Same as entrance.

b. Error Indications

An error indicator is provided in DPADD$ when the value of the binary result exceeds $10^{16}-1$. The A register is set negative in DPADD$ when this error condition exists. Appropriate action is initiated by the macro call line error address operand or by the own code operand via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The double precision binary numbers must conform to the format described in the introduction to this subsection. Any deviation from this specified format will produce unpredictable and erroneous results.

A Fieldata field may be converted to a two word, double precision binary operand by using the appropriate routine (DPFDTOBIN$) described under CONVERSION ROUTINES.

d. Routine Requirements

46 words of storage.

The execution time for DPADD$ may vary from 172.8 μs in the case where both operands are positive to 253.2 μs where operand signs differ. The total execution time for an addition or subtraction of two operands may be determined by adding the individual execution times of either DPAD$ or DPSB$ to the execution time of DPADD$. The execution time for DPAD$ and DPSB$ are given with the description of the individual routines in this subsection.

■ DOUBLE PRECISION SUBTRACTION SIGN CHECK (DPSUB$)

DPSUB$ is a composite routine designed to perform algebraic double precision subtraction of binary operands which are in the specified double precision notation.

The signs of the two operands (minuend and subtrahend) are inspected to determine if a subtract or an add routine is to be actually executed. A numeric indicator with a value of 0,1,2, or 3 is set by the sign check routine prior to executing the Subtract or Add routine. This numeric indicator corresponds to a specific combination of operand sign values. It is subsequently used in DPSB$ or DPAD$ to determine the sign value of the final arithmetic result.

The operand sign combinations, numeric indicators, and related arithmetic routines are indicated in the following table:

Minuend Sign	Subtrahend Sign	Numeric Indicator	Arithmetic Routine
+	–	0	DPAD$
+	+	1	DPSB$
–	+	2	DPAD$
–	–	3	DPSB$

Since DPSUB$ automatically executes DPSB$ or DPAD$ on the basis of sign comparisons, the macro call line used to initiate action consists only of the minuend address, subtrahend address, receiving field address and error address, as the first, second, third and fourth operands, respectively. There is no routine name operand.

Error conditions exist in the double-precision routines. Therefore an error indication operand is provided in the macro call line; however, the coding for this error procedure must be included within the main program as a closed subroutine.

In summary, the main functions of DPSUB$ are to perform a sign comparison and execute either a Subtract (DPSB$) or an Add (DPAD$) operation to obtain the algebraic difference. If the sign values of the binary arithmetic operands are known, the implementation of this routine is unnecessary. In this case, DPSB$ or DPAD$ may be used directly, provided that the required parameter and indicator presettings are made.

More complete explanations of the individual routines, DPSB$ and DPAD$, are given in separate sections.

a. Parameters

Entrance:

B4 Address of the first word of the two word double precision operand, the minuend.

B5 Address of the first word of the two word double precision operand, the subtrahend.

B6 Address of the first word of the two word double precision algebraic sum.

Exit:

Same as entrance.

b. Error Indications

An error indicator is provided in DPSUB$ when the value of the binary result exceeds $10^{16}-1$. The A register is set negative in DPAD$ when this error condition exists. Appropriate action is initiated by the macro call line error address operand or by the own code error address operand via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The double precision binary numbers must conform to the format as described in the introduction to this subsection. Any deviation from this specified format will produce unpredictable and erroneous results. A Fieldata field may be converted to a two word, double precision binary operand by using the appropriate routine (DPFDTOBIN$) described under CONVERSION ROUTINES.

d. Routine Requirements

46 words of storage

The execution time for DPSUB$ may vary from 172.8 μs in the case where both operands are positive to 253.2 μs where operand signs differ. The total execution time for an addition or subtraction of two operands may be determined by adding the individual execution times of either DPSB$ or DPAD$ to the execution time of DPADD$. The execution times for DPAD$ and DPSB$ are given with the description of the individual routines in this subsection.

■ DOUBLE PRECISION ADDITION (DPAD$)

DPAD$ is a routine designed to calculate the sum of numbers in double precision notation.

This routine performs the add operation using operands in the binary two word, double precision format. These operands, the augend and addend, may be produced directly from a numeric field by either performing the conversion routine DPFDTOBIN$ (see CONVERSION ROUTINES) or by the application of some other technique within the main program prior to the execution of DPAD$.

Double precision addition as herein described is a fixed-point operation involving numbers with values not exceeding $10^{16} - 1$, where decimal point alignment of the numbers at the time of the add operation does exist. The technique used in double precision addition is shown as follows: Suppose A+B=C, where double precision configurations are $A_1\ A_2 + B_1\ B_2 = C_1\ C_2$ with assumed decimal point alignment; then by using consecutive memory locations, the add operation is represented by:

Augend		Addend		Initial Sum
A_1	+	B_1	=	$C_1 = A_1 + B_1$
A_2		B_2		$C_2 = A_2 + B_2$

where A_1, B_1 and C_1 are the most significant words of the operands and A_2, B_2 and C_2 are the least significant words.

The addition process in DPAD$ proceeds in the following sequential steps.

1. The corresponding words, A_i and B_i, are added to form the initial sums, C_i, which are placed in consecutive memory locations as shown above (i = 1, 2). The least significant words (i = 2) are added first.

2. Word size adjustment to a maximum value of $10^8 - 1$ (decimal) per word, thus establishing the basis for the carry process in addition; e.g.,

   if $(A_2 + B_2) > 10^8$

   then $(A_2 + B_2) \div 10^8 = Q_2 + R_2$

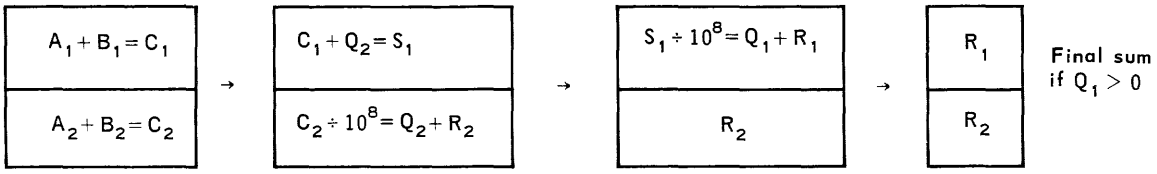
   where R_2 is the least significant word of the final sum, and Q_2 is the carry factor added to the sum of the most significant word which subsequently undergoes word size adjustment, i.e.,

   $A_1 + B_1 + Q_2 = S_1$

   $S_1 \div 10^8 = Q_1 + R_1$

   where $Q_1 = 0$ and R_1 is the most significant word of the final sum. If $Q_1 \neq 0$, a size error condition is present, indicating that the sum exceeds the accepted operand size of $10^{16} - 1$ decimal.

This may be illustrated as follows:



Note: Subtraction of $10^8 - 1$ rather than division by 10^8 is actually used within DPAD$ to obtain maximum efficiency; however, this does not effect the basic theory involved.

3. Setting an error indicator prior to exiting from DPAD$ if a size error condition is detected.

DPAD$ may function as a second level routine within DPADD$ or DPSUB$. It may also function as an independent routine provided the operand parameters are preset along with a special numeric indicator set in the A register. This indicator specifies the sign values of the operands. If both operational signs are positive, the A register is set equal to 0. If both operational signs are negative, then the sign bit positions of the most significant words of each operand must be set to zero and the A register set equal to 2 before executing DPAD$.

a. Parameters

Entrance:

B4 Address of the first word of the two word, double precision operand, the augend.

B5 Address of the first word of the two word, double precision operand, the addend.

B6 Address of the first word of the two word, double precision resultant binary sum.

A register is set as follows to determine sign value of the resultant binary sum

0 Both operands are positive; therefore, the result is positive.

2 Both operands are negative; therefore, the result is negative. (The sign bits of the most significant operand words must be cleared.)

Exit:

B4, B5, B6 same as entrance.

A register contains an error indicator:

$A < 0$ Size error condition.

$A \geq 0$ No error condition.

b. Error Indications

An error indicator is set in DPAD$ when the sum of the binary operand exceeds $10^{16} - 1$. The A register is set negative in DPAD$ when this size error condition is detected. Appropriate action is initiated by the macro call line error address operand or by the own code error address via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The double precision binary numbers must conform to the format described in the introduction to this subsection. Any deviation from this specified format will produce unpredictable and erroneous results.

A Fieldata field may be converted to a two word double precision binary operand by using the appropriate routine (DPFDTOBIN$) described under CONVERSION ROUTINES.

d. Routine Requirements

33 words of storage.

The execution time for DPAD$ may vary from 248 μs, in the case where a size error condition exists, to 280 μs for the addition of two binary operands of maximum size.

■ DOUBLE PRECISION SUBTRACTION (DPSB$)

DPSB$ is a routine designed to calculate the difference of numbers in double precision notation. This routine performs the subtract operation using operands in the binary two word, double precision format. These operands, the minuend and subtrahend, may be produced directly from a numeric field by either performing the conversion routine DPFDTOBIN$ (see CONVERSION ROUTINES), or by the application of some other technique within the main program prior to the execution of DPSB$.

Double precision subtraction as herein described in a fixed-point operation involving numbers with values not exceeding $10^{16}-1$ where decimal point alignment of the numbers at the time of the subtract operation does exist. The technique used in double precision subtraction is shown as follows: Suppose A−B=C, where the double precision configurations are $A_1\ A_2 - B_1\ B_2 = C_1\ C_2$ with assumed decimal point alignment; then by using consecutive memory locations, the subtract operation is represented by

	Minuend		Subtrahend		Initial Difference
word 1	A_1	−	B_1	=	$C_1 = A_1 - B_1$
word 2	A_2		B_2		$C_2 = A_1 - B_2$

where A_1, B_1 and C_1 are the most significant words of the operands, and A_2, B_2 and C_2 are the least significant words.

The subtraction process in DPSB$ proceeds in the following sequential steps:

1. The minuend and subtrahend words are initially compared to determine whether or not the absolute value of the minuend is greater than the absolute value of the subtrahend. If the minuend is smaller, the operands are reversed and the subtraction process is carried out in the normal manner. Therefore, if the subtraction process was initially stated as $A_1A_2 - B_1B_2 = C_1\ C_2$ and it was determined that $|A_1\ A_2| < |B_1\ B_2|$

   then the operands would be reversed and the problem restated as $B_1\ B_2 - A_1\ A_2 = C_1\ C_2$

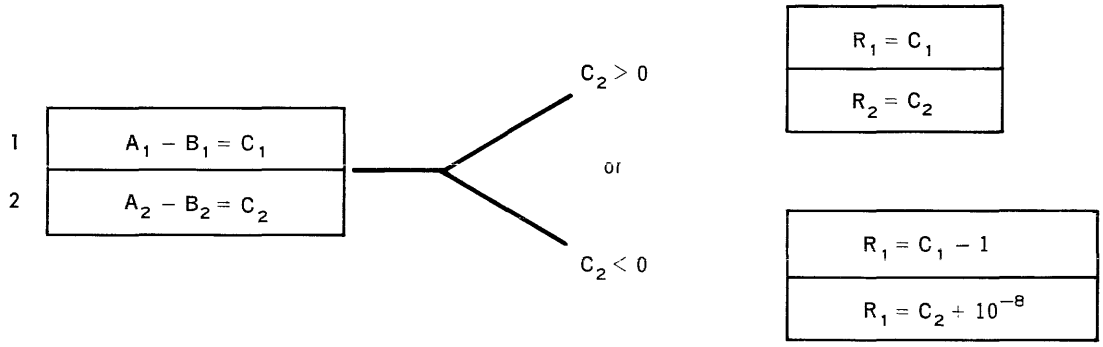
2. The corresponding words, A_i and B_i, are subtracted to form an initial difference C_i which is placed in consecutive memory locations as shown above (i = 1, 2). The least significant words are subtracted first, i.e., $A_2 - B_2 = C_2$ before $A_1 - B_1 = C_1$.

3. Each word, C_1 and C_2, of the initial difference is inspected for negativity, thus establishing the basis for the borrow process in subtraction. The following illustration clarifies this technique. Beginning with the subtraction of the least significant operand words, we have

   $A_2 - B_2 = C_2$

   If $C_2 \geq 0$, no borrow is necessary; therefore, set $R_2 = C_2$ as the least significant difference word, and $R_1 = C_1$ as the most significant word.

   The subtraction process may be illustrated as follows:



DPSB$ may function as a second-level routine within DPADD$ and DPSUB$. It may also function as an independent routine, provided that the operand parameters are preset along with a special numeric indicator. This indicator must be set in the A register to specify the sign values of the operands. The value of the numeric indicator used in DPSB$ depends on the operand sign combination indicated below.

Minuend	Subtrahend	Numeric Indicator	Result Sign
+	–	1	+
–	+	3	–

a. Parameters

Entrance:

B4 Address of the first word of the two word, double precision operand, the minuend.

B5 Address of the first word of the two word, double precision operand, the subtrahend.

B6 Address of the first word of the two word, double precision resultant binary difference.

A register is set as follows to determine sign value of the resultant binary sum:

1 Minuend is positive and subtrahend is negative; therefore, the result is positive. The sign bit of the most significant word of the subtrahend must be cleared.

3 Minuend is negative and subtrahend is positive; therefore, the result is negative. The sign bit of the most significant word of the minuend must be cleared.

Exit:

B4, B5, B6 same as entrance.

b. Error Indications

No error indicator is set in DPSB$ since the difference of the binary operands cannot exceed $10^{16}-1$.

c. Data Format

The double precision binary numbers must conform to the format described in the introduction to this subsection. Any deviation from this specified format will produce unpredictable and erroneous results.

A Fieldata field may be converted to a two word double precision binary operand by using the appropriate routine (DPFDTOBIN$) described under CONVERSION ROUTINES.

d. Routine Requirements

51 words of storage.

The execution time for DPSB$ may vary from 157 μs, in the minimum case, to 392 μs for the subtraction of a negative operand from a positive operand where reversal of operands is necessary.

■ DOUBLE PRECISION MULTIPLICATION (DPMUL$)

DPMUL$ is a routine designed to perform algebraic, double precision multiplication of binary operands which are in the specified double precision notation.

This routine inspects the signs of the two input operands, multiplier and multiplicand, and sets the sign of the resultant product before performing the actual multiplication of the operands. The two input operands are the first and second operands in the macro call line and, when multiplied, the product is placed in the memory locations referenced by the third operand of the macro call line.

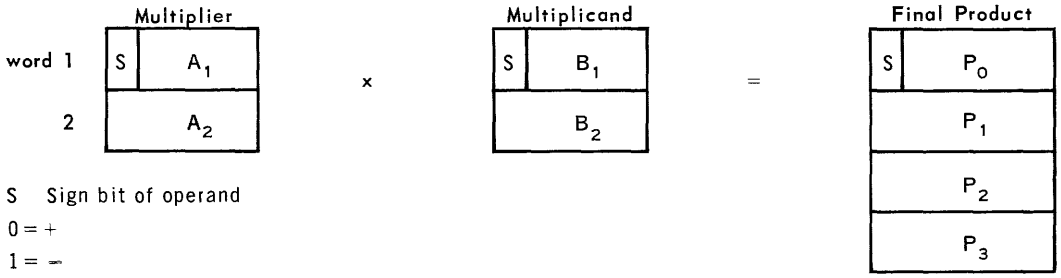
The input operands may be produced directly from a numeric field by either performing the conversion routine DPFDTOBIN$ (see CONVERSION ROUTINES) or by application of some other technique within the main program prior to the execution of DPMUL$.

A size error operand is provided as the fourth operand in the macro call line; however, the coding for this error procedure must be included within the main program as a closed subroutine.

Double precision multiplication as herein described is a fixed-point operation concerned with multiplication of numbers not exceeding $10^{16}-1$ (decimal). Decimal point alignment is not necessary when performing DPMUL$. The maximum size product which may result in a multiply operation is the square of $10^{16}-1$ which is contained in four words. The technique used in double precision multiplication is shown using the following description and illustrations.

Suppose A x B = P, where the double precision configurations are $A_1A_2 \times B_1B_2 = (A_1 \times B_1) + (A_1 \times B_2) + (A_2 \times B_1) + A_2B_2 = P_0P_1P_2P_2$ where the A_i and B_i (i = 1, 2) represent the double precision, two word multiplier and multiplicand and P_i (i=0,1,2,3) represents the maximum four word product.

By using consecutive memory locations, the double precision multiply calculation is given by the following sequential steps with accompanying illustrations based on the simplified operand format.



S Sign bit of operand
0 = +
1 = −

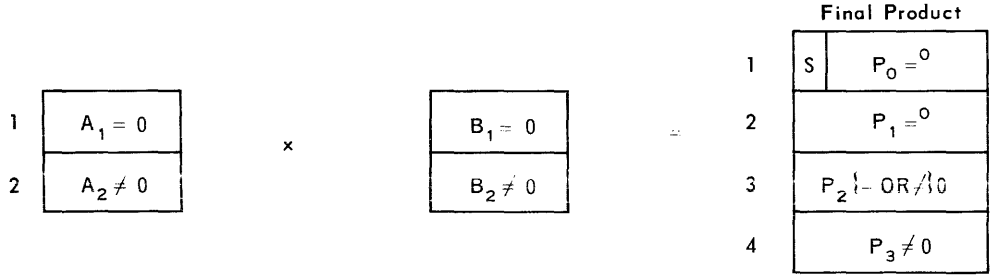
Step 1. Determine and set the final product sign value. If either of the input operands is negative the sign bit position is cleared before multiplication takes place.

Step 2. Determine by means of a jump table the number of words in the input operands. Four distinct cases result.

Case 1. 1 by 1 case

One-word multiplier and

one-word multiplicand.

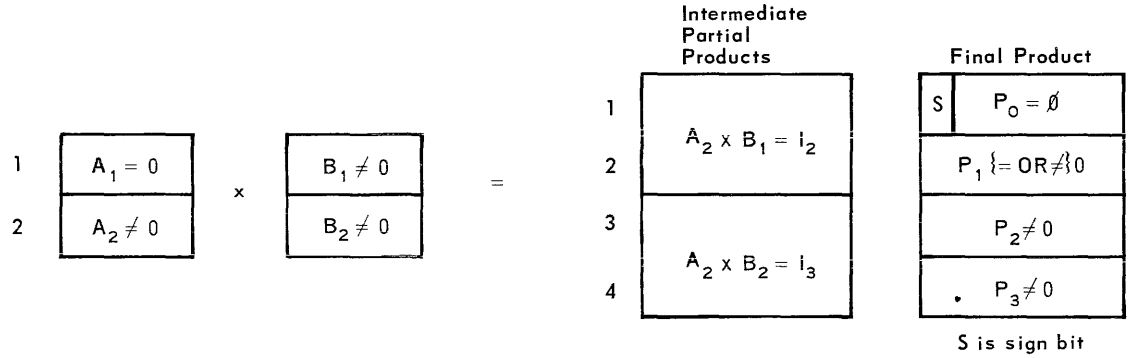


$(A_2 \times B_2)\ 10^8 = P_3 + P_2$ is the least significant product word and P_2 is the most significant product word. In this case $P_1 = P_0 = 0$.

Case 2. 1 by 2 case

One-word multiplier and

two-word multiplicand



The procedure for determining the final product is as follows:

$$A_2 x B_1 B_2 = (A_2 x B_1) + (A_2 \times B_2) = I_2 + I_3$$

$I_3 \div 10^8 = P_3 + Q_3$ where P_2 is the least significant word and Q_3 is the carry factor added to the product of the next high partial product factor which then undergoes word size adjustment; e.g., $(Q_3 + I_2) \div 10^8 = P_2 + Q_2$ where P_2 is the next higher order product word and Q_2 is used to determine the most significant product word as follows:

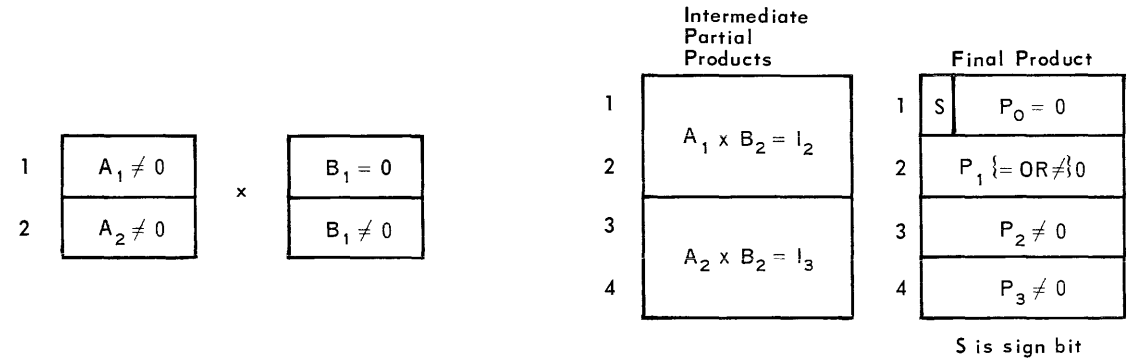
If $Q_2 \neq 0$ then $Q_2 = P_1$ which is the most significant product word with $P_0 = 0$.

If $Q_2 = 0$ then $P_1 = 0$ with $P_0 = 0$.

Case 3. 2 by 1 case

Two-word multiplier and

one-word multiplicand.



The procedure for determining the final product is as follows:

$$A_1 A_2 \times B_2 = (A_1 \times B_2) + (A_2 \times B_2) = I_2 + I_3$$

$I_3 \div 10^8 = P_3 + Q_3$ where P_3 is the least significant word and Q_3 is the carry factor added to the product of the next high partial product factor which then undergoes word size adjustment; e.g., $(Q_3 + I_2) \div 10^8 = P_2 + Q_2$ where P_2 is the next higher order product word and Q_2 is used to determine the most significant product word as follows:

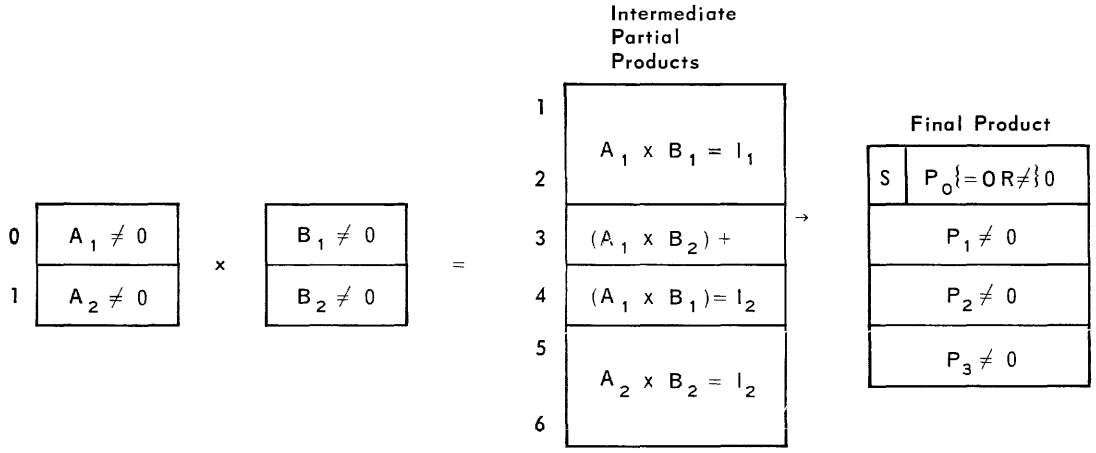
If $Q_2 \neq 0$ the $Q_2 = P_1$ which is the most significant product word with $P_0 = 0$.

If $Q_2 = 0$ then $P_1 = 0$ with $P_0 = 0$.

Case 4. 2 by 2 case

Two-word multiplier and

two-word multiplicand.



The procedure for determining the final product is as follows:

$$A_1A_2 \times B_1B_2 = (A_1 \times B_1) + [A_1 \times B_2) + (A_2 \times B_1)] + (A_2B_2) = I_1 + I_2 + I_3$$

$I_3 \div 10^8 = P_3 + Q_3$ where P_2 is the next higher order product word and Q_2 is the carry factor added to the sum of the high order partial product which then undergoes word size adjustment; e.g., $(Q_2 + I_1) \div 10^8 = P_1 + Q_1$ where P_1 is the next higher order product word and Q_1 is used to determine the most significant product word as follows:

If $Q_1 = 0$ then $Q_1 = P_0 = 0$.

If $Q_1 < 10^8$ then $Q_1 = P_0$ where P_0 is the most significant product word.

Step 3. Setting an error indicator prior to exiting from DPMUL$ if the product exceeds the square $10^{16} - 1$. If $Q_1 \geq 10^8$, then the size error condition indicator is set.

$Q_1 \div 10^8 = P_0 + Q_0$ where P_0 is the most significant product word as illustrated in Case IV.

In summary, DPMUL$ performs three main functions: it performs an operand sign check thereby determining the sign of the final product; it calculates a four-word product to a maximum value equal to the square of $10^{16} - 1$; and it performs a size error check.

a. Parameters:

Entrance:

B4 Address of the first word of two word double precision operand, the multiplier.

B5 Address of the first word of the two word, double precision operand, the multiplicand.

B6 Address of the first word of the four word double precision binary product.

Exit:

Same as entrance.

b. Error Indications

An error indicator is provided in DPMUL$ when the value of the binary result exceeds the square of $10^{16} - 1$. The Q register is set negative when an error condition exists. Appropriate action is initiated by the macro call line error address operand or by the own code error address operand via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The double precision binary numbers must conform to the format described in the introduction to this subsection. Any deviation from this specified format will produce unpredictable and erroneous results. A Fieldata field may be converted to a two word double precision binary operand by using the appropriate routine (DPFDTOBIN$) described under CONVERSION ROUTINES.

d. Routine Requirements

166 words of storage.

The execution time for DPMUL$ may vary from approximately 778.4 μs in the case where operands contain only one word and are positive to 1430 μs where two word negative operands are used.

■ DOUBLE PRECISION DIVISION

DPDIV$ is a routine designed to perform algebraic double precision division of binary operands which are in the specified double precision notation. This routine inspects the signs of the two input operands, divisor and dividend, and sets the sign of the resultant quotient before performing the actual division of the operands. The two input operands are the first and second operands in the macro call line, and when division occurs, the quotient is placed in the memory locations referenced by the third operand of the macro call line.

The input operands may be produced directly from a numeric field by either performing the conversion routine DPFDTOBIN$ (see CONVERSION ROUTINES) or by the application of some other technique within the main program prior to the execution of DPDIV$.

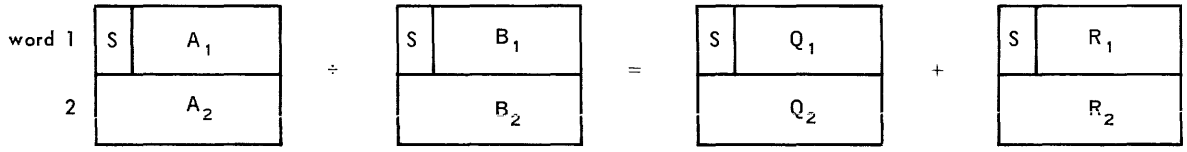
An error indication operand is provided as the fourth operand in the macro call line; however, the coding for this error procedure must be included within the main program as a closed subroutine.

Double precision division as herein described is a fixed point operation concerned with division of numbers not exceeding $10^{16}-1$. Decimal point alignment is not necessary when performing DPDIV$. The technique used in double precision division is shown using the following descriptions and illustrations based on the general Euclidean division algorithm, N = b q + r, where N is the number, b represents a base of 10^8 in double precision division, and q and r are quotient and remainder, respectively.

Suppose $A \div B=Q+R$ where the general double precision configuration is $A_1A_2 \div B_1B_2 = Q_1Q_2 + R_1R_2$ where A_i, B_i, Q_i and R_i represent the double precision two word dividend, divisor, quotient and remainder, respectively (i = 1,2).

The remainder is available to the worker program for rounding purposes by directly referencing the two word working storage area, "DPDIVREM$" which is included within the divide routine coding.

By using consecutive memory locations, the double precision divide calculation is given by the following sequential steps with accompanying illustrations based on the simplified operand format:



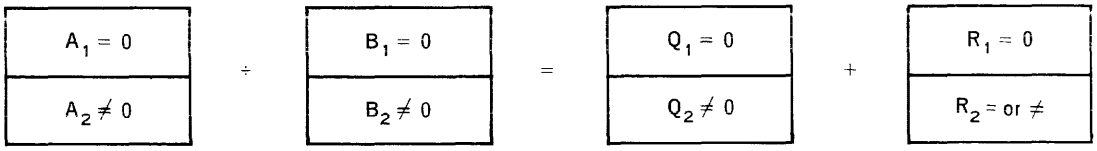
S Sign bit of operand
0 = +
1 = −

Step 1. Determine and set the final quotient sign value. If either of the input operands is negative, the sign bit position is cleared before division takes place.

Step 2. Determine by means of a jump table the number of words in the input operands. Four distinct cases result.

Case 1. 1 by 1 case

One word dividend and one word divisor where the operand range is between 0 and 10^8-1.



$A_1 = B_1 = 0$ and $|A_2| > |B_2|$ (Dividend > Divisor) then $A_2 \div B_2 = Q_2 + R_2$ where Q_2 is the one word quotient, and R_2 is the one word remainder, and $Q_1 = R_1 = 0$.

$A_1 = B_1 = 0$ and $|A_2| < |B_2|$ (Dividend < Divisor) then $Q_1 = Q_2 = 0$, $R_1 = 0$ and $R_2 = A_2$, i.e., the quotient is zero and the remainder is set equal to the original dividend.

Case 2. 1 by 2 case

One word dividend and two word divisor.

$A_1 = 0$ $A_2 \neq 0$	÷	$B_1 \neq 0$ $B_2 \neq 0$	=	$Q_1 = 0$ $Q_2 = 0$	+	$R_1 = 0$ $R_2 = A_2$	

$|A_2| < |B_1\ B_2|$ (Dividend < Divisor)

Therefore $Q_1 = Q_2 = 0$ and the remainder is set equal to the original dividend, i.e., $R_2 = A_2$.

Case 3. 2 by 1 case

Two word dividend and one word divisor where the range of the dividend is between 0 and 10^{16} and the range of the divisor is between 0 and 10^8.

$A_1 \neq 0$ $A_2 \neq 0$	÷	$B_1 = 0$ $B_2 \neq 0$	=	$Q_1 \{= \text{or} \neq\} 0$ $Q_2 = 0$	+	$R_1 = 0$ $R_2 \{= \text{or} \neq\} 0$	

$|A_1\ A_2| > |B_1|$ (Dividend > Divisor)

$A_1 \div B_1 = Q_1 + R_1$ where Q_1 is the most significant quotient word and R_1 is used to calculate the new dividend, D',

where $D' = R_1 \cdot 10^8 + A_2$

then $D' \div B_1 = Q_2 + R_2$ where Q_2 is the least significant quotient word and R_2 is the remainder which may be used for rounding.

Case 4. 2 by 2 case

Two word dividend and two word divisor where the operand range is between 0 and $10^{16}-1$.

$A_1 = 0$ $A_2 \{= \text{or} \neq\} 0$	÷	$B_1 = 0$ $B_2 \{= \text{or} \neq\} 0$	=	$Q_1 = 0$ $Q_2 \neq 0$	+	$R_1 = 0$ $R_2 \{= \text{or} \neq\} 0$	

$|A_1\ A_2| > |B_1\ B_2|$ (Dividend > Divisor)

The calculation of the true quotient, Q_2, and true remainder, R_2, is given in the following sequential steps:

(1) Scale the divisor, B_1B_2, S places such that its value lies between 10^{15} and 10^{16}, e.g.,

$$10^{16} > [(B_1 \cdot 10^8 + B_2)S] > 10^{15}$$

(2) Determine a trial divisor, D_T, from the scaled divisor (indicated above) equivalent in value to the upper eight decimal digits of the divisor incremented by 1, e.g.,

$$[(B_1 \cdot 10^8 + B_2)\, S \div 10^8] + 1 = D_T$$

(3) Determine a trial quotient, Q_T, using the original dividend and trial divisor D_T, e.g.,

$(A_1 \cdot 10^8 + A_2) \div D_T = Q_T + R_T$; R_T is the trial remainder and is ignored.

(4) Using a preset scale factor, S', (derived from original scale factor) (S), scale down the trial quotient, Q_T, to obtain a near approximation of the true quotient,

Q_T (scaled) $\rightarrow Q_{A1} + R_A$ where Q_{A1} is the first approximated quotient and R_A is the ignored remainder.

(5) Determine a new dividend, D_{N1} using the approximated quotient Q_{A1}. This procedure involves the use of double precision multiply and double precision subtract.

$(A_1A_2) - (Q_{A1} \cdot B_1B_2) = D_{N1}$ where A_1 A_2 and B_1 B_2 are in double precision format.

(6) Determine by an iterative subtract method the number of adjustments to the trial quotient, Q_{A1}, to obtain the true quotient Q_2.

If $D_{N1} > B_1B_2$, then $Q_{A1} + 1 = Q_{A2}$ is the new approximated quotient and D_{N2} is the new dividend, e.g., $D_{N1} - (Q_{A2} \cdot B_1B_2) = D_{N2}$

These iterations occur until there is a point at which $D_{Ni} < B_1\, B_2$.

Therefore the true quotient, Q_2, is represented by:

lim $Q_2 = Q_{A1} + \Sigma$ iterations,

and the remainder $R_2 = D_{Ni} - (Q_{Ai+1} \cdot B_1B_2) = D_{Ni+1}$

where $0 < D_{Ni+1} < B_1\, B_2$

$|A_1\, A_2| < |B_1\, B_2|$ (Dividend < Divisor)

In this 2 by 2 case, the quotient is set equal to zero and the remainder is set equal to the dividend, i.e.,

$Q_1 = Q_2 = 0$,

$R_1\, R_2 = A_1\, A_2$.

Step 3. Set an error indicator prior to the exit from DPDIV$ if any error condition is detected during the divide process.

a. Parameters

Entrance:

B4 Address of the first word of the two word double precision operand, the divisor.

B5 Address of the first word of the two word double precision operand, the dividend.

B6 Address of the first word of the four word double precision binary quotient.

Exit:

Same as entrance.

b. Error Indications

An error indicator is provided in DPDIV$ when:

(1) The divisor is zero.

(2) The dividend is zero.

(3) The divisor is larger than the dividend.

When the error conditions occur, the quotient is set equal to zero and the remainder assumes the value of the dividend or zero (when the dividend is zero).

The Q register is set negative when an error condition exists. Appropriate action is initiated by the macro call line error address operand or by the own code error address operand via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The double precision binary numbers must conform to the format described in the introduction to this subsection. Any deviation from this specified format will produce unpredictable and erroneous results. A Fieldata field may be converted to a two word, double precision, binary operand by using the appropriate routine (DPFDTOBIN$) described under CONVERSION ROUTINES.

d. Routine Requirements

216 words of storage.

The execution time for DPDIV$ may vary from approximately 590 μs in the case of a positive, one word divisor and dividend, to approximately 3150 μs when both operands (divisor and dividend) are negative with numeric values between 10^8 and $10^{16} - 1$. The execution time of 3150 μs includes the average execution times of the multiply routine and subtract routine (described in this section). However, the iteration time of 242 μs required for each successive adjustment to the trial quotient to obtain the true quotient is not included. As a result, the maximum execution time may vary considerable depending upon the number of iterations.

5. Fieldata Arithmetic Routines

The UNIVAC 490 Fieldata Arithmetic Routines are part of the SPURT library and provide two basic arithmetic routines, add (FDADD$) and subtract (FDSUB$) in a single precision format. They are useful where a value is incremented or decremented by a small value and then printed, as in a page count tally for a printed form. Complex arithmetic operations are performed more efficiently by other routines in this subsection.

The standard format for these routines consists of a computer word containing Fieldata numeric characters. The largest number which can be represented in a computer word is 10^5-1 which must be represented as six bit Fieldata character codes. For example, the number 9876 in the proper operand format would be represented in Fieldata codes in one computer word as

Character Position	0	1	2	3	4
	60	71	70	67	66

The Fieldata number is always normalized right in a computer word with a Fieldata zero filled in at the left if the number does not occupy a full word.

The operand number will be considered an integer since the decimal point is assumed at the right of the least significant character. The decimal point is, in fact, disregarded in either arithmetic operation; therefore, it is necessary for the worker program to supply techniques for decimal point manipulation, if necessary. For example, in the Add and Subtract operations, point alignment is assumed.

It is important to note that no complementing of operands occurs during any arithmetic operation. The operands are considered positive, therefore the programmer must decide which operation, FDADD$ or FDSUB$ is to be performed with specific operands.

■ FIELDATA ADDITION (FDADD$)

FDADD$ is a routine designed to calculate the sum of numbers in a one word, six bit Fieldata character representation.

This routine adds two positive, five character Fieldata words, augend and addend, and stores the sum as a five character Fieldata word. This is accomplished by performing a series of masking operations in the operands with subsequent addition and overflow check, thus simulating a Fieldata adder.

The two input operands may be set within the main program or may be the result of some operation within the main program.

The augend and addend are the first and second operands in the macro call line. The sum is placed in the memory locations referenced by the third operand of the macro call line.

A size error operand is provided as the fourth operand in the macro call line; however, the coding for this error procedure must be included within the main program as a closed subroutine.

a. Parameters

Entrance:

B4 Address of the one word Fieldata numeric operand, the addend.

B5 Address of the one word Fieldata numeric operand, the augend.

B6 Address of the one word Fieldata resultant sum.

Exit:

Same as entrance.

b. Error Indications

An error indicator is provided in FDADD$ when the value of the Fieldata result exceeds 10^5-1. The Q register is set negative when an error condition exists. Appropriate action is initiated by the macro call line error address operand or by the own code error address operand via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The Fieldata operands must conform to the format as described at the beginning of this subsection. Any deviation from this specified format will produce unpredictable and erroneous results.

d. Routine Requirements

32 words of storage.

The execution time for FDADD$ varies from 232.8 μs in the case of no overflow (size error) to 244.8 μs when an overflow condition exists.

■ FIELDATA SUBTRACT (FDSUB$)

FDSUB$ is a routine designed to calculate the difference of numbers in a one word, six bit Fieldata character representation.

This routine initially performs an operand size check. If the minuend is greater than the subtrahend, normal subtraction is implemented; however, if the minuend is less than the subtrahend, the operands are reversed, an error indication is set and the normal subtraction is carried out. This routine subtracts two five-character Fieldata words, minuend and subtrahend and stores the difference as a five-character Fieldata word. This is accomplished by performing a series of masking operations in the operands with subsequent addition and check, thus simulating a Fieldata subtracter.

The two input operands may be set within the main program or may be the result of some operation within the main program.

The minuend and subtrahend are the first and second operands in the macro call line. The difference is placed in the memory locations referenced by the third operand of the macro call line.

An error indication for operand reversal is provided as the fourth operand in the macro call line; however, the coding for this error procedure must be included within the main program as a closed subroutine.

a. Parameters

Entrance:

B4 Address of the one word, Fieldata numeric operand, the minuend.

B5 Address of the one word, Fieldata numeric operand, the subtrahend.

B6 Address of the one word, Fieldata resultant difference.

Exit:

Same as entrance.

b. Error Indication

An error indicator is provided in FDSUB$ when the initial minuend and subtrahend are reversed. The Q register is set negative when the operands are reversed. Appropriate action is initiated by the macro call line error address operand or by the own code error address operand via a return jump to a closed subroutine which must be included within the main program.

c. Data Format

The Fieldata operands must conform to the format as described at the beginning of this subsection. Any deviation from this specified format will produce unpredictable and erroneous results.

d. Routine Requirements

47 words of storage.

The execution time for FDSUB$ varies from 296.4 μs in the case of no reversal of operands to 320.4 μs with operand reversal.

6. Routine Implementation

The routines that have been described in subsection 1 through 5 may be included in the coding of a program and initiated by either a macro call line or an own code option. This subsection describes the coding required for each option and specifies the variable operands that must be inserted by the user. The page reference in the heading refers to the description of the routine in the text.

CONVERSION ROUTINES

FIELDATA TO BINARY (FDTOBIN$)	Page 14-H-4

```
MACRO:   FDTOBIN$ . a, b, c, EA

OWN
CODE:    ENT . B5 . a

         ENT . B6 . b

         ENT . B7 . c

         EXECUTE . FDTOBIN$

         RJP . EA
```

a	First word address of the two word Fieldata storage.
b	Address of binary storage.
c	Number of characters to be converted.
EA	Entrance to user coded error address.

BINARY TO FIELDATA (BINTOFD$)	Page 14-H-7

```
MACRO:  BINTOFD$ . a, b

OWN
CODE:   ENT . B4 . a

        ENT . B5 . b

        EXECUTE . BINTOFD$
```

a	Address of binary storage.
b	First word address of two word Fieldata storage.

CODE CONVERSION (CODECON$)	Page 14-H-8

```
MACRO:  CODECON$ . a, b, c, d, e

OWN
CODE:   ENT . B4 . a

        ENT . B5 . b

        ENT . B6 . c

        ENT . B7 . d

        ENT . A  . e

        RJP . CODECON$
```

a	The first word address of the pickup field.
b	The first word address of the deposit field.
c	The number of words to be translated.
d	Translation mode.
e	The first word address of the translation table.

DOUBLE PRECISION BINARY TO FIELDATA (DPBINTOFD$) Page 14-H-16

MACRO: DPBINTOFD$. a, b, c, d, e, f

OWN
CODE: ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

ENT . B7 . d

ENT . A . e

EXECUTE . DPFDTOBIN$

RJP . **EA**

a	First word address of pickup field.
b	First word address of deposit field.
c	First character position in the deposit field.
d	Number of characters in the deposit field.
e	Sign option indicator. If this operand is omitted in the macro call line, the operand is automatically made zero by the SPURT assembly system, thus causing sign insertion to be inhibited.
EA	Address of a closed error subroutine.

FIELDATA TO DOUBLE PRECISION BINARY (DPFDTOBIN$) Page 14-H-19

```
MACRO:   DPFDTOBIN$ . a, b, c, d, EA

OWN
CODE:    ENT . B4 . a

         ENT . B5 . b

         ENT . B6 . a

         ENT . B7 . d

         EXECUTE . DPFDTOBIN$

         RJP . EA
```

a	First word address of pickup field.
b	First word address of deposit field.
c	First character position in the pickup field.
d	Number of characters in the pickup field.
EA	Address of a closed error subroutine.

EDITING ROUTINES

ZERO SUPPRESSION (ZSEDIT$)	Page 14–H–23

```
MACRO:   ZSEDIT$ . a, b

OWN
CODE:    ENT . B6 . a

         ENT . B7 . b

         EXECUTE . ZSEDIT$
```

a	Number of characters (words) to zero suppress.
b	Address of first word (n) in the 20 word working storage.

FLOAT DOLLAR SIGN (FLDEDIT$)	Page 14–H–25

```
MACRO:   FLDEDIT$ . a

OWN
CODE:    ENT . B7 . a

         EXECUTE . FLEDIT$
```

a	Address of first word (n) in 20 word working storage.

CHECK PROTECTION (CHKEDIT$)	Page 14–H–27

```
MACRO:   CHECKEDIT$ . a

OWN
CODE:    ENT . B7 . a

         EXECUTE . CHKEDIT$
```

a	Address of first word (n) in 20 word working storage.

FLOAT PLUS OR MINUS SIGN (FLPMEDIT$)	Page 14–H–29

```
MACRO:   FLPMEDIT$ . a, b

OWN
CODE:    ENT . B6 . a

         ENT . B7 . b

         EXECUTE . FLPMEDIT$
```

a	Code to indicate floating plus or minus (0 = plus ; 1 = minus).
b	Address of the first word (n) in the 20 word working storage.

MERGE CHARACTERS (MERGEDIT$)	Page 14–H–34

```
MACRO:   MERGEDIT$ . a

OWN
CODE:    ENT . B7 . a

         EXECUTE . MERGEDIT$
```

a	Address of the first word (n) in the 20 word working storage.

FLOATING POINT ROUTINES

FLOATING POINT ADD OR SUBTRACT (FPAS$)	Page 14-H-39

MACRO: FPAS$ {FDADD$ / FDSUB$}, a, b, c, **EA**

OWN
CODE:
```
CALL . FPAS$
ENT . B4 . a
ENT . B5 . b
ENT . B6 . c
RJP {FPADD$ / FPSUB$}
RJP . EA . QPOS
```

a	Address of the first word of the augend or minuend.
b	Address of the first word of the addend or subtrahend.
c	Address of the first word of the result.
EA	Error address.

FLOATING POINT MULTIPLY OR DIVIDE (FPMD$)	Page 14-H-40

MACRO: FPMD$ {FPMUL$ / FPDIV$}, a, b, c, **EA**

OWN
CODE:
```
CALL . FPMD$
ENT . B4 . a
ENT . B5 . b
```

ENT . B6 . c

RJP { FPMUL$ / FPDIV$ }

RJP . **EA** . QPOS

a	Address of the first word of the multiplicand or dividend.
b	Address of the first word of the multiplier or divisor.
c	Address of the first word of the product or quotient.
EA	Error address.

FLOATING POINT CONVERSION (FPFXFL$)	Page 14–H–40

MACRO: FPFXFL$ { FXTOFL$ / FLTOFX$ } , a, b, c, **EA**

OWN CODE:

CALL . FPFXFL$

ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

RJP { FXTOFL$ / FLTOFX$ }

RJP . **EA** . QPOS

a	Position of the scaling point.
b	Address of the first word of the operand.
c	Address of the first word of the result.
EA	Error address for FLTOFX$.

FLOATING POINT SINE OR COSINE (FPSNCS$)	Page 14–H–41

MACRO: FPSNCS$.{FPSIN$ / FPCOS$} , a, b, **EA**

OWN
CODE: CALL . FPSNCS$

ENT . B4 . a

ENT . B6 . b

RJP .{FPSIN$ / FPCOS$}

RJP . **EA** . QPOS

a	Address of the first word of the operand.
b	Address of the first word of the result.
EA	Error address.

FLOATING POINT ARCTANGENT (FPATAN$)	Page 14–H–41

MACRO: FPATAN$. a, b, **EA**

OWN
CODE: ENT . B4 . a

ENT . B6 . b

[EXECUTE . FPATAN$]

or

[CALL . FPATAN$ / RJP . FPATAN$]

RJP . **EA** . QPOS

a	Address of the first word of the tangent function value.
b	Address of the first word of the result (in radians).
EA	Error address.

FLOATING POINT SQUARE ROOT (FPSQR$)	Page 14-H-42

MACRO: FPSQR$. a, b, **EA**

OWN
CODE: ENT . B4 . a

ENT . B6 . b

[EXECUTE . FPSQR$]

or

[CALL . FPSQR$]
[RJP . FPSQR$]

RJP . **EA** . ANEG

a	Address of the first word of the radicand.
b	Address of the first word of the root.
c	Error address.

FLOATING POINT EXPONENT (FPEXP$)	Page 14-H-42

MACRO: FPEXP$. a, b, **EA**

OWN
CODE: ENT . B4 . a

ENT . B6 . b

[EXECUTE . FPEXP$]

or

[CALL . FPEXP$]
[RJP . FPEXP$]

RJP . EA . QPOS

a	Address of the first word of the operand.
b	Address of the first word of the result.
EA	Error address.

FLOATING POINT NATURAL LOGARITH (FPLOGE$)	Page 14–H–42

MACRO: FPLOGE$. a, b, **EA**

OWN
CODE: ENT . B4 . a

ENT . B6 . b

[EXECUTE . FPLOGE$]

or

[CALL . FPLOGE$
RJP . FPLOGE$]

RJP . **EA** . ANOT

a	Address of the first word of the operand.
b	Address of the first word of the result.
EA	Error address.

FLOATING POINT LOAD AND CONVERT (FPSTART$)	Page 14–H–43

MACRO: (not applicable)

OWN
CODE: [EXECUTE . FPSTART$]

or

[CALL . FPSTART$
RJP . FPSTART$]

Prior to executing FPSTART$, the paper tape containing the input data must be properly set in the reader.

FLOATING POINT CONVERT AND PUNCH (FPPUNCH$)	Page 14–H–44

MACRO: FPPUNCH$. a, b, c

OWN
CODE: ENT . B4 . a

ENT . B5 . c

ENT . B6 . b

[EXECUTE . FPPUNCH$]

or

[CALL . FPPUNCH$
RJP . FPPUNCH$]

a	Address of the first of a two word, floating point number to be converted and punched.
b	Number of decimal equivalents to convert and punch.
c	Flexcode for the desired end of file action.

FLOATING POINT CONVERT AND TYPE (FPTYPE$)	Page 14–H–44

MACRO: FPTYPE$. a, b

OWN
CODE: ENT . B4 . a

ENT . B6 . b

[EXECUTE . FPTYPE$]

or

[CALL . FPTYPE$
RJP . FPTYPE$]

a	Address of the first word of a two word, floating point number to be converted and typed.
b	Number of decimal equivalents to convert and type.

FLOATING POINT SET OUTPUT LENGTH (FPSET$)	Page 14-H-45

MACRO: FPSET$. a

OWN
CODE: ENT . B4 . a

[EXECUTE . FPSET$]

or

[CALL . FPSET$]
[RJP . FPSET$]

a	Contains the value which sets the number of digits in the output number.

FLOATING POINT SCALE (FPSCL$)	Page 14-H-46

MACRO: (not applicable)

OWN
CODE: ENT . B6 . a

ENT . A . W (B6 + 1)

[EXECUTE . FPSCL$]

or

[CALL . FPSCL$]
[RJP . FPSCL$]

RJP . **EA** . QPOS

a	Address of the first word of the two word floating point operand.

EA Error address.

FLOATING POINT DECIMAL CONVERSION (FPCONV$)	Page 14-H-46

MACRO: (not applicable)

OWN
CODE: ENT . B4 . a

[EXECUTE . FPCONV$]

or

CALL . FPCONV$

RJP . FPCONV$

a	Address of the first word of the two word, floating point operand.

DOUBLE PRECISION ROUTINES

DOUBLE PRECISION ADD SIGN CHECK (DPADD$)	Page 14–H–49

MACRO: DPADD$. a, b, c, **EA**

OWN
CODE: ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

EXECUTE . DPADD$

RJP . **EA** . ANEG

a	Address of augend.
b	Address of addend.
c	Address of result.
EA	Error address.

DOUBLE PRECISION SUBTRACT SIGN CHECK (DPSUB$)	Page 14–H–51

MACRO: DPSUB$. a, b, c, **EA**

OWN
CODE: ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

EXECUTE . DPSUB$

RJP . **EA** . ANEG

a	Address of minuend.
b	Address of subtrahend.
c	Address of result.
EA	Error address.

DOUBLE PRECISION ADD (DPAD$)	Page 14-H-52

MACRO: (1) DPADD$. a, b, c, **EA**

(2) DPSUB$. a, b, c, **EA**

DPAD$ is entered through the DPADD$ when signs are like (1) or, through DPSUB$ when signs are unlike (2)

OWN
CODE: ENT . A . NI

ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

EXECUTE . DPAD$

RJP . **EA** . ANEG

a	Address of augend.
b	Address of addend.
c	Address of result.
EA	Error Address.
NI	Numeric indicator which must be set when using the own code option: 0, if both operands are positive; 2, if both operands are negative. The sign bits of the most significant operand words must be cleared before executing DPAD$.

DOUBLE PRECISION SUBTRACT (DPSB$)	Page 14–H–55

MACRO: (1) DPADD$. a, b, c, **EA**

(2) DPSUB$. a, b, c, **EA**

DPSB$ is entered through the DPADD$ when signs are unlike (1) or, through DPSUB$ when signs are like (2).

OWN CODE:

```
ENT . A . NI

ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

EXECUTE . DPSB$
```

a	Address of augend.
b	Address of addend.
c	Address of result.
EA	Error address (when using DPADD$ or DPSUB$).
NI	Numeric indicator which must be set when using the own code option: 1, if minuend positive, subtrahend negative; 3, if minuend negative, subtrahend positive.

DOUBLE PRECISION MULTIPLY (DPMUL$)	Page 14-H-57

MACRO: DPMUL$. a, b, c, **EA**

OWN
CODE:
```
ENT . B4 . a
ENT . B5 . b
ENT . B6 . c
EXECUTE . DPMUL$
RJP . EA . QNEG
```

a	Address of multiplier.
b	Address of multiplicand.
c	Address of product.
EA	Error address.

DOUBLE PRECISION DIVIDE (DPDIV$)	Page 14-H-61

MACRO: DPDIV$. a, b, c, **EA**

OWN
CODE:
```
ENT . B4 . a
ENT . B5 . b
ENT . B6 . c
EXECUTE . DPDIV$
RJP . EA . QNEG
```

a	Address of divisor.
b	Address of dividend.
c	Address of quotient.
EA	Error address.

FIELDATA ROUTINES

FIELDATA ADD (FFADD$)	Page 14-H-66

MACRO: FDADD$. a, b, c, **EA**

OWN
CODE: ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

EXECUTE . FDADD$

RJP . **EA** . QNEG

a	Address of augend.
b	Address of addend.
c	Address of sum.
EA	Error address.

FIELDATA SUBTRACT	Page 14-H-67

MACRO: FDSUB$. a, b, c, **EA**

OWN
CODE: ENT . B4 . a

ENT . B5 . b

ENT . B6 . c

EXECUTE . FDSUB$

RJP . **EA** . QNEG

a	Address of minuend.
b	Address of subtrahend.
c	Address of difference.
EA	Error address.

APPENDIX A. COMPUTER INSTRUCTIONS

Figure A-1 provides a summary of the Computer instructions for the UNIVAC 490 Real-Time System.

FUNCTION CODE DESIGNATOR	INSTRUCTION	OPERATION	CLASS
01	Shift Q Right	Shift (Q) Right by Y	R
02	Shift A Right	Shift (A) Right by Y	
03	Shift AQ Right	Shift (AQ) Right by Y	
04	Compare	*Compare (A) with Y or (Q) with Y or (AQ) with Y; and sense j	
05	Shift Q Left	Shift (Q) Left by Y	
06	Shift A Left	Shift (A) Left by Y	
07	Shift AQ Left	Shift (AQ) Left by Y	
10	Enter Q	Y→Q	
11	Enter A	Y→A	
12	Enter B_J	Y→B_J	
13	Enter External Function	**Y→C_J if j ≠ 0 or 1. Test C_0/C_1; if active, skip NI when j = 0 or 1.	
14	Store Q	(Q)→Y	S
15	Store A	(A)→Y	
16	Store B_J	(B_J)→Y	
17	Store Input Channel	**(C_J)→Y	
20	Add	(A) + Y→A	R
21	Subtract	(A) - Y→A	
22	Multiply	(Q) • Y→AQ	
23	Divide	*(AQ) ÷ Y; Quotient→Q, Remainder→A	
24	Replace Add	(A) + Y→Y and A	RP
25	Replace Subtract	(A) - Y→Y and A	
26	Q Add	*(Q) + Y→Q	R
27	Q Subtract	*(Q) - Y→Q	
30	Add Q and Load A	(Q) + Y→A	
31	Subtract Q and Load A	Y - (Q)→A	
32	Add Q and Store	(A) + (Q)→Y and A	S
33	Subtract Q and Store	(A) - (Q)→Y and A	
34	Replace Add Q	(Q) + Y→Y and A	RP
35	Replace Subtract Q	Y - (Q)→Y and A	
36	Replace Add One	Y + 1→Y and A	
37	Replace Subtract One	Y - 1→Y and A	
40	Enter Logical Product	*L (Y • (Q))→A	R
41	Add Logical Product	L (Y • (Q)) + (A)→A	
42	Subtract Logical Product	(A) - L (Y • (Q))→A	
43	Masked Comparison	(A) - L (Y • (Q)); sense j (A) and (Q) unchanged	
44	Replace Logical Product	*L (Y • (Q))→Y and A	RP
45	Replace Add Logical Product	L (Y • (Q)) + (A)→Y and A	
46	Replace Subtract Logical Product	(A) - L (Y•(Q))→Y and A	
47	Store Logical Product	L (A)(Q)→Y	S
50	Selective Set	Set $(A)_n$ for $(Y)_n = 1$	R
51	Selective Complement	Complement $(A)_n$ for $(Y)_n = 1$	
52	Selective Clear	Clear $(A)_n$ for $(Y)_n = 1$	
53	Selective Substitute	$(Y)_n$→$(A)_n$ for $(Q)_n = 1$	
54	Replace Selective Set	Set $(A)_n$ for $(Y)_n = 1$→Y and A	RP
55	Replace Selective Complement	Complement $(A)_n$ for $(Y)_n = 1$→Y and A	
56	Replace Selective Clear	Clear $(A)_n$ for $(Y)_n = 1$→Y and A	
57	Replace Selective Substitute	$(Y)_n$→$(A)_n$ for $(Q)_n = 1$→Y and A	
60	Arithmetic Jump (normal)	*Y→P per j	R
61	Manual Jump (normal)	*Y→P per j	
62	Input Active Buffer Jump	**If C_J active, jump to Y	
63	Output Active Buffer Jump	**If C_J active, jump to Y	
64	Arithmetic Return Jump	*j = 0, no jump; j = 1 jump to Y + 1, (P)→Y	
65	Manual Return Jump	*Return Jump, stop if STOP key up	
66	Terminate Input	**Terminate input buffer on channel j	
67	Terminate Output	**Terminate output buffer on channel j	
70	Repeat	*NI Y times per j	
71	Index Skip	If (B_J) = Y, skip NI and clear B_J; If (B_J) ≠ Y, Advance B_J and execute NI	
72	Index Jump	If (B_J) = 0, execute NI; If (B_J) ≠ 0, jump to Y; subtract 1 from (B_J)	
73	Initiate Input	**Activate input buffer on channel j	
74	Initiate Output	**Activate output buffer on channel j	
75	Initiate Input, Monitor	**Activate input buffer on channel j	
76	Initiate Output, Monitor	**Activate output buffer on channel j	

LEGEND:

* = Special j designator

** = Input/Output Instructions

R = Read Class Instructions

RP = Replace Class Instructions

S = Store Class Instructions

In Figure A-1, the following conventions are used to describe the operations that are performed by the instructions:

( )	the contents of the storage location or register enclosed within the parentheses.
→	the quantity on the left is transferred to the storage location or register on the right.
A	the accumulator, a 30-bit shift register.
Y	the operand.
AQ	the 60-bit shift register formed by using A and Q together as a single register.
j	the 3-bit branch condition designator or the particular input/output channel designated by j.
(A)i (A)f	the initial and final contents of register A, respectively.
Q	the 30-bit quotient register.
Bj	the particular B register designated by j.
L	the logical product or bit-by-bit product of binary digits.
( )n	the nth bit position of the specified register or address enclosed within the parentheses.
P	the 15-bit program address counter register.
NI	the next instruction to be executed.

A. INSTRUCTION WORD

As shown in Figure A-2, each 30-bit instruction word is composed of five designators.

f Designator

The function code designator, f, is a six-bit code that specifies the principal operation – shift, store, add, and so on – to be performed by a program step. Sixty-two function code values constitute the UNIVAC Real-Time System repertorie of instructions.

y Designator

The y designator is a 15-bit code from which is derived either the address of the operand or the operand itself.

b Designator

The operand address modification designator, b, is a 3-bit code that governs the first modification of y. This modication involves adding to y the contents of a B register designated by b.

k Designator

The operand-interpretation designator, k, is usually a 3-bit code that controls the procedure by which the operand is obtained and/or stored. The effect of k is different for each of three instruction categories: read, store, and replace.

The operand interpretation designator also controls data transmissions during store operations.

j Designator

The branch-condition designator, j, is usually a 3-bit code that may be interpreted as a skip or jump-condition designator, a register designator, or a repeat modification designator.

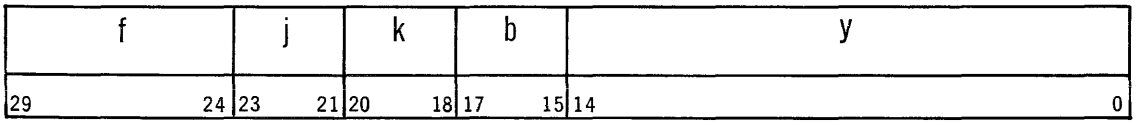


Figure A-2. Instruction Word

B. INSTRUCTION CYCLE

The instruction execution cycle begins with a storage access period that transfers the address of the next instruction from register P to register S. Next the 15-bit contents of the storage address register, (register S) are translated, activating the storage selection system, which transmits a 30-bit instruction from memory into register U.

As soon as it enters register U, the instruction word controls the execution of the remaining phases of the program step.

The instruction word function code and various designators are then translated. If address modification is specified, the contents of the designated index register are added to the address portion (15 low order bits) of the instruction in register U before execution. Now the control mechanism performs the operations designated by the instruction word; that is, it executes the program step – add, subtract, compare, and so on.

The instruction word remains in register U until it is replaced by the succeeding instruction word at the beginning of the next program step.

C. j and k DESIGNATOR INTERPRETATION

The normal j and k designator interpretations, special j designator interpretations, and j and k designator combinations for input/output instructions are shown in the text that follows:

- Normal j and k Designator Interpretation

  Figure A-3 shows the normal j and k designator interpretations. The k designator interpretations that are shown apply to all instructions within a given class. The j designator interpretations apply to all instructions except input/output instructions and those that are involved with the transfer of data to and from the B-registers. In the former case the j designator functions as an input/output channel designator and in the latter case as a B-register designator.

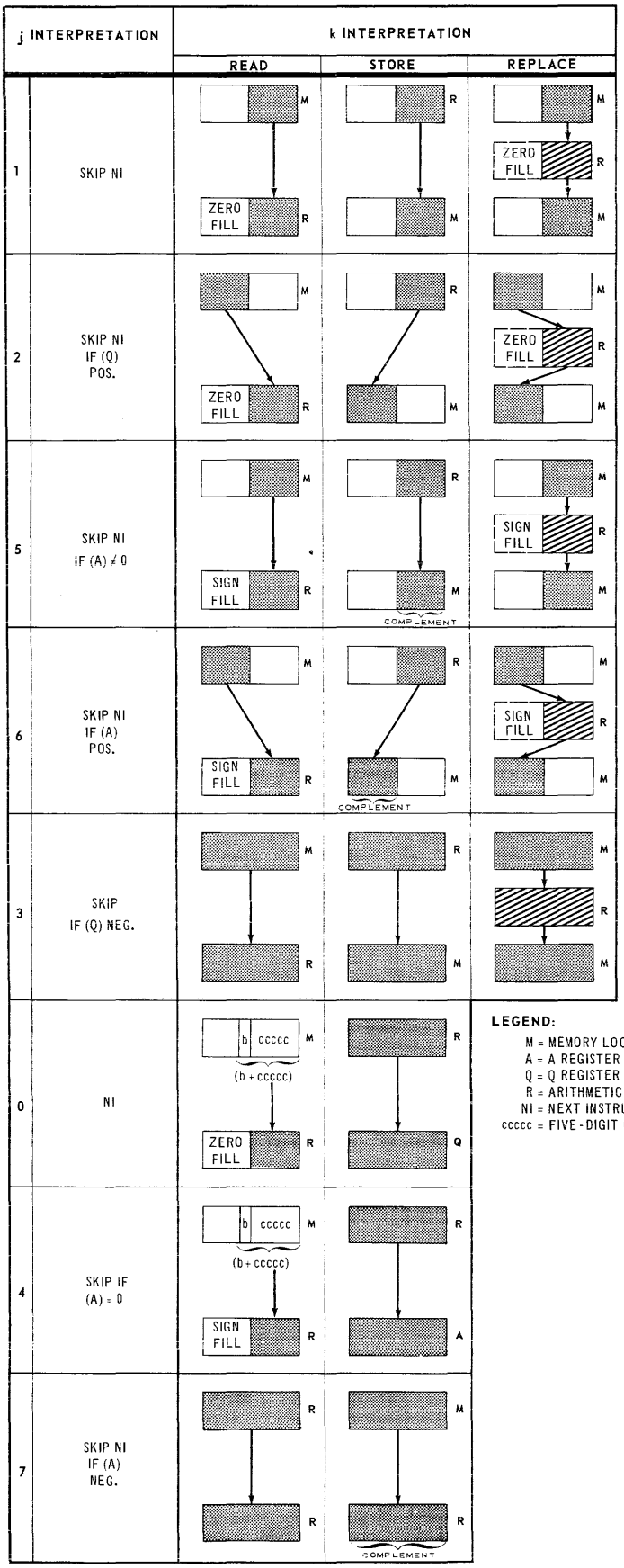


Figure A–3. Normal j and k Designator Interpretation

- Special j Designator Interpretation

The special j designator interpretations that are shown in Figure A-4 apply only to the instructions that are indicated.

j	FUNCTION CODE DESIGNATOR						
	04	23	26/27	40/44	60/64	61/65	70
0	NI	NI	NI	NI	RELEASE SET/INTERLOCK	JUMP	NE: Y = Y
1	SKIP NI	SKIP NI	SKIP NI	SKIP NI	RELEASE SET/INTERLOCK JUMP	JUMP IF KEY 1 SET	NE: Y = Y+1
2	SKIP NI IF Y ≤ (Q)	SKIP NI IF NO OVERFLOW	SKIP NI IF (A) POS.	SKIP NI IF (A) EVEN PARITY	JUMP IF (Q) POS.	JUMP IF KEY 2 SET	NE: Y = Y−1
3	SKIP NI IF Y > (Q)	SKIP NI IF OVERFLOW	SKIP NI IF (A) NEG.	SKIP NI IF (A) ODD PARITY	JUMP IF (Q) NEG.	JUMP IF KEY 3 SET	NE: Y = Y+Bn*
4	SKIP NI IF (Q)≥Y>(A)	SKIP NI IF (A) = 0	SKIP NI IF (Q) = 0	SKIP NI IF (A) = 0	JUMP IF (A) = 0	JUMP STOP	NE: Y = Y+(B6)
5	SKIP NI IF (Q) < Y or Y ≤ (A)	SKIP NI IF (A) ≠ 0	SKIP NI IF (Q) ≠ 0	SKIP NI IF (A) ≠ 0	JUMP IF IF (A) ≠ 0	JUMP STOP IF STOP KEY 5 IS SET	NE: Y = Y+1+(B6)
6	SKIP NI IF Y ≤ (A)	SKIP NI IF (A) POS.	SKIP NI IF (Q) POS.	SKIP NI IF (A) POS.	JUMP IF (A) POS.	JUMP STOP IF STOP KEY 6 IS SET	NE: Y = Y−1+(B6)
7	SKIP NI IF Y > (A)	SKIP NI IF (A) NEG.	SKIP NI IF (Q) NEG.	SKIP NI IF (A) NEG.	JUMP IF (A) NEG.	JUMP STOP IF STOP KEY 7 IS SET	NE: Y = Y+Bn*+(B6)

Add cumulatively the B-register indicated in the repeated operation to its operand during each execution.

Figure A-4. Special j Designator Interpretation

■ j and k Designator Combinations For Input/Output Instructions

The j and k designators for the input/output instructions occupy the same six bit positions in the instruction word as those for non input/output instructions; however, the input/output j designator utilizes four of these six positions and the k designator the remainder. When input/output instructions are written, the six bits that represent the j and k combination appear as two octal digits as do the six bits that represent these designators in all other instructions. In the case of input/output instructions, these octal digits are considered as a unit that represents a specific j and k combination rather than having one digit represent the j designator and one the k designator. The octal digits that represent the j and k combinations for input/output instructions are shown in Figure A-5. For example, assume that an input/output instruction is to be written with the requirement that j = 5 and k = 3. An examination of the j = 5 row and k = 3 in the diagram will show that 27 is the two-digit octal number that meets this requirement.

	k = 0	k = 1	k = 2	k = 3
j = 0	00	01	02	03
j = 1	04	05	06	07
j = 2	10	11	12	13
j = 3	14	15	16	17
j = 4	20	21	22	23
j = 5	24	25	26	27
j = 6	30	31	32	33
j = 7	34	35	36	37
$8_D = 10_8$	40	41	42	43
$9_D = 11_8$	44	45	46	47
$10_D = 12_8$	50	51	52	53
$11_D = 13_8$	54	55	56	57
$12_D = 14_8$	60	61	62	63
$13_D = 15_8$	64	65	66	67

Figure A-5. j and k Combinations for Input/Output Instructions

APPENDIX D. PERIPHERAL SUBSYSTEMS

This appendix provides a detailed description of the functional characteristics of each of the peripheral subsystems that are designed to operate in a UNIVAC 490 Real-Time System.

The function codes and status indications for each subsystem are listed to complete the information necessary for coding basic input/output instructions.

Input/output operations that are controlled by the Real-Time Executive Routine (REX) must be coded with the mnemonic instructions described in Section 6, SPURT INPUT/OUTPUT UNDER EXECUTIVE CONTROL. The brief description of the input/output operations contained in Section 6 should be sufficient. However, if a more detailed description of a peripheral subsystem is desired, it may be found in this appendix.

A. FLYING HEAD – 880 MAGNETIC DRUM SUBSYSTEM

The Flying Head – 880 Magnetic Drum Subsystem is used as a mass-memory storage device in the UNIVAC 490 Real-Time System. Differing from conventional magnetic drums, the storage employed in this system makes use of a unique approach developed by UNIVAC engineers – the "flying" read/write head.

Unlike the fixed heads of conventional drums, the flying head of the Flying Head – 880 Magnetic Drum Subsystem floats on a boundary layer of air along the drum's surface, following its contours in a precise manner. This technique eliminates the recording disadvantages of the wider gap that separated recording heads and drum surfaces that was once necessary to compensate for the surface eccentricities common to all drums. Instead, because the boundary layer is quite thin, the head-to-surface spacing is greatly reduced and a greater recording density is therefore possible. Because rotation is required to create the boundary layer and "fly" the head, a special mechanism lowers the heads after the drum reaches speed, and retracts them for shutdown or in the event of a power failure.

Forty head blocks are positioned around the drum. Mounted in each of these head blocks are 22 "flying" read/write heads, one for each recording track which revolves beneath the block. There are 128 6-track data recording bands across the drum. Each band can store 6,144 computer words, allowing a total of 786,432 computer words or 3,932,160 alpha-numeric characters to be recorded on the drum. The remaining tracks are used for error checking procedures and timing functions. In addition, four tracks function as address tracks to provide an address reference around the circumference of the drum. The read/write heads record information on the drum's surface at a density of 518 bits per inch while it revolves at 1,800 revolutions per minute. Recording frequency is is approximately 1 megacycle. Average access time is one-half drum revolution or 17 milliseconds.

All drums in the system are available to the central processor at any time. However, only one drum may be read or written on at any one time on a single input/output channel. If an operation requires reading from one drum and simultaneously writing on a second drum, two subsystems, each on a separate input/output channel must be employed. Up to eight drums may be connected to each available input/output channel. A Channel Synchronizer/Control Unit is required to control all the drums on a channel.

1. Addressing

   To facilitate reading and writing of information on the magnetic drums, a simplified scheme of addressing has been devised. Each of the 786,432 storage locations on each drum has a unique address composed of three parts: Angular Section (5-bits), Channel (7-bits) and Angular Address (11-bits).

   - Angular Sections

     The angular sections are intermixed around the drum to provide maximum storage with continuous addressability. For each angular address, angular section 1 will be found between angular section 0 and angular section 2.

   - Channels

     There are 128 6-track bands or channels across the surface of the drum. Each channel can store 6,144 computer words.

   - Angular Address

     Around the periphery of each drum are 2,048 angular addresses per section or a total of 6,144 unique addresses.

   - Address Format

     Reading or writing in consecutive addresses will begin at the address (23 bits) specified in the Function Word, and then proceed to the next angular address.

     When angular address 2,048 is reached, the next word will be read or written in angular address 0000 of the next channel. When channel 128 is reached, the next word will be written or read in channel 00 of the next section.

     A particular address is specified by 23 bit positions in a function word.

     The uppermost angular address of any channel is:

     $$2047_{10} = 3777_8 = 11\ 111\ 111\ 11 \text{ (binary)}$$

     The uppermost channel number on any drum is:

     $$127_{10} = 177_8 = 1\ 111\ 111 \text{ (binary)}$$

     The highest angular section number in an eight drum subsystem is:

     $$23_{10} = 27_8 = 10\ 111 \text{ (binary)}$$

Combining these three individual bit patterns and grouping in sets of three for octal interpretation gives:

BIT POSITION	22 18	17 11	10 0
ADDRESS PORTION	ANGULAR SECTION NUMBER	CHANNEL NUMBER	ANGULAR ADDRESS
BINARY	1 0 1 1 1	1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1
OCTAL	2 7	7 7	7 7 7 7

This pattern represents, in octal notation, the uppermost storage address on drum number 7 of the subsystem.

Figure A-1 is a simplified diagrammatic representation of the address structure. Each section is shown occupying a distinct portion of the drum. The sections are in reality intermixed around the drum.

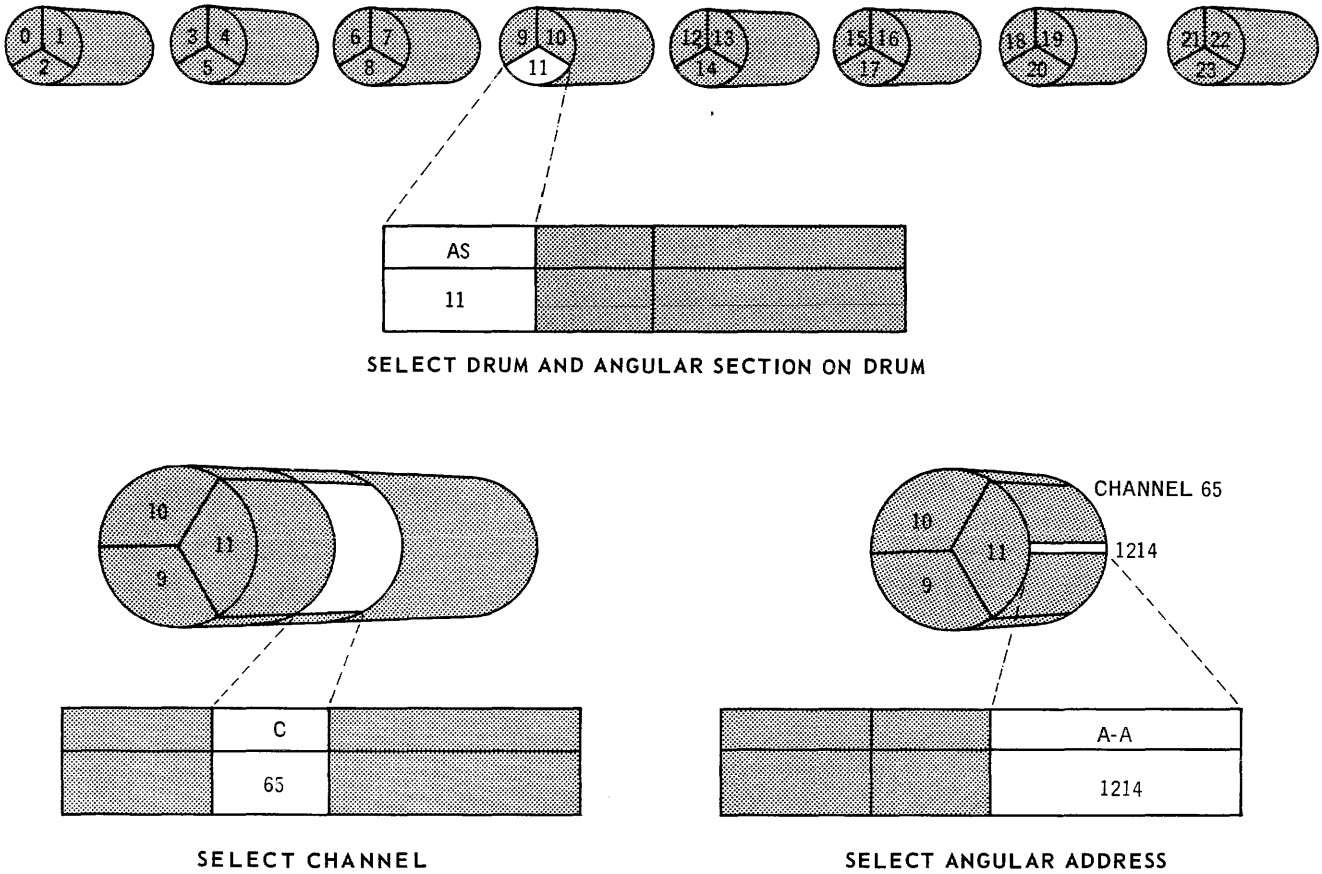


Figure A-1. Address Structure of Flying Head – 880 Magnetic Drum Subsystem

As a practical matter, however, the user is not concerned with the bit patterns of the addressing system because he need specify drum addresses in octal notation only. The range of addresses (in octal) for each drum on an eight drum subsystem is presented in the following table.

UNIT	BEGINNING ADDRESS	ENDING ADDRESS
0	00 000 000	02 777 777
1	03 000 000	05 777 777
2	06 000 000	10 777 777
3	11 000 000	13 777 777
4	14 000 000	16 777 777
5	17 000 000	21 777 777
6	22 000 000	24 777 777
7	25 000 000	27 777 777

Reading from or writing in consecutive addresses begins at the address specified in the Function Word and proceeds to consecutive angular addresses, channels, angular sections and drum units. For instance, starting with the first address in a subsystem, angular section 0 on drum unit 0 is written or read starting with angular address 0, channel 0 and continuing through angular address 2047, channel 127. Then angular sections 1 and 2 on the drum unit are written or read in the same sequence. When angular section 2, angular address 2047, channel 127 has been read, the same procedure is repeated for angular sections 3, 4 and 5 on drum unit 1, and so on.

2. The Channel Synchronizer/Control Unit

The Channel Synchronizer/Control Unit for the subsystem controls the operation of the magnetic drum units during their write and read functions, transfers the data to be written from the central processor to the drums, and transfers the data which has been read from the drums to the central processor.

- Write Function

  During the write function, the Channel Synchronizer/Control Unit receives the Function Word from the central processor and translates it into a write command to the selected drum. Then it receives data words from the central processor and transfers them to the drum for writing.

- Read Function

  During the read function, the Channel Synchronizer/Control Unit receives the Function Word from the central processor and translates it into a read command to the selected drum. Then it receives data words from the drum and transfers them to the central processor.

- Address Selection

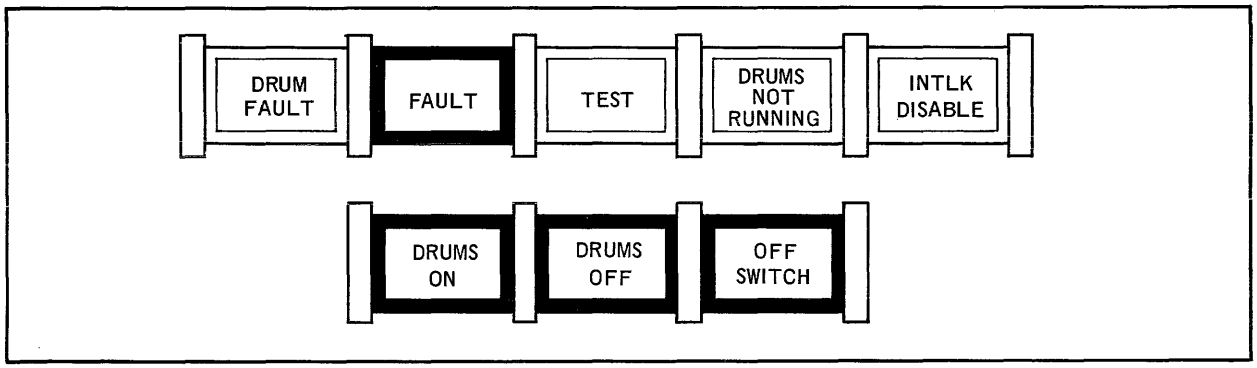
  The Channel Synchronizer/Control Unit decodes addresses contained in the Function Word and selects the requested address.

- Error Detection

  The Channel Synchronizer/Control Unit generates a parity bit for each word transferred from the central processor during a write operation. It also checks for the parity bit in a read operation. Data words are checked for the proper number of characters.

3. OPERATOR CONTROLS

Primary control of the Flying Head – 880 Magnetic Drum Subsystem is accomplished by the program. However, the Control Cabinet has an Operator control panel that shows operating conditions in the subsystem and allows some manual operation.
The control panel indicators and their functions are as follows:



LEGEND:

DRUM FAULT — Lights under any of the following conditions:

- High or low temperature exists in a drum unit.
- A voltage fault occurs in a drum unit.
- One of the drums becomes inoperative after all drums in the subsystem have attained operating speed.

FAULT — Lights whenever the DRUM FAULT light is on or when one of the following is detected:

- Low voltage in Control Unit.
- High current in Control Unit.
- High temperature in Control Unit.
- Loss of air in Control Unit.

Pressing this button after the fault condition is cleared, resets the sensing circuitry.

TEST — Lights when any test toggle switch in the subsystem is in its test position.

DRUMS NOT RUNNING: Lights when any of the following conditions exist:

- All drums are not in the automatic mode of operation.
- Power in a drum unit is off.
- One or more of the drums is not up to full speed.
- Write voltage in a drum unit is off.

INTLK DISABLE — Lights when the Bootstrap-Write switch is in the ON position or when the Synchronizer Interlock switch is bypassed.

DRUMS ON — Pressing this button starts sequential operation of the drums. Logical drum unit 0 starts first and, once it is up to speed, the sequence continues until all drums are running. This indicator lights immediately; however, the DRUMS NOT RUNNING indicator remains on until all drums in the subsystem are up to speed.

DRUMS OFF — Pressing this button removes power from the drums and lights the indicator. This button must be held until the DRUMS ON indicator goes out. Power to the Magnetic Drum Control Cabinet is not affected.

OFF SWITCH — Pressing this button removes power from the Magnetic Drum Control Cabinet unit. Power on is indicated by green light; power off by red light.

4. Operations

Any part of the central processor's internal core storage can be used as an input/output data buffer storage area, with the exception of the few special core storage locations that are re-reserved for the Incremental Clock and the Interrupt Words. Information is transferred between the central processor and the subsystem in the form of 30-bit data words. These words are formed into blocks in core storage. A block may contain any number of words. Words in a block must occupy consecutive core storage addresses, starting with a program determined first word address, and ending with a program determined last word address.

a. Buffer Mode

A buffer mode transfer, which occurs independent of main program control, is used to transfer data between core storage and the drum units. Before execution of a buffer mode transfer of data, the program must perform the following steps:

(1) Activate the channel to be used for the information transfer.

(2) Load the channel index register with the buffer control word. (The lower and upper halves of the buffer control word contain the beginning and ending addresses of the section of core storage involved in the transfer.)

(3) Send the proper Function Word to the Control Units.

Steps 1 and 2 above are accomplished with one of the Initiate Buffer Instructions; step 3 is performed by the Enter External Function instruction. See BASIC INPUT/OUTPUT INSTRUCTIONS (Section 5) for a more detailed discussion of these operations.

Data is then transferred between the central processor core storage and the selected drum without main program intervention. When a word is transferred to or from storage, 1 is added automatically to the lower half of the control word. The data transfer is terminated when:

(1) The central processor senses that the upper and lower halves of the buffer control word are equal.

(2) A Terminate function is executed by the central processor.

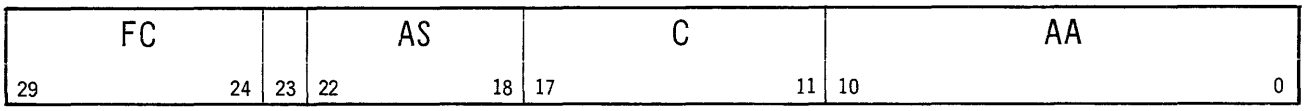
(3) An error is detected.

b. Word Arrangement

The Flying Head – 880 Magnetic Drum Subsystem accommodates four types of computer input/output words. They are the Function Word, Data Word, Identifier Word, Status Word, and End of Block Word.

(1) Function Word

The Function Word designates the operation to be performed by the Drum Unit and the address where the function is to begin. It is arranged as follows:



The six most significant bits of the Function Word are the function code (FC), bit positions 24 through 29. The function code specifies the actual operation to be performed by the Flying Head – 880 Drum Subsystem.

Bit positions 0 through 22 (bit position 23 is ignored) contain the drum address with AS as the angular section, C as the channel number, and AA as the angular address.

The function codes are as follows:

CODE	FUNCTION	DESCRIPTION
02	Write	Write data in consecutive drum addresses starting at the address specified by the Function Word. Stop when no more data is available from the central processor or when terminated by a Terminate Instruction or by an error.
42	Continuous Read	Read data from consecutive drum addresses starting at the address specified by the Function Word and transfer this data to the central processor. Stop when no more data is requested by the central processor, or when terminated by a Terminate Instruction or by an error.
52	Block Read	Read one block of data from consecutive drum addresses starting at the address specified by the Function Word and transfer this data to the central processor. The transfer is completed after the End of Block Word and one more word (overflow word), containing the End of Block status code and the five least significant characters of the overflow word, is transferred. Stop when transfer is completed or when terminated by a Terminate Instruction or by an error.

CODE	FUNCTION	DESCRIPTION
45	Search	After receiving the Identifier Word, read data from consecutive drum addresses starting at the address specified by the Function Word and compare each word read to the Identifier Word. When identical comparison is achieved, transfer the Search Find status code and the address of the "find" to the central processor. Stop when identical comparison is achieved or when terminated by a Terminate Instruction or by an error.
46	Search Read	After receiving the Identifier Word, read data from consecutive drum addresses starting at the address specified by the Function Word and compare each word read to the Identifier Word. When identical comparison is achieved, continue reading and transfer the data to the central processor starting with the Identifier Word. Stop when terminated by a Terminate Instruction or by an error.
55	Block Search	After receiving the Identifier Word, read data from consecutive drum addresses starting at the address specified by the Function Word and compare each word read to the Identifier Word. If identical comparison is achieved before an End of Block is read, transfer the Search Find status code along with the address of the "find", if the End of Block Word is read before the ""find" is made, transfer one more word (overflow word) containing the End of Block status code and the five least significant characters of the overflow word. Stop after the Search Find or End of Block status code has been transferred to the central processor or when terminated by a Terminate Instruction or by an error.
56	Block Search Read	After receiving the Identifier Word, read data from consecutive drum addresses starting at the address specified by the Function Word and compare each word read to the Identifier Word. If an End of Block Word is read before the "find" is made, transfer one more word containing the End of Block status code and the five least significant characters of the word to the central processor; if identical comparison is achieved before the End of Block Word is read, continue reading and starting with the Identifier Word transfer the data remaining in the block to the central processor. Stop when the End of Block status code has been transferred to the central processor or when terminated by a Terminate Instruction or by an error.
40	Bootstrap	Perform a Continuous Read from octal address 0.

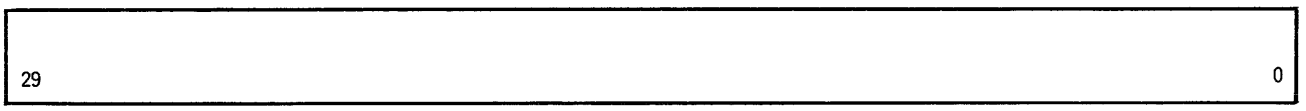
CODE	FUNCTION	DESCRIPTION
50	Bootstrap with Interrupt	Perform a Block Read from octal address 0.
23	Terminate	Terminate an input operation immediately. Terminate an output operation after the last Data Word has been recorded.
33	Terminate With Interrupt	Same as Terminate except the normal completion status code is sent to the central processor.

(2) Data Word

The data word (input or output) may be composed of 30 bits of binary information or it may be composed of five six-bit alphanumeric characters as shown below.

(3) Identifier Word

The Identifier Word is a computer output word that immediately follows a Search or Search-Read Function Word. It can be in any bit configuration, as shown below. It is transmitted to the subsystem accompanied by an External Function Signal. The characters of the Identifier Word are sequentially compared with the drum characters until the appropriate find is accomplished.



(4) Status Word

The Status Word is transmitted by the subsystem into the central processor whenever there is data error information or a function with interrupt has been completed. A Status Word is arranged as follows:

The status code, SC, for the Magnetic Drum Subsystem is contained in the most significant 6 bits of the Status Word. In some cases bit positions 0 through 22 contain a drum address. (See Function Word.) In other cases these bit positions are not used and are transmitted to the Computer as binary zeros. In some special cases, the status code may occupy the four most significant bits of the Status Word.

The status codes are as follows:

CODE	INDICATION	DESCRIPTION
14	Write Fault	This indication occurs if more than one read/write head is selected, if high or low temperature occurs in the unit, or if D.C. power to the unit is dropped.
30	Channel Synchronizer Character Count Error	This indication occurs whenever the character counter in the Channel Synchronizer reflects a character count less than that which is required to make up a computer word.
34	End of File	This indication occurs when the next sequential address is an illegal address or an address on an inoperable unit.
40	Normal Completion	This indication occurs when any function with interrupt has been completed.
50	Illegal Function	This indication occurs if the function word contains a function code that is not valid for the subsystem.
54	Illegal Address	This indication occurs if the function word contained an address that does not exist in the particular sub-system, an address on an inoperable unit, or a bootstrap address for a Write function when the bootstrap area is locked out.
60	Control Unit Sequence Error	This indication occurs when the character timing pulses are not in synchronism with the word mark timing.
64	Continuous Read Parity Error	This indication appears when a parity error has occurred during a Continuous Read function. The word containing the error is held in the Channel Synchronizer/Control Unit and an Input Data Request signal will be sent to the central processor following the acknowledgement of the interrupt and the transfer of data then stops. A Terminate function must be sent to the Channel Synchronizer Control Unit to restore it to a ready condition.
04	End of Block	This indication occurs when an End of Block word (a word that contains binary one's) has been read during a block function. The status word contains this code and the four least significant characters of the overflow word (the word immediately following the End of Block word).
05	Search Find	This indication occurs when the word specified in a Search function has been found. The low order 22 bits of the status word contain the address of the find word.

(5) End of Block Word

The End of Block Word is all binary ones and may be stored at any location on the drum except the last (highest) address in the system. In octal code, the End of Block Word contains 7777777777.

5. Subsystem Characteristics

a. Summary of General Characteristics

DATA BITS (per track)	30,720
DATA BIT CAPACITY (per drum)	23,592,960
DRUM LENGTH (inches)	30
DRUM DIAMETER (inches)	24
TRACKS PER INCH	33
DATA TRACKS PER DRUM	768
HEAD BLOCKS PER DRUM	40
HEADS PER BLOCK	22
DRUM SPEED (revolutions per minute)	1,800
WORDS PER DRUM	786,432
AVERAGE ACCESS TIME (milliseconds)	17
MAXIMUM ACCESS TIME (milliseconds)	34

b. Physical Characteristics

	FLYING HEAD – 880 DRUM CABINET	FLYING HEAD – 880 CONTROL CABINET
HEIGHT (inches) WIDTH (inches) DEPTH (inches) WEIGHT (approximate lbs.)	96 55 34 1,700	96 20 34 550
TEMPERATURE RANGE	60° – 80° F.	
HUMIDITY RANGE	40% – 70%	
HEAT DISSIPATION (BTU/hr.)	3,600	850
AIR FLOW (cu. ft./min.)	300	390
POWER REQUIREMENTS	208/220 VAC 60 cps 3 phase + neutral 11.5 KVA (start) 1.3 KVA (run)	208/220 VAC 60 cps 3 phase 1.5 KVA

MAXIMUM CABLE LENGTH RESTRICTIONS

1. Central processor to Drum Control Cabinet: 300 feet
2. Drum Control Cabinet to Drum: 60 feet

B. FASTRAND SUBSYSTEM

The FASTRAND Subsystem is used as a large capacity random access storage device in the UNIVAC 490 Real-Time System. Each FASTRAND unit contains two magnetic drums that are similar to those used in the Flying Head-880 Magnetic Drum Subsystem. These drums are similar in that they employ flying read/write heads; however, they have an additional feature, the ability to laterally position the read/write heads. In each FASTRAND unit, 64 read/write heads (32 per drum) are connected to a common positioning mechanism that moves these heads in unison. These read/write heads record information on the surface of a drum at a density of 1,000 bits per inch while it revolves at 880 revolutions per minute. Average access time is 92 milliseconds. There are 6,144 recording tracks in each unit (3,072 across each drum). Each track can store 2,112 (30-bit) computer words, allowing a total of 12,976,128 computer words or 64,880,640 alphanumeric characters to be recorded on each unit.

On each unit the recording tracks are divided into 96 positions with 64 tracks per position. Each track in a given position is under a separate read/write head and is divided into 64 sectors (areas that can hold 33 computer words). Since there are 64 tracks in a position, a position contains 4096 sectors. A given position begins with sector 0 of its first track, runs through sector 63 of that track, and continues through the sectors of each succeeding track to the end of the position (sector 63 of the last track).

All units in the subsystem are available to the central processor at any time. However, only one unit may be read from or written on at any one time on a single input/output channel. If an operation requires simultaneous reading from one unit and writing on a second unit, two subsystems each on a separate input/output channel must be employed. Up to eight FASTRAND units may be connected to each available input/output channel. A Channel Synchronizer/Control Unit is required to control all the units on a channel.

1. Addressing

Each of the 393,216 sectors in each FASTRAND unit has a unique address that is composed of four parts: Unit (3 bits), Position (8 bits), Track (6 bits), and Sector (6 bits).

- Unit

  Specifies the particular unit in the subsystem that is involved.

- Position

  Each unit in the subsystem is divided into 96 positions, each of which contains 64 recording tracks. These recording tracks are accessed by 64 read/write heads that are moved laterally from position to position.

- Track

  There are 6,144 recording tracks in each unit. Each track can store 2,112 computer words.

- Sector

  In each recording track there are 64 sectors. Each sector can store 33 computer words. Since there are 64 tracks in a position, a position contains 4,096 sectors.

■ Address Format

The basic storage element in a FASTRAND Subsystem is the sector. Sector addresses are continuous octal numbers within each unit.

A particular sector address is specified by 23 bit positions in a function word. For example:

The uppermost sector in a track is
$63_{10} = 77_8 = 111\ 111$ (binary)

The uppermost track in a position is
$63_{10} = 77_8 = 111\ 111$ (binary)

$95_{10} = 137_8 = 01\ 011\ 111$ (binary)

The highest numbered unit in a subsystem is
$7_{10} = 7_8 = 111$ (binary)

Combining these four individual bit patterns in sets of three for octal interpretation gives:

BIT POSITION	22 20	19 12	11 6	5 0
ADDRESS PORTION	UNIT	POSITION	POSITION RANGE: TRACK	POSITION RANGE: SECTOR
BINARY	1 1 1	0 1 0 1 1 1 1 1	1 1 1 1 1 1	1 1 1 1 1 1
OCTAL	3	5 3	7 7	7 7

This pattern represents the address of the uppermost storage element on unit 7 of the subsystem.

Figure B-1 is a simplified diagrammatic representation of the address structure of the FASTRAND subsystem. Although a position is shown as occupying a distinct portion of a unit, the tracks that comprise a position in reality are not adjacent to each other but are intermixed throughout the unit.

As a practical matter, however, the user is not concerned with the bit patterns in the addressing system because he need specify sector addresses in octal notation only. The range of addresses (in octal) for each unit in an eight unit subsystem is presented in the following table.

UNIT	BEGINNING ADDRESS	ENDING ADDRESS
0	0000 0000	0137 7777
1	0400 0000	0537 7777
2	1000 0000	1137 7777
3	1400 0000	1537 7777
4	2000 0000	2137 7777
5	2400 0000	2537 7777
6	3000 0000	3137 7777
7	3400 0000	3537 7777

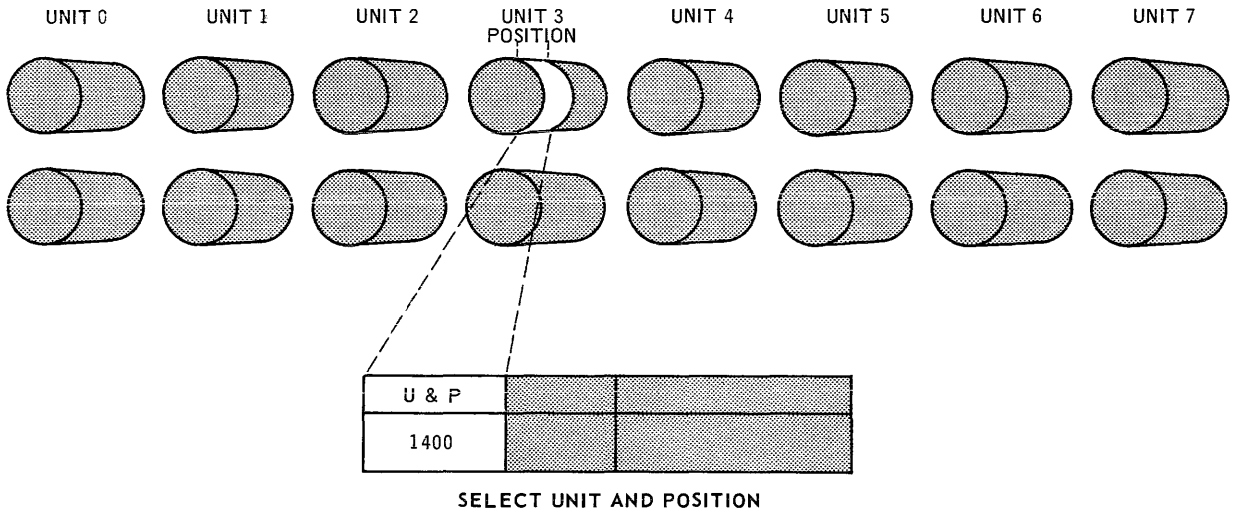


Figure B-1. Address Structure of FASTRAND Subsystem

It should be noted that a movement of the read/write heads is required each time the starting sector address is not accessible within the current position. In these cases, the head movement will add to the amount of time that is normally required for an operation.

Reading or writing in consecutive sectors begins at the sector address specified by the function word and proceeds to consecutive sectors, tracks, and positions. For instance, starting with the first sector address in a subsystem, sector 0 in track 0 of position 0, data is read or written, and continues through sector 63. Then sectors 0-63, in track 1 of this position are read or written in the same sequence. This process continues until sector 63 in track 63 of position 95 has been read or written.

2. Channel Synchronizer/Control Unit

The Channel Synchronizer Control Unit for the FASTRAND Subsystem controls the operation of the storage units during their write and read functions, transfers the data to be written from the central processor to the storage units, and transfers data that has been read to the central processor.

- Write Function

  During the write function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a write command to the selected unit. Then it receives data words from the central processor and transfers them to the storage unit for writing.

- Read Function

  During the read function, the Channel Synchronizer/Control Unit receives a function word and translates it into a read command to the selected storage unit. Then it receives data words from the storage unit and transfers them to the central processor.

- Address Selection

  The Channel Synchronizer/Control Unit decodes addresses contained in the function word and selects the requested address.

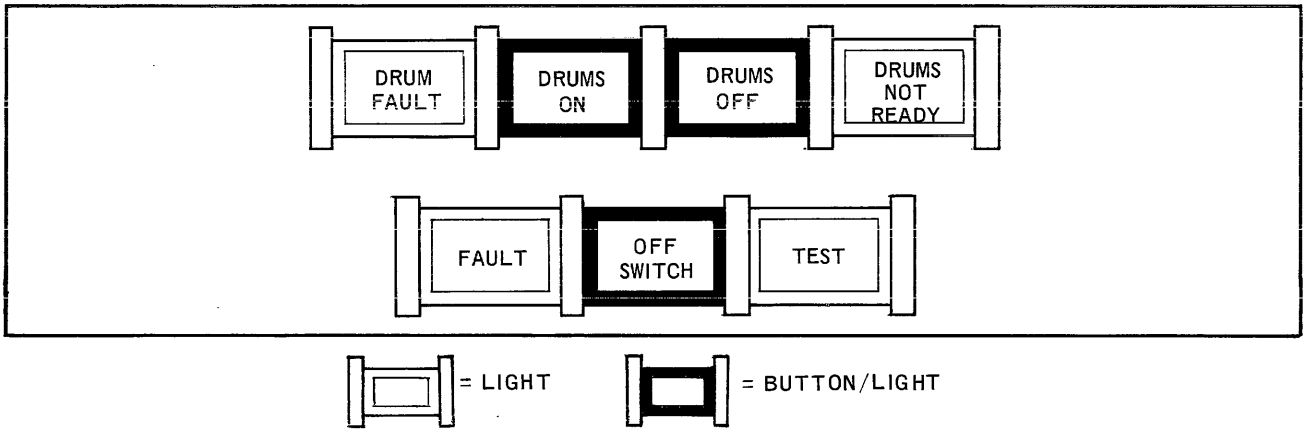
- Error Detection

  Various internal checks, such as parity and phase checks, are performed on all data that is transferred to or from the FASTRAND Subsystem.

3. Operator Controls

Primary control of the FASTRAND Subsystem is accomplished by the program. However, the Control Cabinet and the individual FASTRAND units have operator panels that show operating conditions in the subsystems and which allow some manual operations.

The Control Cabinet panel indicators and their functions are as follows:



DRUM FAULT — Lights when there is a fault in one of the FASTRAND units.

DRUMS ON — Pressing this button starts the FASTRAND units. Logical unit 0 starts first and, once it is up to speed, the sequence continues until all units are running. This indicator lights immediately; however, the DRUMS NOT READY indicator remains on until all units in the subsystem are up to speed. This button is also used to clear the DRUM FAULT indicator.

DRUMS OFF — Pressing this button removes power from all the FASTRAND units, and lights the indicator. Power to the Control Unit is not affected.

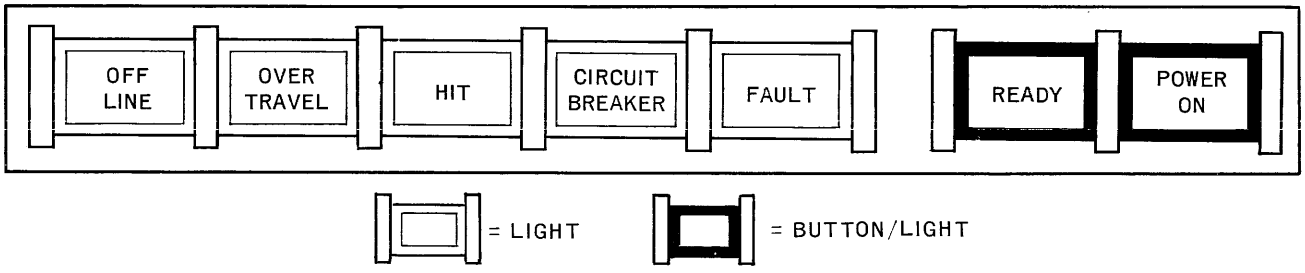
DRUMS NOT READY — Lights when one or more of the FASTRAND units in the subsystem is not ready for operation.

FAULT — Lights when there is a fault in one of the FASTRAND units or in the Control Unit.

OFF SWITCH — Depressing this button removes power from the Control Unit. Power on is indicated by green light; power off by red light.

TEST — Lights when a FASTRAND unit or the Control Unit is under test.

The FASTRAND Unit panel indicators and their functions are as follows:



OFF-LINE — Lights when one or more switches on the maintenance panel is in the non-normal position.

OVER TRAVEL — This indicator is lit when read/write head over travel (either right or left) has occurred.

HIT — This indicator is lit when one or more read/write heads hit the drum surface.

CIRCUIT BREAKER — This indicator is lit when one or more circuit breakers in the unit are open due to overload.

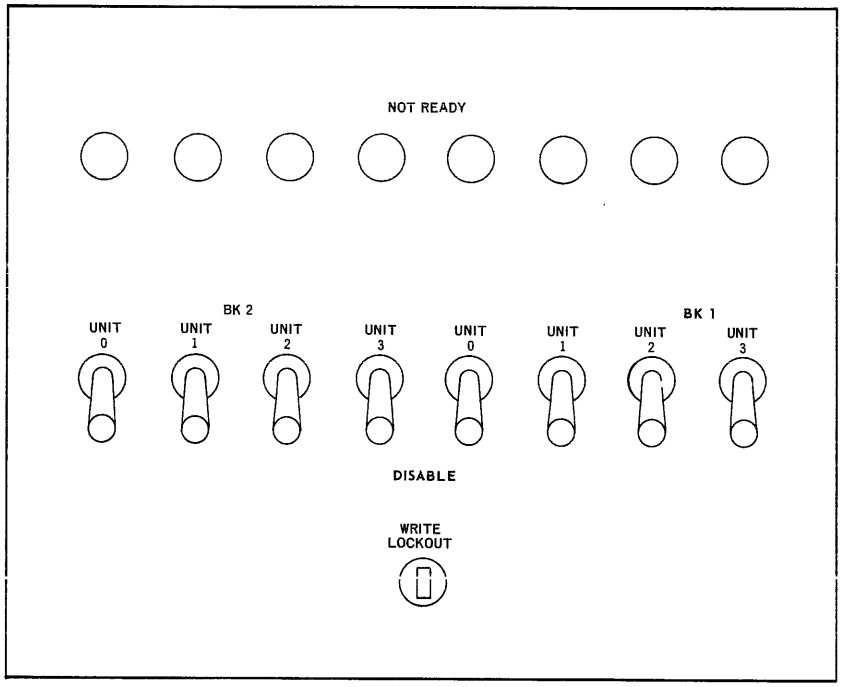
FAULT — This indicator is lit when any of the following conditions occur:

- Loss of cooling air supply due to failure of a blower or cooling compressor.
- DC power supply failure.

READY — This indicator is lit during normal operation. If the unit is not ready because a fault condition exists, the indicator is not lit. After the fault condition has been corrected, this button must be pressed to reset the fault sensing circuits.

POWER ON-OFF — Lights when power is on in the unit. Pressing this button applies power to the unit, provided that power is on in the Control Cabinet or the OFF-LINE indicator is lit.

The Control Cabinet also has an auxiliary operator's panel that is located behind its front door. An illustration of this panel and descriptions of the indicators and switches it contains are provided in the text that follows:



As shown in the illustration, this panel contains a toggle switch for each unit in the subsystem. If any of these switches are placed in the up position, the unit that is associated with it will be disconnected from the fault circuits, the Control Unit logic, and the indicators that appear on the other operator panels. In addition, this panel contains indicators that light when a particular unit is not ready and a key lock switch that can be used to prevent write functions from being executed.

4. Operations

Any part of the internal core storage of the central processor can be used as an input/output data buffer storage area, with the exception of the few special core storage locations that are reserved for the Incremental Clock and the Interrupt Words.

Information is transferred between the central processor and the FASTRAND Subsystem in the form of 30 - bit data words. These words are formed into blocks in core storage. A block may contain any number of words. Words in a block must occupy consecutive core storage addresses, starting with a program determined first word address, and ending with a program determined last word address.

a. Buffer Mode

A buffer mode transfer, which occurs independent of program control, is used to transfer data to and from core storage and the FASTRAND units. Before execution of a buffer mode transfer of data, the program must perform the following steps:

(1) Activate the channel to be used for the data transfer.

(2) Load the channel index register with the buffer control word. (The lower and upper halves of which contain the beginning and ending addresses of the section of core storage involved in the transfer.)

(3) Send the proper function word to the FASTRAND Control Unit.

Steps 1 and 2 are accomplished with one of the Initiate Buffer instructions; step 3 is performed by the Enter External Function instruction. See BASIC INPUT/ OUTPUT INSTRUCTIONS (Section 5) for a more detailed discussion of these operations.

Data is then transferred between the central processor core storage and the selected FASTRAND unit without main program intervention. When a word is transferred to or from core storage, 1 is added to the lower half of the buffer control word. The data transfer is terminated when:

(1) The central processor senses that the upper and lower halves of the buffer control word are equal.

(2) A terminate function is executed by the central processor.

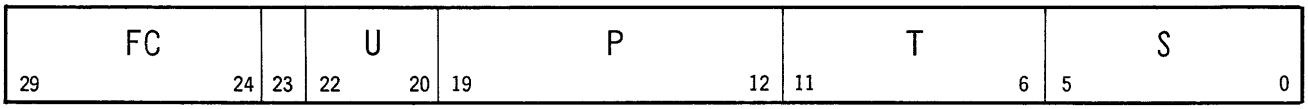
(3) An error is detected.

b. Word Arrangement

The FASTRAND Subsystem accommodates four types of computer input/output words. They are the Function Word, Data Word, Identifier Word, and Status Word.

(1) Function Word

The function word designates the operation to be performed by the FASTRAND unit and the address where the operation is to begin. It is arranged as follows:



The six most significant bits of the function word specify the function code, FC, bit positions 24 through 29. Bit positions 0 through 22 (bit position 23 is ignored) contain the sector address with U as the unit, P as the position, T as the track, and S as the sector.

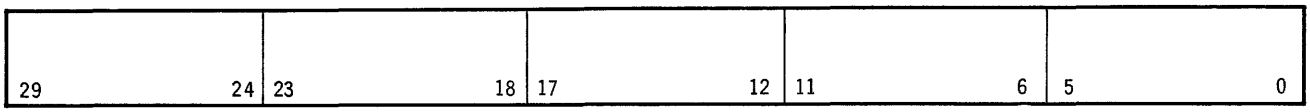
The function codes are as follows:

CODE	FUNCTION	DESCRIPTION
12	Write	Write data in consecutive sectors starting at the sector address specified in the function word. Data is transferred until the function is ended by a Terminate function, an error, a Time Out, or an End of Position.
52	Continuous Read	Read data from consecutive sectors starting with the sector address specified in the function word and transfer this data to the central processor. Data is transferred until the function is ended by a Terminate function, an error, a Time Out, or an End of Position.
20	Position	Move the read/write heads in the specified FASTRAND unit to the position address specified in the function word.
30	Position With Interrupt	Same as the Position function except that when the function is completed, a status word and an external interrupt signal are transmitted to the central processor.
56	Search All Words (long search)	Read each word of a sector, starting at the sector address specified in the function word, and compare it for equality with an identifier word that has been placed in the Channel Synchronizer by the program. If a find is made within a sector, the subsystem goes immediately to the Continuous Read mode. The find word, the balance of the sector, and the words in succeeding sectors are transferred to the central processor. This continues until the function is ended by a Terminate function, an error, a Time Out, or an End of Position.

CODE	FUNCTION	DESCRIPTION
		If a find is not made within a sector, the search continues through each succeeding sector (until the end of the position is reached) until a find is made or the function is ended by a Terminate function, an error, a Time Out, or an End of Position.
57	Search All Words (short search)	Same as Search All Words (long search) function except that a maximum of 64 sectors will be searched; that is, the search will be confined to one track within a position rather than the entire position.
54	Search First Word (long search)	Read the first word of a sector, starting at the sector address specified in the function word, and compare it for equality with an identifier word that has been placed in the Channel Synchronizer by the program. If a find is made within a sector, the subsystem goes immediately to the Continuous Read mode. The find word, the balance of the sector, and the words in succeeding sectors are transferred to the central processor. This continues until the function is ended by a Terminate function, an error, a Time Out, or an End of File.
55	Search First Word (short search)	Same as Search First Word (long search) function except that a maximum of 64 sectors will be searched; that is, the search will be confined to one track within a position rather than the entire position.
53	Data Recovery Read	Read data from a sector, whose address is specified in the function word, and transfer this data to the central processor. When the end of the sector is reached, the transfer is stopped and a status word containing the Data Recovery Status Code will be sent to the central processor provided that a Phase Check error, a non-recoverable error, or a Time Out has not occurred.
23	Terminate	If the subsystem is in the input mode, the operation is stopped immediately, registers are cleared, and an Output Data Request signal is sent to the central processor. If the subsystem is in the output mode, the operation is stopped after the current sector has been written.
33	Terminate With Interrupt	Same as Terminate Without Interrupt function except that a status word and an external interrupt signal are transmitted to the central processor immediately following the termination.

(2) Data Word

The data word (input or output) may be composed of 30 bits of binary information or it may be composed of five six-bit alphanumeric characters as shown below.



(3) Identifier Word

The identifier word is a computer output word that immediately follows a search function word. It can be in any bit configuration, as shown below. When the identifier is transferred to the FASTRAND Subsystem it is accompanied by an external function signal.

The characters of the identifier word are compared sequentially with the characters in words that are read from consecutive addresses, starting with the address specified in the function word. When identical comparison is made, further operation is determined by the type of search that was ordered.

(4) Status Word

The status word is transmitted by the FASTRAND Subsystem to the central processor whenever there is data error information or a function with interrupt has been completed. A status word is arranged as follows:

The status code, SC, for the FASTRAND Subsystem is contained in the most significant six bit positions of the status word, bit positions 24 through 29. In some cases bit positions 0 through 22 (bit position 23 is ignored) contain an error address. In other cases these bit positions are not used.

The status codes are as follows:

CODE	INDICATION	DESCRIPTION
04	Time Out (Input)	This indication occurs if the central processor does not accept data at the rate required by the subsystem.

CODE	INDICATION	DESCRIPTION
05	End Of Position (Input)	This indication occurs whenever a change in the position of the read/write heads is required to continue the transfer of data. It will also occur if a find is not made during a search operation; it indicates the end of the track has been reached.
06	Phase Error	This indication occurs when the phase monitoring circuits detect an incorrect phase condition.
07	Non-Recoverable Error	This indication will appear during a read function when two or more phase errors have occurred in a sector or, an error in the longitudinal parity is detected but no phase errors were detected in the sector.
10	Normal Ending (data recovery)	This indication will appear after a Data Recovery function has been performed provided that a Time Out, a Phase Error, or a Non-Recoverable error has not occurred.
14	Address Error	Indicates that the required sector address cannot be found.
20	Channel Synchronizer Sequence Error	This indication occurs during a write function when the central processor has not kept up with the transfer rate.
24	End Of Position (Output)	This indication occurs whenever a change in the position of the read/write heads is required to continue the transfer of data.
34	Sector Length Error	This indication will occur whenever the control unit attempts to read from or write into a sector more than 33 computer words.
40	Normal Ending (Write)	This indication appears when a Write or Position function has been completed.
50	Invalid Function Code	This indication occurs if the function word contains a function code that is not valid for the subsystem.
54	Invalid Address	This indication occurs if the Control Unit receives an address that does not exist in the particular subsystem.
60	Write Error	This indication appears when a malfunction occurs during the writing of data.
74	Non-Operable Condition	This indication occurs if the subsystem is not ready for operation.

5. Subsystem Characteristics

a. Summary of General Characteristics

DATA BITS (per track)	63,360
DATA BIT CAPACITY (per unit)	389,283,840
DRUM LENGTH (inches)	61.2
DRUM DIAMETER (inches)	24
TRACKS PER INCH	53
DATA TRACKS PER UNIT	6,144
NUMBER OF READ/WRITE HEADS PER UNIT (moveable)	64
DRUM SPEED (revolutions per minute)	880
WORDS PER UNIT	12,976,128
AVERAGE ACCESS TIME (milliseconds)	92
MAXIMUM ACCESS TIME (milliseconds)	156

b. Physical Characteristics

	FASTRAND UNIT	FASTRAND CONTROL CABINET
HEIGHT (INCHES)	63.38	96
WIDTH (INCHES)	121.5	20
DEPTH (INCHES)	32.5	34.5
WEIGHT (APPROX. LBS.)	5,450	700
TEMPERATURE RANGE	60° – 80° F.	
HUMIDITY RANGE	40% – 70%	
HEAT DISSIPATION (BTU/HR.)	19,454	2,600
AIR FLOW (CU. FT./MIN.)	1,000	400
POWER REQUIREMENTS	208/220 VAC 60 cps 3 phase 12.6 KVA (start), 35 amps per phase 7.0 KVA (run), 20 amps per phase	208/220 VAC 60 cps 3 phase 1.5 KVA

Maximum Cable Length Restrictions

1. Control Cabinet to first FASTRAND Unit: 50 ft.
2. Control Cabinet to all other FASTRAND Units: 75 ft.
3. FASTRAND unit to FASTRAND unit: 30 ft.

C. UNISERVO IIA MAGNETIC TAPE SUBSYSTEM

The UNISERVO IIA Magnetic Tape Subsystem is an integral part of the UNIVAC 490 Real-Time System. It consists of a Channel Synchronizer/Control Unit and from two to twelve UNISERVO IIA Tape Units that are connected to the central processor via an input/output channel.

A subsystem that is on a particular input/output channel is capable of reading or writing data in the form of 30-bit binary words (five six-bit characters per word) on any one of the tape units at any one time. If simultaneous reading and writing is required, two subsystems on separate input/output channels will be required. Data can be read or written at densities of 125 or 250 characters per inch. In addition, a subsystem can also read data that has been written by a UNITYPER* at a density of 50 characters per inch.

The UNISERVO IIA Tape Unit utilizes 0.5 inch wide 1500 foot reels of metallic tape or 2400 foot reels of plastic tape.

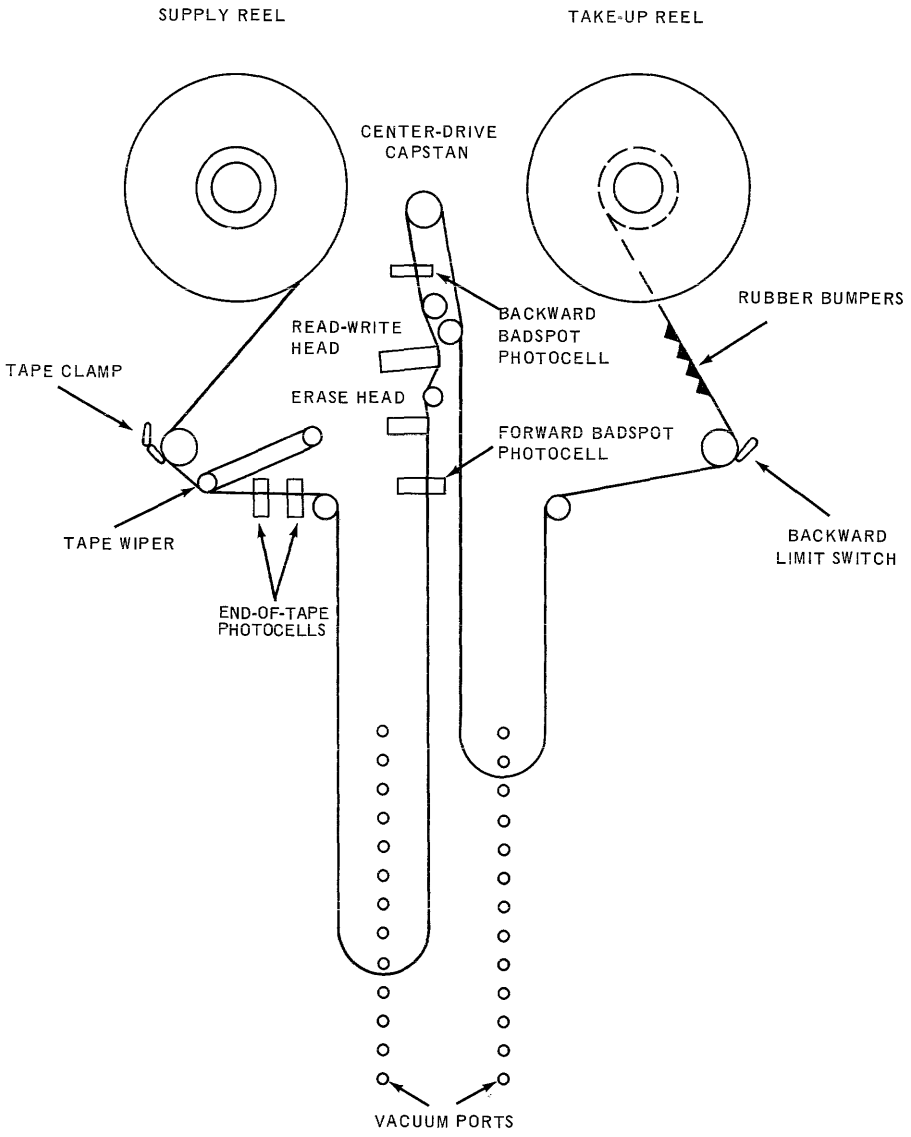


Figure C-1. UNISERVO IIA Tape Unit Schematic

* *Trademark of Sperry Rand Corporation*

As shown in Figure C–1, the reel of tape that contains data to be processed or the blank reel upon which data is to be written is mounted on the left. The right hand reel is used to store the tape as it is read or written on. The tape is threaded around and through guide rollers which control its path between the supply reel and take-up reel. A pre-threaded leader is permanently attached to the take-up reel. When it becomes necessary to mount a new supply reel, the end of the tape on the supply reel is attached to this leader, thus eliminating the need for manual threading.

The read/write head performs the actual reading or writing of data as the tape passes beneath it at the rate of 100 inches per second. Since the tape density may be 50, 125, or 250 characters per inch, information will be transferred at the rate of 5,000 (reading only), 12,500, or 25,000 characters per second.

The erase head is used only during a write operation. It erases information on the tape before new information is written. Accidental erasure of a previously recorded tape can be prevented by the insertion of a metal ring into the reel.

The vacuum columns, located below the tape reels are used to store the tape before it passes beneath the read/write head or is wound on the take-up reel. These columns also insure that the tape tension is even and that there is the proper amount of slack to permit rapid acceleration and deceleration.

The subsystem can detect premarked bad spots on tape where data cannot be written. Tapes are pretested by maintenance personnel, and any bad spots are marked so that they are recognizable. During processing the two bad spot photocells on the tape unit continuously monitor the tape for bad spots. When a bad spot is detected, the read or write operation that is in progress is suspended until the bad spot has passed the read/write head.

1. Tape Format

   Data appears on tape in the form of frames that are written across its width in eight channels or bit positions (6 data, 1 parity, and 1 sprocket). Five frames represent one 30-bit computer word. These words are written in groups called blocks (one or more words). Figure C–2 provides an example of how data appears on tape.

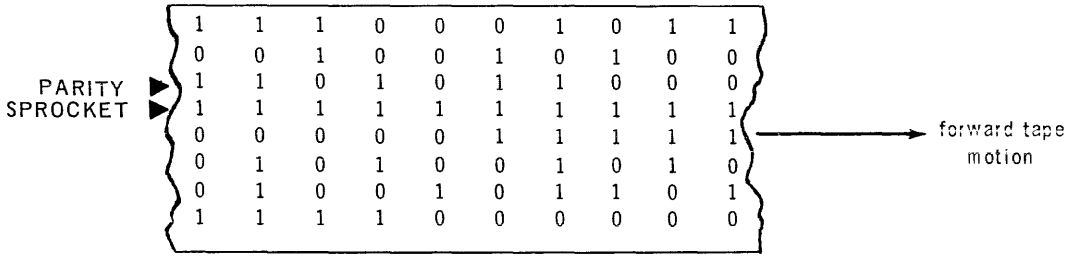


Figure C-2. UNISERVO IIA Tape Format

As shown in the diagram, a sprocket and a parity channel are provided on the tape.

The sprocket channel is used to generate interval timing signals during reading.

The parity channel helps assure that every character that is written will be read accurately. This is accomplished by having the subsystem generate a 1 bit in the parity channel of every frame that is written which does not contain an odd number of 1 bits. Therefore, every correctly written frame will contain an odd number of 1 bits. When data is read, the parity status of each frame is checked before the information is transferred to the central processor. If an improper parity condition is found, this fact is stored by the subsystem until all of the current block of data has been read. When entire block has been read, the subsystem will notify the central processor that an improper parity condition exists.

In addition to being able to read or write data at high or low densities, the UNISERVO IIA Magnetic Tape Subsystem can also read or write variable or fixed length blocks of data.

In the case of variable length blocks, the only requirement is that a block must be at least one computer word.* The spacing between adjacent blocks is 1.05 inches and the end of file space is at least 4.5 inches. Fixed block length can be used to insure tape compatibility with other UNIVAC Systems.

If data is written in fixed length blocks at high density, the data blocks must be 720 characters (144 computer words) in length. The spacing between adjacent blocks is the same as in the variable block length situation.

If data is written in fixed length blocks at low density, the data blocks must be 720 characters (144 computer words) in length. When a block is written on tape, it is subdivided into six 120 character (24 computer words) blockettes that are spaced 1.05 inches apart. The spacing between adjacent blocks is 2.4 inches.

The tape spacing for variable and fixed length blocks is shown in Figure C–3.

2. Channel Synchronizer/Control Unit

The Channel Synchronizer/Control Unit for the UNISERVO IIA Magnetic Tape Subsystem controls the operation of the tape units during their write and read functions, transfers the data to be written from the central processor to the tape units, and transfers the data which has been read from the tape units to the central processor.

- Write Function

  During the write function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a write command to the selected tape units. Then it receives data words from the central processor and transfers them to the tape unit for writing.

- Read Function

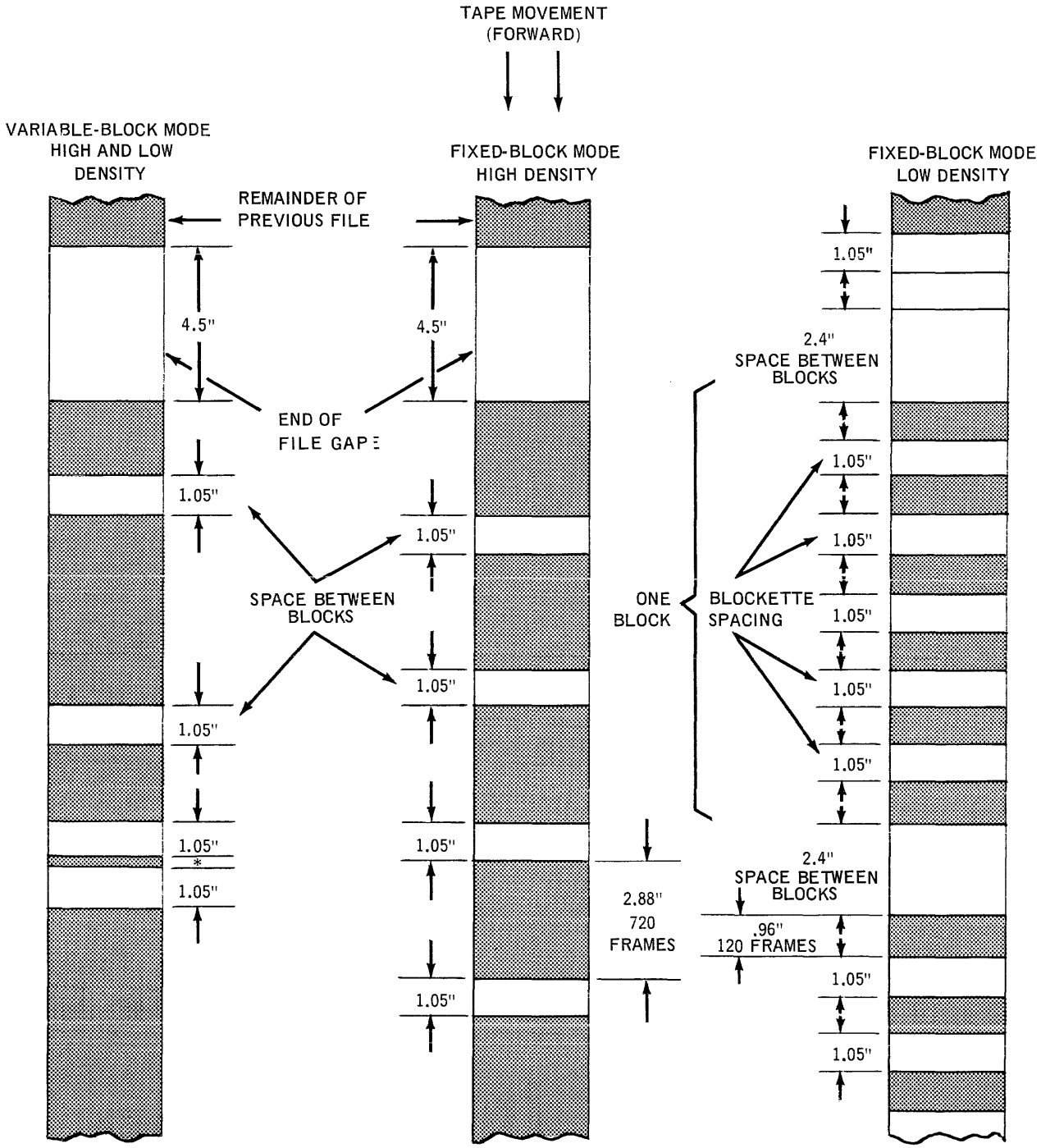
  During the read function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a read command to the selected tape unit. Then it receives data words from the tape unit and transfers them to the central processor.

- Unit Selection

  The Channel Synchronizer/Control Unit decodes the function word and selects the specified tape unit.

---

* *It should be noted that if the space occupied by a data block is less than the space between blocks, it may be difficult to recover from tape errors.*



* MINIMUM BLOCK LENGTH (5 FRAMES)

.02" AT 250 CHARACTERS PER INCH (HIGH DENSITY)

.04" AT 125 CHARACTERS PER INCH (LOW DENSITY)

Figure C-3. UNISERVO IIA Tape Spacing

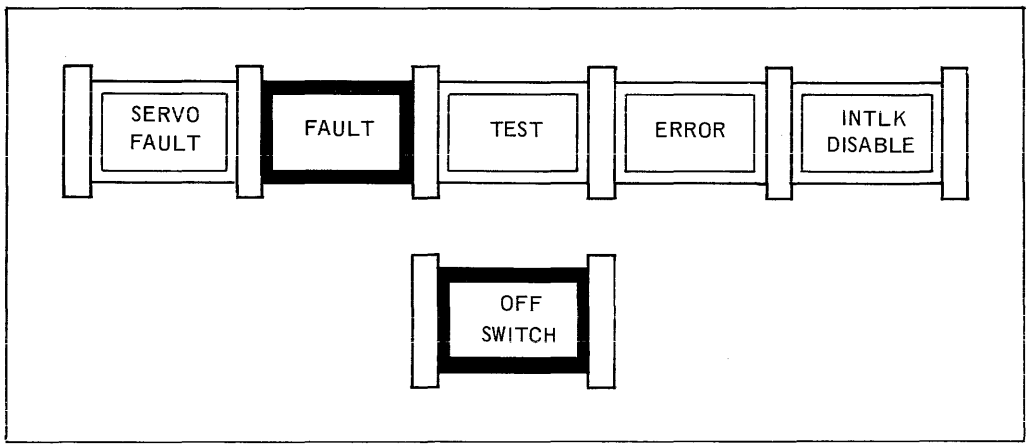
- Error Detection

  Detection circuits in the Channel Synchronizer Control Unit check the data characters as they are transferred to or from the central processor. These circuits also check the data words to see if they contain the proper number of characters.

3. Operator Controls

Primary control of the UNISERVO IIA Magnetic Tape Subsystem is accomplished by the program. However, the Control Cabinet and the individual tape units have operator panels that show operating conditions in the subsystem and which allow some manual operations.

The Control Cabinet panel indicators and their functions are as follows:



LEGEND:

SERVO FAULT — Lights when there is an open fuse or open circuit breaker in one of the tape units in the subsystem.

FAULT — Lights when any of the following conditions occur:

- Under voltage in the Control Unit.
- Overcurrent in the Control Unit.
- An interlock fault in the Control Unit.

If the fault has been corrected, the indicator light may be extinguished by pressing this button.

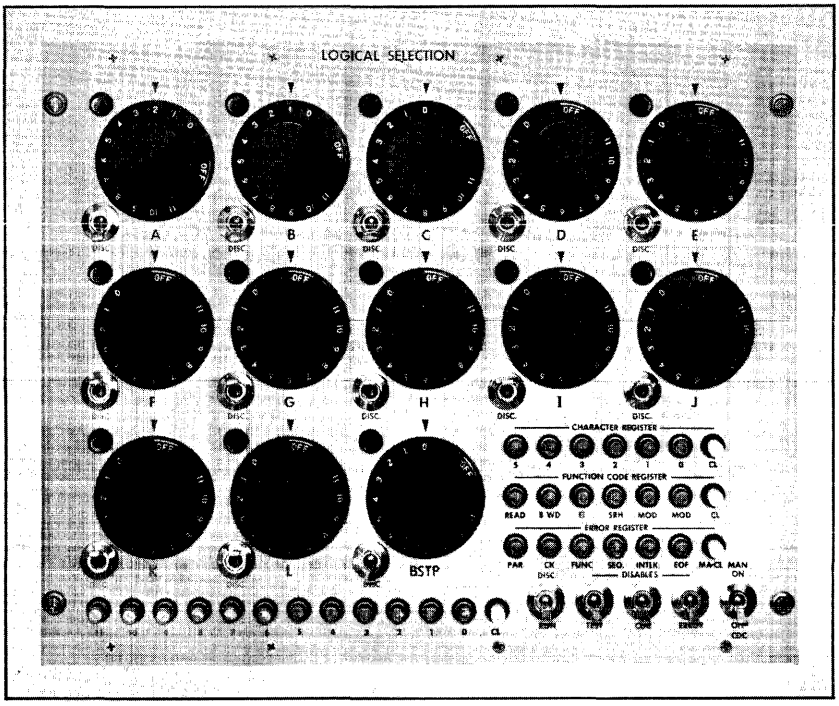
TEST — Lights when the subsystem is isolated for test purposes.

ERROR — Lights when two or more tape units have the same logical number.

INTLK DISABLE — Lights when the Interlock Disable Switch on the maintenance panel is in the up position.

OFF SWITCH — When all power is on in the Control Cabinet this button is green. When power is off, this button is red. Power in the Control Cabinet can be turned off by pressing this button; however, once the power has been turned off, this button cannot be used to turn the power on again. Power can be turned on only by resetting a circuit breaker in the rear of the cabinet.

The Control Cabinet also has an auxiliary operator panel that is located behind its front door. This panel is used to assign logical numbers to the tape units in the subsystem. An illustration of this panel and descriptions of the switches and indicators that it contains are provided in the text that follows:

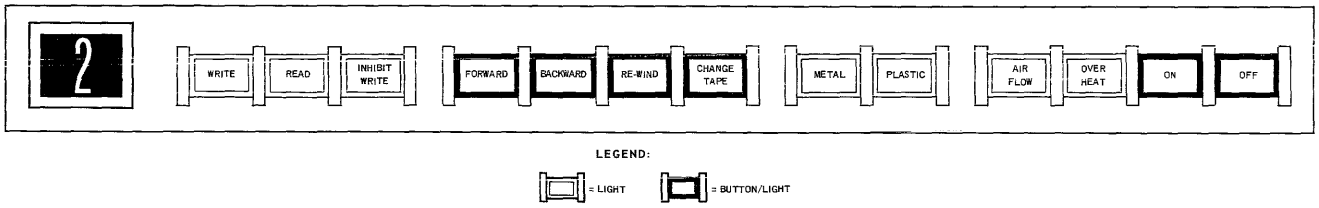


As shown in the illustration, this panel contains 13 rotary switches. Each of these switches has 12 positions that represent the logical numbers that can be assigned to the tape units and an OFF position. The switches labeled A through L are used to assign logical numbers to the respective physical tape units. The switch labeled BSTP is used to designate the logical number of the tape unit that will be used for Bootstrap* operation. A disconnect switch, labeled DISC, is provided for each of the rotary switches. These disconnect switches, located below and to the left of the rotary switches, enable (down position) or disconnect (up position) the tape units and the Bootstrap operation. When a rotary switch is disconnected (the disconnect switch for that particular rotary switch is in the up position), an indicator (located above and to the left of the rotary switch) will be lit.

In addition to the rotary switches, this panel also contains visual displays of the Character Register, the Function Code Register, and the Unit Select Register.

** To facilitate the initial loading of a program into core storage, a Bootstrap operation is provided in the central processor. This is a small program which, when activated, will bring some other program on magnetic tape into core storage. Once the program to be executed is loaded, the Bootstrap program is no longer required and may be removed from core storage. The Bootstrap program is activated or deactivated by a manual switch that is connected to the Central processor.*

The Tape Unit panel indicators and their functions are as follows:



UNIT NUMBER	Displays the physical number of the individual unit.
WRITE	Lights when a Write function is initiated.
READ	Lights when a Read function is initiated.
INHIBIT WRITE	Lights if a reel of tape that contains an inhibit write ring has been mounted on the unit.
FORWARD	Lights when the tape unit is conditioned for forward motion, or when the tape is moving in a forward direction. Pressing this button conditions the unit for forward motion.
BACKWARD	Lights when the tape unit is conditioned for backward motion, or when the tape is moving in a backward direction. Pressing this button conditions the unit for backward motion.
REWIND	Lights when the tape is being rewound. Pressing this button initiates the rewind operation if the unit has been conditioned for backward tape motion.
CHANGE TAPE	Lights when a tape is rewinding with interlock, or has been rewound with interlock. Pressing this button stops tape movement during any rewind.
METAL	Lights when the metal-plastic switch (located below the tape transport mechanism) is set for metallic tape.
PLASTIC	Lights when the metal-plastic switch is set for plastic tape.
AIR FLOW	Lights when the unit has shut off due to an insufficient air flow in the cabinet.
OVERHEAT	Lights when power has been turned off because the cabinet temperature has exceeded allowable limits.
ON	Lights when the unit is on. Pressing this button turns the unit on.
OFF	Lights when the unit is off. Pressing this button turns the unit off.

4. Operations

Any part of the central processor's internal core storage can be used as an input/output data buffer storage area with the exception of the few special core storage locations that are reserved for the Incremental Clock and Interrupt Words. Information is transferred between the central processor and the UNISERVO IIA Magnetic Tape Subsystem in the form of 30-bit data data words. These words are formed into blocks in core storage. A block may contain any number of words. Words in a block must occupy consecutive core storage addresses, starting with a program determined first word address and ending with a program determined last word address.

a. Buffer Mode

A buffer mode transfer, which occurs independent of main program control, is used to transfer data between core storage and the UNISERVO IIA Tape Units. Before execution of a buffer mode transfer of data, the program must perform the following steps:

(1) Activate the channel to be used for the information transfer.

(2) Load the channel index register with the buffer control word. (The lower and upper halves of the buffer control word contain the beginning and ending addresses of the section of core storage involved in the transfer.)

(3) Send the proper Function Word to the UNISERVO IIA Magnetic Tape Control Unit.

Steps 1 and 2 above are accomplished with one of the Initiate Buffer instructions; step 3 is performed by the Enter External Function instruction. See BASIC INPUT/OUTPUT INSTRUCTIONS (Section 5) for a more detailed discussion of these operations.

Data is then transferred between the central processor core storage and the selected UNISERVO IIA Tape Unit without main program intervention. When a word is transferred to or from storage, 1 is added automatically to the lower half of the buffer control word. The data transfer is terminated when:

(1) The central processor senses that the upper and lower halves of the buffer control word are equal.

(2) A Terminate instruction is executed by the central processor.

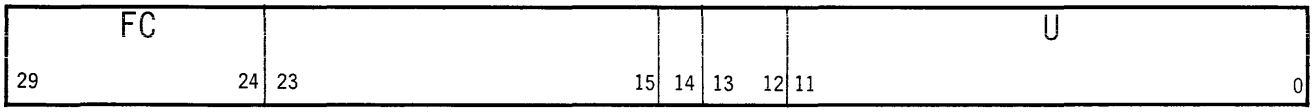
(3) An error is detected.

b. Word Arrangement

The UNISERVO IIA Magnetic Tape Subsystem accommodates four types of computer input/output words. They are the Function Word, Data Word, Identifier Word, and Status Word. These words are explained and illustrated in the following paragraphs:

(1) Function Word

The Function Word designates the operation to be performed by the UNISERVO IIA Magnetic Tape Subsystem. It is arranged as follows:



The six most significant bits of the Function Word are the function code, FC, bit positions 24 through 29. The function code specifies the actual operation to be performed by the subsystem.

Bit position 14 designates variable or fixed block length; that is, a 1 designates fixed block length, and a 0 designates variable block length.

Bit positions 0 through 11 specify the tape unit, U, that will be selected.

The function codes are as follows:

CODE	FUNCTION	DESCRIPTION
01	Write Low Density	Write one block on tape at a density of 125 characters per inch.
11	Write Low Density With Interrupt	Same as Write Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
02	Write High Density	Write one block on tape at a density of 250 characters per inch.
12	Write High Density With Interrupt	Same as Write High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
41	Read Forward Low Gain	Read one block from tape in a forward direction at low gain.
51	Read Forward Low Gain With Interrupt	Same as Read Forward Low Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
42	Read Forward Normal Gain	Read one block from tape in a forward direction at normal gain.
52	Read Forward Normal Gain With Interrupt	Same as Read Forward Normal Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
43	Read Forward High Gain	Read one block from tape in a forward direction at high gain.

CODE	FUNCTION	DESCRIPTION
53	Read Forward High Gain With Interrupt	Same as Read Forward High Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
61	Read Backward Low Gain	Read one block from tape in a backward direction at low gain.
71	Read Backward Low Gain With Interrupt	Same as Read Backward Low Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
62	Read Backward Normal Gain	Read one block from tape in a backward direction at normal gain.
72	Read Backward Normal Gain With Interrupt	Same as Read Backward Normal Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
63	Read Backward High Gain	Read one block from tape in a backward direction at high gain.
73	Read Backward High Gain With Interrupt	Same as Read Backward High Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
45	Search Read Forward Low Gain	Read the tape in a forward direction at low gain; compare the first word of each block with the identifier word; when equal comparison is made, read that block.
55	Search Read Forward Low Gain With Interrupt	Same as Search Read Forward Low Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
46	Search Read Forward Normal Gain	Read the tape in a forward direction at normal gain; compare the first word of each block with the identifier word; when equal comparison is made, read that block.
56	Search Read Forward Normal Gain With Interrupt	Same as Search Read Forward Normal Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

CODE	FUNCTION	DESCRIPTION
47	Search Read Forward High Gain	Read the tape in a forward direction at high gain; compare the first word of each block with the identifier word; when an equal comparison is made, read that block.
57	Search Read Forward High Gain With Interrupt	Same as Search Read Forward High Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
65	Search Read Backward Low Gain	Read the tape in a backward direction at low gain; compare the last word in each block with the identifier word; when an equal comparison is made, read that block.
75	Search Read Backward Low Gain With Interrupt	Same as Search Read Backward Low Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
66	Search Read Backward Normal Gain	Read the tape in a backward direction at normal gain; compare the last word of each block with the identifier word; when an equal comparison is made, read that block.
76	Search Read Backward Normal Gain With Interrupt	Same as Search Read Backward Normal Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
67	Search Read Backward High Gain	Read the tape in a backward direction at high gain; compare the last word of each block with the identifier word; when an equal comparison is made, read that block.
77	Search Read Backward High Gain With Interrupt	Same as Search Read Backward High Gain except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
20	Rewind	Rewind the tape on the specified unit.
30	Rewind With Interrupt	Same as Rewind except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
21	Rewind With Interlock	Rewind the tape on the specified unit with interlock.

CODE	FUNCTION	DESCRIPTION
31	Rewind With Interlock and Interrupt	Same as Rewind With Interlock except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
23	Terminate	Terminate after completion of the present function (at the end of the current block during a search function).
33	Terminate With Interrupt	Same as Terminate except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
40	Bootstrap	Rewind the tape on the tape unit specified for Bootstrap operation; read one block from tape in a forward direction.
50	Bootstrap With Interrupt	Same as Bootstrap except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

(2) Data Word

The data word (input or output) may be composed of 30 bits of binary information or it may be composed of five six bit alphanumeric characters as shown below:

(3) Identifier Word

The Identifier Word is a computer output word that immediately follows a Search Read Function word. It can be in any bit configuration, as shown below. It is transmitted to the subsystem accompanied by an External Function signal. The characters of the Identifier are sequentially compared with characters read from the first word of each block on tape until the appropriate find is accomplished. When this occurs, the block containing the identical word is read from tape.

(4) Status Word

The status word is transmitted by subsystem to the central processor whenever an abnormal condition occurs, or an operation with interrupt has been completed. A status word is arranged as follows:

The status code, SC, is contained in bit positions 24 through 29. All other bit positions are not used. The status word is accompanied by an external interrupt signal when it is transmitted to the central processor.

The status codes are as follows:

CODE	INDICATION	DESCRIPTION
10	Tape Unit Rewinding (without Interlock)	This indication occurs when the central processor requests a tape unit that is in the process of rewinding without interlock.
20	Channel Synchronizer Sequence Error	This indication occurs if the Channel Synchronizer/Control Unit receives a Normal Completion signal from the Control Unit and the Channel Synchronizer/Control Unit frame counter is not at the proper count.
24	Control Unit Sequence Error	This indication occurs if the following conditions exist during variable block operation: end of block, frame count of 2, 3, or 4; and more data detected within 15 milliseconds after the end of block. During fixed block operation, this indication occurs when the block just read or searched contains 727 or more frames.
30	Channel Synchronizer Counter Error	This indication occurs when the Channel Synchronizer/Control Unit character counter does not contain the proper count.
34	End of File	This indication occurs when no data is detected within the allowable distance after a data block.
40	Normal Completion	This indication occurs when a function with interrupt has been successfully completed.
50	Illegal Function	This indication occurs if the function word contains a function code that is not valid for the subsystem.

CODE	INDICATION	DESCRIPTION
54	Illegal Unit Address	This indication occurs when a tape unit was not specified in the function word, or when more than one unit was specified in the function word.
60	Parity Error	This indication appears at the completion of a read function when a parity error has occurred in one or more frames in the block that has been read. This indication also appears during search read functions after a block that contains a parity error has been searched.
70	Frame Count Error	This indication occurs at the end of the block during variable block operation, if the number of frames transferred to or from tape is not a multiple of five (five frames constitute one computer word) and the character counter does not contain the proper count. This indication also occurs during fixed block operation when an erased tape gap after 709 to 719 or 721 to 726 frames have been counted.
44	End of Tape	This indication occurs at the end of the block when the clear leader at the end of the tape is detected during a write function.
74	Interlock	This indication occurs when: (1) the central processor requests a non-existent tape unit (2) a write function is requested on a tape unit in which writing has been inhibited by an inhibit write ring (3) an interlocked tape unit is requested (a door is open on the unit, the unit is rewinding with interlock, and so on.)

5. Subsystem Characteristics

a. Summary of General Characteristics

TRANSFER RATE	12,500 and 25,000 characters per second.
RECORDING DENSITY	125 and 250 6 bit characters per inch.
TAPE SPEED	100 inches per second
TAPE WIDTH	0.5 inch
TAPE LENGTH	1,500 feet (metallic) 2,400 feet (plastic)
TAPE THICKNESS	1 mil (metallic) 1.5 mil (plastic)
BLOCK LENGTH	Variable
SPACE BETWEEN BLOCKS	1.05 inches
CHANNELS ON TAPE	8 channels 6 data 1 parity 1 sprocket
READ/WRITE OPERATION	Reading in forward and backward directions. Writing in the forward direction only.

b. Physical Characters

	UNISERVO IIA TAPE UNIT	POWER SUPPLY	UNISERVO IIA CONTROL CABINET
HEIGHT (inches)	69	96	96
WIDTH (inches)	30.875	66	20
DEPTH (inches)	30.5	32.75	34.5
WEIGHT (approximate lbs.)	908	2800	625
TEMPERATURE RANGE	60° – 80° F.		
HUMIDITY RANGE	40% – 70%		
HEAT DISSIPATION (BTU/hr.)	7,200	10,200	2,060
AIR FLOW (cu. ft./min.)	300	2,300	390
POWER REQUIREMENTS	Supplied by power supply	208 VAC 60 cps 3 phase 47.3 kw	208 VAC 400 cps 3 phase 0.6 kw 60 cps .35 KVA unreg.

Maximum Cable Length Restrictions

1. Control Cabinet to Central Processor: 300 ft.
2. Control Cabinet to Power Supply: 75 ft.
3. Control Cabinet to first UNISERVO IIA Tape Unit: 150 ft.
4. First UNISERVO IIA Tape Unit to Power Supply: 25 ft.

D. UNISERVO IIIC MAGNETIC TAPE SUBSYSTEM

The UNISERVO IIIC Magnetic Tape Subsystem differs from the UNISERVO IIA Magnetic Tape Subsystem and the UNISERVO IIIA Magnetic Tape Subsystem in that it can read magnetic tapes that were written by unrelated data processing equipment.

A UNISERVO IIIC Magnetic Tape Subsystem consists of a Channel Synchronizer/Control Unit, a Tape Adapter, and from two to twelve UNISERVO IIIC Tape Units that are connected to the central processor via an input/output channel.

A subsystem that is on a particular input/output channel is capable of reading or writing data in the form of 30-bit binary words, on any one of the tape units at any one time. Data can be read or written at densities of 200 or 556 characters per inch in either binary or binary coded decimal format. If simultaneous reading and writing is required, two subsystems on separate input/output channels will be required.

The UNISERVO IIIC Tape Unit utilizes 0.5 inch wide 2400 foot reels of plastic tape.

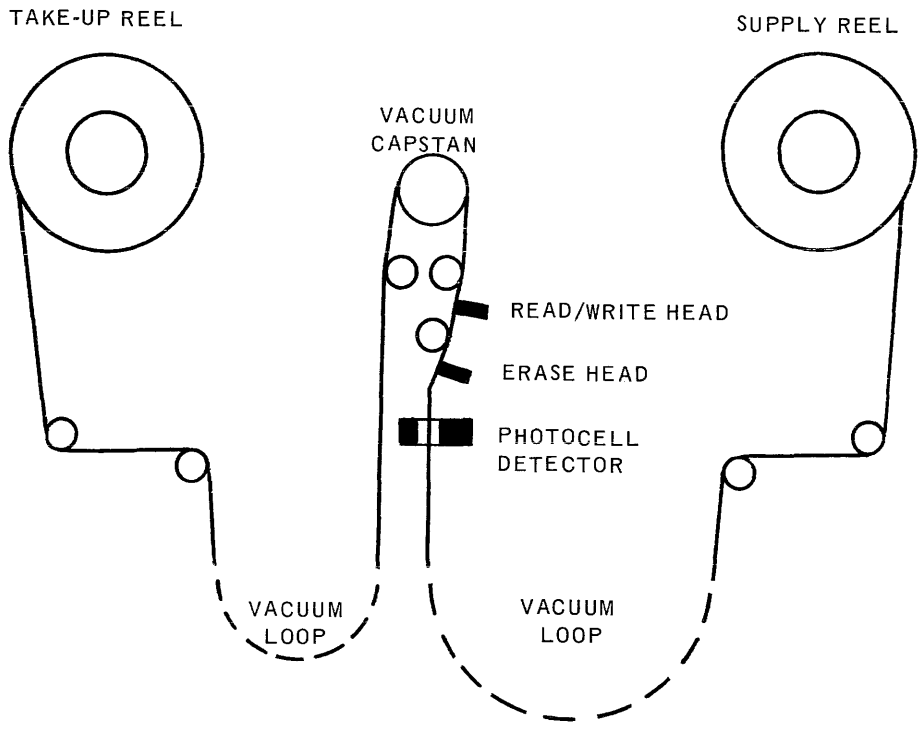


Figure D-1. UNISERVO IIIC Tape Unit Schematic

As shown in Figure D-1, the reel of tape that contains data to be processed or the blank reel upon which data is to be written is mounted on the right. The left hand reel is used to store the tape as it is read or written on. The tape is threaded around and through guide rollers which control its path between the supply reel and take-up reel.

The read/write head performs the actual reading or writing of data as the tape passes beneath it at the rate of 112.5 inches per second. Since the tape density may be 200 or 556 characters per inch, information can be transferred at the rate of 22,500 or 62,500 characters per second.

The erase head is used only during a write operation. It erases information on the tape before new information is written. Accidental erasure of a previously recorded tape can be prevented by removing the plastic ring from the tape before it is mounted on a tape unit.

Vacuum columns that are located below the tape reels are used to store the tape before it passes beneath the read-write head, or is wound on the take-up reel. These columns also insure that the tape tension is even and that there is a proper amount of slack to permit rapid acceleration and deceleration.

A photocell located below the erase head is used to detect reflective spots on the tape. These spots (See Figure D-2) indicate the load point (the initial point on the tape where reading or writing begins) and the end of tape warning point (the point that indicates that the physical end of tape is near).

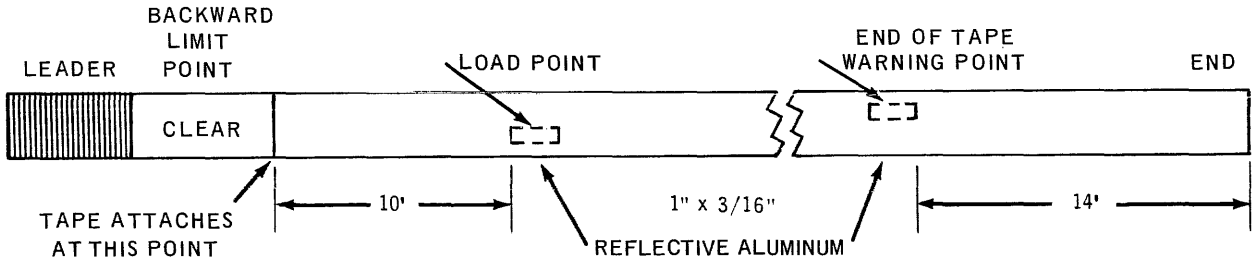


Figure D-2. UNISERVO IIIC Tape Diagram

Bad spots on tape are detected by reading the data back immediately after it has been written. The data read is then transferred to high and low gain registers in the Tape Adapter where a comparison is made. If the contents of these registers do not compare exactly, or there is a parity error, a status word is sent to the central processor. At this point it is the responsibility of the program to handle any rewriting attempts. The method that is chosen will depend upon block length, recording density, and other factors.

1. Tape Format

   Data appears on tape in the form of frames that are written across its width in seven channels or bit positions (6 data and 1 parity). Five frames represent one 30 bit computer word. Figure D-3 provides an example of how data appears on tape.

   As shown in Figure D-3, a parity channel is provided on tape. The parity channel helps assure that every character that is written will be read accurately.

   If data is written in binary coded decimal format, the subsystem will assure that the lateral parity is even by generating a 1 bit in the parity channel position of every frame that does not contain an even number of 1 bits.

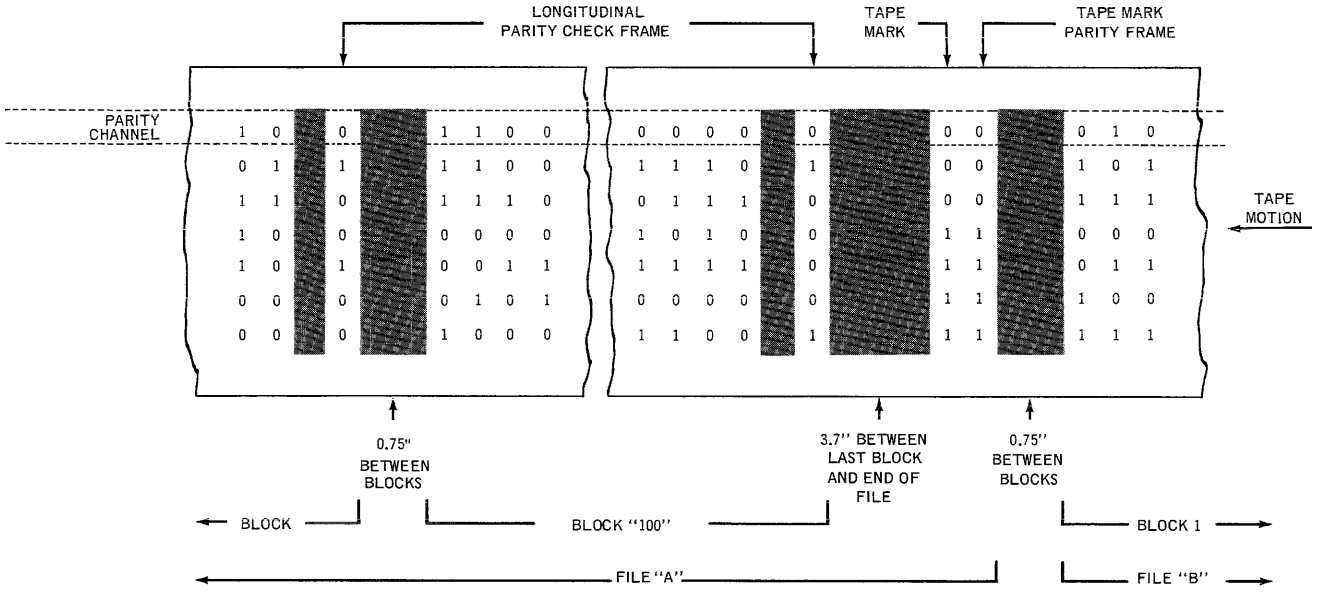


Figure D-3. UNISERVO IIIC Tape Format

If data is written in binary format, the subsystem will assure that the lateral parity is odd by generating a 1 bit in the parity channel position of every frame that does not contain an odd number of 1 bits.

In addition, the subsystem also assures that the longitudinal parity of data that is written is even. This is accomplished by having the subsystem count the number of 1 bits in each channel on tape when the writing of a block has been completed. If a channel does not contain an even number of 1 bits, the subsystem will generate a 1 bit in that channel position in the longitudinal parity check frame that is written after the end of the block.

When data is read from tape, it is checked for lateral and longitudinal parity before it is transferred to the central processor. If an improper parity condition is found, this fact will be stored by the subsystem until all of the current block of data has been read. When the entire block has been read, the subsystem will notify the central processor by means of a status word that an improper parity condition exists.

Data can be read or written in binary or binary coded decimal format. The binary format is normally used when data is exclusively numeric. When this format is used, it is possible to represent a positive or negative value up to $2^{29} - 1$ in each computer word. The binary coded decimal format allows the use of alphabetic characters and punctuation marks.

The UNISERVO IIIC Magnetic Tape Subsystem can read or write data blocks of any length.* The spacing between adjacent blocks is 0.75 inches and the end of file gap is 3.7 inches (See Figure D-3).

** It should be noted that if the space occupied by a data block is less than the space between blocks, it may be difficult to recover from tape errors.*

2. Variable by Character Binary Coded Decimal Format

In general, the minimum block length is one computer word. It is possible, however, to write blocks in binary coded decimal format in which the number of character's per block is not an integral multiple of 5. This is accomplished by using the Variable by Character Binary Coded Decimal Mode. In this mode the end of a block is marked by placing a stop code (000000) after the last valid character in the block. When a block is written in this mode, writing ceases when the stop code is detected. The stop code is not written on tape. For example, assume that a one character block (the letter A) is to be written on tape in this mode. This would be accomplished by having the program place the letter A in the first character position of the output buffer and the stop code in the second character as shown below

A	00			
29 24	23 18	17 12	11 6	5 0

When a tape that has been recorded in this mode is read, the first character in a block is placed in the first character position of the input buffer, the second character is placed in the second character position, and so on, until all of the characters have been transferred to the input buffer. When this has been completed, the remaining character positions in the last word of the input buffer are then filled with the binary equivalent of the number of valid characters that are present in this word. To illustrate, assume that a block that contains the characters A, B, C, D, E, F, G, and H is read. This block would appear in the input buffer as follows

A	B	C	D	E
29 24	23 18	17 12	11 6	5 0

F	G	02	02	02
29 24	23 18	17 12	11 6	5 0

If a tape that has been recorded in this mode is to be searched and the identifier word is less than five characters in length, the unused character positions of the identifier word must be filled with the binary equivalent of the number of valid characters. For example, if a search identifier consists of the letters D, G, T, and B, the identifier word that is sent to the subsystem is in the following form:

D	G	T	B	04
29 24	23 18	17 12	11 6	5 0

3. Translation

If the user wishes to avoid including translation routines in programs that require the reading and writing of tapes in binary coded decimal format, a translator is available. The translator is an optional device which can be incorporated into the UNISERVO IIIC Channel Synchronizer/Control Unit. The translator will translate Fieldata characters to binary coded decimal characters on output and on input will translate binary coded decimal characters to Fieldata characters. Translation will be performed in accordance with the input and output translation tables that are supplied by the user.

4. Channel Synchronizer/Control Unit

The Channel Synchronizer/Control Unit for the UNISERVO IIIC Magnetic Tape Subsystem controls the operation of the tape units during their write and read functions, transfers the data to be written from the central processor to the tape units, and transfers the data that has been read from the tape units to the central processor.

- Write Function

  During the write function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a write command to the selected tape unit. Then it receives data words from the central processor and transfers them to the tape unit for writing.

- Read Function

  During the read function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a read command to the selected tape unit. Then it receives data words from the tape unit and transfers them to the central processor.

- Unit Selection

  The Channel Synchronizer/Contrc Unit decodes the function word and selects the specified tape unit.

- Error Detection

  Detection circuits in the Channel Synchronizer/Control Unit check the data characters for lateral parity as they are transferred to or from the central processor. These circuits also check the data words to see if they contain the proper number of characters.

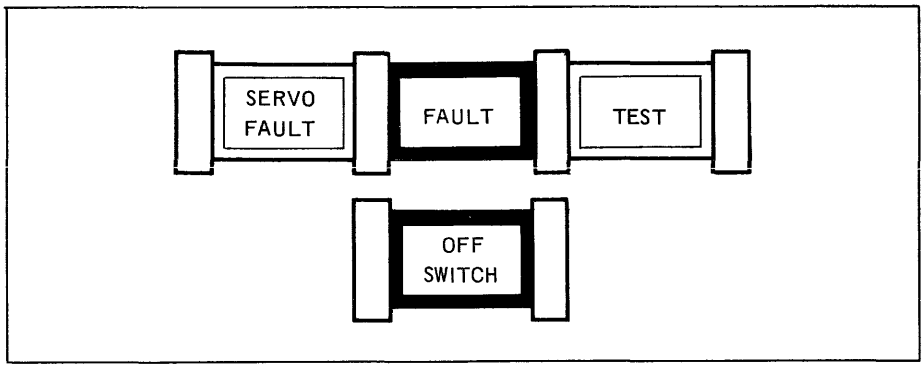
5. Tape Adapter Unit

The Tape Adapter Unit is located logically between the Channel Synchronizer/Control Unit and the tape units. It contains circuits that check data that is read from or written on tape for lateral and longitudinal parity.

6. Operator Controls

Primary control of the UNISERVO IIIC Magnetic Tape Subsystem is accomplished by the program. However, the Control Cabinet, the Tape Adapter Cabinet, and the individual tape units have operator panels that show operating conditions in the subsystem and allow some manual operation.

The Control Cabinet panel indicators and their functions are as follows:



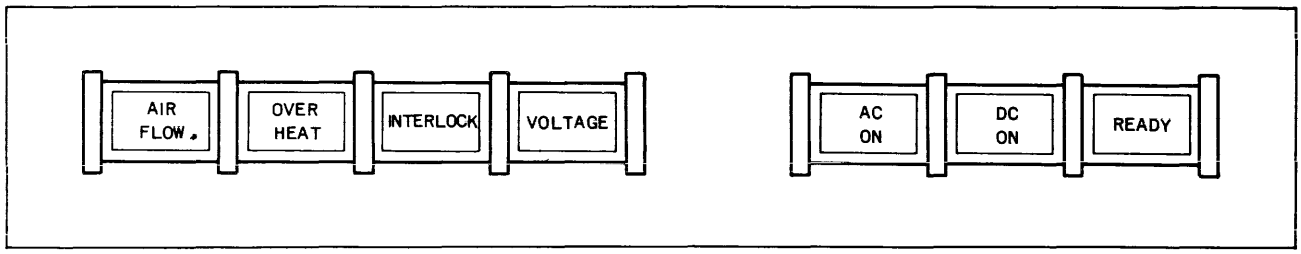
LEGEND:

SERVO FAULT	Lights when there is a fault in one of the tape units, in the Tape Adapter Cabinet, or in the power supply.
FAULT	Lights when there is a power fault in the subsystem. If the fault has been corrected, the indicator light may be extinguished by pressing this button.
TEST	Lights when the subsystem is isolated for test purposes.
OFF SWITCH	When all power is on in the subsystem, this button is green. When power is off, this button is red. Power can be turned off by pressing this button; however, once the power has been turned off, this button cannot be used to turn the power on again. Power is turned on by other means.

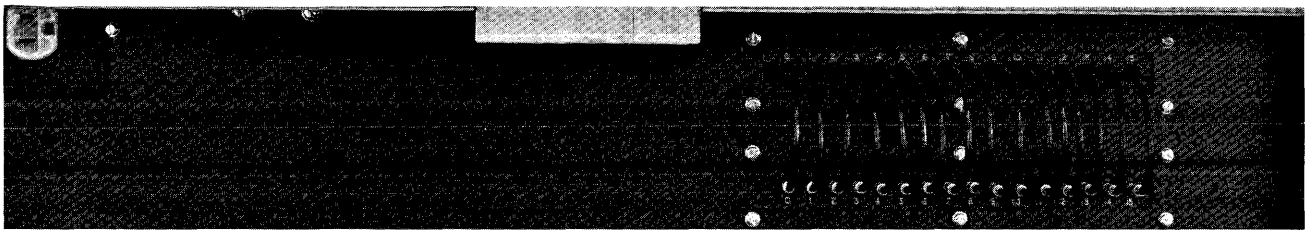
The Control Cabinet also has an auxiliary operator panel that is located behind its front door. This panel is used to assign a specific tape unit for Bootstrap* operation and to monitor certain signals from the tape units.

* *To facilitate the initial loading of a program into core storage, a Bootstrap operation is provided in the central processor. This is a small program which, when activated, will bring some other program on magnetic tape into core storage. Once the program to be executed is loaded, the Bootstrap program is no longer required and may be removed from core storage. The Bootstrap program is activated or deactivated by a manual switch that is connected to the central processor.*

The Tape Adapter Cabinet panel indicators and their functions are as follows:



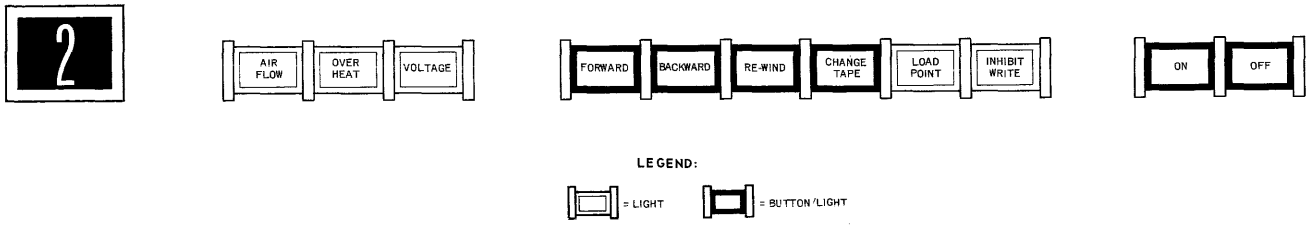
LEGEND:



AIR FLOW FLOW	Lights when power has been turned off because of insufficient air flow.
OVERHEAT	Lights when power has been turned off because the cabinet temperature has exceeded allowable limits.
INTERLOCK	Lights when a door on the cabinet is open.
VOLTAGE	Lights when cabinet voltage supply has failed due to power supply or interlock failures.
AC ON	Lights when AC power is applied to circuitry.
DC ON	Lights when DC power is applied to circuitry.
READY	Lights when all power is applied within tolerances to all circuitry.

As shown in the illustration, the Tape Adapter Cabinet also contains a plugboard. This plugboard is located behind the front door. It is used to assign logical numbers to the respective physical tape units. The numbers that are shown at the top of the plugboard represent the physical numbers of the tape units and those at the bottom represent the logical numbers.

The Tape Unit panel indicators and their functions are as follows:



UNIT NUMBER — Displays the physical number of the individual unit.

AIR FLOW — Lights when power has been turned off because of insufficient air flow.

OVERHEAT — Lights when power has been turned off because the cabinet temperature has exceeded allowable limits.

VOLTAGE — Lights when a circuit breaker within the unit is open.

FORWARD — Lights when the tape is moving in the forward direction, or the unit is ready to do so. Pressing this button conditions the unit for forward motion independent of program control.

BACKWARD — Lights when the tape is moving in the backward direction, or the unit is ready to do so. Pressing this button conditions the unit for backward motion independent of program control.

REWIND — Lights when the tape is being rewound. Pressing this button initiates the rewind operation if the unit has been conditioned for backward tape motion.

CHANGE TAPE — Lights when a tape is rewinding with interlock or has been rewound with interlock. Pressing this button will move the tape to the load point.

LOAD POINT — Lights when the tape is positioned at its load point (beginning point), ready for reading or writing.

INHIBIT WRITE — Lights if a reel of tape that does not contain a write enable ring has been mounted on the unit.

ON — Lights when all power is on in the unit. Pressing this button applies power.

OFF — Lights when all power is off in the unit. Pressing this button removes power.

7. Operations

Any part of the Central Processor's internal core storage can be used as an input/output data buffer storage area with the exception of the few special core storage locations that are reserved for the Incremental Clock and Interrupt Words. Information is transferred between the central processor and the UNISERVO IIIC Magnetic Tape Subsystem in the form of 30-bit data words. These words are formed into blocks in core storage. A block may contain any number of words. Words in a block must occupy consecutive core storage addresses, starting with a program determined first word address and ending with a program determined last word address.

a. Buffer Mode

A buffer mode transfer, which occurs independent of main program control, is used to transfer data between core storage and the UNISERVO IIIC Tape Units. Before execution of a buffer mode transfer of data, the program must perform the following steps:

(1) Activate the channel to be used for the information transfer.

(2) Load the channel index register with the buffer control word. (The lower and upper halves of the buffer control word contain the beginning and ending addresses of the section of core storage that is involved in the transfer).

(3) Send the proper function word to the UNISERVO IIIC Control Unit.

Steps 1 and 2 above are accomplished with one of the Initiate Buffer Instructions; step 3 is performed by the Enter External Function instruction. See BASIC INPUT/OUTPUT INSTRUCTIONS (Section 5) for a more detailed discussion of these operations. Data is then transferred between the central processor core storage and the selected UNISERVO IIIC Tape Unit without main program intervention. When a word is transferred to or from core storage, a 1 is added to the lower half of the buffer control word. The data transfer is terminated when:

(1) Thecentral processor senses that the upper and lower halves of the buffer control word are equal.

(2) A Terminate instruction is executed by the central processor.

(3) An error is detected.

b. Word Arrangement

The UNISERVO IIIC Magnetic Tape Subsystem accommodates four types of computer input/output words. They are the function word, data word, identifier word, and status word. These words are explained and illustrated in the following paragraphs.

(1) Function Word

The function word designates the operation to be performed by the UNISERVO IIIC Magnetic Tape Subsystem. It is arranged as follows:

The function code, FC, occupies bit positions 18 through 29 of the function word. The function code specifies the actual operation to be performed by the subsystem. Bit positions 0 through 11, U, specify the tape unit that will be selected. This is accomplished by placing a 1 bit in the bit position that corresponds to the desired unit.

The function codes are as follows:

CODE	FUNCTION	DESCRIPTION
0200	Write Binary High Density	Write one block on tape in binary format at a density of 556 characters per inch.
1200	Write Binary High Density With Interrupt	Same as Write Binary High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
0300	Write Binary Low Density	Write one block on tape in binary format at a density of 200 characters per inch.
1300	Write Binary Low Density With Interrupt	Same as Write Binary Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
0220	Write Binary Coded Decimal High Density	Write one block on tape in binary coded decimal format at a density of 556 characters per inch.
1220	Write Binary Coded Decimal High Density With Interrupt	Same as Write Binary Coded Decimal High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
0320	Write Binary Coded Decimal Low Density	Write one block on tape in binary coded decimal format at a density of 200 characters per inch.
1320	Write Binary Coded Decimal Low Density With Interrupt	Same as Write Binary Coded Decimal Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
0240	Write Multiblock Binary High Density	Write one block on tape in binary format at a density of 556 characters per inch; stop only if new data is not made available within 4.0 milliseconds after the block has been written.
1240	Write Multiblock Binary High Density With Interrupt	Same as Write Multiblock Binary High Density except that after each block is written, a status word and an external interrupt signal are sent to the central processor.
0340	Write Multiblock Binary Low Density	Write one block on tape in binary format at a density of 200 characters per inch; stop only if new data is not made available within 4.0 milliseconds after the block has been written.

CODE	FUNCTION	DESCRIPTION
1340	Write Multi-block Binary Low Density With Interrupt	Same as Write Multiblock Binary Low Density except that after each block is written, a status word and an external interrupt signal are sent to the central processor.
0260	Write Multi-block Binary Coded Decimal High Density	Write one block on tape in binary coded decimal format at a density of 556 characters per inch; stop only if new data is not made available within 4.0 milliseconds after the block has been written.
1260	Write Multi-block Binary Coded Decimal High Density With Interrupt	Same as Write Multiblock Binary Coded Decimal High Density except that after each block is written, a status word and an external interrupt signal are sent to the central processor.
0230	Write End Of File High Density	Write one special block on tape, consisting of the Tape Mark frame (0001111) and its logitudinal parity check frame, at a density of 556 characters per inch.
1230	Write End Of File High Density With Interrupt	Same as Write End Of File High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
0330	Write End Of File Low Density	Write one special block on tape, consisting of the Tape Mark frame (0001111) and its longitudinal parity check frame, at a density of 200 characters per inch.
1330	Write End Of File Low Density With Interrupt	Same as Write End Of File Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4200	Read Binary High Density	Read one block from tape in binary format at a density of 556 characters per inch.
5200	Read Binary High Density With Interrupt	Same as Read Binary High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4300	Read Binary Low Density	Read one block from tape in binary format at a density of 200 characters per inch.
5300	Read Binary Low Density With Interrupt	Same as Read Binary Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

CODE	FUNCTION	DESCRIPTION
4220	Read Binary Coded Decimal High Density	Read one block from tape in binary coded decimal format at a density of 556 characters per inch.
5220	Read Binary Coded Decimal High Density With Interrupt	Same as Read Binary Coded Decimal High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4320	Read Binary Coded Decimal Low Density	Read one block from tape in binary coded decimal format at a density of 200 characters per inch.
5320	Read Binary Coded Decimal Low Density With Interrupt	Same as Read Binary Coded Decimal Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4600	Search Binary High Density	Read the tape in binary format at a density of 556 characters per inch and compare the first word of each block with the identifier word. When an identical comparison is made, read that block from tape.
5600	Search Binary High Density With Interrupt	Same as Search Binary High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4700	Search Binary Low Density	Read the tape in binary format at a density of 200 characters per inch and compare the first word of each block with the identifier word. When an identical comparison is made, read that block from tape.
5700	Search Binary Low Density With Interrupt	Same as Search Binary Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4620	Search Binary Coded Decimal High Density	Read the binary coded decimal format at a density of 556 characters per inch and compare the first word of each block with the identifier word. When an identical comparison is made, read that block from tape.
5620	Search Binary Coded Decimal High Density With Interrupt	Same as Search Binary Coded Decimal High Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

CODE	FUNCTION	DESCRIPTION
4720	Search Binary Coded Decimal Low Density	Read the tape in binary coded decimal format at a density of 200 characters per inch and compare the first word of each block with the identifier word. When an identical comparison is made, read that block from tape.
5720	Search Binary Coded Decimal Low Density With Interrupt	Same as Search Binary Coded Decimal Low Density except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
2010	Rewind	Rewind the tape on the specified tape unit to the load point.
3010	Rewind With Interrupt	Same as Rewind except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
2110	Rewind With Interlock	Rewind the tape on the specified tape unit to the load point with interlock.
3110	Rewind With Interlock And Interrupt	Same as Rewind With Interlock except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
4000	Bootstrap	Rewind the tape on the tape unit specified for Bootstrap operation; read one block from tape at a density of 556 character per inch.
5000	Bootstrap With Interrupt	Same as Bootstrap except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
2030	Backspace Block	Move the tape backward on the specified tape unit, until the load point, tape mark frame, or a space between blocks is encountered.
3030	Backspace Block With Interrupt	Same as Backspace Block except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
2130	Backspace File	Move the tape backward on the specified tape unit until the load point or a block that contains four frames or less is encountered.
3130	Backspace File With Interrupt	Same as Backspace File except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

CODE	FUNCTION	DESCRIPTION
0030	Erase	Move the tape on the specified tape unit forward and erase approximately four inches of tape.
1030	Erase With Interrupt	Same as Erase except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
2300	Terminate	Terminate the present function at the end of the current block.
3300	Terminate With Interrupt	Same as Terminate except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*0221	Write Binary Coded Decimal High Density Translation Mode	Translate the data to be written into binary coded decimal format and write one block on tape at a density of 556 characters per inch.
*1221	Write Binary Coded Decimal High Density Translation Mode With Interrupt	Same as Write Binary Coded Decimal High Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*0321	Write Binary Coded Decimal Low Density Translation Mode	Translate the data to be written into binary coded decimal format and write one block on tape at a density of 200 characters per inch.
*1321	Write Binary Coded Decimal Low Density Translation Mode With Interrupt	Same as Write Binary Coded Decimal Low Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*0261	Write Multi-block Binary Coded Decimal High Density Translation Mode	Translate the data to be written into binary coded decimal format and write one block on tape at a density of 556 characters per inch; stop only if new data is not made available within 4.0 milliseconds after the block is written.

* *Used only when a Translator has been installed in the Channel Synchronizer/Control Unit.*

CODE	FUNCTION	DESCRIPTION
*1261	Write Multi-block Binary Coded Decimal High Density Translation Mode With Interrupt	Same as Write Multiblock Binary Coded Decimal High Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*0361	Write Multi-Block Binary Coded Decimal Low Density Translation Mode	Translate the data to be written into binary coded decimal format and write one block on tape at a density of 200 characters per inch; stop only if new data is not made available within 4.0 milliseconds after the block is written.
*1361	Write Multi-block Binary Coded Decimal Low Density Translation Mode With Interrupt	Same as Write Multiblock Binary Coded Decimal Low Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*4221	Read Binary Coded Decimal High Density Translation Mode	Read one block from tape in binary coded decimal format at a density of 556 characters per inch and translate the data that has been read into Fieldata format.
*5221	Read Binary Coded Decimal High Density Translation Mode With Interrupt	Same as Read Binary Coded Decimal High Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*4321	Read Binary Coded Decimal Low Density Translation Mode	Read one block from tape in binary coded decimal format at a density of 200 characters per inch and translate the data that has been read into Fieldata format.
*5321	Read Binary Coded Decimal Low Density Translation Mode With Interrupt	Same as Read Binary Coded Decimal Low Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

* *Used only when a Translator has been installed in the Channel Synchronizer/Control Unit.*

CODE	FUNCTION	DESCRIPTION
*4621	Search Binary Coded Decimal High Density Translation Mode	Read the tape in binary coded decimal format at a density of 556 characters per inch and compare the first word of each block with the identifier word. When an identical comparison is made, read that block and translate the data that has been read into Fieldata format.
*5621	Search Binary Coded Decimal High Density Translation Mode With Interrupt	Same as Search Binary Coded Decimal High Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*4721	Search Binary Coded Decimal Low Density Translation Mode	Read the tape in binary coded decimal format at a density of 200 characters per inch and compare the first word of each block with the identifier word. When an identical comparison is made, read that block and translate the data that has been read into Fieldata format.
*5721	Search Binary Coded Decimal Low Density Translation Mode With Interrupt	Same as Search Binary Coded Decimal Low Density Translation Mode except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

* *Used only when a Translator has been installed in the Channel Synchronizer/Control Unit.*

(2) Data Word

The data word (input or output) is arranged in five groups of 6-bit characters if the data is in binary coded decimal format. A data word in binary coded decimal format is arranged as follows:

If a data word is in binary format, it is arranged as follows:

(3) Identifier Word

The identifier word is a computer output word that immediately follows a search function word. It can contain any bit configuration as shown below. It is transmitted to the subsystem accompanied by an External Function signal. The identifier word is compared with first word of each block on tape until the appropriate find is accomplished. When this occurs, the block containing the identical word is read from tape.

(4) Status Word

A status word is transmitted by the subsystem to the central processor whenever an abnormal condition or an operation with interrupt has been completed. A status word is arranged as follows:

The status code, SC, occupies bit positions 24 through 29 of the status word. All other bit positions are not used. The status word is accompanied by an external interrupt signal when it is transmitted to the central processor.

The status codes are as follows:

CODE	INDICATION	DESCRIPTION
34	Function Word Error	This indication occurs when the function word contains a function code that is not valid for the subsystem, there is a unit select error (more than one unit has been specified, or none has been specified), or if the first character encountered during a binary coded decimal write function is 00.
24	Tape Unit Rewinding (without interlock)	This indication occurs when the central processor requests a tape unit that is in the process of rewinding without interlock.
54	End Of File	This indication occurs when the Tape Mark frame and its longitudinal parity check frame are detected or if an echo error is detected after the first character during a binary coded decimal write function.

CODE	INDICATION	DESCRIPTION
50	Control Unit Sequence Error	This indication occurs when the central processor has not issued an input acknowledge signal in time to remove a word from the subsystem register before the subsystem begins to put a new word into the register; or the central processor delays in issuing an output acknowledge signal and data with the result that the subsystem begins to write and verify the longitudinal parity check frame.
74	Interlock Fault	This indication occurs when the central processor requests a non-existent tape unit, when a write function is requested on a tape unit that has a tape mounted that does not contain a write enable ring, or when an interlocked tape unit is requested (the unit is rewinding with interlock, a door is open on the unit, and so on).
44	Repeat Operation	This indication occurs when a lateral or longitudinal parity error is detected during a read or search function, or a comparison error is detected during the write-check portion of a write function.
70	Abnormal Frame Count	This indication occurs when the total number of significant frames in a block on tape is not a multiple of five.
60	Tape Limit Reached	This indication occurs when the load point or the end of tape warning mark is detected during a read, search, write, erase, backspace block, or backspace file function.
30	Channel Synchronizer Character Count Error	This indication occurs when the channel synchronizer character counter does not contain the proper count.
20	Channel Synchronizer Sequence Error	This indication occurs when the channel synchronizer has received an end of block signal and the character counter does not contain the proper count.
40	Normal Completion	This indication occurs when a function with interrupt has been successfully completed.

8. Subsystem Characteristics

a. Summary of General Characteristics

TRANSFER RATE	22,500 and 62,500 characters per second
RECORDING DENSITY	200 and 556 6 bit characters per inch
TAPE SPEED	112.5 inches per second
TAPE WIDTH	0.5 inch
TAPE LENGTH	2,400 feet
TAPE THICKNESS	1.5 mils
BLOCK LENGTH	variable
SPACE BETWEEN BLOCKS	0.75 inch
CHANNELS ON TAPE	7 channels 6 data 1 parity
READ/WRITE OPERATION	Reading and writing operations proceed in the forward direction only.

b. Physical Characteristics

	UNISERVO IIIC TAPE UNIT	POWER SUPPLY	UNISERVO IIIC CONTROL CABINET	TAPE ADAPTER CABINET
HEIGHT (inches)	96	96	96	96
WIDTH (inches)	31	66	20	36
DEPTH (inches)	30	33	34	30
WEIGHT (approximate lbs.)	350	2,200	310	800
TEMPERATURE RANGE	60°–82° F.	68° – 82° F.		
HUMIDITY RANGE	40%–70%	40%–60%	40%–70%	40%–60%
HEAT DISSIPATION (BTU/hr.)	7,500	10,200	598	1,360
AIR FLOW (cu. ft./min.)	350	2,300	390	600
POWER REQUIREMENTS	Supplied by power supply via Tape Adapter Cabinet	208 VAC 60 cps 33phase 43.7 kw	208 VAC – 0.1% 60 cps 3 phase 0.6 kw	Supplied by power supply.

MAXIMUM CABLE LENGTH RESTRICTIONS

1. Tape Adapter Cabinet must be adjacent to the left side of the first UNISERVO IIIC Tape Unit.
2. Power Supply to Tape Adapter Cabinet: 60 ft.
3. UNISERVO IIIC Control Cabinet to Tape Adapter Cabinet: 80 ft.
4. Central Processor to UNISERVO IIIC Control Cabinet: 75 ft.

E. UNISERVO IIIA Magnetic Tape Subsystem

The UNISERVO IIIA Magnetic Tape Subsystem differs from the UNISERVO IIA Magnetic Tape Subsystem and the UNISERVO IIIC Magnetic Tape Subsystem in that its storage capacity per reel of tape is substantially greater. This is accomplished by recording data at a higher density.

A UNISERVO IIIA Magnetic Tape Subsystem consists of a Channel Synchronizer/Control Unit, and from two to sixteen UNISERVO IIIA Tape Units that are connected to the central processor via an input/output channel.

A subsystem that is on a particular input/output channel is capable of reading or writing data in the form of 30-bit binary words on any one of the tape units at any one time. Data can be read or written at a density of 1,250 characters per inch. If simultaneous reading and writing is required, options are available which permit this.*

The UNISERVO IIIA Tape Unit utilizes 600 foot, 1,500 foot, 1,800 foot, and 3,600 foot reels of plastic tape that are 0.5 inch wide and 1 mil in thickness; or 2,400 foot reels of plastic tape that are 0.5 inches wide and 1.5 mils in thickness.

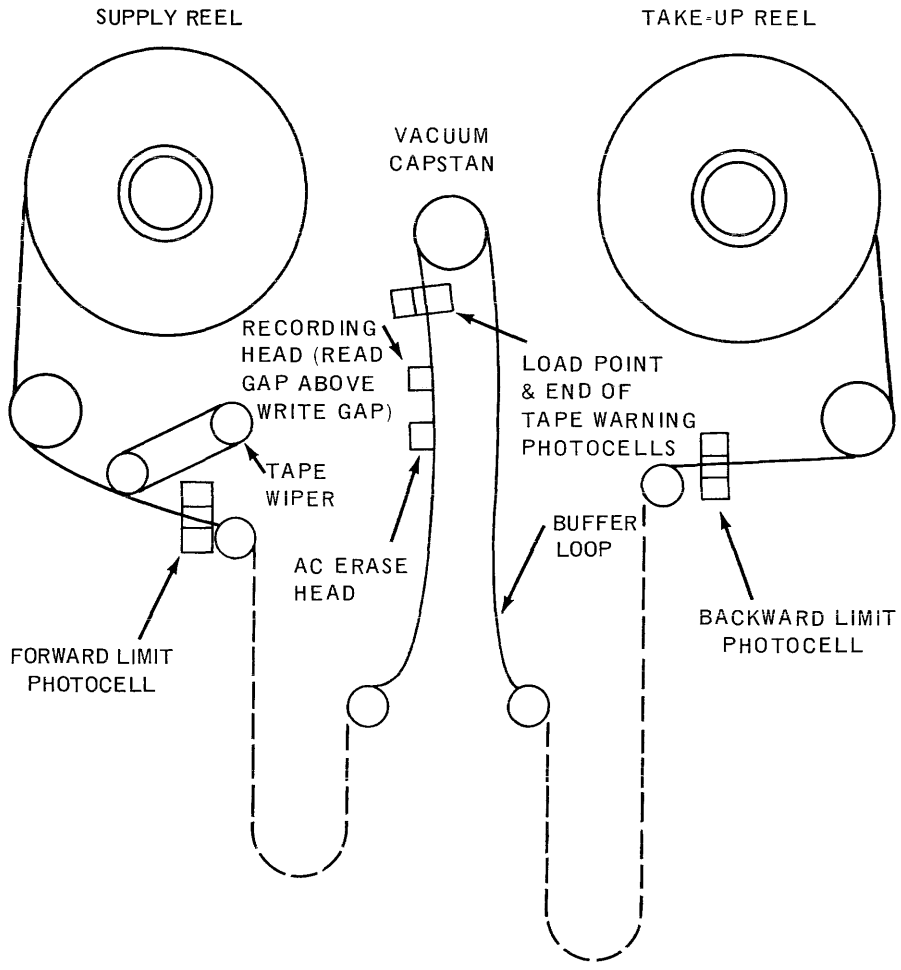


Figure E-1. UNISERVO IIIA Tape Unit Schematic

* *See Subsection 5, Control Options.*

As shown in Figure E-1, the reel of tape that contains data to be processed, or the blank reel upon which data is to be written is mounted on the left. The right hand reel is used to store the tape as it is read or written on. The tape is threaded around and through guide rollers which control its path between the supply reel and take-up reel. A pre-threaded leader is permanently attached to the take-up reel. When it becomes necessary to mount a new supply reel, the end of the tape on the supply reel is attached to this leader.

The read/write head performs the actual reading or writing of data as the tape moves beneath it at the rate of 100 inches per second. Since the tape density is 1,250 characters per inch, information is transferred at the rate of 125,000 characters per second.

The erase head is used only during a write operation. It erases information on the tape before new information is written. Accidental erasure of a previously recorded tape can be prevented by removing the write enable ring from the tape before it is mounted on a tape unit.

Vacuum columns, that are located below the tape reels, are used to store the tape before it passes beneath the read/write head or is wound on the take-up reel. These columns also insure that the tape tension is even and that there is a proper amount of slack to permit rapid acceleration and deceleration.

The tape unit contains four photocells. Two of these, located above the read/write head, are used to detect windows on the tape. These windows (see Figure E-2) indicate the load point (the initial point on the tape where reading or writing begins) and the end of tape warning point (the point that indicates that the physical end of tape is near). The other two photocells, one of which is located below the supply reel, and the other beneath the take-up reel, are used to detect the forward and backward limits of the tape.

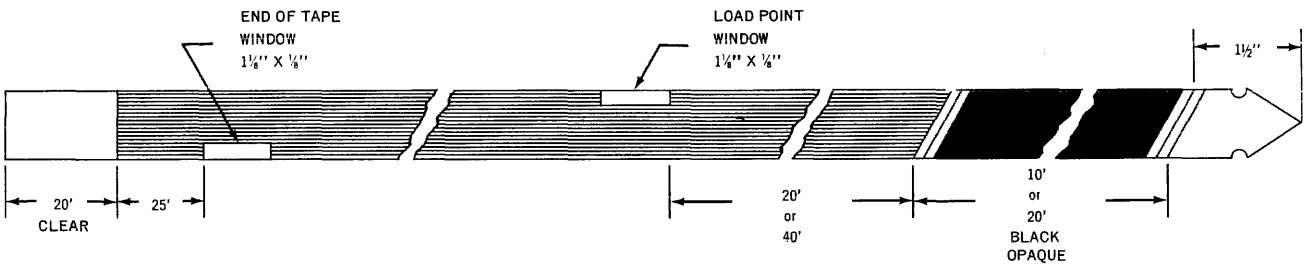


Figure E-2. UNISERVO IIIA Tape Diagram

Bad spots on tape are detected by reading the data back immediately after it has been written, and checking it for parity. If there is a parity error, the writing of data stops and the subsystem writes a specific pattern on the tape that will indicate that this portion of the tape is defective.

1. Tape Format

   Data appears on tape in the form of frames that are written across its width in nine channels or bit positions (8 data and 1 parity). The 30 bits of a computer word are placed in four frames** on tape; that is, 30 data bits and two 0 bits for fill. These words are written in groups called blocks (one or more words). Figure E-3 provides an example of how data appears on tape.

** *See subsection 5, Control Options.*

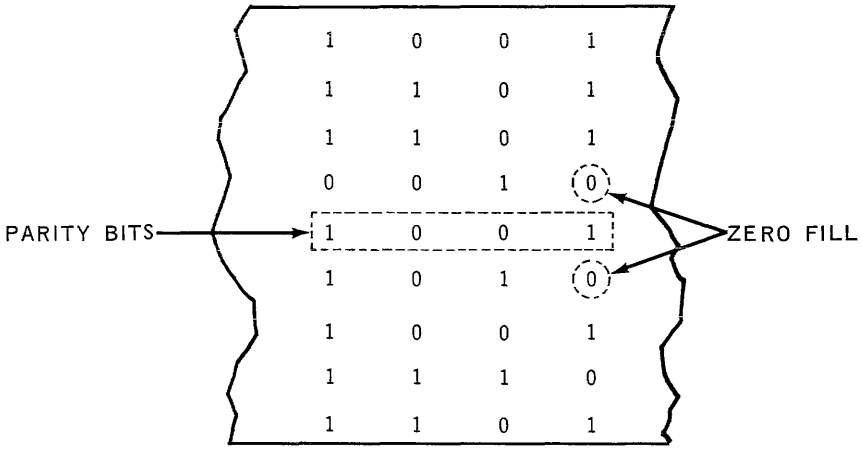


Figure E-3. UNISERVO IIIA Tape Format

As shown in Figure E-3, a parity channel is provided on tape. The parity channel helps assure that every character that is written will be read accurately.

When data is written, the subsystem will assure that the parity is even by generating a 1 bit in the parity channel position of each frame that does not contain an even number of 1 bits.

When data is read from tape, it is checked for parity before it is transferred to the central processor. If an improper parity condition is found, the subsystem will store this fact until all of the current block of data has been read. When the entire block has been read, the subsystem will notify the central processor by means of a status word that an improper parity condition exists.

The UNISERVO IIIA Magnetic Tape Subsystem can read or write data blocks of any length.* The only requirement is that a block must be at least one computer word. The spacing between adjacent blocks is 0.75 inches and the end of file gap is 2.5 inches.

As each data block is written, the subsystem places a start pattern and a start sentinel before the data, and a stop sentinel, a stop pattern, and an ending pattern after the data. The start pattern and the stop pattern each consist of 27 frames of 1 bits. The start sentinel consists of one frame of 0 bits followed by two frames of 1 bits. The stop sentinel consists of two frames of 1 bits followed by one frame of 0 bits. The ending pattern consists of 225 frames that contain 0 bits in odd channels only. When the tape is subsequently read, these patterns and sentinels will indicate the beginning and end of data to the reading circuitry. Figure E-4 illustrates how these sentinels and patterns appear on tape.

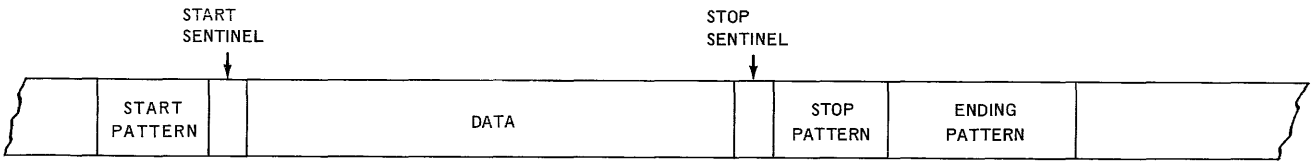


Figure E-4. Data Block Sentinels and Patterns

* *It should be noted that if the space occupied by a data block is less than the space between blocks, it may be difficult to recover from tape errors.*

In addition, if an error (frame count error, parity error, or bad spot) is detected during the writing of a data block, the subsystem will place a special pattern on tape after the data. This pattern consists of 225 frames that contain 0 bits in odd channels only (an ending pattern), 250 frames that contain 0 bits in all channels, and 250 frames that contain 0 bits in even channels only. When the tape is subsequently read, the presence of this special pattern will indicate to the reading circuitry that this block is improperly written. Figure E-5 illustrates how this special pattern appears on tape. (The stop sentinel and stop pattern will not be written if a bad spot or parity error is detected.)

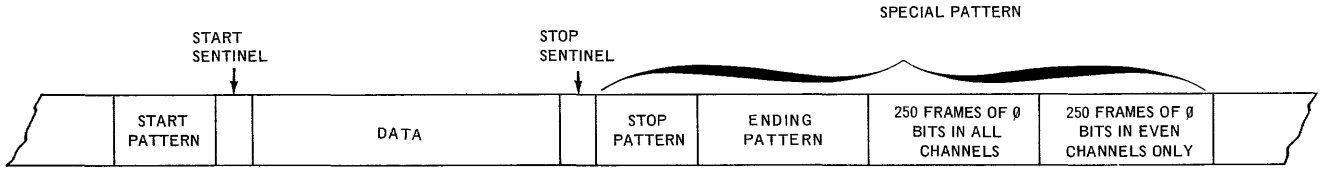


Figure E-5. Special Pattern for Improperly Written Block

2. Channel Synchronizer/Control Unit

The Channel Synchronizer/Control Unit for the UNISERVO IIIA Magnetic Tape Subsystem controls the operation of the tape units during their write and read functions, transfers the data to be written from the central processor to the tape units, and transfers the data that has been read from the tape units to the central processor.

- Write Function

  During the write function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a write command to the selected tape unit. Then it receives data words from the central processor and transfers them to the tape unit for writing.

- Read Function

  During the read function, the Channel Synchronizer/Control Unit receives a function word from the central processor and translates it into a read command to the selected tape unit. Then it receives data words from the tape unit and transfers them to the central processor.

- Unit Selection

  The Channel Synchronizer/Control Unit decodes the function word and selects the specified tape unit.

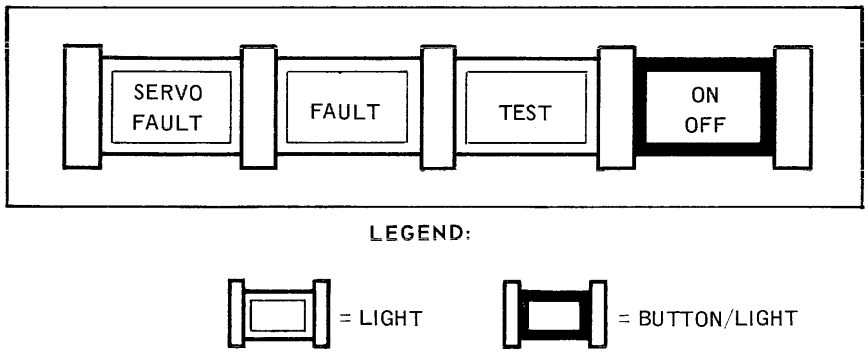
- Error Detection

  Detection circuits in the Channel Synchronizer/Control Unit check the characters for parity as they are transferred to and from the central processor. These circuits also check the data words to see if they contain the proper number of characters.

3. Operator Controls

Primary control of the UNISERVO IIIA Magnetic Tape Subsystem is accomplished by the program. However, the Control Cabinet and the individual tape units have operator panels that show operating conditions in the subsystem and allow some manual operation.

The Control Cabinet panel indicators and their functions are as follows:



SERVO FAULT — Lights when there is a voltage supply fault in any of the tape units.

FAULT — Lights when any of the following conditions occur:

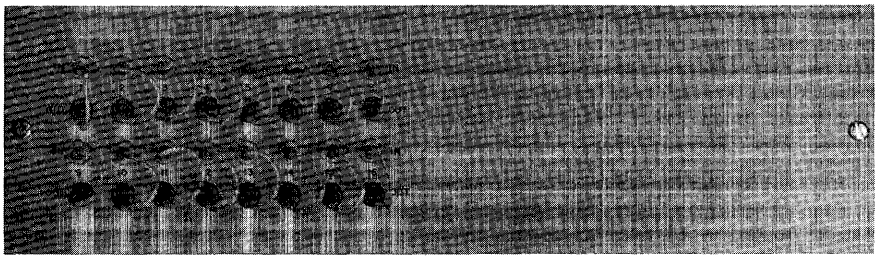
- Undervoltage in the Control Cabinet.
- Overcurrent in the Control Cabinet.
- Insufficient air flow in the Control Cabinet.
- Mechanical interlock failure in the Control Cabinet.

TEST — Lights when the subsystem is isolated for test purposes.

ON OFF — Pressing this button applies power to or removes power from the Control Cabinet. This button does not affect the blowers or the fault indicators.

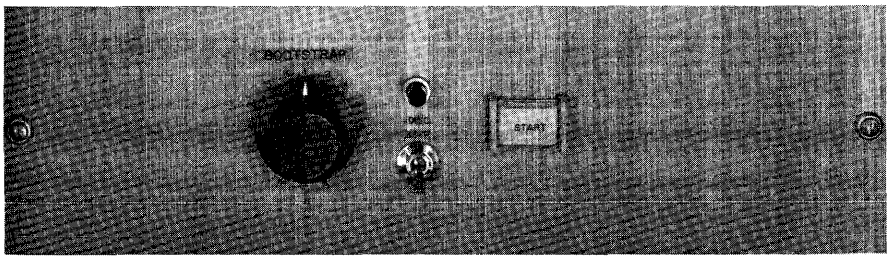
The Control Cabinet also contains an assignment panel and an auxiliary panel. These panels and their functions are described in the text that follows.

The assignment panel, which is used to assign logical numbers to the tape units, is illustrated below.



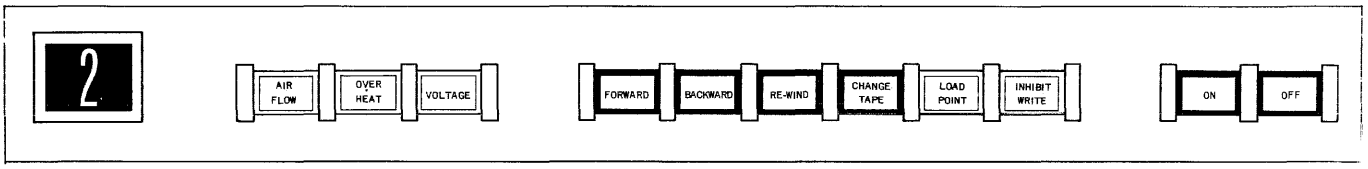
As shown in the illustration, the assignment panel consists of a plugboard that contains red IN jacks and green OUT jacks that are numbered 1 through 16. The IN jacks represent the numbers of the physical units and the OUT jacks represent the logical numbers.

The auxiliary panel, which is used to designate the logical number of the tape unit that will be used for Bootstrap* operation, is illustrated below.



As shown in the illustration, the auxiliary panel contains a rotary switch labeled BOOTSTRAP, a disconnect switch labeled DISC BSTP, and a button labeled START. The BOOTSTRAP switch has 16 positions which represent the logical numbers of the tape units. The DISC BSTP switch enables (down position) or disconnects (up position) the BOOTSTRAP switch. The START button is used only for maintenance purposes.

The Tape Unit panel indicators and their functions are as follows:



LEGEND:

UNIT NUMBER	Displays the physical number of the individual unit.
AIR FLOW	Lights when there is an insufficient air flow in the tape unit cabinet.
OVERHEAT	Lights when the temperature in the tape unit cabinet exceeds 130°F.
VOLTAGE	Lights when the tape unit voltage fails.
FORWARD	Lights when the tape is moving in a forward direction or the unit is ready to do so. Pressing this button conditions the unit for forward motion independent of program control.
BACKWARD	Lights when the tape is moving in a backward direction or the unit is ready to do so. Pressing this button conditions the unit for backward motion independent of program control.
REWIND	Lights when the tape is being rewound. Pressing this button initiates the rewind operation if the unit has been conditioned for backward tape motion.

** To facilitate the initial loading of a program into core storage, a Bootstrap operation is provided in the central processor. This is a small program which, when activated, will bring some other program on magnetic tape into core storage. Once the program to be executed is loaded, the Bootstrap program is no longer required and may be removed from core storage. The Bootstrap program is activated and deactivated by a manual switch that is connected to the central processor.*

CHANGE TAPE	Lights when a tape is rewinding with interlock or has been rewound with interlock. Pressing this button will move the tape to the load point.
LOAD POINT	Lights when the tape is positioned at its load point (beginning point), ready for reading or writing.
INHIBIT WRITE	Lights when a tape that does not contain a write enable ring has been mounted on the unit.
ON	Lights when all power is on in the unit. Pressing this button applies power.
OFF	Lights when all power is not on in the unit. Pressing this button removes power.

4. Operations

Any part of the central processor's internal core storage can be used as an input/output data buffer storage area, with the exception of the few special core storage locations that are reserved for the Incremental Clock and the Interrupt Words. Information is transferred between the central processor and the UNISERVO IIIA Magnetic Tape Subsystem in the form of 30-bit data words. These words are formed into blocks in core storage. A block may contain any number of words. Words in a block must occupy consecutive core storage addresses, starting with a program determined first word address, and ending with a program determined last word address.

a. Buffer Mode

A buffer mode of transfer, which occurs independent of program control, is used to transfer data between core storage and the UNISERVO IIIA Tape Units. Before execution of a buffer mode transfer of data, the program must perform the following steps:

(1) Activate the channel to be used for the information transfer.

(2) Load the channel index register with the buffer control word. (The lower and upper halves of the buffer control word contain the beginning and ending addresses of the section of the section of core storage that is involved in the transfer).

(3) Send the proper function word to the UNISERVO IIIA Control Unit.

Steps 1 and 2 above are accomplished with one of the Initiate Buffer instructions; step 3 is performed by the Enter External Function instruction. See BASIC INPUT/OUTPUT INSTRUCTIONS (Section 5) for a more detailed discussion of these operations.

Data is then transferred between the central processor core storage and the selected UNISERVO IIIA Tape Unit without main program intervention. When a word is transferred to or from core storage, a 1 is added to the lower half of the buffer control word. The data transfer is terminated when:

(1) The central processor senses that the upper and lower halves of the buffer control word are equal.

(2) A Terminate instruction is executed by the central processor.

(3) An error is detected.

b. Word Arrangement

The UNISERVO IIIA Magnetic Tape Subsystem accommodates five types of computer input/output words. They are the Function Word, Data Word, Identifier Word, Mask Word, and Status Word. These words are explained and illustrated in the following paragraphs.

(1) Function Word

The function word designates the operation to be performed by the UNISERVO IIIA Magnetic Tape Subsystem. It is arranged as follows:

The function code, FC, occupies bit positions 24 through 29 of the function word. The function code specifies the actual operation to be performed by the subsystem.

Bit positions 0 through 3, U, specify the tape unit that will be selected.

The function codes are as follows:

CODE	FUNCTION	DESCRIPTION
01	Write	Write one block on tape.
11	Write With Interrupt	Same as Write except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
02	Contingency Write	Write 0 bits in even channels only on tape for 3 inches.
12	Contingency Write With Interrupt	Same as Contingency Write except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
03	Write File Separator	Erase 2.5 inches of tape.
13	Write File Separator With Interrupt	Same as Write File Separator except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
41	Read Forward	Read one block from tape in a forward direction.
51	Read Forward With Interrupt	Same as Read Forward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

CODE	FUNCTION	DESCRIPTION
61	Read Backward	Read one block from tape in a backward direction.
71	Read Backward With Interrupt	Same as Read Backward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
42	Reposition Read Forward	Skip the block of data that is at the read head and read the next block from tape. This function is used to recover from an error caused by a block which was apparently written and checked without error. It permits skipping this block and reading the next block.
52	Reposition Read Forward With Interrupt	Same as Reposition Read Forward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
62	Reposition Read Backward	Read one block from tape in a backward direction. This is used in the case where an error is detected in a block with a normal ending pattern and the erased tape gap that follows is smaller than normal. When this occurs, the tape will stop with the read head positioned over the next block. The Reposition Read Backward function takes care of this situation by ignoring the information in the next block and reading the troublesome block in a backward direction.
72	Reposition Read Backward With Interrupt	Same as Reposition Read Backward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
45	Search Forward	Read the tape in a forward direction and compare the first word of each block with the identifier word. When an identical comparison is made, read that block from tape.
55	Search Forward With Interrupt	Same as Search Forward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
65	Search Backward	Read the tape in a backward direction and compare the last word of each block with the identifier word. When an identical comparison is made, read that block from tape.
75	Search Backward With Interrupt	Same as Search Backward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.

CODE	FUNCTION	DESCRIPTION
46	Masked Search Forward	Read the tape in a forward direction and compare those portions of the first word of each block that are selected by the mask word with those portions of the identifier word that are selected by the mask word. When an identical comparison is made, read that block from tape.
56	Masked Search Forward With Interrupt	Same as Masked Search Forward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
66	Masked Search Backward	Read the tape in a backward direction and compare those portions of the last word of each block that are selected by the mask word with those portions of the identifier word that are selected by the mask word. When an identical comparison is made, read that block from tape.
76	Masked Search Backward With Interrupt	Same as Masked Search Backward except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
20	Rewind	Rewind the tape on the specified tape unit to the load point.
30	Rewind With Interrupt	Same as Rewind except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
21	Rewind With Interlock	Rewind the tape on the specified tape unit to the unload point with interlock.
31	Rewind With Interlock and Interrupt	Same as Rewind With Interlock except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
40	Bootstrap	Rewind the tape on the tape unit specified for Bootstrap operation; read one block from tape in a forward direction.
50	Bootstrap With Interrupt	Same as Bootstrap except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
23	Terminate	Terminate the current function.
33	Terminate With Interrupt	Same as Terminate except that when the function is completed, a status word and an external interrupt signal are sent to the central processor.
*24	Release Control	Central processor that has control of the subsystem relinquishes control.

* *Used only with dual UNIVAC 490 Real-Time Systems.*

CODE	FUNCTION	DESCRIPTION
*34	Release Control With Interrupt	Same as Release Control except that when the function is completed, a status word and an external interrupt signal are sent to the particular central processor that is involved.
*26	Request Control	Non-controlling central processor requests control of the subsystem.
*27	Request Control With Interrupt	Same as Request Control except that when the function is completed, a status word and an external interrupt signal are sent to the particular central processor that is involved.
*25	Demand Control	Non-controlling central processor demands control of the subsystem. If this function is used, control of the subsystem is given immediately to the central processor that demands it.
*35	Demand Control With Interrupt	Same as Demand Control except that when the function is completed, a status word and an external interrupt signal are sent to the particular central processor that is involved.

* *Used only with dual UNIVAC 490 Real-Time Systems.*

(2) Data Word

The data word (input or output) may be composed of 30 bits of binary information or it may be composed of five six bit alphanumeric characters as shown below:

(3) Identifier Word

The identifier word is a computer output word that follows a search function word. It can contain any bit configuration as shown below. The identifier word is compared with the first word of each block on tape until the appropriate find is accomplished. When this occurs, the block containing the identical word is read from tape.

(4) Mask Word

The mask word is a computer output word that is used in a masked search operation to select the portion of the first word of each block that is to be compared with the identifier word. It can contain any bit configuration as shown below. The portion of the first word of each block that is selected is determined by the presence of 1 bits in specific bit positions in the mask word; that is, only those bit positions in the first word of each block that correspond to the 1 bits in the mask word will be compared with the identifier word.

When a masked search operation is requested, the masked search function word is followed by the mask word and the mask word is followed by the identifier word.

(5) Status Word

A status word is transmitted by the subsystem to the central processor whenever an abnormal condition or a function with interrupt has been completed. A status word is arranged as follows:

The status code, SC, occupies bit positions 24 through 29 of the status word. Bit positions 0 through 5, EI, are used to indicate the type of error that has occurred. A 1 bit in bit position 0 indicates a skew error, a 1 bit in bit position 1 indicates a parity error, a 1 bit in bit position 2 indicates a frame count error, a 1 bit in bit position 3 indicates an interlock error, a 1 bit in bit position 4 indicates a data pile-up, and a 1 bit in bit position 5 indicates an illegal function. The status word is accompanied by an external interrupt signal when it is transmitted to the central processor.

The status codes are as follows:

CODE	INDICATION	DESCRIPTION
64	Function Terminated	This indication occurs when a function is terminated before it has been completed.
50	Illegal Function Code	This indication occurs when the function word contains a function code that is not valid for the subsystem.
74	Manual Intervention Required	This indication occurs if the requested tape unit has an interlock set.
10	Tape Unit Rewinding	This indication occurs when the central processor requests a tape unit that is in the process of rewinding without interlock.
30	Maintenance Intervention Required	This indication occurs when there is a tape unit signal malfunction, the load point is detected when reading or searching backwards, or there is a loss of power in the tape unit.
34	End of File	This indication occurs when the file separator is encountered.
60	Repeat Operation	This indication occurs when a parity error is detected when reading a previously determined valid data block, the central processor resumes accepting data after a pile up has happened during a read function, or the central processor resumes sending data after the tape unit stops writing during a write function.
54	Improperly Written Block	This indication occurs when a parity or frame count error is detected during a write function, or when the special pattern that indicates an improperly written block is detected during a read function.
70	Frame Count Error	This indication occurs when the total number of frames in a block or tape is not a multiple of four.
44	Tape Marker Detected	This indication occurs when the end of tape warning point is detected during a write, contingency write, or write file separator function; or when the load point is detected during an attempt to execute a function that calls for backward tape movement.
40	Normal Completion	This indication occurs when a function with interrupt has been successfully completed.
14	Tape Unit Controlled By Other Control Unit	This indication occurs when the subsystem has the dual control option and the tape unit that is requested by one control unit is under the control of the other control unit.

5. Control Options

The following control options are available:

a. Single UNIVAC 490 Real-Time System, Dual Control

This option permits simultaneous reading from one tape unit and writing on another or simultaneous reading from two tape units. It consists of an additional Channel Synchronizer/Control Unit, connected to a second input/output channel, that controls the same tape units as the first. The second Channel Synchronizer/Control Unit cannot perform a write function; consequently, simultaneous reading and writing is performed by using the first Channel Synchronizer/Control Unit to write on one tape unit and the second to read from another tape unit. Simultaneous reading is performed by using both Channel Synchronizer/Control units to read from separate tape units.

d. Dual UNIVAC 490 Real-Time Systems, Dual Control

This option allows the user to have two UNIVAC 490 Real-Time Systems utilize the same UNISERVO IIIA Magnetic Tape Subsystem. In this case, either system can perform simultaneous reading from one tape unit and writing on another or simultaneous reading from two tape units; or, either system can read from one tape unit and the other write on a tape simultaneously.

c. Dual UNIVAC 490 Real-Time Systems, Single Control

This option permits the user to have two UNIVAC 490 Real-Time Systems utilize the same UNISERVO IIIA Magnetic Tape Subsystem via a single common Channel Synchronizer/Control Unit. In this case, only one UNIVAC 490 Real-Time System may utilize the subsystem at a time.

d. Five Frame Tape Format

The normal tape format, as described in subsection 1, utilizes four frames (8 bits per frame) to represent one 30 bit computer word. If desired, the Channel Synchronizer/Control Unit can be modified internally so that a computer word is represented by five frames (6 bits per frame). If this modification is made, data will appear on tape as shown in Figure E-6.

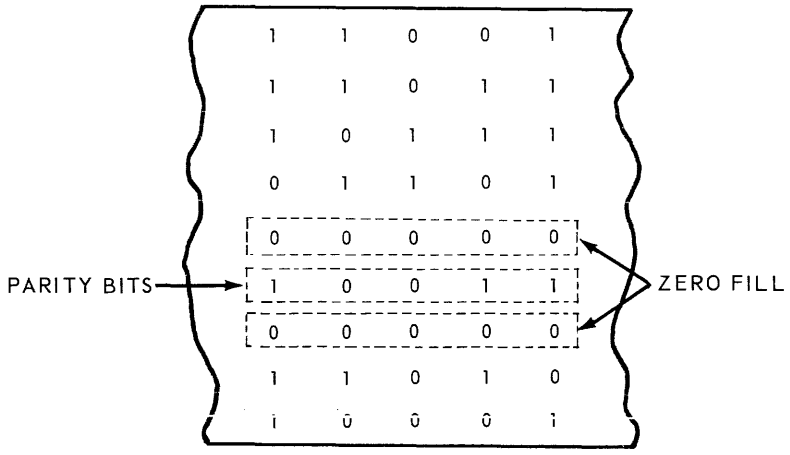


Figure E-6. UNISERVO IIIA Five Frame Tape Format

6. Subsystem Characteristics

a. Summary of General Characteristics

TRANSFER RATE	125,000 characters per second.
RECORDING DENSITY	1,250 6 bit characters per inch.
TAPE SPEED	100 inches per second.
TAPE WIDTH	0.5 inch
TAPE LENGTH	600, 1,500, 1,800, 2,400 or 3,600 feet
TAPE THICKNESS	1 mil (1.5 mils for 2,400 foot tapes)
BLOCK LENGTH	variable
SPACE BETWEEN BLOCKS	0.75 inch
CHANNELS ON TAPE	9 channels
	8 data
	1 parity
READ/WRITE OPERATION	Reading in forward and backward directions; writing in the forward direction only.

b. Physical Characteristics

	UNISERVO IIIA TAPE UNIT	POWER SUPPLY	UNISERVO IIIC CONTROL CABINET
HEIGHT (inches)	96	96	96
WIDTH (inches)	31	66	20
DEPTH (inches)	30	32.75	34.5
WEIGHT (approximate lbs.)	750	2,800	625
TEMPERATURE RANGE	60° – 80° F.		
HUMIDITY RANGE	40% – 70%		
HEAT DISSIPATION (BTU/hr.)	7,500	10,200	3,000
AIR FLOW (cu. ft./min.)	350	2,300	350
POWER REQUIREMENTS	Supplied by power supply.	208 VAC 60 cps 3 phase 3.8 KVA	208 VAC 3 phase 400 cps regulated 600 W 60 cps unregulated 200 W

Maximum Cable Length Restrictions

1. Central Processor to UNISERVO IIIA Control Cabinet: 380 ft.
2. UNISERVO IIIA Control Cabinet to UNISERVO IIIA Tape Unit: 50 ft.
3. Power Supply to UNISERVO IIIA Tape Unit: 50 ft.

SPURT

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

UP-3900