Mitre Corporation
D-1

THE MITRE CORPORATION
Lexington 73, Massachusetts

Technical Memorandum TM—15 #1

Sheet 1 of 20 Sheets

TITLE:

Subject: Basic XD-1 Information: Introduction to SAGE and XD-1

To: J. D. Porter

From: J. R. Tobey

Date: 5 March 1959

Approved: JHBurrows
James H. Burrows

ABSTRACT

Acknowledgement is made to the System Development Corporation for portions of Field Note 726; and to Philip Bagley, Lincoln Laboratory, for portions of 6M-5754, contained herein.

J. R. Tobey
J. R. Tobey

JRT/hht

DISTRIBUTION LIST

J. R. Tobey (25 copies)

1. This document has been prepared for internal use only. It has not been reviewed by Office of Security Review, Department of Defense, and therefore, is not intended for public release. Further dissemination or reproduction in whole or in part of the material within this document shall not be made without the express written approval of The MITRE Corporation (Technical Information Department).

2. The research reported in this document was supported by the Department of the Air Force under Air Force Contract AF-33 (600) 37882.

~~AD-496~~
TM-15 #1

TABLE OF CONTENTS

Chapter 1.

Introduction to SAGE

1.1 The Meaning of SAGE

The word SAGE is an abbreviation of "Semi-Automatic Ground Environment". The system on which we are working has been given this designation because the AN/FSQ-7 computer enables the functions relating to air defense which are carried out on the ground (observation of radar scopes, correlation of tracks, etc.) to be performed in a semi-automatic fashion. Men are still needed to make most of the decisions, but the computer does the routine calculations.

1.2 Mission of the Air Defense Command

This is a major command of the U. S. Air Force, and has the responsibility of destroying the enemy during an air attack. Generally speaking, there are four steps involved in air defense: (1) Detection - The first step is the detection of all aircraft with the area to be protected. The air defense can be no better than the detection system used.

Some of the tools for detection are:

a) Long range radar (LRI) antennas sweep large areas with their beams and transmit to radar display scopes all objects in the covered area; gap filler radar (GFI) antennas cover a much smaller area and fill in the gaps in the air picture not covered by the LRI's. Height finder radar sets obtain information regarding the altitude of aircraft, to supplement the information given by LRI's and GFI's.

b) Picket ships, bearing powerful radar antennas and transmitters, patrol the waters off the U. S. and are able to detect planes far away from the country, thus extending the range of land based radars. Airborne Early Warning (AEW) planes fly many miles off the shores of the country and supplement the early-warning protection offered by picket ships.

c) The Ground Observer Corps (GOC), composed of civilian volunteers who man observation posts scattered throughout the country, supplies vital information on low flying aircraft.

d) Texas Towers, which are metal platforms erected on the shallow continental shelf off the coast of the U. S., contain radar equipment, and are another link in the early detection and warning of aircraft which are still far away from the country.

(2) Identification - The second step in air defense is the identification, as friendly or hostile, of aircraft which have been detected in the area to be defended. Most civilian and all military planes are required to file, with a central agency, the plans of their flight, stating the time and place of take-off and landing, together with the course which will be flown and other pertinent information. When a plane is detected by radar, GOC or some other means, the course of that plane is immediately telephoned or telegraphed into a central plotting center which compares the observed course of the plane with the flight plans for that area.

If there is a flight plan corresponding to the plane's course, it is assumed that the plane is the friendly aircraft which filed the flight plan. If the course of the plane does not, however, compare with any flight then other means of identification must be used. If an airplane cannot be identified through these or other means, then it becomes necessary for ADU interceptors to be sent to intercept the plane and identify it.

(3) Interception and (4) Destruction - When a plane has been identified as hostile, it becomes necessary to intercept and destroy it in order to protect the United States. This is the end purpose of the Air Defense Project.

1.3 Advantages of Computer for the SAGE Air Defense Project

The SAGE Air Defense Project may easily be likened to a human organism. Radar, GOC, AEW planes, picket ships, etc., are the eyes and ears, or sensing mechanisms, for the project. The computer is the "brain", in that it receives messages from these sensing mechanisms and "remembers" and associates the many varied messages which it receives. The computer, however, makes no decisions about what should be done. These are left to Air Force officers who, at the various display consoles, are shown the air situation picture which the computer has made up, and who then decide on the actions to be taken by pushing buttons or via other means.

The "hands" or means for action of the SAGE project are the interceptors, anti-aircraft batteries, missiles, etc., which according to directions given by the computer and by Air Force officers, actually do the work of destroying enemy attackers.

1.4 The Combat Direction Center

Air Defense Combat Direction Centers have the job of compiling and processing the air movements information within a given area; of displaying to the various consoles pictures depicting aspects of the air picture (e.g., one display might show the tracks of all known friendly planes whereas another display might show the tracks of all planes which haven't yet been identified); and of supplying information to air defense weapons (e.g., interceptors, missiles) on the basis of information supplied to the computer from picket ships, GOC, AEW planes, weather stations, etc., and the decision of Air Force officers.

Direction Centers are equipped with the AN/FSQ-7 computer.

1.5 The AN/FSQ-7 & 8 Computers

These computers are large storage, single address, random access, parallel operations, binary, general purpose, duplex (with the exception of XD-1), digital computers.

Some of the components are:

(1) Core Memory - This consists of tiny ferrite (iron) cores which can be magnetized in either of two planes, thus representing either a binary "0" or a "1". It has a basic access time of six millionths of a second. The great advantage of core memory is its rapid access time together with the fact that there is random access, i.e., you can immediately recall the information from any specified register in core memory without having to start at the first register and going through each one in turn until the correct register is reached.

(2) Drum System - The computer has twelve drums, eleven of these contain six drum fields; one, the RD drum, contains nine drum fields. Each field is capable of storing 2048 bit binary words. The average access time for the drum is ten thousandths of a second. Unlike core memory, there is no random access for the drums, i.e.; the drums are constantly revolving, in order to read or write onto any register on a drum field it is necessary for the computer to read the address of each register and wait until the correct register comes under the read or write head. Although drums do not have as rapid an access time as does core memory, their advantage is in the fact that they can store a great deal more information than can core memory.

(3) Tapes - At present there are six tape drives being used in conjunction with each computer. Although the access time for tapes is slower than that for core memory or drums, one tape reel can store more information than can all of the drums and core memory together. However, tapes will mot be used in actual Direction Center operation. They will only be used while the programs are being checked out, and, afterwards, to store copies of the DCA

program systems on in case some failure causes the programs to be lost from the drums, thus requiring that they be started over from tapes.

(4) Display System - This system received information from the drums, generates displays from the binary information which was prepared by the Central Computer System and then sends the displays to the various consoles. Situation displays cover air movement information and may be compared to plotting boards in manual air defense centers. Digital displays contain statistical type informations, rather than representations of air pictures.

(5) Input System - Inputs to the computer come from several sources: Long Range Radars, Gap Filler Radars; Mark X IFF (Identification Friend-or-Foe); crosstell messages from other Combat Direction Centers; backtell messages from Combat Control Centers; weather stations, AEW planes, GOC, etc., reports entered manually via IBM-type cards; light gun actions, area discriminators; manual input console switches.

(6) Output System - The main outputs, once the computer has received, sorted and computed the inputs, are the making up of situation and digital displays for Air Force officers, and, finally, the making up of messages directing the defensive actions to be taken against hostile aircraft.

(7) Central Computer System - This system performs all of the computations involved in the air defense problem. It consists of the following elements:

a. Core memory - provides a very rapid random access storage medium;

b. Arithmetic element - receives data from core memory, performs necessary calculations on this data, and returns the information to core memory;

c. Instruction control element - directs the step-by-step operations of the arithmetic element; the instruction control element receives and interprets the program instructions and sends to the arithmetic element a series of electrical impulses necessary to perform the given program instruction;

d. Program element - controls transfers of information between different elements of the central computer; transfers instructions from core memory to the instruction control element;

e. Selection control element - selects, and, if necessary, operates, mechanical units involved in IO transfers;

f. Maintenance control unit - used in testing the central computer system, in loading programs into core memory and in determining whether the central computer is responding correctly to errors and alarm conditions;

g. External IO units - technically included in the central computer system are the card reader, card punch, line printer and magnetic tape units.

(8) Warning Light System - the purpose of this system is to notify Air Force officers and technicians at the display console, via neon indicators and/or audible alarms, of air defense situations which require the immediate attention or action; the system is controlled by the Central Computer System.

1.6 The Direction Center Active (DCA) Program

The purpose of the DCA system is to perform all the computational functions involved in air defense. There are, altogether about eighty DCA programs involving some 90,000 instructions. Radar and other sources supply information on the air picture (planes, clouds, storms, etc.,) to the computer. The computer, directed by the DCA programs, masks out such things as clouds or storms, determines which detected objects are actually airplanes, and then calculates the speed and heading of these planes. It then displays these planes symbolically on cathode ray display scopes using special symbols to represent the planes. Speed, heading and other information are also displayed. Air Force officers who man the scopes at the consoles can request, via switches, the computer to perform many other computations so that they may make decisions as to whether a plane is hostile, and if it is, what weapons to use to destroy it. The computer will then compute the course to be used by interceptors and will also compute the return-to-base course to be followed by the interceptors.

All these functions of marking out clouds and storms, tracking planes, determining speed and heading, making up displays, calculating attack and return-to-base courses for interceptors, plus many others, are all performed by the computer at the direction of the DCA air defense program. For a brief description of each DCA program, see document 6M-4469.

1.6.1 DCA Program Classes

The DCA programs are divided into three "classes", according to how often they will be operated in a computer "frame", which is the basic computer program cycle. During a single frame nearly all of the DCA programs will be operated. This means that just about all of the air defense functions - masking out clouds and storms, calculating tracks, making up displays, computing courses for interceptors, etc. - will be repeated during each frame.

The frame, in turn, is divided into three subframes, in terms of time. Most important of the DCA programs, as for example the tracking programs, will be operated once every subframe. These programs, which are in the "A" class, will thus operate three times per frame. Programs which operate only once per frame are in the "B" class. Other programs are operated once every two or more frames, and are called "C" class.

Chapter 2 - Introduction to XD-1

2.1 General

AN/FSQ-7 (XD-1) Combat Direction Centrals are essentially complete data-processing machines designed to handle large amounts of military tactical data associated with a number of aircraft detection and search devices. Because of operational considerations the Combat Direction Centrals (CDC's) are divided into six separate systems, each of which has its own special functions to perform. These systems as shown in figure 2-1 are:

a. Input System
b. Drum System
c. Central Computer System
d. Output System
e. Display System
f. Power Supply and Marginal Checking System

The function of the Input System is to control the flow of information to the Drum System from radar sites and other CDC's. The Drum System is a medium-speed, medium-capacity storage device. It stores slowly recurring intermittent data from the Input System and transfers these data to the Central Computer System. It is the responsibility of the Central Computer System to correlate and process these data and to store the results back in the Drum System. In addition, the Drum System provides storage space, in the form of the auxiliary memory fields, to supplement the storage capacity integrated with the Central Computer System. Data which have been processed by the Central Computer System and stored on the drums are further distributed to the Output and Display Systems. The Output System controls the flow of these processed data to the radar and weapons sites, while the Display System presents these data in visual form to human observers for interpretation and control. The Power Supply and Marginal Checking System provides the a-c and d-c voltages needed for electronic operation and also provides the voltages needed for the marginal checking of CDC's.

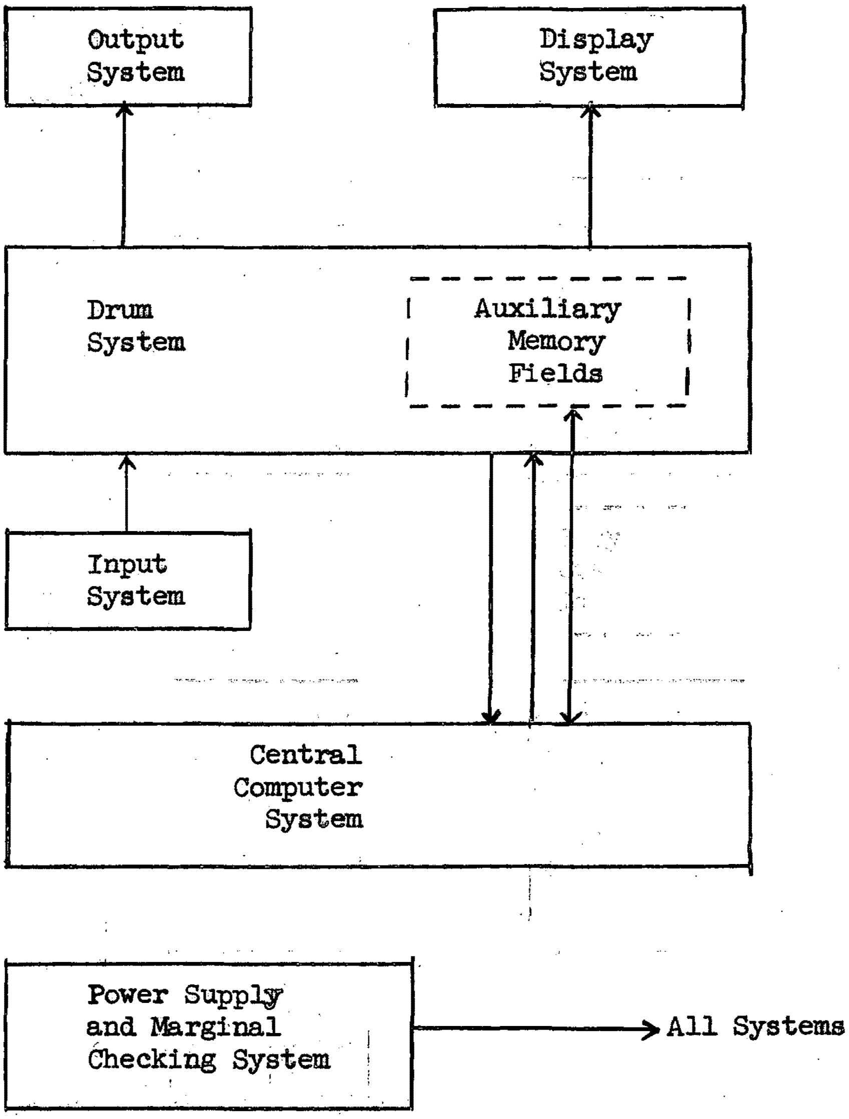


FIGURE 2-1.

Combat Directional Central, Simplified Block Diagram

As shown in figure 2-1, the Central Computer System occupies a central position relative to the flow of data within the CDC. Thus, the Central Computer System receives tactical information from the Input System via the Drum System, processes it, and transmits it to the Output and Display System via the Drum System. In addition to the transfers between the Central Computer System and the Drum System, information may also be transferred between the Central Computer System and various input-output (IO) units. The IO units consist of auxiliary storage devices which are capable of storing data received from, or transferring data into the Central Computer System.

2.2 Function

The Central Computer System operates internally in the binary number system and is controlled by means of a stored program. The Central Computer System (CCS), incorporates an internal memory device capable of storing binary numbers and providing rapid access to any one of these numbers. This memory device is termed magnetic core memory, or more briefly, core memory, and stores all the information which the CCS must use during the execution of its computations.

Information stored in the core memory of the CCS is of two types, instructions and numerical data. An instruction is an order to the CCS which causes it to execute a specific operation such as addition, subtraction, multiplication, etc. The numerical data are the binary numbers representing tactical information and are stored in core memory from the Drum System. Before any processing of this numerical data may be accomplished the CCS must be supplied with a series of instructions describing the exact processing procedure that is to take place. Such a series of instructions is termed a program. It is the function of the CCS to subject the tactical numerical data to processing operations in accordance with the series of instructions, or program, which is stored in core memory. The results of these processing or computational operations are stored back in core memory after the operations are completed and are ultimately transferred to the Drum, Output, and Display Systems, as determined by the program.

2.3 Description

The CCS is functionally divided into six sections or logical elements. Five of these are used for control and calculation, while the sixth is utilized for CCS maintenance and testing. These six elements are:

a. Arithmetic element
b. Instruction control element
c. Maintenance control element

d. Memory element
e. Core memory element
f. Selection and IO control element

These elements are illustrated in figure 2-2.

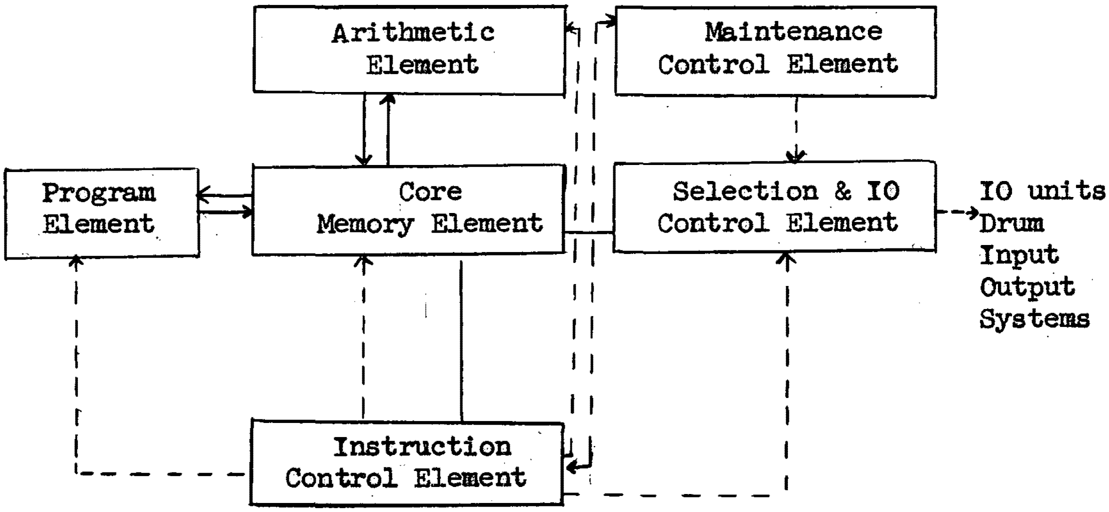

Arithmetic Element
Maintenance Control Element
Program Element
Core Memory Element
Selection & IO Control Element
IO units
Drum
Input
Output
Systems
Instruction Control Element

FIGURE 2-2.

Central Computer System, Simplified Block Diagram

The instruction control, program, and selectionand IO elements act in conjunction with each other to control and co-ordinate the operation of the CCS and associated systems in the execution of the program. This function of co-ordination, or control, may be differentiated into internal control and external control. Internal control refers to the control of operations and calculations taking place within the CCS's, while external control refers to the control of operations in other associated systems. External control deals principally with the co-ordination of the CCS and the Drum System and IO units so as to effect the transfer of information between them.

The arithmetic element provides those circuits necessay for the execution of the arithmetic computations of the CCS program, while the memory element stores the data involved in these computations and also the program itself. The flow of information associated with these calculations is quite straightforward, in that numerical data are transferred out of the memory element to the arithmetic element, where they are subjected to operations of a strictly mathematical nature. After the computations are complete, the resultant numerical data may be transferred from the arithmetic element back into the memory element, where it is stored for future reference. Thus, it may be seen that the computational operation of the CCS is one of continuously circulating data between the memory and arithmetic elements.

The flow of numerical information and the mathematical operations to which this information is subjected are sequenced and controlled by the instruction control and program elements. These elements provide the circuits necessary for internal control of the CCS. During the execution of the CCS program, the instructions of which it is composed are transferred one at a time to the instruction control and program elements. The former element causes the CCS to execute the operations dictated by each instruction as it is received; the latter controls the memory element so as to obtain any additional information required by an instruction, and also sequences the transfer of each consecutive instruction out of core memory.

If information is to be transferred between the CCS and the other systems of AN/FSQ-7 (XD-1) CDC, the transfer is directed by the program. It is the function of the selection and IO control and program elements to co-ordinate and effect this informational transfer; i.e., to perform the external control function. It should be noted that the program element contains circuits associated with both the internal control functions of the instruction control element and the external control functions of the selection and IO control element. The process of transferring information into or out of the memory element from or to input-output devices is called an input-output (IO) operation. This operation is initiated and co-ordinated, under the control of the CCS program, by the selection and IO control element. The program element contains those circuits which are utilized to provide transfer paths into or out of the CCS, and it also dictates the originating point or designation in the memory element of the information involved in the IO operation. The maintenance control element provides controls for use by CCS personnel to start, shut down, and service the CCS.

2.4 Digital Words

All information held in the CCS for control and computation is in binary coded form. This information is handled in groups of 16 or 32 binary digits (bits), such groups being termed binary words. Binary coded numbers as well as instructions and other non-numerical information are usually termed words. The number of bits composing a word is called the word length. The unit of information in the CCS is a 32-bit word composed of two half-words having 16 bits each. These two half-words are distinguished from each other by the terms, right half-word (RHW) and left half-word (LHW). The symbolic layout of a full 32-bit Central Computer System word is shown in table 1.

TABLE 1. COMPOSITION OF A WORD

Left Half-word																Right Half-word															
S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

It can be seen that a 16-bit half-word consists of 15 bits numbered from 1 through 15 plus an S bit which is called the sign bit of the word. The S bit has significance only if the half-word contains numerical information. By definition, if the S bit is 0, the number contained in bits 1 through 15 is positive and in true binary from; if the S bit is 1, the number is negative and is in 1's complement form. When referring to a single bit in a 32-bit word, a special notation is employed. For instance, L8 refers to the 8 bit of the left half-word, and R14 refers to the 14 bit of the right half-word.

2.4.1 Instruction Words

An instruction word is a full 32-bit word containing a binary coded command to the Central Computer System which when decoded and acted upon, will cause the CCS to execute a designated operation. Because of their special importance, the left and right half-words of a full instruction word have been assigned distinctive names. The left half-word is termed the operation portion and the right half-word is termed the address portion. These names are derived from the names of the registers in the CCS in which the left and right half-words are stored while the instruction is being executed.

The operation portion, or left-hand portion of the instruction word, contains a binary coded instruction or command to the CCS describing the operation it is to perform. The S bit is meaningless in the operation portion and bits 1 through 15 may be broken down into groups having their own operational significance. The manner in which the operational portion bits are grouped and the names assigned to these groups is shown in table 2.

TABLE 2. BIT DESCRIPTION OF LEFT HALF-WORD

Bit	LS	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15
Desig-			Index			Class			Variation							
nation		Indicator				Operation		Code					Index Interval			

Bits L1 through L3 are termed the index indicator since they are used to specify one of three index registers. The utilization of an index register, under the control of bits L1 through L3, is called indexing, and provides a means of altering or cycling the Central Computer System program. There are 59 instructions associated with AN/FSQ-7 (XD-1) Combat Direction Centrals. These 59 instructions are grouped into eight classes.

Bits L4 through L6 indicate the class withing which a specified instruction is contained. Within a given class, there may be as many as nine distinct instructions. Bits L7 through L10 are used to specify the particular instruction, called the variation, involved in the class determined by bits L4 through L6. The combination of bits L4 through L10 is termed the operation code of an instruction and completely specifies the instruction by the class and variation. The separate instructions within a class are determined by the variation of the class; the combination of class and variation bits serves to completely identify an instruction. Bits L10 through L15 are termed the index interval and are used to provide additional information required by particular instructions. The representation of the index interval varies with the instruction specified by bits L4 through L10. It should be noted that bit L10 is utilized in both the operation code and the index interval. However, this causes no ambiguity since, by design, the instructions which use the index interval do not require bit L10 for their identification. Thus, when the index interval is utilized, the appropriate instruction is completely specified by only six bits of the operation code instead of the usual seven.

The address portion, or right-hand portion of an instruction word, denotes the location in the memory element or other storage elements of the CCS from which additional information pertinent to the instruction specified by the operation portion may be obtained. This additional information usually takes the form of operands required to execute an instruction specifying a mathematical operation. For example, if the machine is to add, an addend must be obtained. It is the purpose of the address portion to specify the location in the memory element where these operands may be found.

TABLE 3. BIT DESIGNATION OF RIGHT HALF-WORD

Bit	RS	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15
Designation		Memory Unit Selector			Address											

Additional words may be stored in the Drum System. The Drum System of AN/FSQ-7 (XD-1) Combat Direction Centrals is divided into 39 sections, called fields. Each field provides 2048 storage locations, any one of which may be specified by a number from decimal 0 through 2047. It is the purpose of the address portion of the instruction word to designate the location of a word in the CCS memory element or on a particular field of the Drum System. In the case where Drum System words are designated, the index interval bits of the operation portion specify on which field of the drum the word may be found and the address portion specifies the exact location of the word on the field. The layout of the address portion bits are shown in table 3.

Since the address portion does not contain information of a numerical nature, the S bit is meaningless. Bits R1 through R3 are called the memory unit selector, since they specify the core memory unit in which the desired word may be found. Bits R4 through R15 designate the address, since they spcify the exact location in the core memory unit where the desired word is stored.

2.5 XD-1 Instructions

The abbreviations used in the following descriptions of XD-1 instructions are:

L = left

R = right

Acc = accumulator(s)

Cl = clock

Reg. = register(s)

µ = microseconds

Ix = index

Itlk = interlock

I-O = input-output

Prog. = program

Adr = address

Wd = word

Ctr = counter

x = selected memory register

n = number of words, or number of steps

K = interleave factor

The expression, "augmented right half-word" is used to designate the 17-bit expression formed of the contents of bit positions LS, RS-R15.

The "Instruction action" is a brief statement of what happens when the instruction is executed, and of the final contents of the following registers, if they are altered by the instruction:

Register X (the selected memory register)

Accumulators

B Registers

Right A Register

Program Counter

Index Registers

In-Out Word Counter

In-Out Address Counter

OPERATIONS LISTED BY OCTAL OPERATION CODE AND CLASS

Class	Abbreviation	Octal Operation Code	Page	Class	Abbreviation	Octal Operation Code	Page
Miscellaneous:	HLT	000		Store:	FST	324	
	ETR	004			LST	330	
	PER	01-			RST	334	
	CSW	020			STA	340	
	SLR	024			AOR	344	
	LDB	030			ECH	350	
	CMM	040			DEP	360	
	CDM	041		Shift:	DSL	400	
	CMR	042			DSR	404	
	CDR	043			ASL	420	
	CML	044			ASR	424	
	CDL	045			LSR	440	
	CMF	046			RSR	444	
	CDF	047			DCL	460	
	TOB	050			FCL	470	
	TTB	054		Branch	BPX	51-	
Add:	CAD	100			BSN	52-	
	ADD	104			BFZ	540	
	TAD	110			BFM	544	
	ADB	114			BLM	550	
	CSU	130			BRM	554	
	SUB	134		In-Out:	LDC	600	
	TSU	140			SDR	61-	
	CAM	160			SEL	62-	
	DIM	164			RDS	670	
	CAC	170			WRT	674	
Multiply:	MUL	250		Indexing:	XIN	754	
	TMU	254			XAC	764	
	DVD	260			ADX	770	
	TDV	264					

ADD, ADB, ADX, AOR

Name: ADD Abbr. ADD

Instruction action: Add the numbers in the left and right halves of register x to the numbers in L and R Accs respectively; leave the results in Accs.
Put +0 into A Regs.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (104 octal)
L12:	Configuration control
L13-L15:	Overflow control
LS,RS-R15:	Memory address

Execution time (μsec): 12

Name: ADD B REGISTERS TO ACCUMULATOR REGISTERS (ADD B REGISTERS) ADB

Instruction action: Add the numbers in L and R B Regs to the numbers in L and R Accs respectively; leave the results in Accs.
Put +0 into A Regs.

Instruction word layout:

L4-L10:	Operation code (114 octal)
L12:	Configuration control
L13-L15:	Overflow control

Execution time (μsec): 12

Name: ADD INDEX REGISTER ADX

Instruction action: Add the numerical contents (ignoring sign) of the specified Ix Reg to the augmented right half of the instruction word; leave the 17-bit result in R A Reg.
Overflow is possible but it does not affect overflow alarm or overflow sense units.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (770 octal)
LS,RS-R15:	Number

Execution time (μsec): 6

Name: ADD ONE (ADD ONE TO RIGHT) AOR

Instruction action: Add the octal number 0.00001 to the right half word in register x; leave the result in R Acc and in the right half of register x.
If the augmented configuration is selected, clear L Acc 1-15.
Put into A Regs the original contents of Reg x.

(Cont.)

AOR (Continued)

Instruction word layout:	L1-L3:	Index register selection
	L4-L10:	Operation code (344 octal)
	L12:	Configuration control
	L13-L15:	Overflow control
	LS,RS-R15:	Memory address
Execution time (µsec):	18	

Name: SHIFT ACCUMULATORS LEFT (ACCUMULATORS SHIFT LEFT) ASL

Instruction action: Shift the numbers in the L and R Accs (with the exception of the contents of the sign bit positions) n places to the left, the bit positions thus made vacant being filled with Acc sign bits, and bits shifted left out of Accs bit position 1 being lost.

Instruction word layout:	L4-L10:	Operation code (420 octal)
	R10-R15:	Amount of shift (number of places, n)
Execution time (µsec):	$3.5 + \frac{n}{2}$ (minimum is 6.0)	

Name: SHIFT ACCUMULATORS RIGHT (ACCUMULATORS SHIFT RIGHT) ASR

Instruction action: Shift the numbers in the L and R Accs (with the exception of the contents of the sign bit positions) n places to the right, the bit positions thus made vacant being filled with Acc sign bits, and bits shifted right out of Accs bit position 15 being lost.

Instruction word layout:	L4-L10:	Operation code (424 octal)
	R10-R15:	Amount of shift (number of places, n)
Execution time (µsec):	$3.5 + \frac{n}{2}$ (minimum is 6.0)	

Name: BRANCH ON MINUS (BRANCH ON FULL MINUS) BFM

Instruction action: If the numbers in Accs are both negative, put into R A Reg the original contents of the Prog Ctr plus 1, and put into the Prog Ctr the augmented right half word of the instruction.
If the numbers in Accs are not both negative, put +0 into A Regs.

Instruction word layout:	L4-L10:	Operation code (544 octal)
	LS,RS-R15:	Memory address
Execution time (µsec):	6	

BFZ, BLM, BPX

Name: BRANCH ON ZERO (BRANCH ON FULL ZERO) BFZ

Instruction action: If the numbers in Accs are both zero (any combination of +0 and/or -0), put into R A Reg the original contents of the Prog Ctr plus 1, and put into the Prog Ctr the augmented right half word of the instruction.

If the numbers in Accs are not both zero, put +0 into A Regs.

Instruction word layout: L4-L10: Operation code (540 octal)
LS,RS-R15: Memory address

Execution time (µsec): 12

Name: BRANCH ON LEFT MINUS BLM

Instruction action: If the number in L Acc is negative, put into R A Reg the original contents of the Prog Ctr plus 1, and put into the Prog Ctr the augmented right half word of the instruction.

If the number in L Acc is positive, put +0 into A Regs.

Instruction word layout: L4-L10: Operation code (550 octal)
LS,RS-R15: Memory address

Execution time (µsec): 6

Name: BRANCH AND INDEX (BRANCH ON POSITIVE INDEX) BPX

Instruction action: If the contents of the specified Ix Reg (1, 2, 4, or 5) are positive, put into R A Reg the original contents of the Prog Ctr plus 1, and put into the Prog Ctr the augmented right half word of the instruction; subtract the positive number (termed the "index interval") given by the contents of bit positions L10-L15 of the instruction word from the contents of the specified Ix Reg; if the result of this subtraction is positive, put the result in the specified Ix Reg, otherwise set the Ix Reg to its special representation of -0.

If the contents of the specified Ix Reg are negative, put +0 into A Regs.

If Ix Reg 0 or 3 is specified, put in R A Reg the original contents of the Prog Ctr plus 1, and put in Prog Ctr the augmented right half word of the instruction.

If Ix Reg 6 or 7 is specified, put +0 into A Regs.

(Cont.)

BPX, BRM, BSN, CAC

BPX (Continued)

Instruction word layout:

L1-L3:	Index register selection
L4-L9:	Operation code (51 octal)
L10-L15:	Index interval
LS,RS-R15:	Memory address

Execution time (µsec): 6

Name: BRANCH ON RIGHT MINUS BRM

Instruction action:

If the number in R Acc is negative, put into R A Reg the original contents of the Prog Ctr plus 1, and put into Prog Ctr the augmented right half word of the instruction.

If the number in R Acc is positive, put +0 into A Regs.

Instruction word layout:

L4-L10:	Operation code (554 octal)
LS,RS-R15:	Memory address

Execution time (µsec): 6

Name: SENSE (BRANCH ON SENSE) BSN

Instruction action:

Sense the unit specified by the sense code. (See Sense Codes.)

If the branch condition is present, put into R A Reg the original contents of the Prog Ctr plus 1, and put into the Prog Ctr the augmented right half word of the instruction.

If the branch condition is not present in the specified unit, or if a non-existent unit is specified, put +0 into A Regs.

Instruction word layout:

L4-L9:	Operation code (52 octal)
L10-L15:	Sense code
LS,RS-R15:	Memory address

Execution time (µsec): 12

Name: CLEAR AND ADD CLOCK CAC

Instruction action:

Provided that the address part of the instruction is a test memory address (377760 thru 377777 octal), replace the contents of R Acc by the contents of the Clock Register, and put +0 in L Acc.

(Cont.)

If the address part of the instruction word is not a test memory address, it may cause a memory parity.
Put +0 into A Regs.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (170 octal
LS,RS-R15: Test memory address ($377,777_8$ by convention)

Execution time (μsec): 12

Name: CLEAR AND ADD CAD

Instruction action: Replace the contents of Accs by the contents of register x.
Put +0 into A Regs.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (100 octal
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: CLEAR AND ADD MAGNITUDE CAM

Instruction action: Replace the contents of Accs by the positive magnitudes of the contents of the right and left halves of register x.
Put +0 into A Regs.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (160 octal)
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: COMPARE AND DIFFERENCE FULL WORDS CDF

Instruction action: Compa re the contents of Accs with the contents of register x; if the words thus compared are not identical bit-for-bit, add 1 to Prog Ctr (in order to skip the next instruction).
Subtract the numbers in the L and R Accs from the numbers in the left and right halves of register x, respectively; leave results in Accs.
Put +0 into A Regs.

(Cont'd)

Instruction word layout:	L1-L3:	Index register selection
	L4-L12:	Operation code (047 octal)
	L13-L15:	Overflow control
	LS,RS-R15:	Memory address
Execution time (µsec)	12	

Name: COMPARE AND DIFFERENCE LEFT HALF WORDS CDL

Instruction action:

Compare the contents of L Acc with the left half word in register x; if the half words thus compared are not identical bit-for-bit, add 1 to Prog Ctr (in order to skip the next instruction).

Subtract the numbers in the L and R Accs from the number in the left and right halves of register x, respectively; leave the results in Accs.

Put +0 into A Regs.

Instruction word layout:	L1-L3:	Index register selection
	L4-L12:	Operation code (045 octal)
	L13-L15:	Overflow control
	LS,RS-R15:	Memory address
Execution time (µsec):	12	

Name: COMPARE AND DIFFERENCE MASKED BITS CDM

Instruction action:

For each "1" bit in B Regs, compare the contents of the corresponding bit position of Accs with the contents of the corresponding bit position of register x; if not all of the bit pairs thus compared are identical, add 1 to Prog Ctr (in order to skip the next instruction); for the special case where the B Regs contain +0, do not add to Prog Ctr.

Change the contents of Accs to the value obtained by the following steps:

1. Logical multiply the 1's complement of the contents of Accs by contents of B Regs.
2. Logical add the contents of register x to the 1's complement of the contents of B Regs.
3. Add, in normal dual arithmetic, the results of steps 1 and 2 and place results in Accs. (This, in effect, subtracts the numbers in Accs, as defined by a mask, from the corresponding numbers in register x.)

Put +0 in A Regs.

(Cont.)

CDM (Continued)

Instruction word layout:
L1-L3: Index register selection
L4-L12: Operation code (041 octal)
L13-L15: Overflow control
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: COMPARE AND DIFFERENCE RIGHT HALF WORDS CDR

Instruction action:

Compare the contents of R Acc with the right half word in register x; if the half words thus compared are not identical bit-for-bit, add 1 to Prog Ctr (in order to skip the next instruction).

Subtract the numbers in L and R Accs from the numbers in the left and right halves of register x, respectively; leave the results in Accs.

Put +0 into A Regs.

Instruction word layout:
L1-L3: Index register selection
L4-L12: Operation code (043 octal)
L13-L15: Overflow control
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: COMPARE FULL WORDS CMF

Instruction action:

Compare the contents of Accs with the contents of register x; if the words thus compared are not identical bit-for-bit, add 1 to Prog Ctr (in order to skip the next instruction).

Put +0 into A Regs.

Instruction word layout:
L1-L3: Index register selection
L4-L12: Operation code (046 octal)
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: COMPARE LEFT HALF WORDS CML

Instruction action:

Compare the contents of L Acc with the left half word in register x; if the half words thus compared are not identical bit-for-bit, add 1 to Prog Ctr (in order to skip the next instruction).

Put +0 into A Regs.

Instruction word layout:
L1-L3: Index register selection
L4-L12: Operation code (044 octal)
LS,RS-R15: Memory address

Execution time (μsec): 12

CMM, CMR, CSU

Name: COMPARE MASKED WORDS CMM

Instruction action: For each "1" bit in B Regs, compare the contents of the corresponding bit position of Accs with the contents of the corresponding bit position of register x; if not all of the bit pairs thus compared are identical, add 1 to Prog Ctr (in order to skip the next instruction); for the special case where the B Regs contain +0, do not add to Prog Ctr.

For each "0" bit in B Regs, place a "1" in the corresponding bit positions of Accs.

Put +0 into A Regs.

Instruction word layout:	L1-L3:	Index register selection
	L4-L12:	Operation code (040 octal)
	LS,RS-R15:	Memory address

Execution time (μsec): 12

Name: COMPARE RIGHT HALF WORDS CMR

Instruction action: Compare the contents of R Acc with the right half word in register x; if the half words thus compared are not identical bit-for-bit, add 1 to Prog Ctr (in order to skip the next instruction).

Put +0 into A Regs.

Instruction word layout:	L1-L3:	Index register selection
	L4-L12:	Operation code (042 octal)
	LS,RS-R15:	Memory address

Execution time (μsec): 12

Name: CLEAR AND SUBTRACT CSU

Instruction action: Replace the contents of Accs by the complement of the contents of register x.

Put +0 into A Regs.

Instruction word layout:	L1-L3:	Index register selection
	L4-L10:	Operation code (130 octal)
	LS,RS-R15:	Memory address

Execution time (μsec): 12

CSW, DCL, DEP

Name: CLEAR AND SUBTRACT WORD COUNTER (COPY WORD COUNTER) CSW

Instruction action:

If the normal configuration is selected, replace the contents of R Acc by the contents of the right 16 bits of the I-0 Wd Ctr. (RS-R15).

If the augmented configuration is selected, replace the contents of Acc LS and R Acc by the entire contents of the I-0 Wd Ctr.

Instruction word Layout:
L4-L10: Operation code (020 octal)
L12: Configuration control

Execution time (μsec): 6

Name: CYCLE LEFT (DUAL CYCLE LEFT) DCL

Instruction action:

Shift the entire contents of L Acc and L B Reg and the entire contents of R Acc and R B Reg n places to the left, the bits being shifted left out of the sign bit position of each Acc being reinserted into bit position 15 of the associated B Reg so that no bits are lost.

Instruction word layout:
L4-L10: Operation code (460 octal)
R10-R15: Amount of shift (number of places, n)

Execution time (μsec): $3.5 + \frac{n}{2}$ (minimum is 6.0)

Name: DEPOSIT (SELECTIVE STORE) DEP

Instruction action:

For each "1" bit in B Regs, replace the contents of the corresponding bit position of register x by the contents of the corresponding bit position of Accs; then replace the contents of Accs by the new contents of register x.

Put into each bit position of A Regs the logical sum of the contents of the corresponding bit positions of B Regs and register x. The table of logical sums is:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (360 octal)
LS,RS-R15: Memory address

Execution time (μsec): 18

DIM, DSL, DSR

Name: DIFFERENCE MAGNITUDES (DIFFERENCE OF MAGNITUDES) DIM

Instruction action: Subtract the positive magnitudes of the numbers in the left and right halves of register x from the positive magnitudes of the numbers in L and R Accs respectively; leave the results in Accs.
Put +0 into A Regs.
Put the original contents of Accs into B Regs.

Instruction word layout:

L1-L13	Index register selection
L4-L10:	Operation code (164 octal)
L12:	Configuration control
LS,RS-R15:	Memory address

Execution time (μsec): 12

Name: SHIFT LEFT (DUAL SHIFT LEFT) DSL

Instruction action: Shift the number in L Acc and L B Reg and the number in R Acc and R B Reg (with the exception of the contents of the sign bit positions of the Accs) n places to the left, the bit positions thus made vacant being filled with Acc sign bits, and bits shifted left out of Acc bit position 1 being lost.

Instruction word layout:

L4-L10:	Operation code (400 octal)
R10-R15:	Amount of shift (number of places, n)

Execution time (μsec): $3.5 + \frac{n}{2}$ (minimum is 6.0)

Name: SHIFT RIGHT (DUAL SHIFT RIGHT) DSR

Instruction action: Shift the number in L Acc and L B Reg and the number in R Acc and R B Reg (with the exception of the contents of the sign bit positions of the Accs) n places to the right, the bit positions thus made vacant being filled with Acc sign bits, and the bits shifted right out of B Regs bit position 15 being lost.

Instruction word layout:

L4-L10:	Operation code (404 octal)
R10-R15:	Amount of shift (number of places, n)

Execution time (μsec): $3.5 + \frac{n}{2}$ (minimum is 6.0)

DVD, ECH, ETR

Name: DIVIDE DVD

Instruction action: Divide the number in the L Acc and L B Reg and the number in the R Acc and R B Reg by the numbers in the left and right halves, respectively, of register x; leave the 16 numerical bits of both quotients in B Regs, and remainders in Accs, the signs of the remainders also being the signs of the respective quotients.

If the quotient digit in R B Reg 15 is a "1" put into R A Reg the negative magnitude of the number in the right half of register x; if a "0", the positive magnitude.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (260 octal)
LS,RS-R15:	Memory address

Execution time (μsec): 51

Name: EXCHANGE ECH

Instruction action: Exchange the contents of Accs with the contents of register x.

Put into A Regs the original contents of register x.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (350 octal)
LS,RS-R15:	Memory address

Execution time (μsec): 18

Name: EXTRACT (LOGICAL MULTIPLY) ETR

Instruction action: For each bit position of register x which contains a binary "0", set to zero the corresponding bit in the L or R Acc.

Put into A Reg the original contents of register x.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (004 octal)
LS,RS-R15:	Memory address

Execution time (μsec): 12

FCL, FST, HLT, LDB, LDC

Name: CYCLE ACCUMULATORS LEFT (FULL CYCLE LEFT) FCL

Instruction action: Shift the entire contents of L and R Accs n places to the left, the bits shifted left out of the sign bit position of each Acc being reinserted into bit position 15 of the OTHER Acc so that no bits are lost.

Instruction word layout:
L4-L10: Operation code (470 octal)
R10-R15: Amount of shift (number of places, n)

Execution time (μsec): $3.5 + \frac{n}{2}$ (minimum is 6.0)

Name: STORE (FULL STORE) FST

Instruction action: Replace the contents of register x by the contents of Accs.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (324 octal)
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: PROGRAM STOP (HALT) HLT

Instruction action: If the I-O Interlock is on, wait until it is turned off.
Add 1 to the contents of the Prog Ctr; stop the computer

Instruction word layout: L4-L10: Operation code (000 octal)

Execution time (μsec): 12

Name: LOAD B REGISTERS LDB

Instruction action: Replace the contents of B Regs by the contents of register x.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (030 octal)
LS,RS-R15: Memory address

Execution time (μsec): 12

Name: LOAD INPUT-OUTPUT ADDRESS COUNTER (LOAD ADDRESS COUNTER) LDC

Instruction action: If I-O Interlock is on, wait until it is turned off.
Put the augmented right half word of this instruction (suitably modified if an Ix Reg is specified) into the I-O Adr Ctr.

(Cont.)

Instruction word layout: L1-L3: Index register selection
L4-L10: Operation code (600 octal)
LS,RS-R15: Memory address

Execution time (µsec); 6

Name: LEFT ELEMENT SHIFT RIGHT LSR

Instruction action: Shift the number in L Acc and L B Regs (with the exception of the contents of the sign bit position of L Acc) n places to the right, the bit positions thus made vacant being filled with Acc sign bits, and the bits shifted right out of L B Reg bit position 15 being lost.

Instruction word layout: L4-L10: Operation code (440 octal)
R10-R15: Amount of shift (number of places, n)

Execution time (µsec): $3.5 + \frac{n}{2}$ (minimum is 6.0)

Name: LEFT STORE LST

Instruction action: Replace the left half-word in register x by the contents of L Acc.
Put into A Regs the original contents of register x.

Instruction word layout: L1-L3: Index register selection
L4-L10: Operation code (330 octal
LS,RS-R15: Memory address

Execution time (µsec): 18

Name: MULTIPLY MUL

Instruction action: Multiply the numbers in L and R Accs by the numbers in the left and right halves, respectively, of register x; leave the results in Accs and B Regs.
The final contents of bit position 15 of the L and R B Regs are identical to the contents of the sign bit positions of the L and R Accs respectively.
Put into A Regs the magnitudes of the numbers in the left and right halves of register x.

Instruction word layout: L1-L3: Index register selection
L4-L10: Operation code (250 octal
LS,RS-R15: Memory address

Execution time (µsec): 16.5

PER, RDS, RSR

Name: OPERATE (PERFORM) PER

Instruction action: Activate the unit specified by the Operate Unit code. (See Operate Unit Codes.) If the Operate Unit code specifies a non-existent unit, take no action.

Instruction word layout:
L4-L9: Operation code (01 octal)
L10-L15: Operate unit

Execution time (μsec): 12

Name: READ RDS

Instruction action: If I-O Interlock is on, wait until it is turned off.
Provided that a suitable in-out unit has been previously selected, and that it is ready to operate, initiate the transfer of the number of words specified by the augmented right half word of this instruction (suitably modified if an Ix Reg is specified): the words will be transferred from the currently selected input-output unit to a block of consecutive registers in memory starting at the address contained in the I-O Adr Ctr.
Turn on the I-O Interlock.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (670 octal)
L13-L15: Interleave (used with address modes only
LS,RS-R15: Length of transfer (number of words)

Execution time (μsec): 6

Name: RIGHT ELEMENT SHIFT RIGHT RSR

Instruction action: Shift the number in R Acc and R B Reg (with the exception of the contents of the sign bit position of R Acc) n places to the right, the bit positions thus made vacant being filled by Acc sign bits, and the bits shifted right out of R B Reg bit position 15 being lost.

Instruction word layout:
L4-L10: Operation code (444 octal)
R10-R15: Amount of shift (number of places, n)

Execution time (μsec): $3.5 + \frac{n}{2}$ (minimum is 6.0)

Name: RIGHT STORE RST

Instruction action:

Replace the right half-word in register x by the contents of R Acc.

Put into L A Regs the original contents of left half of register x and clear the right half.

Instruction word layout:

L1-L3	Index register selection
L4-L10:	Operation code (334 octal)
L12:	Configuration control
LS,RS-R15:	Memory address

Execution time (µsec): 18

Name: SELECT DRUM SDR

Instruction action:

If I-O Interlock is on, wait until it is turned off.

Select the specified drum field for subsequent reading or writing in the specified mode, beginning at drum register x if in address mode; deselect any previously selected drum field or in-out unit. (See Drum Field and Mode Selection Codes).

Instruction word layout:

L1-L3:	Index register selection (used with address and identity modes only)
L4-L9:	Operation code (61 octal)
R1, L10-L15:	Drum field and mode
R5-R10:	
R11-R15:	Identity (used with identity modes
R14-R15:	only)
R5-R15:	Drum address (used with address mode only)

Execution time (µsec): 12

SEL, SLR

Name: SELECT SEL

Instruction action: If I-O Interlock is on, wait until it is turned off.
Select for subsequent input-output operations the specified input-output unit; deselect any previously selected drum field or in-out unit. (See Input-Output Unit Selection Codes.)

Instruction word layout:
L4-L9 Operation code (62 octal)
L10-L15: In-out unit code

Execution time (μsec): 12

Name: SHIFT LEFT AND ROUND SLR

Instruction action: Shift the number in L Acc and L B Reg and the number in R Acc and R B Reg (with the exception of the contents of the sign bit positions of the Accs) n places to the left, the bit positions thus made vacant being filled with Acc sign bits, and bits shifted left out of Accs bit position 1 being lost.
Round off the results to 15 numerical bits in each Acc. Roundoff of a positive number is accomplished by adding 1 to the contents of Acc 15 if the S bit of the corresponding B Reg contains a 1. Roundoff of a negative number is accomplished by subtracting 1 from the contents of Acc 15 if the S bit of the corresponding B Reg contains a 0.
Put +0 into A Regs and B Regs.

Instruction word layout:
L4-L10: Operation code (024 octal)
L13-L15: Overflow control
R10-R15: Amount of shift (number of places, n)

Execution time (μsec): $5.5 + \frac{n}{2}$ (minimum is 6.0)

STA, SUB, TAD

Name: STORE ADDRESS STA

Instruction action: If the normal configuration is selected, replace the right half word in register x by the contents of the R A Reg RS-R15.
If the augmented configuration is selected, replace the contents of LS and RS-R15 of register x by the contents of R A Reg LS (Sic!) and RS-R15.

Instruction word layout:	L1-L3:	Index register selection
	L4-L10:	Operation code (340 octal)
	L12:	Configuration control
	LS,RS-R15:	Memory address

Execution time (μsec): 18

Name: SUBTRACT SUB

Instruction action: Subtract the numbers in the left and right halves of register x from the numbers in L and R Accs respectively; leave the results in Accs.
Put +0 into A Regs.

Instruction word layout:	L1-L3:	Index register selection
	L4-L10:	Operation code (134 octal)
	L12:	Configuration control
	L13-L15:	Overflow control
	LS,RS-R15:	Memory address

Execution time (μsec): 12

Name: TWIN AND ADD TAD

Instruction action: Add the number in the left half of register x to the numbers in the L and R Accs; leave the results in Accs.
Put +0 into A Regs.

Instruction word layout:	L1-L3:	Index register selection
	L4-L10:	Operation code (110 octal)
	L12:	Configuration control
	L13-L15:	Overflow control
	LS,RS-R15:	Memory address

Execution time (μsec): 12

TDV, TMU, TOB

Name: TWIN AND DIVIDE — TDV

Instruction action: Divide the number in the L Acc and L B Reg and the number in the R Acc and R B Reg by the number in the left half of register x; leave the 16 numerical bits of both quotients in B Regs, and remainders in Accs, the signs of the remainders also being the signs of the respective quotients.

If the quotient digit in R B Reg 15 is a "1", put into R A Reg the negative magnitude of the number in the left half of register x; if a "0", the positive magnitude.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (264 octal)
LS,RS-R15: Memory address

Execution time (μsec): 51

Name: TWIN AND MULTIPLY — TMU

Instruction action: Multiply the numbers in L and R Accs by the number in the left half of register x; leave the results in Accs and B Regs.

The final contents of bit position 15 of the L and R B Regs are identical to the contents of the sign bit positions of the L and R Accs respectively.

Put into R A Reg the magnitude of the number in the left half of register x.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (254 octal)
LS,RS-R15: Memory address

Execution time (μsec): 16.5

Name: TEST ONE BIT — TOB

Instruction action: Add to Prog Ctr the contents of the designated bit position of register x; if bit thus sensed is a "1", the next instruction will be skipped.

Instruction word layout:
L1-L3: Index register selection
L4-L10: Operation code (050 octal)
L11-L15: Bit position (0 thru 31 decimal)
LS,RS-R15: Memory address

Execution time (μsec): 12

TSU, TTB

Name: TWIN AND SUBTRACT TSU

Instruction action: Subtract the number in the left half of register x from the numbers in and L and R Accs; leave the results in Accs. Put +0 into A Regs.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (140 octal)
L12:	Configuration control
L13-L15:	Overflow control
LS,RS-R15:	Memory address

Execution time (µsec): 12

Name: TEST TWO BITS TTB

Instruction action: If the rightmost bit position of the designated pair of bit positions is other than 0 or 16 decimal, add to Prog Ctr the contents of the designated pair of bit positions of register x; if the bit pair thus sensed is binary 01, 10, or 11, then the next 1, 2 or 3 instructions respectively will be skipped.

If the rightmost bit position of the designated pair of bit positions is either 0 or 16 decimal, add to Prog Ctr the contents only of the single designated bit position of register x; if the bit thus sensed is a "1", the next instruction will be skipped.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (054 octal)
L11-L15:	Rightmost bit position (0 thru 31 decimal)
LS,RS-R15:	Memory address

Execution time (µsec): 12

WRT, XAC

Name: WRITE WRT

Instruction action:

If I-0 Interlock is on, wait until it is turned off.

Provided that a suitable in-out unit has been previously selected, and that it is ready to operate, initiate the transfer of the number of words specified by the augmented right half word of this instruction (suitably modified if an Ix Reg is specified); the words will be transferred to the currently selected input-output unit from a block of consecutive registers in memory starting at the address contained in the I-0 Adr Ctr.

Turn on the I-0 Interlock.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (674 octal)
L13-L15:	Interleave (used with address modes only)
LS,RS-R15:	Length of transfer (number of words)

Execution time (µsec): 6

Name: RESET INDEX REGISTER FROM RIGHT ACCUMULATOR XAC

Instruction action:

If the normal configuration is selected, replace the numerical contents of the specified Ix Reg by the contents of the R Acc and set the Ix Reg sign positive ("0").

If the augmented configuration is selected and the contents of L Acc S is "0" replace the entire contents of the specified Ix Reg by the contents of the L Acc S and the R Acc.

If the augmented configuration is selected and the contents of L Acc S is "1", set the contents of the specified IX Reg to its special representation of -0.

The R Acc cannot be used as an Ix Reg with this instruction.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (764 octal)
L12:	Configuration control

Execution time (µsec): 6

XIN

Name: RESET INDEX REGISTER (RESET INDEX REGISTER FROM INSTRUCTION) XIN

Instruction action:

If the contents of LS of the instruction word is "0", replace the entire contents of the specified Index Register by the contents of the augmented right half of the instruction word.

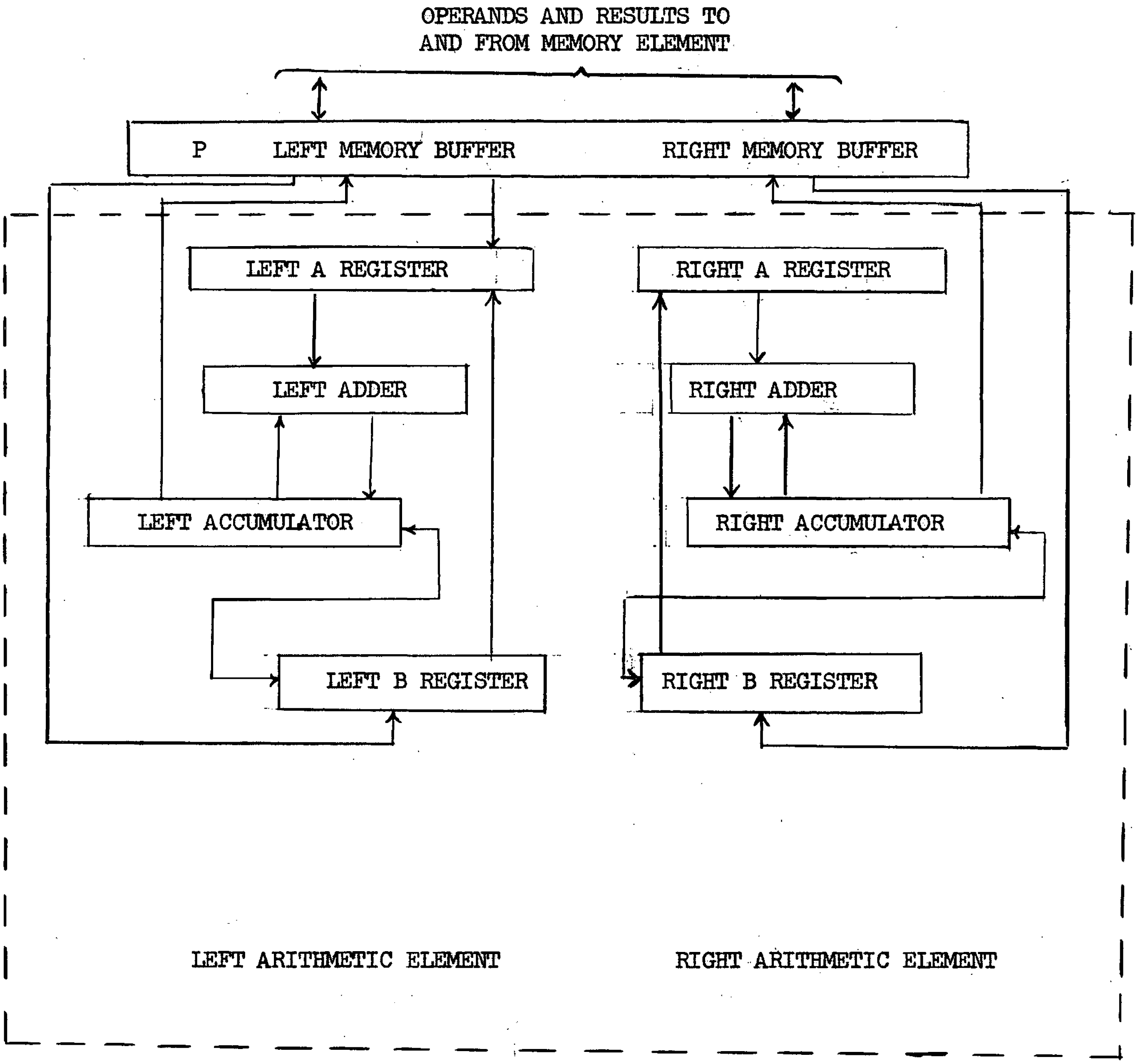
If the contents of LS of the instruction word is a "1", set the contents of the specified Ix Reg to its special representation of -0.

The R Acc cannot be used as an Ix Reg with this instruction.

Instruction word layout:

L1-L3:	Index register selection
L4-L10:	Operation code (754 octal)
LS,RS-R15:	Number

Execution time (μsec): 6



NOTE:
COMMANDS NOT SHOWN

APPENDIX A

FIGURE 2-3. ARITHMETIC ELEMENT INFORMATION FLOW

Mitre Lib.
D-19

THE MITRE CORPORATION
Lexington 73, Massachusetts

Technical Memorandum TM—15 #2
Sheet 1 of 16 Sheets

TITLE:

Subject: BASIC XD-1 INFORMATION: INTRODUCTION TO THE BINARY AND OCTAL NUMBERING SYSTEM

To: J. D. Porter

From: J. R. Tobey

Date: 27 April 1959

Approved: J. H. Burrows

ABSTRACT

Acknowledgement is made to Lincoln Laboratory for permission to reproduce Lincoln Manual No. 13.

DISTRIBUTION LIST.

J. R. Tobey (1 copy)

TM-15 no. 2

1. This document has been prepared for internal use only. It has not been reviewed by Office of Security Review, Department of Defense, and therefore, is not intended for public release. Further dissemination or reproduction in whole or in part of the material within this document shall not be made without the express written approval of The MITRE Corporation (Technical Information Department).

2. The research reported in this document was supported by the Department of the Air Force under Air Force Contract AF-33 (600) 37882.

TABLE OF CONTENTS

~~TM-15~~ #2

INDEX TO TABLES

INTRODUCTION TO THE BINARY AND OCTAL NUMBERING SYSTEMS

I. NUMBERING SYSTEMS

Numbering systems may be classified by the total number of different symbols used in counting. This total number is called the base of the system.

A. Decimal

The decimal system is a number scheme based on ten different symbols – 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. This system probably was developed because primitive men used their ten fingers as a means of keeping tally of a number of objects. Many young children still use this method when first learning to count.

B. Octal

If primitive man had been born with four digits on each hand he might well have counted in the octal system which has eight symbols – 0, 1, 2, 3, 4, 5, 6, and 7. Some modern computing machines require information to be fed to them in the octal system and in turn give out the results in the octal system.

C. Duodecimal

Another numbering system which has been suggested is the duodecimal system which is based on twelve different symbols – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, X and Ɛ – the latter two symbols are named dec and elf. This system allows certain common fractions such as 1/3, 1/4, 2/3, 3/4, 1/6, etc., to be expressed in a simpler way than in the decimal system. Some mathematical tables have been tabulated in this system.

D. Binary

The number system which is most widely used in digital computing machines is the binary system which has only two symbols – 0 and 1. The reason for this is that many devices which are used in electronic circuits have two different electrical states during operation. Thus, a relay or switch may be on or off; a vacuum tube may be conducting or nonconducting; a crystal diode conducts very well in one direction but becomes a high resistance to current in the opposite direction. It appears then that the natural counting scheme for electronic circuits is the binary system.

II. DECIMAL SYSTEM – REVIEW

Inasmuch as most people are familiar with the decimal system of counting through everyday experience, it will be helpful to review its symbol-structure as an aid to the understanding of the binary system.

A. Counting

Since the decimal system has ten symbols (including 0), one may count up to nine using only one symbol at a time. (Table I, column 1.)

In order to count past nine, combinations containing two symbols each are used. This allows

TABLE I

DECIMAL NUMBER COMBINATIONS

1	2	3	4	5	6	7
0	10	20	90	100	190	990
1	11	21	91	101	191	991
2	12	22	92	102	192	992
3	13	23	93	103	193	993
4	14	24	94	104	194	994
5	15	25	95	105	195	995
6	16	26	96	106	196	996
7	17	27	97	107	197	997
8	18	28	98	108	198	998
9	19	29	99	109	199	999

us to count up to ninety-nine. The combinations are selected in an orderly way. The symbol "1" is first used in conjunction with all the symbols (Table I, column 2), next the symbol "2" is used in conjunction with all the symbols. This process is continued until all possible combinations have been exhausted, which occurs at ninety-nine. To count further, three symbols must be used. Proceeding in a similar fashion, the symbol "1" is used together with all the previously developed two-symbol combinations. This allows us to go to 199. Proceeding in this fashion we discover that all combinations of three symbols are exhausted at nine-hundred and ninety-nine. Higher numbers require combinations containing four or more symbols. In general, if one uses combinations containing m decimal symbols it is possible to count up to, but not including, 10^m. With three decimal symbols (m = 3), $10^3 = 1000$, so the highest number that we can count up to is 999. In algebraic notation, if M represents the highest number, $M = 10^m - 1$.

TABLE II

POWERS OF TEN

$10^0 = 1$	$10^0 = 1$
$10^1 = 10$	$10^{-1} = 0.1$
$10^2 = 100$	$10^{-2} = 0.01$
$10^3 = 1{,}000$	$10^{-3} = 0.001$
$10^4 = 10{,}000$	$10^{-4} = 0.0001$
$10^5 = 100{,}000$	$10^{-5} = 0.00001$
$10^6 = 1{,}000{,}000$	$10^{-6} = 0.000001$

B. Powers of Ten

The power of a number "N", it may be remembered, is the number of times "N" is multiplied together. A partial table of powers of 10 is given in Table II, where negative powers are defined by $10^{-x} = 1/10^x$.

Since the base of the decimal system is ten, all numbers may be expressed as a sum of powers of ten (each power being multiplied by an appropriate number). For example, take the number 1063.401. If we examine each digit we find this number may be considered thus:

$$\begin{array}{cccccccccccccc}
 & 1 & & 0 & & 6 & & 3 & & 4 & & 0 & & 1 \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 & (1\times10^{3}) & + & (0\times10^{2}) & + & (6\times10^{1}) & + & (3\times10^{0}) & + & (4\times10^{-1}) & + & (0\times10^{-2}) & + & (1\times10^{-3}) \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
= & 1000 & & 0 & & 60 & & 3 & & 0.4 & & 0 & & 0.001
\end{array}$$

Adding these numbers we get:

```
1000.
 000.
  60.
   3.
   0.4
   0.00
   0.001
--------
1063.401
```

III. BINARY SYSTEM

A. Introduction

A binary symbol is often called a "bit", thus a seven-bit number is a binary number containing seven binary symbols. As mentioned before, the binary system uses only two symbols – 0 and 1. With such brevity of symbols it is apparent that larger combinations of symbols will be necessary to express most numbers than if one were using the decimal system which has ten different symbols to use. As a matter of fact, it will be seen that with a combination containing n binary symbols it is possible to count up to, but not including, 2^n. In algebraic notation, if N is the highest number, $N = 2^n - 1$. Thus, with three binary symbols ($n = 3$) $2^3 = 8$, so that the highest number which may be expressed with three binary symbols is $N = 8 - 1 = 7$.

TABLE III
RATIO OF BINARY SYMBOLS TO DECIMAL SYMBOLS

Decimal Number	Binary Number	Decimal Symbols Used	Binary Symbols Used	Ratio (Binary to Decimal)
1	1	1	1	1
2	10	1	2	2
4	100	1	3	3
8	1000	1	4	4
10	1010	2	4	2
16	10000	2	5	2-1/2
32	100 000	2	6	3
64	1 000 000	2	7	3-1/2
100	1 100 100	3	7	2-1/3
128	10 000 000	3	8	2-2/3
256	100 000 000	3	9	3
512	1 000 000 000	3	10	3-1/3
1000	1 111 101 000	4	10	2-1/2
1024	10 000 000 000	4	11	2-3/4
2048	100 000 000 000	4	12	3
4096	1 000 000 000 000	4	13	3-1/4
8192	10 000 000 000 000	4	14	3-1/2
10000	10 011 100 010 000	5	14	2-4/5

Compare this with the number 999 obtained with three decimal symbols in Sec. II-A. Table III tabulates the ratio between the number of symbols required in the binary system to the number of symbols required in the decimal system. All possible ratios are given for numbers up to 10,000.

It is necessary to accustom oneself to the handling of rather bulky binary numbers which may seem rather small in magnitude when considered on the decimal scale. The lack of economy in size, however, is more than compensated for by the ease with which these numbers are processed in electronic computer systems.

B. Positive Whole Numbers (Binary)

Table IV is a partial table of binary numbers, and may be referred to for this discussion.

TABLE IV
BINARY NUMBERS 0 – 15

0	0	8	1000
1	1	9	1001
2	10	10	1010
3	11	11	1011
4	100	12	1100
5	101	13	1101
6	110	14	1110
7	111	15	1111

The one-symbol combination is exhausted after a count of one. The one- and two-symbol combinations allow a count to three, the three-symbol combination to seven, and the four-symbol combination to fifteen. The combinations are arranged in an orderly fashion. As soon as all combinations of say two symbols are formed, a "1" is put in the next place to the left and all previous combinations are repeated. Note that any number may be thought of as having zeros to left of the first place on the left. All of the numbers of Table IV may be written as four-symbol combinations, by adding an appropriate number of zeros to the left of the given number. This changes nothing significant, but is a convenience in most uses. It is easier to keep track of numbers if they are all of the same symbol length. Table V is a more complete table and uses seven binary places for all numbers.

C. Powers of Two

Just as decimal numbers may be written as a sum of powers of ten, a binary number may be written as a sum of powers of two. A partial table of powers of two is found in Table VI. In a whole binary number, the first bit on the right is the 2^0 term, the others follow in sequence from right to left. As an example, take the binary number 1 0 1 0011; separating it into its component parts, we have the following:

$$\begin{array}{ccccccccccccccc}
 & 1 & & 0 & & 1 & & 0 & & 0 & & 1 & & 1 \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
 & (1 \times 2^6) & + & (0 \times 2^5) & + & (1 \times 2^4) & + & (0 \times 2^3) & + & (0 \times 2^2) & + & (1 \times 2^1) & + & (1 \times 2^0) \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
= & 64 & & 0 & & 16 & & 0 & & 0 & & 2 & & 1
\end{array}$$

$$\begin{array}{rr}
 & 64 \\
 & 0 \\
 & 16 \\
 & 0 \\
 & 0 \\
 & 2 \\
 & \underline{1} \\
\text{sum} & 83
\end{array}$$

That this number is, in fact, 83 may be verified by an examination of Table IV.

Conversely, one may obtain the binary number of a given decimal number by the inverse process. The process is to take the decimal number and subtract successively the powers of two, given in Table VI, starting with the largest number possible and working toward the smaller until the remainder is zero. The binary number may then be written directly by writing a "1" whenever it was possible to subtract a power of two and writing a "0" otherwise. In order to illustrate this method, take the number 873. The largest number in Table VI which may be subtracted from this number is $2^9 = 512$. The remainder is 361. Subtracting $2^8 = 256$ next we get 105. This is smaller than 2^7 which is passed over. The next is $2^6 = 64$ which, when subtracted, gives a remainder of 41. Finally, when 2^5, 2^3 and 2^0 are subtracted there is a remainder of zero.

$$\begin{array}{rr}
 & 873 \\
2^9 & -512 \\ \hline
 & 361 \\
2^8 & -256 \\ \hline
 & 105 \\
2^6 & -\ 64 \\ \hline
 & 41 \\
2^5 & -\ 32 \\ \hline
 & 9 \\
2^3 & -\ 8 \\ \hline
 & 1 \\
2^0 & -\ 1 \\ \hline
\text{Remainder} & 0
\end{array}$$

The binary number may now be written

$$(1 \times 2^9) + (1 \times 2^8) + (0 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0).$$

When this is written in a more abbreviated form we have:

1 101 101 001 = 873

The reader may wish to develop his skill in these conversion processes by working out the following examples.

1) 10 000 001 = 129
2) 10 001 000 = 136
3) 11 111 111 = 255
4) 10 101 010 = 170
5) 11 100 111 = 231
6) 00 000 011 = 3
7) 00 000 110 = 6
8) 00 001 100 = 12
9) 00 011 000 = 24
10) 00 110 000 = 48
11) 01 100 000 = 96
12) 11 000 000 = 192

13) 100 = 01 100 100
14) 200 = 11 001 000
15) 251 = 11 111 011
16) 197 = 11 000 101
17) 87 = 01 010 111
18) 5 = 00 000 101
19) 10 = 00 001 010
20) 20 = 00 010 100
21) 40 = 00 101 000
22) 80 = 01 010 000
23) 160 = 10 100 000

TABLE V

BINARY NUMBERS 0 – 127

0	0 000 000	32	0 100 000	64	1 000 000	96	1 100 000
1	0 000 001	33	0 100 001	65	1 000 001	97	1 100 001
2	0 000 010	34	0 100 010	66	1 000 010	98	1 100 010
3	0 000 011	35	0 100 011	67	1 000 011	99	1 100 011
4	0 000 100	36	0 100 100	68	1 000 100	100	1 100 100
5	0 000 101	37	0 100 101	69	1 000 101	101	1 100 101
6	0 000 110	38	0 100 110	70	1 000 110	102	1 100 110
7	0 000 111	39	0 100 111	71	1 000 111	103	1 100 111
8	0 001 000	40	0 101 000	72	1 001 000	104	1 101 000
9	0 001 001	41	0 101 001	73	1 001 001	105	1 101 001
10	0 001 010	42	0 101 010	74	1 001 010	106	1 101 010
11	0 001 011	43	0 101 011	75	1 001 011	107	1 101 011
12	0 001 100	44	0 101 100	76	1 001 100	108	1 101 100
13	0 001 101	45	0 101 101	77	1 001 101	109	1 101 101
14	0 001 110	46	0 101 110	78	1 001 110	110	1 101 110
15	0 001 111	47	0 101 111	79	1 001 111	111	1 101 111
16	0 010 000	48	0 110 000	80	1 010 000	112	1 110 000
17	0 010 001	49	0 110 001	81	1 010 001	113	1 110 001
18	0 010 010	50	0 110 010	82	1 010 010	114	1 110 010
19	0 010 011	51	0 110 011	83	1 010 011	115	1 110 011
20	0 010 100	52	0 110 100	84	1 010 100	116	1 110 100
21	0 010 101	53	0 110 101	85	1 010 101	117	1 110 101
22	0 010 110	54	0 110 110	86	1 010 110	118	1 110 110
23	0 010 111	55	0 110 111	87	1 010 111	119	1 110 111
24	0 011 000	56	0 111 000	88	1 011 000	120	1 111 000
25	0 011 001	57	0 111 001	89	1 011 001	121	1 111 001
26	0 011 010	58	0 111 010	90	1 011 010	122	1 111 010
27	0 011 011	59	0 111 011	91	1 011 011	123	1 111 011
28	0 011 100	60	0 111 100	92	1 011 100	124	1 111 100
29	0 011 101	61	0 111 101	93	1 011 101	125	1 111 101
30	0 011 110	62	0 111 110	94	1 011 110	126	1 111 110
31	0 011 111	63	0 111 111	95	1 011 111	127	1 111 111

TABLE VI
POSITIVE AND NEGATIVE POWERS OF TWO

Positive Powers		Negative Powers			
2^0 =	1				
2^1 =	2	2^{-1}	= 1/2	= 0.5	
2^2 =	4	2^{-2}	= 1/4	= 0.25	
2^3 =	8	2^{-3}	= 1/8	= 0.125	
2^4 =	16	2^{-4}	= 1/16	= 0.062	5
2^5 =	32	2^{-5}	= 1/32	= 0.031	25
2^6 =	64	2^{-6}	= 1/64	= 0.015	625
2^7 =	128	2^{-7}	= 1/128	= 0.007	813
2^8 =	256	2^{-8}	= 1/256	= 0.003	906
2^9 =	512	2^{-9}	= 1/512	= 0.001	953
2^{10} =	1,024	2^{-10}	= 1/1024	= 0.000	977
2^{11} =	2,048	2^{-11}	= 1/2048	= 0.000	488
2^{12} =	4,096	2^{-12}	= 1/4096	= 0.000	244
2^{13} =	8,192	2^{-13}	= 1/8192	= 0.000	122
2^{14} =	16,384	2^{-14}	= 1/16,384	= 0.000	061
2^{15} =	32,768	2^{-15}	= 1/32,768	= 0.000	031
2^{16} =	65,536	2^{-16}	= 1/65,536	= 0.000	015
2^{17} =	131,072	2^{-17}	= 1/131,072	= 0.000	008
2^{18} =	262,144	2^{-18}	= 1/262,144	= 0.000	004
2^{19} =	524,288	2^{-19}	= 1/524,288	= 0.000	002
2^{20} =	1,048,576	2^{-20}	= 1/1,048,576	= 0.000	001

Note: Decimal values have been rounded off to the nearest millionth place.

D. Fractional Binary Numbers

In the decimal system, it is remembered, fractions are represented by numbers to the right of the decimal point, the positions being successive powers of ten, 0.1, 0.01, 0.001, etc. (see Sec. II-B).

In the binary system, fractions are represented in an analogous manner, by a "1" or "0" to the right of the binary point, the positions being successive negative powers of two, starting with 2^{-1}, 2^{-2}, 2^{-3}, etc., or 1/2, 1/4, 1/8, etc., going from left to right. The binary fraction is then found by adding together the numerical values of these negative powers whenever a "1" occurs in a bit. As an example take the binary fraction 0.11010 and consider each place separately.

$$\begin{array}{cccccccccccc}
 & 0 & & 1 & & 1 & & 0 & & 1 & & 0 \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
= & (0 \times 2^{0}) & + & (1 \times 2^{-1}) & + & (1 \times 2^{2}) & + & (0 \times 2^{-3}) & + & (1 \times 2^{4}) & + & (0 \times 2^{-5}) \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
= & (0 \times 1) & + & (1 \times 1/2) & + & (1 \times 1/4) & + & (0 \times 1/8) & + & (1 \times 1/16) & + & (0 \times 1/32) \\
 & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\
= & 0.0000 & + & 0.5000 & + & 0.2500 & + & 0.0000 & + & 0.0625 & + & 0.0000 \\
 & & & & & & & & & & & \\
= & 0.8125 & & & & & & & & & &
\end{array}$$

Stated in fractional form we have $1/2 + 1/4 + 1/16 = 13/16 = 0.8125$. The process for converting a fraction from decimal form to binary form is similar to the one given for whole numbers. The table of negative powers found in Table VI will be found useful. Let us convert the decimal 0.839 to binary form. First, select the largest number in the table of negative powers which may be subtracted from 0.839. This is 2^{-1} or 0.500. The remainder is 0.339. The next largest number which may be subtracted is 2^{-2} or 0.250, leaving 0.089 as a remainder. After 2^{-4}, 2^{-6}, 2^{-7}, 2^{-9}, and 2^{-10} have been subtracted there is a remainder of 0.000132. Since the original decimal was given to the nearest one-thousandth place, the 2^{-10} place is sufficient to give approximately the same accuracy (0.1 per cent).

$$\begin{array}{crl}
 & 0.839 & \\
2^{-1} & -0.500 & \\ \hline
 & 0.339 & \\
2^{-2} & -0.250 & \\ \hline
 & 0.089 & \\
2^{-4} & -0.0625 & \\ \hline
 & 0.0265 & \\
2^{-6} & -0.015625 & \\ \hline
 & 0.010875 & \\
2^{-7} & -1.007813 & \\ \hline
 & 0.003062 & \\
2^{-9} & -0.001953 & \\ \hline
 & 0.001109 & \\
2^{-10} & -0.000977 & \\ \hline
 & 0.000132 & \text{(remainder)}
\end{array}$$

Adding these together we have

$$(0 \times 2^{0}) + (1 \times 2^{-1}) + (1 \times 2^{-2}) + (0 \times 2^{-3}) + (1 \times 2^{-4}) + (0 \times 2^{-5}) + (1 \times 2^{-6}) + (1 \times 2^{-7}) + (0 \times 2^{-8}) + (1 \times 2^{-9}) + (1 \times 2^{-10})$$

or in the more compact form,

$$0.839 = 0.110\ 101\ 101\ 1 \quad \text{(approximately)}$$

The "round off" in the above example is equivalent to what is done in the decimal system when representing, decimally, the fraction "two-thirds" by 2/3 = 0.667. Actually, the decimal contains an infinite number of sixes. When representing a fraction in the decimal system, the only time there is a zero remainder is when the fraction is equal to the sum of a series of negative powers of ten, as $1/20 = 1/100 + 1/100 + 1/100 + 1/100 + 1/100 = 5 \times 10^{-2} = 0.0500$ or $1/5 = 1/10 + 1/10 = 2 \times 10^{-1} = 0.200$. Other fractions can never be completely represented by a finite number of decimals, such as 1/3 = 0.3333. . . , or 1/9 = 0.111. . .

In the same way, when representing a fraction in the binary system, the only time there is a zero remainder is when the fraction is equal to the sum of a series of negative powers of two, as $3/4 = 1/2 + 1/4 = 1 \times 2^{-1} + 1 \times 2^{-2} = 0.11000. . .$, or 7/8 = 1/2 + 1/4 + 1/8 = 0.11100. . .

"Round off" may sometimes be deliberately avoided when it is anticipated that additional accuracy and, hence, more bits may be required in the future. The additional bits may be added without changing the original number. Also, this sometimes simplifies the changing of the original circuitry.

Table VII lists decimal to binary conversions for decimals from 0.01 to 1.00 in steps of 0.01.

Table VIII lists fractional to binary conversions for fractions from 1/64 to 63/64 in steps of 1/64.

In general, when given a decimal number to convert to binary form, one should also know the required accuracy of representation. Thus, five bits allows an accuracy of one part in 32 or 1/32 which is approximately 3 per cent. Seven bits allows an accuracy of one part in 128 or 1/128 which is about one per cent, while ten bits allows an accuracy of 1/1024 or about 0.1 per cent. That this is true is easily seen since a "1" in the tenth binary place adds 1/1024 to the final sum. A table of percentage accuracy of representation is given in Table IX. This table may be used whether the binary number is a whole number, fractional, or a combination. It is the total number of bits which is significant.

PROBLEMS

Verify the following:

1) 3/64 = 0.000011
2) 6/64 = 0.000110
3) 12/64 = 0.001100
4) 24/64 = 0.011000
5) 48/64 = 0.110000

6) 0.640625 = 0.101001

approx. $\left\{ \begin{array}{l} \text{7) } .0732 = 0.000\ 100\ 101\ 1 \\ \text{8) } 0.11011 = 0.84 \\ \text{9) } 0.011011 = 0.42 \end{array} \right.$

TABLE VII
DECIMAL TO BINARY CONVERSION TABLE

0.00	0.000 000 00						
0.01	0.000 000 11	0.26	0.010 000 11	0.51	0.100 000 11	0.76	0.110 000 11
0.02	0.000 001 01	0.27	0.010 001 01	0.52	0.100 001 01	0.77	0.110 001 01
0.03	0.000 010 00	0.28	0.010 010 00	0.53	0.100 010 00	0.78	0.110 010 00
0.04	0.000 010 10	0.29	0.010 010 10	0.54	0.100 010 10	0.79	0.110 010 10
0.05	0.000 011 01	0.30	0.010 011 01	0.55	0.100 011 01	0.80	0.110 011 01
0.06	0.000 011 11	0.31	0.010 011 11	0.56	0.100 011 11	0.81	0.110 011 11
0.07	0.000 100 10	0.32	0.010 100 10	0.57	0.100 100 10	0.82	0.110 100 10
0.08	0.000 101 00	0.33	0.010 101 00	0.58	0.100 101 00	0.83	0.110 101 00
0.09	0.000 101 11	0.34	0.010 101 11	0.59	0.100 101 11	0.84	0.110 101 11
0.10	0.000 110 10	0.35	0.010 110 10	0.60	0.100 110 10	0.85	0.110 110 10
0.11	0.000 111 00	0.36	0.010 111 00	0.61	0.100 111 00	0.86	0.110 111 00
0.12	0.000 111 11	0.37	0.010 111 11	0.62	0.100 111 11	0.87	0.110 111 11
0.13	0.001 000 01	0.38	0.011 000 01	0.63	0.101 000 01	0.88	0.111 000 01
0.14	0.001 001 00	0.39	0.011 001 00	0.64	0.101 001 00	0.89	0.111 001 00
0.15	0.001 001 10	0.40	0.011 001 10	0.65	0.101 001 10	0.90	0.111 001 10
0.16	0.001 010 01	0.41	0.011 010 01	0.66	0.101 010 01	0.91	0.111 010 01
0.17	0.001 011 00	0.42	0.011 011 00	0.67	0.101 011 00	0.92	0.111 011 00
0.18	0.001 011 10	0.43	0.011 011 10	0.68	0.101 011 10	0.93	0.111 011 10
0.19	0.001 100 01	0.44	0.011 100 01	0.69	0.101 100 01	0.94	0.111 100 01
0.20	0.001 100 11	0.45	0.011 100 11	0.70	0.101 100 11	0.95	0.111 100 11
0.21	0.001 101 10	0.46	0.011 101 10	0.71	0.101 101 10	0.96	0.111 101 10
0.22	0.001 110 00	0.47	0.011 110 00	0.72	0.101 110 00	0.97	0.111 110 00
0.23	0.001 110 11	0.48	0.011 110 11	0.73	0.101 110 11	0.98	0.111 110 11
0.24	0.001 111 01	0.49	0.011 111 01	0.74	0.101 111 01	0.99	0.111 111 01
0.25	0.010 000 00	0.50	0.100 000 00	0.75	0.110 000 00	1.00	1.000 000 00

Note: Six-place decimal numbers from Table VI have been used to calculate the above table. The binary number has been rounded off by adding "1" to the 2^{-8} bit if there is a "1" in the 2^{-9} bit.

TABLE VIII
FRACTION TO BINARY CONVERSION TABLE

Fractional						Decimal	Binary
0/64	0/32	0/16	0/8	0/4	0/2	0.000000	0.000000
1/64						0.015625	0.000001
2/64	1/32					0.031250	0.000011
3/64						0.046875	0.000011
4/64	2/32	1/16				0.062500	0.000101
5/64						0.078125	0.000101
6/64	3/32					0.083750	0.000110
7/64						0.094375	0.000111
8/64	4/32	2/16	1/8			0.125000	0.001000
9/64						0.140625	0.001001
10/64	5/32					0.156250	0.001010
11/64						0.171875	0.001011
12/64	6/32	3/16				0.187500	0.001100
13/64						0.203125	0.001101
14/64	7/32					0.218750	0.001110
15/64						0.345375	0.001111
16/64	8/32	4/16	2/8	1/4		0.250000	0.010000
17/64						0.265625	0.010001
18/64	9/32					0.281250	0.010010
19/64						0.296875	0.010011
20/64	10/32	5/16				0.312500	0.010100
21/64						0.328125	0.010101
22/64	11/32					0.343750	0.010110
23/64						0.359375	0.010111
24/64	12/32	6/16	3/8			0.375000	0.011000
25/64						0.390625	0.011001
26/64	13/32					0.416250	0.011010
27/64						0.431875	0.011011
28/64	14/32	7/16				0.447500	0.011100
29/64						0.463125	0.011101
30/64	15/32					0.478750	0.011110
31/64						0.494375	0.011111

(Table VIII continued on next page)

TABLE VIII (Continued)							
Fractional						Decimal	Binary
32/64	16/32	8/16	4/8	2/4	1/2	0.500000	0.100000
33/64						0.515625	0.100001
34/64	17/32					0.531250	0.100010
35/64						0.546875	0.100011
36/64	18/32	9/16				0.562500	0.100100
37/64						0.588125	0.100101
38/64	19/32					0.603750	0.100110
39/64						0.619375	0.100111
40/64	20/32	10/16	5/8			0.635000	0.101000
41/64						0.650625	0.101001
42/64	21/32					0.666250	0.101010
43/64						0.681875	0.101011
44/64	22/32	11/16				0.687500	0.101100
45/64						0.703125	0.101101
46/64	23/32					0.718750	0.101110
47/64						0.734375	0.101111
48/64	24/32	12/16	6/8	3/4		0.750000	0.110000
49/64						0.765625	0.110001
50/64	25/32					0.781250	0.110010
51/64						0.796875	0.110011
52/64	26/32	13/16				0.812500	0.110100
53/64						0.828125	0.110101
54/64	27/32					0.843750	0.110110
55/64						0.859375	0.110111
56/64	28/32	14/16	7/8			0.875000	0.111000
57/64						0.890625	0.111001
58/64	29/32					0.906250	0.111010
59/64						0.921875	0.111011
60/64	30/32	15/16				0.937500	0.111100
61/64						0.953125	0.111101
62/64	31/32					0.968750	0.111110
63/64						0.984375	0.111111

TABLE IX
PERCENTAGE ACCURACY vs NUMBER OF BINARY BITS

No. of Bits	Approximate Percentage Accuracy	No. of Bits	Approximate Percentage Accuracy
1	50	11	0.05
2	25	12	0.025
3	13	13	0.012
4	6	14	0.006
5	3	15	0.003
6	1.6	16	0.0015
7	0.7	17	0.0008
8	0.4	18	0.0004
9	0.2	19	0.0002
10	0.1	20	0.0001

E. Whole and Fractional (Mixed) Binary Numbers

When a decimal number such as 121.56 (which is the sum of a whole and a fractional number) is converted to binary form, each part is converted separately. The binary number is then written with a binary point separating the two parts just as in the decimal case.

As an example, let us convert the above number 121.56 to binary form. It is apparent that the whole number 121 requires 7 bits to represent it.

121 = 1 111 001

Assume that the fractional number is accurate to one part in the second decimal place or to one per cent. If we refer to Table IX we note that 7 bits will allow 0.7 per cent accuracy, so we conclude that 7 bits will be required to represent the fraction. If we convert the decimal 0.56 we get:

0.56 = 1 000 111

When the whole number is combined with this decimal we get

1 111 001.100 011 1 = 121.56

F. Negative Binary Numbers

It has been common practice to introduce high-school students to negative numbers by referring to the thermometer scale which is a part of everyday experience. In this case, we use the same symbols to designate both above and below zero. In order to differentiate between these numbers a minus sign (–) is affixed to those numbers below zero and a plus sign (+) is affixed to those numbers above zero. The + and – might be thought of as additional symbols which enable us to double the range of application of the set of numbers.

If one is working with 5-bit binary numbers, numbers from 0 to 31 may be expressed; 0 = 00000 and 31 = 11111. Now if it is also desired to use negative binary numbers from –1 to –31, an additional bit or a total of six bits must be used, since we are effectively doubling the range of

TABLE X

POSITIVE AND NEGATIVE BINARY NUMBERS

0	$\bar{0}00000$	0	$\bar{0}00000$
+1	$\bar{0}00001$	–1	$\bar{1}11111$
+2	$\bar{0}00010$	–2	$\bar{1}11110$
+3	$\bar{0}00011$	–3	$\bar{1}11101$
+4	$\bar{0}00100$	–4	$\bar{1}11100$
+5	$\bar{0}00101$	–5	$\bar{1}11011$
+6	$\bar{0}00110$	–6	$\bar{1}11010$
+7	$\bar{0}00111$	–7	$\bar{1}11001$
+8	$\bar{0}01000$	–8	$\bar{1}11000$
+9	$\bar{0}01001$	–9	$\bar{1}10111$
+10	$\bar{0}01010$	–10	$\bar{1}10110$
+11	$\bar{0}01011$	–11	$\bar{1}10101$
+12	$\bar{0}01100$	–12	$\bar{1}10100$
+13	$\bar{0}01101$	–13	$\bar{1}10011$
+14	$\bar{0}01110$	–14	$\bar{1}10010$
+15	$\bar{0}01111$	–15	$\bar{1}10001$
+16	$\bar{0}10000$	–16	$\bar{1}10000$
+17	$\bar{0}10001$	–17	$\bar{1}01111$
+18	$\bar{0}10010$	–18	$\bar{1}01110$
+19	$\bar{0}10011$	–19	$\bar{1}01101$
+20	$\bar{0}10100$	–20	$\bar{1}01100$
+21	$\bar{0}10101$	–21	$\bar{1}01011$
+22	$\bar{0}10110$	–22	$\bar{1}01010$
+23	$\bar{0}10111$	–23	$\bar{1}01001$
+24	$\bar{0}11000$	–24	$\bar{1}01000$
+25	$\bar{0}11001$	–25	$\bar{1}00111$
+26	$\bar{0}11010$	–26	$\bar{1}00110$
+27	$\bar{0}11011$	–27	$\bar{1}00101$
+28	$\bar{0}11100$	–28	$\bar{1}00100$
+29	$\bar{0}11101$	–29	$\bar{1}00011$
+30	$\bar{0}11110$	–30	$\bar{1}00010$
+31	$\bar{0}11111$	–31	$\bar{1}00001$

numbers. The additional bit is sometimes known as the "sign bit." This additional bit must now be used with both the negative and positive numbers. In the system we will use, this "sign bit" is "0" with positive numbers and "1" with negative numbers and is identified as $\bar{0}$ or $\bar{1}$. The rule for obtaining negative numbers is as follows:

> **Change all ones to zeros and all zeros to ones (this is called complementing each bit) and add a one to the last bit on the right.**

Example: take the number 23 = 10111

$$\begin{array}{rl} & \text{sign} \\ & \downarrow \\ +23 = & \bar{0}10111 \\ \hline \text{complement} & \bar{1}01000 \\ \text{add "1"} & \underline{\qquad +1} \\ -23 = & \bar{1}01001 \end{array}$$

Example: take the number 18 = 10010

$$\begin{array}{rl} & \text{sign} \\ & \downarrow \\ +18 = & \bar{0}10010 \\ \hline \text{complement} & \bar{1}01101 \\ \text{add "1"} & \underline{\qquad +1} \\ & \bar{1}01110 \end{array}$$

When "1" is added to "1" the sum is "0" with "1" to carry. This carry is added to the adjacent bit on the left. This property may be verified by adding ones successively to a number thus forming a binary sequence:

$$\begin{array}{ccccccc} 0 & 1 & 10 & 11 & 100 & 101 & 110 \\ \underline{+1} & \underline{+1} & \underline{+1} & \underline{+1} & \underline{+\ 1} & \underline{+\ 1} & \underline{+\ 1} \\ 1 & 10 & 11 & 100 & 101 & 110 & 111 \end{array}$$

Let us compare positive and negative binary numbers which are formed by the rule stated. Table X lists these numbers up to 31.

An examination of Table X quickly shows that the positive numbers form a binary sequence going up while negative numbers form a binary sequence going down.

To show that what has been formed is in truth a table of negative and positive binary numbers, let us perform some simple arithmetical operations using six-bit numbers. Only numbers less than 31 are considered. Add two positive numbers, 5 and 9

$$\begin{array}{lrr} & 5 & \bar{0}00101 \\ \text{Add} & \underline{+9} & \underline{\bar{0}01001} \\ & 14 & \bar{0}01110 \end{array}$$

Add two negative numbers, −3 and −8

$$\begin{array}{lrr} & -3 & \bar{1}11101 \\ \text{Add} & \underline{-8} & \underline{\bar{1}11000} \\ & -11 & (1)\bar{1}10101 \\ & & \uparrow\qquad \\ & & \text{discard}\quad \end{array}$$

The carry out of the end bit on the left is discarded, since we are dealing with only six-bit numbers and a seven-bit number is not defined in this system. Note that adding three ones together yields a "1" with a "1" to carry.

Add a positive and negative number, +12 and −7

$$\begin{array}{lrr} & +12 & \bar{0}01100 \\ \text{Add} & -7 & \bar{1}11001 \\ \hline & +5 & (1)\bar{0}00101 \\ & & \uparrow \\ & & \text{discard} \end{array}$$

Add a positive and negative number, +4 and −13

$$\begin{array}{lrr} & +4 & \bar{0}00100 \\ \text{Add} & -13 & \bar{1}10011 \\ \hline & -9 & \bar{1}10111 \end{array}$$

Thus it is seen that the negative numbers formed by the rule given do, in fact, behave in the same way as do the negative numbers of a thermometer scale or the negative numbers in algebra. Negative numbers larger than those given in Table X may be formed by the rule given and will be found to follow the same "count down" sequence indicated in the table.

The process of finding the decimal magnitude of a negative binary number consists of finding the positive equivalent by the inverse of the method just given and then converting this number to the decimal form. In order to get the positive equivalent, a "1" is subtracted from the last bit on the right and then each bit is complemented. The examples below illustrate this process:

$$\begin{array}{rl} -9 = & \bar{1}10111 \\ \text{Subtract "1"} & \quad\;\; -1 \\ \cline{2-2} & \bar{1}10110 \\ \text{Complement} & \bar{0}01001 = +9 \end{array} \tag{1}$$

$$\begin{array}{rl} -12 = & \bar{1}10100 \\ \text{Subtract "1"} & \quad\;\; -1 \\ \cline{2-2} & \bar{1}10011 \\ \text{Complement} & \bar{0}01100 = +12 \end{array} \tag{2}$$

In the second example, it was necessary to borrow "1" in order to subtract a "1" from the "0" in the last bit. Since the adjacent bit was also "0", it was necessary to go to the third bit from the right to borrow a "1"; this made it "0" and the other two bits "1". The process of subtracting one is equivalent to a "count down" of one and is the same as the method used in ordinary arithmetic, whether binary or decimal.

It is interesting to note that the process of subtracting "1" from a binary number gives the same result as adding a number of the same length as the given binary number and consisting solely of ones. Reconsider the two examples just given.

$$\begin{array}{rl} -9 = & \bar{1}10111 \\ \text{Add} & \underline{\bar{1}11111} \\ & (1)\bar{1}10110 \\ & \uparrow \\ & \text{discard} \\ \text{Complement} & \bar{0}01001 = +9 \end{array}$$

$$\begin{array}{rl} -12 = & \bar{1}10100 \\ \text{Add} & \underline{\bar{1}11111} \\ & (1)\bar{1}10011 \\ & \uparrow \\ & \text{discard} \\ \text{Complement} & \bar{0}01100 = +12 \end{array}$$

The reason for this interesting property may be understood if one considers what happens when a one is added to a number consisting solely of ones.

$$\begin{array}{r} \bar{1}1111 \\ \underline{+1} \\ (1)\bar{0}0000 \\ \uparrow \\ \text{discard} \end{array}$$

The only number which may be added to +1 to give zero is −1, hence, $\bar{1}1111$ in the above example is equivalent to −1; or stated in algebraic form, if A = $\bar{1}1111$

$$\begin{aligned} A + 1 &= 0 \\ A &= -1. \end{aligned}$$

This fact may also be seen by referring to Table X.

The rule for forming negative whole numbers also applies to fractional and mixed numbers. It is important to remember that any positive binary number may be considered as having zeros to the left of the most significant digit, hence in the process of complementing each bit, these additional bits are complemented. It is usually only necessary to make note of this fact mentally so that the proper number of bits are kept.

In all the following problems and examples, no positive or negative number with magnitude greater than 15 or with more than 1/16 accuracy will be encountered. All numbers will have six bits to the left of the decimal place (including the sign bit) and four bits to the right. (See Tables VIII and X.)

<u>Example:</u> express −3 3/16 in binary form

$$\begin{array}{rl} 3\ 3/16 = & \bar{0}00011.0011 \\ \text{Complement} & \bar{1}11100.1100 \\ & \underline{\qquad\qquad\quad +1} \\ -3\ 3/16 = & \bar{1}11100.1101 \end{array}$$

Verify:

$$\begin{array}{rl} -11\ 7/16 = & \bar{1}10100.1001 \\ -14\ 10/16 = & \bar{1}10001.0110 \\ +8\ 4/16 = & \bar{0}01000.0100 \\ -8\ 4/16 = & \bar{1}10111.1100 \end{array}$$

Example:

Add −3 3/16 and −11 7/16

$$\begin{array}{rl} -3\ 3/16 = & \bar{1}11100.1101 \\ \text{Add } -11\ 7/16 = & \underline{\bar{1}10100.1001} \\ & (1)\bar{1}10001.0110 = -14\ 10/16 \\ & \ \uparrow \\ & \text{discard} \end{array}$$

Verify:

$11\ 7/16 - 3\ 3/16 = \bar{0}01000.0100 = +8\ 4/16$

$3\ 3/16 - 11\ 7/16 = \bar{1}10111.1100 = -8\ 4/16$

It might be wondered just what application negative binary numbers would have. It is often convenient, in certain kinds of calculations, to use negative numbers. One example of this is the process of subtraction. While subtraction can be done by certain rules it sometimes becomes awkward to implement this process. The process of subtracting +B from +A is exactly the same as adding −B to +A. The process of obtaining a negative number is quite simple, and usually adders have already been built in the system so the subtraction is easily done. Also, many functions have both negative and positive values. A familiar example of this is the sine of an angle which varies from 0 to +1 to −1 and back to zero as the angle goes from 0 to 360 degrees.

IV. ARITHMETIC PROCESSES IN THE BINARY SYSTEM

A. Addition

The process for adding binary numbers – whole, fractional and negative – has been illustrated in the examples given in past sections. Table XI below gives all cases. The carry is added to the adjacent bit on the left.

In the cases where one is dealing with negative numbers, the given number of bits to be used (including the sign bit) is decided in advance. This number is determined by the largest number to be encountered and the fractional accuracy desired. A carry out of the sign bit is discarded since an additional bit has not been defined in the selected system.

The process of addition proceeds, as in the decimal case, from right to left as each bit is added in turn.

TABLE XI ADDITION TABLE
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 0 (and 1 to carry)
1 + 1 + 1 = 1 (and 1 to carry)

TABLE XII SUBTRACTION TABLE
0 − 0 = 0
0 − 1 = 1 (and borrow 1)
1 − 0 = 1
1 − 1 = 0

The special case of adding a series of ones to another number of equal length is the same as subtracting one or "counting down" by one as previously explained.

B. Subtraction

The process of subtraction is the converse of addition. Table XII gives the various cases.

It is necessary to keep careful tab on the "borrowing" of ones which occurs more frequently than in the decimal analogue. Because of this, it is sometimes more convenient to change the subtrahend to a negative binary number (adding a sign bit on the left), and then add this negative number to the other number.

$$A - B = A + (-B)$$

The special case of subtracting one from the last bit of a number is equivalent to adding a number of the same length consisting solely of ones. This is the converse of the special case stated in the section on addition.

C. Multiplication

In the decimal case, multiplication is carried out by successive additions of multiples of powers of ten, each multiplied by the multiplicand and each positioned according to its power.

Example:

$$\begin{array}{rrl} & 357 & \\ & \underline{134} & \\ \text{Add}\left\{\begin{array}{c}\\ \\ \\ \end{array}\right. & \begin{array}{r}1428\\ 1071\\ \underline{357}\end{array} & \begin{array}{l}\text{multiplicand} \times 4 \times 10^0\\ \text{multiplicand} \times 3 \times 10^1\\ \text{multiplicand} \times 1 \times 10^2\end{array}\\ & 47838 & \text{product}\end{array}$$

The binary case is analogous. Multiplication is carried out by successive additions of powers of two (multiplied by 1 or 0) each multiplied by the multiplicand and positioned in accordance to its power.

Example:

$$\begin{array}{rrl} & 10111 & = 23\\ & \underline{101} & = 5\\ \text{Add}\left\{\begin{array}{c}\\ \\ \\ \end{array}\right. & \begin{array}{r}10111\\ 00000\\ \underline{10111}\end{array} & \begin{array}{l}\text{multiplicand} \times 1 \times 2^0\\ \text{multiplicand} \times 0 \times 2^1\\ \text{multiplicand} \times 1 \times 2^2\end{array}\\ & 1110011 & = 115 \text{ (product)}\end{array}$$

In the special decimal case, where one multiplies by a power of ten as 10, 100, 1000, etc., it is merely necessary to move the decimal place to the right. Similarly, in the binary case where one multiplies by a power of two as 2, 4, 8, 16, etc., it is merely necessary to move the binary point to the right.

Example:

$$\begin{array}{lcrcr} 101.0000 & = & 5 \times 1 & = & 5\\ 1010.000 & = & 5 \times 2 & = & 10\\ 10100.00 & = & 5 \times 4 & = & 20\\ 101000.0 & = & 5 \times 8 & = & 40\\ 1010000. & = & 5 \times 16 & = & 80\end{array}$$

TABLE XIII

COMPARISON OF DECIMAL, OCTAL AND BINARY NUMBERS

Decimal	Octal	Binary	Decimal	Octal	Binary
0	0	000000	32	40	100000
1	1	000001	33	41	100001
2	2	000010	34	42	100010
3	3	000011	35	43	100011
4	4	000100	36	44	100100
5	5	000101	37	45	100101
6	6	000110	38	46	100110
7	7	000111	39	47	100111
8	10	001000	40	50	101000
9	11	001001	41	51	101001
10	12	001010	42	52	101010
11	13	001011	43	53	101011
12	14	001100	44	54	101100
13	15	001101	45	55	101101
14	16	001110	46	56	101110
15	17	001111	47	57	101111
16	20	010000	48	60	110000
17	21	010001	49	61	110001
18	22	010010	50	62	110010
19	23	010011	51	63	110011
20	24	010100	52	64	110100
21	25	010101	53	65	110101
22	26	010110	54	66	110110
23	27	010111	55	67	110111
24	30	011000	56	70	111000
25	31	011001	57	71	111001
26	32	011010	58	72	111010
27	33	011011	59	73	111011
28	34	011100	60	74	111100
29	35	011101	61	75	111101
30	36	011110	62	76	111110
31	37	011111	63	77	111111

The reader may recall that the problems given previously included such an example.

D. Division

The process of division is the inverse of multiplication. In the decimal case, multiples of powers of ten multiplied by the divisor are successively subtracted from the number. There may or may not be a remainder.

In the binary case, powers of two (multiplied by 1 or 0) are multiplied by the divisor and are successively subtracted from the number. There may or may not be a remainder.

Example: divide 65 by 13

$$\begin{array}{lrl}
 & 101 & = 5 \\
1101\,) & 1000001 & \\
(-) & 01101 & \text{divisor} \times 1 \times 2^{2} \\
 & 00110 & \\
(-) & 00000 & \text{divisor} \times 0 \times 2^{1} \\
 & 1101 & \\
(-) & 1101 & \text{divisor} \times 1 \times 2^{0} \\
 & 0000 & \text{no remainder}
\end{array}$$

The special case of dividing a number by a power of two consists merely of moving the binary point the proper number of places to the left.

Example:

$$\begin{array}{rcl}
1100000. & = 96/1 & = 96 \\
110000.0 & = 96/2 & = 48 \\
11000.00 & = 96/4 & = 24 \\
1100.000 & = 96/8 & = 12 \\
110.0000 & = 96/16 & = 6 \\
11.00000 & = 96/32 & = 3
\end{array}$$

V. THE OCTAL NUMBERING SYSTEM

A. Octal to Decimal Conversion

Because octal numbers are frequently used in certain computing systems, a short description of their use will be given.

As mentioned in Sec. I, the octal numbering system uses eight symbols; viz., 0, 1, 2, 3, 4, 5, 6, and 7. Counting in the octal system follows the pattern given for the decimal system; i.e., as soon as all the symbols are used singly, combinations of two symbols are used, etc. Table XIII gives the counting table for the decimal, octal, and binary numbers in parallel columns. As in the other systems described, numbers in the octal system are written in terms of the powers of the base which is "8" in the octal system. The zero power is immediately to the left of the octal point. Table XIV is a table of positive and negative powers of eight. An example of an octal number is 3075.2. The process for converting this to decimal form is as follows:

$$\begin{array}{ccccccccccc}
3 & & 0 & & 7 & & 5 & . & 2 & & \text{(octal)} \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
(3 \times 8^{3}) & + & (0 \times 8^{2}) & + & (7 \times 8^{1}) & + & (5 \times 8^{0}) & + & (2 \times 8^{-1}) & & \\
\downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \\
1536 & + & 0 & + & 56 & + & 5 & + & 2/8 & &
\end{array}$$

$$= 1597.25 \text{ (decimal)}$$

TABLE XIV
POWERS OF OCTAL NUMBERS

Positive Powers	Negative Powers	
$8^0 =$ 1	$8^0 = 1$	= 1.
$8^1 =$ 8	$8^{-1} = 1/8$	= 0.125
$8^2 =$ 64	$8^{-2} = 1/64$	= 0.015625
$8^3 =$ 512	$8^{-3} = 1/512$	= 0.001953
$8^4 =$ 4,096	$8^{-4} = 1/4096$	= 0.000244
$8^5 =$ 32,768	$8^{-5} = 1/32768$	= 0.000031
$8^6 =$ 262,144	$8^{-6} = 1/262,144$	= 0.000004
$8^7 =$ 2,097,152	$8^{-7} = 1/2,097,152$	= 0.0000005

TABLE XV
OCTAL TO BINARY CONVERSION

0 = 000	4 = 100
1 = 001	5 = 101
2 = 010	6 = 110
3 = 011	7 = 111

TABLE XVI
OCTAL ADDITION TABLE

+	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	10
2	2	3	4	5	6	7	10	11
3	3	4	5	6	7	10	11	12
4	4	5	6	7	10	11	12	13
5	5	6	7	10	11	12	13	14
6	6	7	10	11	12	13	14	15
7	7	10	11	12	13	14	15	16

B. Octal to Binary Conversion

The base of the octal system is "8." The base of the binary system is "2." Since $2^3 = 8$ one might expect some simple relationship to exist between numbers in the two systems. The relation is this: one octal digit may be replaced by a three-bit binary number which represents that number (see Table XV).

Take the example just given:

```
        3       0       7       5       2   (octal)
        ↓       ↓       ↓       ↓       ↓
      (011)   (000)   (111)   (101)   (010)
  or
             011 000 111 101.010  (binary)
```

The reason the octal system is often desirable in computing machines is because the input and output numbers may be in the octal system thus reducing the length of numbers by a factor of three times. Once in the machine, the simple relation between octal and binary numbers, shown above, allows the information to be processed in the binary system which is the natural way for logical switching networks to handle information. Finally, the answer is converted to the octal system again for operating, say an electric typewriter.

C. Octal Arithmetic

Arithmetic operations in the octal system are similar to those in other systems. The addition and multiplication tables are given in Table XVI as an aid to these operations. The use of these tables will be illustrated in the following two examples.

Add the two octal numbers 547 and 304. If we refer to Table XVI, 4 + 7 = 13, so a 3 is put in the last place and a 1 is carried to the next place where it is added to 4, giving 5. Finally, 5 is added to 3, giving 10. Parallel operations in the octal and decimal systems are shown below.

```
   547  (octal) =  359  (decimal)
  +304  (octal) = +196  (decimal)
  ----            ----
  1053  (octal) =  555  (decimal)
```

Multiply the two octal numbers 372 and 53. If we refer to Table XVII, 3 × 2 = 6 which is placed in the last place. 7 × 3 = 25 so 5 is put in the next place with 2 to carry. 3 × 3 = 11 and add 2 which was carried giving 13. Multiplication by 5 is similarly done. These products are added giving the answer. Parallel operations in the octal and decimal systems are given below.

TABLE XVII
OCTAL MULTIPLICATION TABLE

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	10	12	14	16
3	0	3	6	11	14	17	22	25
4	0	4	10	14	20	24	30	34
5	0	5	12	17	24	31	36	43
6	0	6	14	22	30	36	44	52
7	0	7	16	25	34	43	52	61

```
   372  (octal) =    250  (decimal)
  × 53  (octal) =  × 43  (decimal)
  ----            -----
  1356               750
 2342              1000
 -----            -----
 24776  (octal) = 10750  (decimal)
```

The special cases where there is multiplication or division by a power of eight are handled by merely shifting the octal point the proper number of places to the right or to the left respectively. Thus decimal 6 = octal 6, decimal 48 = 6 × 8 = octal 60 and decimal 384 = 6 × 64 = octal 600.

D. Negative Octal Numbers

Before we give the rule for forming negative octal numbers, let us briefly discuss negative numbers in general. Merely affixing a negative symbol (–) to a number does not change the number itself. Rather it indicates the kind of operation which is to be performed when this symbol is present. Thus, in the familiar decimal system when +24 is added to –15 the actual operation is one of subtraction. In order to actually add the positive number +24 to the negative number –15, the negative number must be formed together with a sign digit for distinction purposes. True negative numbers, thus formed, may be added just as positive numbers.

One definition of a negative number "–A" might be as follows:

$$(+A) + (-A) = 0$$

Suppose we talk about the set of one-digit decimal symbols. Let us tabulate numbers, which when added together, give "10." Since we are dealing only with the set of one-digit symbols, the "1" in "10" is discarded leaving "0."

Numbers to be made Negative ↓		Numbers which must be added to get Zero ↓		Discard ↓				9's Complement ↓		
0	+	10	=	(1)0	≡	0	+	(9-0)	+	1
1	+	9	=	(1)0	≡	1	+	(9-1)	+	1
2	+	8	=	(1)0	≡	2	+	(9-2)	+	1
3	+	7	=	(1)0	≡	3	+	(9-3)	+	1
4	+	6	=	(1)0	≡	4	+	(9-4)	+	1
5	+	5	=	(1)0	≡	5	+	(9-5)	+	1
6	+	4	=	(1)0	≡	6	+	(9-6)	+	1
7	+	3	=	(1)0	≡	7	+	(9-7)	+	1
8	+	2	=	(1)0	≡	8	+	(9-8)	+	1
9	+	1	=	(1)0	≡	9	+	(9-9)	+	1

The quantity within the parentheses is known as the 9's complement of a digit. From the above tabulation it is seen that if a digit's 9's complement plus 1 is added to the original digit, the result is 0 (in our single-digit set). Hence, it behaves as a negative number.

In order to distinguish between negative and positive numbers a sign digit is used which is $\overline{0}$ for positive numbers and $\overline{9}$ for negative numbers. As an example, we may talk about positive and negative numbers which do not exceed 9 in magnitude. A table for this set may now be constructed using the rule of forming the 9's complement and adding 1.

Positive	Negative
$+1 = \bar{0}1$	$-1 = \bar{9}9$
$+2 = \bar{0}2$	$-2 = \bar{9}8$
$+3 = \bar{0}3$	$-3 = \bar{9}7$
$+4 = \bar{0}4$	$-4 = \bar{9}6$
$+5 = \bar{0}5$	$-5 = \bar{9}5$
$+6 = \bar{0}6$	$-6 = \bar{9}4$
$+7 = \bar{0}7$	$-7 = \bar{9}3$
$+8 = \bar{0}8$	$-8 = \bar{9}2$
$+9 = \bar{0}9$	$-9 = \bar{9}1$

Let us perform operations whose results do not fall outside our set of numbers from −9 to +9.

$$\begin{array}{lr} & -1 = \bar{9}9 \\ \text{add} & -4 = \bar{9}6 \\ \hline & -5 = \underset{\substack{\uparrow \\ \text{discard}}}{(1)}\bar{9}5 \end{array} \qquad \begin{array}{lr} & -7 = \bar{9}3 \\ \text{add} & +4 = \bar{0}4 \\ \hline & -3 = \bar{9}7 \end{array} \qquad \begin{array}{lr} & -4 = \bar{9}6 \\ \text{add} & +6 = \bar{0}6 \\ \hline & +2 = \underset{\substack{\uparrow \\ \text{discard}}}{(1)}\bar{0}2 \end{array}$$

Larger numbers may be handled in a similar fashion by choosing a larger number of digits, reserving the digit on the extreme left as a sign digit.

The reader may now recall how negative numbers were formed in the binary system. In this case, the 1's complement of each bit was formed and 1 was added to the extreme bit on the right. This is the exact analogue of the 9's complement rule in the decimal case, and the reason for it follows the same reasoning given above.

The above method of forming negative numbers may be extended to other bases in a similar manner by forming the "(base −1) complement" and adding 1. Thus, for the octal system the rule is as follows:

Subtract each digit in turn from 7 (7's complement) and add 1 to the number formed.

In order to keep track of signs, a sign digit is placed on the left of the extreme left digit. This digit is $\bar{0}$ for positive numbers and $\bar{7}$ for negative numbers. Table XVII is a table of negative and positive numbers. The negative numbers form a count down sequence as in the binary system. It is necessary to specify the maximum number to be encountered so that the sign digit is to the left of the first number digit which might occur.

Table XVIII gives a comparison of negative and positive octal numbers from 0 to 31 (decimal).

Example: express −16 (decimal) in octal form.

$$\begin{array}{lrl} 16 \text{ (decimal)} & = \bar{0}020 & \text{(octal)} \\ 7\text{'s complement} & = \bar{7}757 & \\ & +1 & \\ \hline -16 \text{ (decimal)} & = \bar{7}760 & \text{(octal)} \end{array}$$

Example: express −27 (decimal) in octal form.

$$\begin{array}{lrl} 27 \text{ (decimal)} & = \bar{0}033 & \text{(octal)} \\ 7\text{'s complement} & = \bar{7}744 & \\ & +1 & \\ \hline -27 \text{ (decimal)} & = \bar{7}745 & \text{(octal)} \end{array}$$

TABLE XVIII

COMPARISON OF NEGATIVE AND POSITIVE OCTAL NUMBERS

Decimal	Octal	Decimal	Octal
0	$\bar{0}00$	0	$\bar{0}00$
+1	$\bar{0}01$	–1	$\bar{7}77$
+2	$\bar{0}02$	–2	$\bar{7}76$
+3	$\bar{0}03$	–3	$\bar{7}75$
+4	$\bar{0}04$	–4	$\bar{7}74$
+5	$\bar{0}05$	–5	$\bar{7}73$
+6	$\bar{0}06$	–6	$\bar{7}72$
+7	$\bar{0}07$	–7	$\bar{7}71$
+8	$\bar{0}10$	–8	$\bar{7}70$
+9	$\bar{0}11$	–9	$\bar{7}67$
+10	$\bar{0}12$	–10	$\bar{7}66$
+11	$\bar{0}13$	–11	$\bar{7}65$
+12	$\bar{0}14$	–12	$\bar{7}64$
+13	$\bar{0}15$	–13	$\bar{7}63$
+14	$\bar{0}16$	–14	$\bar{7}62$
+15	$\bar{0}17$	–15	$\bar{7}61$
+16	$\bar{0}20$	–16	$\bar{7}60$
+17	$\bar{0}21$	–17	$\bar{7}57$
+18	$\bar{0}22$	–18	$\bar{7}56$
+19	$\bar{0}23$	–19	$\bar{7}55$
+20	$\bar{0}24$	–20	$\bar{7}54$
+21	$\bar{0}25$	–21	$\bar{7}53$
+22	$\bar{0}26$	–22	$\bar{7}52$
+23	$\bar{0}27$	–23	$\bar{7}51$
+24	$\bar{0}30$	–24	$\bar{7}50$
+25	$\bar{0}31$	–25	$\bar{7}47$
+26	$\bar{0}32$	–26	$\bar{7}46$
+27	$\bar{0}33$	–27	$\bar{7}45$
+28	$\bar{0}34$	–28	$\bar{7}44$
+29	$\bar{0}35$	–29	$\bar{7}43$
+30	$\bar{0}36$	–30	$\bar{7}42$
+31	$\bar{0}37$	–31	$\bar{7}41$

Example: express −59 (decimal) in octal form.

$$\begin{array}{lll}
59 \text{ (decimal)} & = \bar{0}073 & \text{(octal)} \\
7\text{'s complement} & = \bar{7}704 & \\
 & \ \ \ \underline{+1} & \\
-59 \text{ (decimal)} & = \bar{7}705 & \text{(octal)}
\end{array}$$

Examples of arithmetic operations involving negative numbers are given below:

	Decimal		Octal
	−16	=	$\bar{7}760$
Add	−27	=	$\bar{7}745$
	−43	=	(1)$\bar{7}725$
			↑ discard

	Decimal		Octal
	−59	=	$\bar{7}705$
Add	−59	=	$\bar{7}705$
	−118	=	(1)$\bar{7}612$
			↑ discard

	Decimal		Octal
	−16	=	$\bar{7}760$
Add	+27	=	$\bar{0}033$
	+11	=	(1)$\bar{0}013$
			↑ discard

	Decimal		Octal
	−59	=	$\bar{7}705$
Add	+27	=	$\bar{0}033$
	−32	=	$\bar{7}740$

James R. Tobey

James R. Tobey

THE MITRE CORPORATION
Lexington 73, Massachusetts

Technical Memorandum TM— 15 #3
Sheet 1 of 37 Sheets

TITLE:

Subject: Basic XD-1 Information: COMPASS

To: J. D. Porter

From: J. R. Tobey

Date: 26 February 1959

Approved: JHBurrows
James H. Burrows

ABSTRACT

COMPASS is a group of utility programs written for the AN/FSQ-7, designed to: a) translate symbolic information into binary information; b) manipulate information stored on tape; c) process tapes. This document describes, from the programmer's and the computer operator's standpoints, how to use the COMPASS programs.

The initial version of compass will operate on either the old 8K Q-7 or the modified 65K machine, with a minimum of changes. Presumably, at some time in the future, a version will be prepared to take full advantage of the expanded machine. As significant additions to this initial version are made, supplements of this document will be published. Acknowledgement is made to the System Development Corporation for FN-654-1.

1. This document has been prepared for internal use only. It has not been reviewed by Office of Security Review, Department of Defense, and therefore, is not intended for public release. Further dissemination or reproduction in whole or in part of the material within this document shall not be made without the express written approval of The MITRE Corporation (Technical Information Department).

2. The research reported in this document was supported by the Department of the Air Force under Air Force Contract AF-33 (600) 37882.

Table of Contents

1.0 BRIEF DESCRIPTIONS OF THE COMPASS PROGRAMS

There are ten programs, one table (the COMPASS Compool), two system subroutines and nine related programs in the COMPASS system. Each of these is described in detail in later chapters; they are listed here by way of introduction.

1.1 COMPASS Programs

1. COMPASS Tape-to-Drum Program (0350). Reads the contents of the COMPASS tape onto drums, and then reads the COMPASS Control Program (0368) and the General Input (0360) and General Output (0361) subroutines into core from drums. This program then turns control over to the COMPASS Control Program.

2. Memory Print (0352). As a convenience in preparing COMPASS, it was decided to include a routine to print out the contents of any part of memory, This is not a function necessary to the successful operation of COMPASS; it is useful primarily as a checkout tool in the preparation of new versions of COMPASS.

3. Assemble Compool (0363). This program produces, from symbolic cards, a binary COMPASS Compool for the use of the Program Translator, Assemble Sequence Parameters and Environment Simulation Programs. Its output is on cards or tape.

4. Assemble Sequence Parameters (0364). This program accepts symbolic input defining the sequence of operations and transfers in the DCA program. It produces the binary sequence tables required by the DCA program. As with the Program Translator, it writes its output on cards or tape.

5. Assemble Geography (0365). This program is a revised version of the program SGO (Fixed Geography Situation Display - DCA 5.3.3.0). Its input is symbolic cards listing the geography display requirements for a SAGE sector. The program makes up the display messages, stores them on the display drums and also preserves the binary messages on cards or tape.

6. Environment Simulation (0366) It is convenient to be able to specify the contents of DCA tables and items in symbolic form, since the Compool contains the information as to how and where this is to be stored. This program will convert such symbolic data to binary form. It will not, however, store this information in the Compool-specified location, since the COMPASS system utilizes virtually all of the computer memory; its output is stored on cards or tape, in a form in which it can readily be read into memory when some specific test is to be run.

7. File Maintenance (0367) As will appear from the descriptions in the body of this document, COMPASS is most effectively used when its input and output are on tape, rather than on cards. Ideally, the only cards it should be required to read are control cards; it should punch cards only for special purposes and not as a general rule. Therefore, facility is required to handle these tapes in the same way that a programmer or EAM equipment can handle punched cards. The File Maintenance Program is this facility. It allows the programmer to rearrange, change, delete, insert, merge and examine symbolic or binary tape records.

8. COMPASS Control (0368) This control program determines the sequence of operations to be performed and controls the core environment of the other COMPASS programs. To do this, it reads and interprets control cards, stores the information from these cards, and branches control to the required program. When the operation has been completed, the program branches back to the Control Program, which takes the next request from the card reader. The COMPASS Control Program performs no useful utility function itself, but only directs the operation of the other COMPASS programs.

9. Program Translator (0370, 0371) This is the basic assembly program at the heart of the COMPASS system. It reads symbolic input (from cards or prestored tape) and converts the information into binary instructions and constants. It preserves this binary data on either cards or tape. It will also produce, if requested, a listing of the program it has assembled, showing the symbolic input and the corresponding output in octal form. Although this program consists of two separate decks, it is operated as a single program, and it is treated as such.

10. Link (0372) Translates a prestored, Lincoln symbolic deck into a COMPASS symbolic tape.

1.2 COMPASS Compool

An integral part of the COMPASS system is the binary compool. This is a complex series of tables of about 6000 registers length.

1.3 COMPASS Subroutines

1. General Input (0360) General Input is a closed subroutine allowing the programmer to specify, by means of calling sequences, various input operations and conversions. It will read cards or Hollerith tape records, and convert any field from octal or decimal to binary. It will read, as one operation, one record of up to 20 words from tape. If tape records are longer than this, information in the later words will be lost. Two calling sequences are needed to create a full binary word.

2. General Output (0361) General Output is a closed subroutine enabling the programmer to produce the equivalent of a 120 character print line in any format desired. The program, under the direction of the programmer, will convert binary information to Hollerith characters and store them in a 24 word "tape image." It will write the record on tape, convert and print the 24 words as one line on the line printer, or convert and punch the first 64 characters on a card.

1.4 Other Programs

Associated with COMPASS, but not a part of it, are several programs which are available only as binary card decks. As requirements for such programs change, the list tends to grow.

1. Core Read-In. A one-card program which reads binary cards into core. After read-in, it halts. "Program Continue" branches to the starting address, as specified on the binary END card. The program also halts on checksum error. "Program Continue" causes the error to be ignored, and the card to be processed in the normal manner.

2. Drum Read-In. A one-card program which reads in binary cards and stores their contents on drums. After read-in, it halts. The program also halts on checksum error. "Program Continue" causes the error to be ignored, and the card to be processed in the normal manner.

3. Environment Simulation Read-In. A two-card program which reads into core or onto drums binary cards of the format produced by Environment Simulation. After read-in, the program halts. It also halts on checksum error. "Program Continue" causes the error to be ignored and the card to be processed in the normal manner.

4. Octal Load. Reads into core or drums cards containing octal addresses and octal values.

5. Tape Read-In. Reads one program or a set of data from binary tape (which is in COMPASS format). It stores in core or on drums, as specified in the tape record. After reading in one set of data, it halts. "Program Continue" causes the next set of records to be read in. If the program on binary tape has been assembled for core, "Program Continue" with sense switch 1 down will operate the program since the TAPE READ IN program will branch to the address specified in the core operate record.

6. Universal Card Load. A program to read in any COMPASS binary cards or octal correction cards. For complex decks, this will simplify the read-in process. Its disadvantage is that it is longer than any of the simple read-in programs.

7. Checksum Corrector. Reads one card, computes a new checksum, and punches out a card with a corrected checksum. This operation will continue until EOF condition is reached in the card reader.

8. Punch 24-Word Cards. Produces binary cards from cards punched in symbolic form.

9. Storage Dump. A one-card program which brings the Memory Print Program (0352) from its COMPASS drum location.

2.0 OPERATING INSTRUCTIONS FOR COMPASS PROGRAMS: 1) PROGRAM TRANSLATOR; 2) ASSEMBLE COMPOOL; 3) ASSEMBLE SEQUENCE PARAMETERS; 4) ASSEMBLE GEOGRAPHY; 5) ENVIRONMENT SIMULATION; 6) FILE MAINTENANCE; AND MISCELLANEOUS INFORMATION

2.1 Starting COMPASS Operation from Scratch

1. Put COMPASS master tape on tape drive #1.

2. Put COMPASS Control Card into card reader (see individual program descriptions in this document for format of card).

3A. Decks to be loaded from cards:
Insert symbolic deck into card reader, following the COMPASS Control Card (see individual program descriptions in this document for format of decks). NOTE: If you wish to process several decks, a COMPASS Control Card should precede each deck.

3B. Decks to be loaded from tape:
Put tape on which the symbolic deck(s) has been loaded on tape drive #2. NOTE: If you wish to process several decks, they may be loaded on the tape one after another, with the order of the COMPASS Control Cards in the card reader corresponding to the order of the decks on tape.

4A. Binary output on tape desired:
Place blank tape on tape drive #3.

4B. Binary output on cards desired:
Place sufficient blank cards in card punch, and make punch ready.

5A. Symbolic output on tape desired:
Place blank tape on tape drive #4.

5B. Symbolic output on printer desired:
Place sufficient blank paper in direct line printer and make printer ready.

6. Lincoln Utility Test Memory plugboard should be used.

7. Press "Start from Test Memory."

2.2 If COMPASS Is On Drums (and no part of it has been destroyed):

1. Perform steps 2 to 5 in Section 2.1, above.

2. Press "Load From AM Drums."

2.3 Use of a Subroutine Library With The Program Translator (See Section 4.0 of this document for definitions of terms used below).

The library tape is loaded on Tape Drive #1. There is no conflict with the COMPASS master since once COMPASS is on drums, it does not refer to the master again. If the Program Translator finds a binary tape on Tape Drive #1 (presumably the COMPASS master) it will halt after logging "LOAD SUBR LIBR ON 11". The assembly will be continued after the library tape is switched on and "Program Continue" action is taken.

2.4 General Printouts

1. "READY READER" = Card reader is empty; COMPASS Control Program halts. If further work involving the COMPASS System is to be performed, put a COMPASS Control Card in the reader, followed by a symbolic deck (or a tape on the proper tape unit) and press "Continue".

2. "MORE INFORMATION REQUIRED" = COMPASS Control Card is not complete (if columns 18 - 23 are not complete or are incorrect "Illegal Card" is logged) or is incorrect; COMPASS Control Program halts; correct the card and press "Continue."

3. "ILLEGAL CARD" = Columns 18 - 23 of the COMPASS Control Card are not complete or are incorrect; COMPASS Control Program halts; correct the card and press "Continue."

2.5 "WAIT" COMPASS Control Card (Other COMPASS Control Cards are discussed in individual program descriptions in later sections of this document.)

There is, at present, a control card with "WAIT" in columns 18 - 21. This is a directive to the COMPASS Control Program to halt before reading the next control card. WAIT causes COMPASS Control to halt while select lines are switched or tapes are loaded, Upon "Program Continue", COMPASS Control will read and carry out the next control card in the reader.

2.6 General Statement -- Loading and Using COMPASS

COMPASS is available as a binary tape. The format of this tape is consistent with that described in the chapter on formats, since COMPASS is naturally capable of assembling itself as a binary tape, in the same fashion as it assembles any program system.

The first program record is a drum loading routine designed to be read in by the test memory program which is used for the Lincoln Utility System. "Drum Load" loads the entire COMPASS system on drums, and halts after logging "COMPASS ON DRUMS." The binary COMPASS tape is no longer required, unless drums are erased or destroyed by some means external to COMPASS. The system will under no conditions destroy itself.

Once COMPASS is on drums, "LOAD FROM AM DRUMS" reads in the COMPASS Control Program, and the General Input and General Output subroutines. General Input and General Output are essentially system subroutines for COMPASS. They are always in core at fixed locations and are used by all COMPASS programs. The COMPASS Control Program is controlled by cards of the format described below. It examines each card for errors and omissions. If the card is incomplete, it is logged with "MORE INFO REQ". If the card is unrecognizable, it is logged with "ILLEGAL CARD" below it. In either case the control program halts, Prepared to read another control card upon "Program Continue."

If the card is acceptable, CC calls in the required program and stores the control information in binary form in a location accessible to that routine. For the Program Translator, for example, the control information defines the modes of input and output, and whether one program is to be assembled or all programs on the input medium. Each routine returns control to the COMPASS Control Program after it has completed its operation; the COMPASS Control Program then reads in the next control card and the process continues.

2.7 All COMPASS programs which produce a binary card output automatically feed two extra cards through the punch. The user need no longer press STOP, FEED, twice on the punch, to get the binary output of a run.

2.8 The COMPASS tape-to-drum program now performs a load from AM drums function after logging COMPASS ON DRUMS. This has the effect of bringing the Control Program, General Input, and General Output into core. Whereupon the Control Program immediately tries to read a control card. To start COMPASS operating:

1. COMPASS Master on unit 1
2. Set up tapes and/or cards
3. Control cards in reader
4. Ready reader
5. Start from test memory

2.9 COMPASS Control Cards

STARTING COLUMN	18	25	30	36	42	47	
MEANING OF FIELDS	FUNCT	SYMB IN	BIN OUT	SYMB OUT	TYPE STOR		EXPLANATION
ASSEMBLY CARDS							
A01	ASSEMB	TAPE	TAPE	TAPE	DRUM	ALL	TAPES IF REQ 2,3,4, RESPECT
A02	ASSEMB	TAPE	TAPE	TAPE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A03	ASSEMB	TAPE	TAPE	TAPE	CORE	ALL	TAPES IF REQ 2,3,4, RESPECT
A04	ASSEMB	TAPE	TAPE	TAPE	CORE		TAPES IF REQ 2,3,4, RESPECT
A05	ASSEMB	TAPE	TAPE	DIR	DRUM	ALL	TAPES IF REQ 2,3,4, RESPECT
A06	ASSEMB	TAPE	TAPE	DIR	DRUM		TAPES IF REQ 2,3,4, RESPECT
A07	ASSEMB	TAPE	TAPE	DIR	CORE	ALL	TAPES IF REQ 2,3,4, RESPECT
A08	ASSEMB	TAPE	TAPE	DIR	CORE		TAPES IF REQ 2,3,4, RESPECT
A09	ASSEMB	TAPE	TAPE	NONE	DRUM	ALL	TAPES IF REQ 2,3,4, RESPECT
A10	ASSEMB	TAPE	TAPE	NONE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A11	ASSEMB	TAPE	TAPE	NONE	CORE	ALL	TAPES IF REQ 2,3,4, RESPECT
A12	ASSEMB	TAPE	TAPE	NONE	CORE		TAPES IF REQ 2,3,4, RESPECT
A13	ASSEMB	TAPE	CARD	TAPE	DRUM	ALL	TAPES IF REQ 2,3,4, RESPECT
A14	ASSEMB	TAPE	CARD	TAPE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A15	ASSEMB	TAPE	CARD	TAPE	CORE	ALL	TAPES IF REQ 2,3,4, RESPECT
A16	ASSEMB	TAPE	CARD	TAPE	CORE		TAPES IF REQ 2,3,4, RESPECT
A17	ASSEMB	TAPE	CARD	DIR	DRUM	ALL	TAPES IF REQ 2,3,4, RESPECT
A18	ASSEMB	TAPE	CARD	DIR	DRUM		TAPES IF REQ 2,3,4, RESPECT
A19	ASSEMB	TAPE	CARD	DIR	CORE	ALL	TAPES IF REQ 2,3,4, RESPECT
A20	ASSEMB	TAPE	CARD	DIR	CORE		TAPES IF REQ 2,3,4, RESPECT
A21	ASSEMB	TAPE	CARD	NONE	DRUM	ALL	TAPES IF REQ 2,3,4, RESPECT
A22	ASSEMB	TAPE	CARD	NONE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A23	ASSEMB	TAPE	CARD	NONE	CORE	ALL	TAPES IF REQ 2,3,4, RESPECT
A24	ASSEMB	TAPE	CARD	NONE	CORE		TAPES IF REQ 2,3,4, RESPECT
A25	ASSEMB	CARD	TAPE	TAPE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A26	ASSEMB	CARD	TAPE	TAPE	CORE		TAPES IF REQ 2,3,4, RESPECT
A27	ASSEMB	CARD	TAPE	DIR	DRUM		TAPES IF REQ 2,3,4, RESPECT
A28	ASSEMB	CARD	TAPE	DIR	CORE		TAPES IF REQ 2,3,4, RESPECT
A29	ASSEMB	CARD	TAPE	NONE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A30	ASSEMB	CARD	TAPE	NONE	CORE		TAPES IF REQ 2,3,4, RESPECT
A31	ASSEMB	CARD	CARD	TAPE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A32	ASSEMB	CARD	CARD	TAPE	CORE		TAPES IF REQ 2,3,4, RESPECT
A33	ASSEMB	CARD	CARD	DIR	DRUM		TAPES IF REQ 2,3,4, RESPECT
A34	ASSEMB	CARD	CARD	DIR	CORE		TAPES IF REQ 2,3,4, RESPECT
A35	ASSEMB	CARD	CARD	NONE	DRUM		TAPES IF REQ 2,3,4, RESPECT
A36	ASSEMB	CARD	CARD	NONE	CORE		TAPES IF REQ 2,3,4, RESPECT

```
     ASSEMBLE COMPOOL CARDS

C01             COMPOL TAPE TAPE  TAPE              TAPES IF REQ 2,3,4   RESPECT
C02             COMPOL TAPE TAPE  DIR               TAPES IF REQ 2,3,4   RESPECT
C03             COMPOL TAPE TAPE  NONE              TAPES IF REQ 2,3,4   RESPECT
C04             COMPOL TAPE CARD  TAPE              TAPES IF REQ 2,3,4   RESPECT
C05             COMPOL TAPE CARD  DIR               TAPES IF REQ 2,3,4   RESPECT
C06             COMPOL TAPE CARD  NONE              TAPES IF REQ 2,3,4   RESPECT
C07             COMPOL CARD TAPE  TAPE              TAPES IF REQ 2,3,4   RESPECT
C08             COMPOL CARD TAPE  DIR               TAPES IF REQ 2,3,4   RESPECT
C09             COMPOL CARD TAPE  NONE              TAPES IF REQ 2,3,4   RESPECT
C10             COMPOL CARD CARD  TAPE              TAPES IF REQ 2,3,4   RESPECT
C11             COMPOL CARD CARD  DIR               TAPES IF REQ 2,3,4   RESPECT
C12             COMPOL CARD CARD  NONE              TAPES IF REQ 2,3,4   RESPECT
C13             COMPOL            TAPE         DCP  TAPE 4
C14             COMPOL            DIR          DCP

     ASSEMBLE SEQUENCE PARAMETER CARDS

S01             SEQPAR TAPE TAPE  TAPE              TAPES IF REQ 2,3,4,  RESPECT
S02             SEQPAR TAPE TAPE  DIR               TAPES IF REQ 2,3,4,  RESPECT
S03             SEQPAR TAPE TAPE  NONE              TAPES IF REQ 2,3,4,  RESPECT
S04             SEQPAR TAPE CARD  TAPE              TAPES IF REQ 2,3,4,  RESPECT
S05             SEQPAR TAPE CARD  DIR               TAPES IF REQ 2,3,4,  RESPECT
S06             SEQPAR TAPE CARD  NONE              TAPES IF REQ 2,3,4,  RESPECT
S07             SEQPAR CARD TAPE  TAPE              TAPES IF REQ 2,3,4,  RESPECT
S08             SEQPAR CARD TAPE  DIR               TAPES IF REQ 2,3,4,  RESPECT
S09             SEQPAR CARD TAPE  NONE              TAPES IF REQ 2,3,4,  RESPECT
S10             SEQPAR CARD CARD  TAPE              TAPES IF REQ 2,3,4,  RESPECT
S11             SEQPAR CARD CARD  DIR               TAPES IF REQ 2,3,4,  RESPECT
S12             SEQPAR CARD CARD  NONE              TAPES IF REQ 2,3,4,  RESPECT

     ASSEMBLE ENVIRONMENT SIMULATION CARDS

E01             ENVSIM TAPE TAPE  TAPE  DRUM        TAPES IF REQ 2,3,4,  RESPECT
E02             ENVSIM TAPE TAPE  TAPE  CORE        TAPES IF REQ 2,3,4,  RESPECT
E03             ENVSIM TAPE TAPE  DIR   DRUM        TAPES IF REQ 2,3,4,  RESPECT
E04             ENVSIM TAPE TAPE  DIR   CORE        TAPES IF REQ 2,3,4,  RESPECT
E05             ENVSIM TAPE TAPE  NONE  DRUM        TAPES IF REQ 2,3,4,  RESPECT
E06             ENVSIM TAPE TAPE  NONE  CORE        TAPES IF REQ 2,3,4,  RESPECT
```

```
E07              ENVSIM TAPE CARD  TAPE  DRUM    TAPES IF REQ 2,3,4, RESPECT
E08              ENVSIM TAPE CARD  TAPE  CORE    TAPES IF REQ 2,3,4, RESPECT
E09              ENVSIM TAPE CARD  DIR   DRUM    TAPES IF REQ 2,3,4, RESPECT
E10              ENVSIM TAPE CARD  DIR   CORE    TAPES IF REQ 2,3,4, RESPECT
E11              ENVSIM TAPE CARD  NONE  DRUM    TAPES IF REQ 2,3,4, RESPECT
E12              ENVSIM TAPE CARD  NONE  CORE    TAPES IF REQ 2,3,4, RESPECT
E13              ENVSIM CARD TAPE  TAPE  DRUM    TAPES IF REQ 2,3,4, RESPECT
E14              ENVSIM CARD TAPE  TAPE  CORE    TAPES IF REQ 2,3,4, RESPECT
E15              ENVSIM CARD TAPE  DIR   DRUM    TAPES IF REQ 2,3,4, RESPECT
E16              ENVSIM CARD TAPE  DIR   CORE    TAPES IF REQ 2,3,4, RESPECT
E17              ENVSIM CARD TAPE  NONE  DRUM    TAPES IF REQ 2,3,4, RESPECT
E18              ENVSIM CARD TAPE  NONE  CORE    TAPES IF REQ 2,3,4, RESPECT
E19              ENVSIM CARD CARD  TAPE  DRUM    TAPES IF REQ 2,3,4, RESPECT
E20              ENVSIM CARD CARD  TAPE  CORE    TAPES IF REQ 2,3,4, RESPECT
E21              ENVSIM CARD CARD  DIR   DRUM    TAPES IF REQ 2,3,4, RESPECT
E22              ENVSIM CARD CARD  DIR   CORE    TAPES IF REQ 2,3,4, RESPECT
E23              ENVSIM CARD CARD  NONE  DRUM    TAPES IF REQ 2,3,4, RESPECT
E24              ENVSIM CARD CARD  NONE  CORE    TAPES IF REQ 2,3,4, RESPECT

ASSEMBLE GEOGRAPHY CARDS

G01              GEOGR  TAPE TAPE                TAPES IF REQ 2,3    RESPECT
G02              GEOGR  TAPE CARD                TAPES IF REQ 2,3    RESPECT
G03              GEOGR  CARD TAPE                TAPES IF REQ 2,3    RESPECT
G04              GEOGR  CARD CARD                TAPES IF REQ 2,3    RESPECT

  TAPE FILE MAINTENANCE CARDS

T01              POS 1R            REW THEN POS TAPE 1 TO IDT IN COLS 25-30
T02              POS 1                      POS TAPE 1 TO IDT IN COLS 25-30
T03              POS 2R            REW THEN POS TAPE 2 TO IDT IN COLS 25-30
T04              POS 2                      POS TAPE 2 TO IDT IN COLS 25-30
T05              POS 3R            REW THEN POS TAPE 3 TO IDT IN COLS 25-30
T06              POS 3                      POS TAPE 3 TO IDT IN COLS 25-30
T07              POS 4R            REW THEN POS TAPE 4 TO IDT IN COLS 25-30
T08              POS 4                      POS TAPE 4 TO IDT IN COLS 25-30
T09.1            LOG 1             CATALOG BINARY OR HOLLERITH TAPE ON UNIT X
T09.2            LOG 2             CATALOG BINARY OR HOLLERITH TAPE ON UNIT X
T09.3            LOG 3             CATALOG BINARY OR HOLLERITH TAPE ON UNIT X
T09.4            LOG 4             CATALOG BINARY OR HOLLERITH TAPE ON UNIT X
T10.1            DUPLIC 1
```

T10.2	DUPLIC 2	
T10.3	DUPLIC 3	
T11	CMPARE	COMPARE TAPES ON UNITS 2 AND 3
T12	PUNCH	PUNCH A BINARY DECK OF THE IDT GROUP ON 3
T13	FILE	WRITE A FILE IDT REC ON TAPE 3 COLS 25-30
T14	END	WRITE AN END IDT REC ON TAPE 3 COLS 25-30
T15	UPDATE CARD	UPDATE HOLLERITH TAPE ON 4 NEW TAPE ON 2
T16	MERGE	BIN TAPES MASTER ON 2 CHANGE ON 4 NEW ON 3
T17	UPDATE TAPE	UPDATE HOL TAPE OLD 4 CHANGE 3 NEW 2
	WAIT	CHANGE TAPE SELECT LINES AND CONTINUE
	LINK	

3.0 Binary Card and Tape Formats

The assembly programs of the COMPASS system (Program Translator, Assemble Compool, Assemble Fixed Geography, Assemble Sequence Parameters, and Environment Simulation, and also certain of the File Maintenance functions) produce a binary output on either tape or cards. Shown below are the formats of such information. COMPASS programs will only handle binary information organized in the following manner, with the exception of DUPLICATE and COMPARE, which do not examine format.

3.1 Binary Cards

Ident. characters are gangpunched in columns 1 - 6 of each card. If the binary deck contains a program, columns 7 - 9 of each card will contain the compool with which this deck was assembled.

1. Cards with drum locations

9's left row: Bit 02 equals "1" indicates a drum card.
Bits 04 - 09 contain drum field.
Bits 10 - 15 contain number of words per card. (Does not include itself or checksum word, hence, maximum value is 22)
Bits 16 - 31 contain drum location.
Bit 17 equals "1" indicates auxiliary drums.

9's right row: Contains the complement checksum, that is, the complement of the sum of all the words on the card not including itself (the complement checksum).

8's left row to 12's right row: Contains the binary information, maximum of 22 words.

2. Cards with core locations. Exactly as described above with the following exceptions in the;

9's left row: Bit 02 equals zero
Bits 04 - 09 equal zero
Bits 00 and 16 - 31 contain core location

3. End card. Applies to either core or drum cards

9's left row: Bit 01 equals one
Bits 00 and 16 - 31 -- core location for the read-in program to branch to.

9's right row: Contains the complement checksum.

4 Environment Simulation Cards

9's left row

LH Word: Bit 03 equals "1" indicates ENVSIM card.

RH Word: Bits 10 - 15 contain number of words per card. (Does not include itself or checksum word, hence maximum value is 22.)

9's right row

Full Word: Contains the complement checksum.

8's left row

LH Word: Bits 10 - 15 contain drum field; blank when core address is involved.

RH Word: Bits 1 - 15 contain address.

8's right row

Full Word: Contains item value CONVERSION. The rest of the card is filled to the maximum of alternating addresses and contents as in the 8's row, left and right.

3.2 Binary tapes produced by COMPASS have five different types of records

NOTE: Type of record is a number such that either bits 30 or 31, or both, of word 1, contains a "1", thereby identifying this as a binary tape. A Hollerith tape contains zero in these bits.

1. The file identification, storage identification and end identification records are three-word records and differ only in bits 28 - 31 of word 1. These bits contain the record type. The format of these records follows:

Word	Bits	Contents
1	00-23	4 Hollerith characters identifying the file or data.
	28-31	Type of record
2	18-29	2 Hollerith characters completing the identification of the file or data.
3	00-31	Complement checksum

File identification (Type 1)
This is the first record of a file. A "file" is a programming system (e.g., DCA, TADOR, SGS, COMPASS). A file is structurally made up of identification groups.

Storage identification (Type 2)
This precedes one or more storage records and contains identification for them. This record could identify a program, the binary Compool, or any kind of binary data, therefore, the storage identification record and the storage record or records it identifies are called an identification group.

End identification (Type 5)
This is the last record of a file and is followed by an end-of-file mark.

2. The storage record contains the actual binary data. There are two types of storage records -- core or drum.

Word	Bits	Contents
1	00-15	Number of data words. Maximum of 2048 or one drum field.
	28-31	Type of record
2	00-31	Contains the information necessary to locate the data in core or on drums
3 - (n-1)	00-31	The binary data information
n	00-31	Complement checksum

NOTE: n is a number between 4 and 2051 inclusive.

Core record (Type 6)
Word 2 contains in bits 00 and 16 - 31 the starting core location for this block data.

Drum record (Type 3)
Word 2 contains an SDR complete with drum field and the starting drum location for this block of data.

3. Environment Simulation Records (Type 9)

Word	Bits	Contents
1	00-15	Number of data words, maximum of 200
	28-31	Type of record
2	00-31	Blank
3 to (n-1)	00-31	The binary data information
n	00-31	Complement checksum ($4 \leq n \leq 203$)

4. Core operate (Type 7)

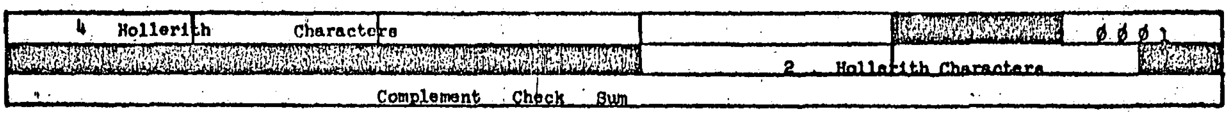
This is a three-word record following one or more storage records for drum or core containing a program. Word 1 contains the type of record in bits 28 - 31. Word 2 contains the core address specified in the END card of the program when it was assembled. The third word contains the complement checksum.

This record provides the same function for a program assembled to binary tape that an end card on a binary deck performs. That is, a program may be assembled to binary tape and if core assembly is specified, it may be read in and operated. The tape read-in program branches to the address specified in the core operate record.

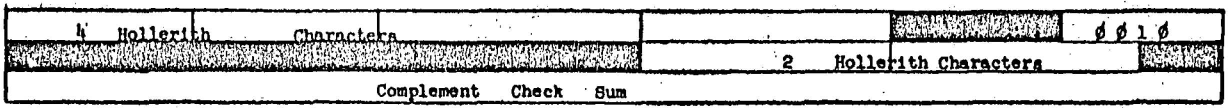
Binary Tape Format

ø 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

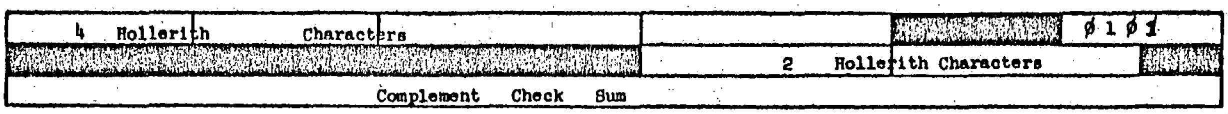
File Identification



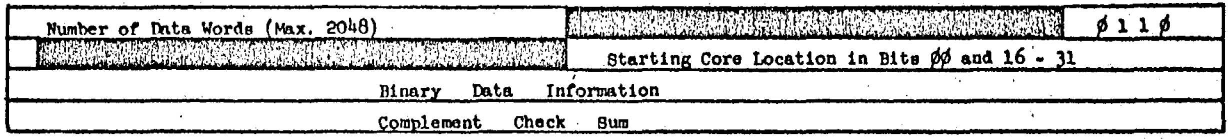
Storage Identification



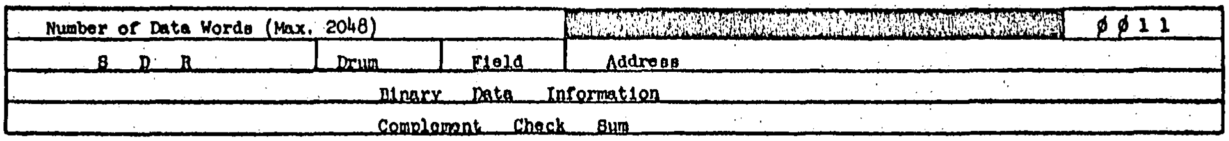
End Identification



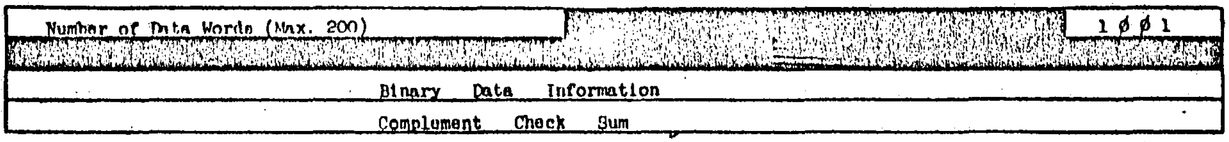
Core Storage Record


Number of Data Words (Max. 2048)
ø 1 1 ø
Starting Core Location in Bits øø and 16 - 31
Binary Data Information
Complement Check Sum

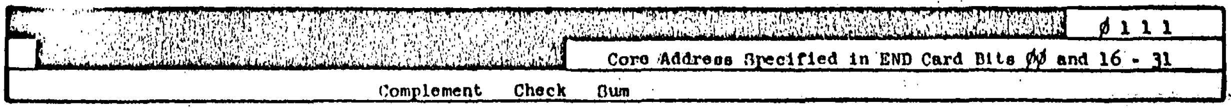
Drum Storage Record



Environment Simulation Record



Core Operate Record



4.0 PROGRAM Program Translator

4.1 IDENT 0370, 0371

4.2 FUNCTIONS

Program Translator is a two-pass assembly program. It converts symbolic cards in the format specified below into binary words recognized by the computer's circuits and stores the binary form of the program on a permanent storage medium (either punched cards or magnetic tape). In addition, if desired, a listing of the assembled program in symbolic and absolute, with programmer's comments and an error printout, may be produced on delayed tape or directly on the line printer.

Note: "Compool Dependent" programs are ones which use tags defined in the COMPASS Compool. DCA and CCA programs are examples of "Compool Dependent" programs. Any programming system using a Compool is "Compool Dependent."

4.3 OPERATION See Section 2.0 of this document.

4.4 INPUTS

4.4.1 COMPASS Control Card

cols. 25 - 28 = "TAPE" or "CARD" (symbolic input medium)
cols. 30 - 33 = "TAPE" or "CARD" (binary output medium)
cols. 36 - 39 = "TAPE" (symbolic output on tape) or "DIR" (cols. 36 - 38, symbolic output on printer) or "NONE" (no symbolic output)
"TAG" (tag table on the direct line printer, Cols. 36 - 38)
cols. 42 - 45 = "DRUM" or "CORE" (type of storage)
cols. 47 - 49 = "ALL" or (blank) ("ALL" permits assembly of all programs on the input tape with one COMPASS control card. If these columns are left blank, only one program is assembled and control is returned to COMPASS control).

4.4.2 Symbolic Input Cards

Note: There is a COMPASS plugboard wired and available in the Santa Monica card room for the card-to-tape machine. It rearranges columns 17 - 80 of the instruction card in the following manner.

Card Columns:		
	17 - 71, 78 - 80	no change
	8 - 13	now in columns 72 - 77

This allows the card numbers to appear on the listing produced by the Program Translator. At the cost of 6 columns of comments, the programmer has in one listing all the information required to develop or make changes to a program being worked on, especially those programs maintained by the card room.

cols. 18 - 22 = symbolic tag

Symbolic Location

a. For Compool Dependent Programs = DDL (digit - digit - letter); no zero suppression

b. For Non-Compool Dependent Programs = any alphanumeric characters and/or blanks

Items

a. For Compool Dependent Programs = LLLL (letter - letter - letter - letter)

b. For Non-Compool Dependent Programs = any alphanumeric characters and/or blanks

Table for use with Compool Override Cards

a. For Compool Dependent Programs = LLLD (letter - letter - letter - digit) or LLL (letter - letter - letter)

b. For Non-Compool Dependent Programs = any alphanumeric characters and/or blanks

col. 24 = index register

a. For indexable instructions, blank, or a number between 1 and 5 inclusive is permissible.

b. Compool override cards

(1) "I" defines this card as an item card

(2) "T" defines this card as a table card

cols. 25 - 27 = operation code

a. Standard Q-7 instruction codes

b. All the new 65K memory instructions (CMF, CML, CMR, CMM, CDF, CDL, CDR, CDM, TOB, TTB, CAC).

c. Assembly instructions

IDT - Identification. This is the first card of every deck. The characters which appear in the director of this card will be gang-punched on all binary cards of the deck produced (in cols. 1 - 6); it appears in the storage identification record, if the output is to tape. If the first character of the director is a 5, this program is presumed to be a Compool Dependent program: Compool dependent restrictions apply and the Compool is used.

LOC - Location. This is not needed with Compool dependent programs, for the starting core address is specified in the Compool. If the LOC card is inserted in a Compool dependent program deck the location specified overrides the core address contained in the Compool. The octal number in the director of this card defines the core location of the succeeding instructions or constants. As many LOC cards as are desired may be inserted. The number of LOC cards in an instruction deck is up to the programmer.

DRM - Drum. This is not required with Compool dependent programs, for the reason given under LOC above. This card does the same thing for a drum location as LOC does for core locations. The drum field is indicated in the index interval. Aux. drum locations are indicated by a 4 in column 37 of the director. Columns 38 - 41 of the director contain the drum location in octal. The number of DRM cards in an instruction deck is up to the programmer.

SKP - Skip. Program Translator is to skip the decimal number of registers specified in the director at this point.

DIT - Ditto. The previous instruction or constant is to be repeated the number of times specified in the director. The content of the director is interpreted as a decimal number.

END - The END card goes at the end of the symbolic deck and can contain an absolute location, or a symbolic tag in the director. This location is put in the last card of the binary deck or in the core operate record on binary tape. This is so that the read-in program may branch to this location after the program is read into the computer. If the director of the END card is left blank it is assumed to be the core location specified in the first LOC card of the deck.

d. Pseudo-instructions

POS - Position. To be used with an item tag in the director. The instruction will cause the Program Translator to substitute the machine code for an FCL, with a right-half word computed such that the item will be cycled in order to put the least significant bit in the accumulator bit position specified in the index interval of the POS instruction.

RES - Restore. This will cause Program Translator to substitute an FCL with the right-half word computed such that the item is positioned ready for depositing in its original storage location.

SDX - Program Translator will put a "1" into bit 17 to address aux. drums. In other respects, this instruction is the same as SDR.

HOL - Hollerith. The 5 characters appearing in columns 36 - 40 of the director are to be converted to 6-bit Hollerith in normal prestored form (inverted) and stored either as a full 32-bit constant word or as an RC word, depending on whether HOL appears in the op-code field or as the left half of an RC word.

CPO - Compool override. These cards permit a Compool dependent program to override the Compool, with respect to item or table tags. They permit non-Compool dependent programs to set up tables and items outside the range of a program. In effect they permit a non-Compool dependent program to set up a private Compool for itself. These cards are actually an extension of symbolic coding.

For table compool override card, the director (cols. 36 - 41) contains the octal address of the first word of the table.

For item compool override card, the index interval contains the number of bits in decimal, columns 34 - 35 of the RC word left half contains the most significant bit position of the item and the director (columns 36 - 40) contains the symbolic tag of the table in which this item is located.

TOB, TTB - When used with an item tag in the director, the index interval will contain the relative bit position of the most significant bit to be tested. Thus, if TSTS is a five-bit item and the two least significant bits to be tested, the instruction TTBØ3 should be used. To ascertain the relative bit position of any bit of an item call the most significant bit Ø, the next bit 1.

e. General Input and General Output Calling Sequences, etc.

Two general purpose programs dealing with in/out are provided, to save the programmer the time-consuming job of conversions. General Input and General Output are directed by 2-word calling sequences, immediately following a BPX to them. As many consecutive sets of these calling sequences as are needed may follow one reference to either General Input or General Output (BPX). A list of General Input and General Output calling sequence instructions follows:

CLR --	Clear input or output image
MOV --	Move Hollerith characters
RCD --	Read a card
RTP --	Read tape
WPR --	Write printer
WPU --	Write punch
WTP --	Write tape
OCI --	Octal integer
OCA --	Octal address
OCF --	Octal fraction
SDI --	Signed decimal integer
UDI --	Unsigned decimal integer
SDF --	Signed decimal fraction
UDF --	Unsigned decimal fraction

For more information on General Input and General Output, see Sections 12.0 and 13.0 of this document.

In the first word of a calling sequence, the right-half word is either a symbolic or an absolute address. The letters ACC (accumulator) may be written in columns 36 - 38 of the director.

In the second word of the calling sequence, the instruction field is divided into 2 parts, columns 24 - 26 and columns 27 - 29. Two unsigned decimal integers are written here with the least significant digit to the right of the column groups, and leading zeros suppressed. The right-half word has an unsigned decimal integer with the least significant bit in column 41 and leading zeros suppressed. The programmer may use the first word of a calling sequence as an RC word; but he may not use the second word of a calling sequence in an RC word.

cols. 28 - 29 = index interval

a. Col. 28:
An "A" with ADD, ADB, SUB, AOR, XAC, STA, CSW, RST, TAD, TSU operation codes designates a 17 bit address. A "1" is inserted in bit position 12 of the binary instruction by the Program Translator.

b. Col. 29
Concerns instructions causing overflow.
(1) Blank -- Set indicators, no suppression
(2) L -- Set indicators, suppress left alarm
(3) R -- Set indicators, suppress right alarm
(4) B -- Set indicators, suppress both alarms
(5) E -- (Everything) - Suppress all overflow alarms and indicators.

c. Cols. 28 - 29
(1) Octal number with BSN, PER, SDR, SDX, SEL, DRM.
(2) Decimal number with BPX, POS, RES, TOB, TTB, CPO, RDS, WRT (interleave codes) and all General Input or General Output calling sequence instructions requiring an index interval.

cols. 30 - 45 = reference

a. RC words-columns 30 - 45
The letters "RC" stand for "register containing" and the function performed by these type words is primarily one of ease of programming. Program Translator assigns an absolute location at the end of the program as the right-half word of the instruction calling for an RC word and stores in that location the value specified. An example follows:

```
ETR        177400        000377
```

Program Translator assigns a location to the RC word at the end of the assembled program and fills the right-half word of the ETR with that address.

IDT, END, DIT, SKP, LOC, DRM, CPO: these characters and the second word of a calling sequence may not be used in the RC word field. The instructions which will take an RC word are Add class, Multiply class. Compare class. ETR, LDB, and LDC.

b. The director, cols. 36 - 41
The self reference symbol. A $ in column 36 of an instruction card is understood by the Program Translator to refer to the contents of the location counter at that instruction, for example:

```
BSN14        $
```

No tag is needed.

This may be used in conjunction with the increment or decrement feature. To wit: if it is desired to turn off the left overflow light and operate the next instruction:

```
BSN12        $ + 1
```

The director field may contain either symbolic tags or constants. Constants are written at the right of the director field. Symbolic tags are written starting at the left of the director field. The rules for tags for non Compool dependent programs are the same as the rules for tags in the location field - any five alphanumeric characters. A list of permissible tags for Compool dependent programs follows:

Location tags:	DDL (digit - digit - letter)
Item tags:	LLLL (letter - letter - letter - letter)
Table tags:	LLLD (letter - letter - letter - digit)
Program tags:	LLL (letter - letter - letter)
Temporary storage	TPYØ

Each Compool dependent program must have a TPYØ table set up for itself in section E of the Compool if temporary storage registers are used. TPYØ may also be defined using a CPO card.

"T" -- if test memory in the 65K memory computer is desired, a T in column 41 will cause Program Translator to fill in an address of 377760 which corresponds to 20000 in the 8K machine. Relative addressing may be used to get successive registers of test memory. For example, T + Ø15 is the live register.

c. Increment or decrement, columns 42 - 45.
A "+" or a "-" in column 42 and a three-digit decimal number ranging from 1-999 in columns 43 - 45. A blank is interpreted as a zero.

4.5 OUTPUTS

4.5.1 Cards -- Program Translator produces a binary deck with the identification characters (from IDT card) gangpunched in columns 1 - 6, and the Compool used to assemble this program is gangpunched in columns 7 - 9, of each card in the deck. The format of the binary deck is described in the chapter of this document on binary format(Section 3.0).

4.5.2 Tape -- Program Translator produces binary records on tape. The format of these records is specified in the chapter of this document on binary format. The three-word storage identification record contains the six characters of the director field of the IDT card and the Compool used to assemble the program in 6-bit Hollerith. One or more storage records following the storage identification record contains the program. More than one storage record is produced for one program if:

1. The program is over one drum field in length (2048 words).
2. There are SKP cards in the symbolic deck.
3. There is more than one LOC card and the program is being assembled with core addresses.
4. There is more than one DRM card and the program is being assembled with Drum addresses.

4.5.3 PRINTOUTS

4.5.3.1 Listing of assembled program (see sample listing at end of this chapter). All assemble instruction cards and CPO cards do not print 1, 3, 4, and 5 below. The RC words at the end of the assembled program also appear in the listing. The listing for RC words prints 1, 3, 4 and 5 below, only.

1.	Print wheels 1 - 8:	Drum field and drum location.
2.	Print wheels 11 - 45:	Contents of columns 18 - 45 inclusive of the input instruction card.
3.	Print wheels 48 - 53:	Core location of this instruction or constant, as an octal address.
4.	Print wheels 57 - 62:	Contents of the left-half word of this instruction or constant, in octal.
5.	Print wheels 66 - 71:	Contents of the right-half word of this instruction or constant, in octal.
6.	Print wheels 75 - 107:	The contents of the comment field of the input instruction card, columns 46 - 77 inclusive.
7.	Print wheels 111 - 120:	Errors in the assembled program detected by the second pass of Program Translator.

4.5.3.2 For the convenience of field personnel, an optional symbolic output is produced by the Program Translator. Since most field sites do not have tape-to-printer equipment, a tag table may be had instead of the usual listing. The control card for assembly must have the word TAG in columns 36 - 38, the remainder of the card is described in Section 4.4.1.

4.5.4 SPECIAL POINTS

1. Errors not causing the program to halt. Errors detected in the assembled program. All error lines of the listings are logged direct whatever the form of symbolic output requested.

   a. "INDETR LHW": Indeterminate left-half word is generally caused by either a special symbol in the instruction left-half word field, or a decimal integer such as 8 or 9 in an octal constant, or an op-code punched in the wrong columns of the instruction left-half word field.

   b. "INDETR RHW": Indeterminate right-half word is generally caused by a special symbol such as an asterisk in the director field.

   c. "UNDEFINED": This appears when Program Translator cannot find the symbolic tag contained in the director field in either the tag table or the Compool override table, and, for Compool dependent programs, cannot find it in the Compool either.

   d. "DUPL TAG": Duplicate tag appears on the line which contains the duplicate in the director. The address of the first duplicate is used for all the others.

   e. "ILL USE RC": Illegal use of the RC word. This is caused by the use of an RC word with an instruction left-half word which does not take an RC word or an illegal instruction in the left-half RC word field, such as DIT or CPO.

   f. "INDETR RC": Indeterminate RC. This is caused by the same sort of error mentioned under indeterminate left-half word except the error would appear in the left-half word in the RC field.

   g. "EXCESS RC": Excessive number of RC words. At present this will be caused by a program having more than 200 different RC words in it.

   h. "MEMORY OVERFLOW": This means the program wraps around memory.

   i. "XXXXX ILLEG TAG": Generally caused by an illegal symbol in the tag field of the instruction card. The X's will contain the illegal tag.

2. Errors causing the program to halt. All error lines of listings are logged direct.

   a. "NO IDT CARD": If symbolic input is from cards, attach IDT card to deck, put deck in card reader, and press "CONTINUE." If input is from tape, nothing can be done.

b. "READY I/O UNIT 01": Card reader not ready. Ready card reader and continue. This error printout also means there may not be an END card on the deck being assembled in the card reader. In this case, add the remainder of the deck with END card and press "continue." If there is no END card on the tape, no recovery is possible.

c. "READY I/O UNIT 1X": Tape X not ready. Ready tape X and press program "continue."

d. "READY I/O UNIT 02": Punch not ready. Ready card punch and press program "continue."

e. "TOO MANY TAGS": This occurs if the program being assembled has over 900 tags. No recovery possible.

f. "LOAD SUBR LIB ON 11": If the COMPASS master is on unit 11 when the subroutine library tape is needed, load subroutine library on 11, and "continue."

3. Information printouts: If assembly is from prestored tape, the IDT card of each program assembled will be logged on the line printer. All errors will be logged directly, even if no symbolic output is indicated.

4. Definitions Pertaining to Subroutine Libraries

<u>Primary subroutine</u> - A subroutine which is self sufficient, that is, it does not refer to any other subroutine.

<u>Second order subroutine</u> - This subroutine refers to one or more primary subroutines.

<u>Third order subroutine</u> - This refers to one or more subroutines, at least one of which is a second order.

The extension to subroutines of higher order is obvious.

<u>Subroutine Location tags</u> - Tags of any five alphanumeric characters or blanks with the exception of all blanks or all numeric characters. For use with compool dependent programs, DDLBB (digit - digit - letter - blank - blank) is the format.

<u>Subroutine reference</u> - This is a subroutine location tag which defines a subroutine. This tag is in the director of the IDT card of the subroutine, columns 36 - 40. It appears as the location tag of the first instruction of the subroutine, always a STAA (Store Address). It also identifies the subroutine in the index and appears as the director of the SBR op code.

Subroutine request - This is an SBR with a subroutine reference in the director. This is used to call for a particular subroutine and can be written in the program to be assembled, or on a subroutine in the library.

Subroutine index - A statement of the relationships between the subroutines in the library. The subroutine index is made up of a number of cards. The format of a subroutine index card is as follows: Columns 17 - 76 of the card are partitioned into 12 fields of five columns each. The first field, starting at the left, contains a non-primary subroutine reference. Successive fields contain the different subroutine references directly used in the non-primary routine, up to a maximum of eleven. Each non-primary subroutine must be defined in this way. The whole of it is the index.

Library Tape - This is a prestored tape consisting of the index and an END card; then follows the subroutines and one end-of-file. Each subroutine is enclosed, has an IDT card with its subroutine reference in the director, the STAA with the subroutine reference in the location field, followed by the remainder of the subroutine and an END card.

5. A statement on the limits of the present (8K) Program Translator

   1. Tags: maximum, 900; minimum, 3, with the exception that a program without any tags may be assembled.

   2. RC words: maximum, 200.

   3. Cards in input deck: maximum, 6120.

   4. CPO cards: maximum, 70.

6. The Program Translator accepts symbolic compool references to drums, such as:

```
SDR      TDT1
LDC      TDT1
RDSØ3    TDT1 - (Read TDT1, TDT2, TDT3)
```

7. At present, the Program Translator will not accept compool parameter tags, although this feature may be included for the combat center program.

8. Program Translator does not convert the letter "O" to Ø or vice versa.

9. Compool dependent programs may not suppress zeros in internal location tags.

10. If a table or program used by a compool dependent program being assembled is not in section E (Formerly section 4) of the compool for this program, the tag will be undefined since the Program Translator does not look in Section T (indexable) or Section P (non-indexable) sections of the compool.

11. Further definition of the "Director" of an instruction card. The director field on the symbolic card is interpreted by the Program Translator in several ways, depending on its context. If it is symbolic, it may in some cases (POS, RES, LDB, ETR) be interpreted as a function of bit position of an item; in other cases (SDR, RDS, WRT) it may be interpreted as drum location or number of words. With a General Input or Output calling sequence, instruction ACC is translated to zeros and means the accumulator. In all other cases, the symbol is replaced by its correlative 17 bit binary core address. Core address is always 17 bits long. This has some consequences which should not be overlooked, particularly when the left-half word contains a constant rather than an instruction. Consider this example:

    - Ø, 6ØA becomes Ø.77777, Ø.Ø226Ø

    6ØA is register 226Ø and notice the constant left-half word has been changed in value.

12. Programs over a drum field in length.

    A program being assembled for drums having more than 2048 (decimal) words will have the excess (over 2048) assigned to the next higher drum field number. The programmer should take care that the drum field specified for assembly plus one is an existent drum, for the Program Translator will not.

```
AB*C  ILLEGAL TAG                                                                                                    (1)

                  IDT              SAMPLE03Z                                A PICTURE IS WORTH 1000 WORDS
                  DRM02                                                     DRUM FIELD 2 LOCATION ZERO
                  LOC                 1000                                  STARTING CORE LOCATION IN OCTAL
                                                                          THE FOLLOWING ARE VARIOUS CONST.
002 0000  ONE                                    001000   000000   000000   NOT A COMMENT CARD BECAUSE TAG
002 0001  CONST        4           +    10     001001   000014   000012   OCTAL LHW  DECIMAL RHW
002 0002          -    9            175231     001002   177766   175231   DECIMAL LHW OCTAL RHW
002 0003               9                 1     001003   000000   000000   LHW CONSIDERED OCTAL NO + OR -     INDETR LHW
002 0004  FRACT   +.8923            -.1234     001004   071067   170063   SIGNED DECIMAL FRACTION LHW.RHW
                                                                          DECIMAL DIRECTOR INSTRUCTIONS
002 0005  ABCD3   DSL                   10     001005   004000   000012   A SAMPLING OF SHIFT CLASS
002 0006  EFGH4   ASL                    9     001006   004200   000011      NOTE  VARIOUS TAGS
002 0007  IJKL5   OCL                   17     001007   004600   000021   AND CYCLE
002 0010  MNOP6   SLR                   15     001010   000243   000017   MISCELLANEOUS CLASS
002 0011  QRST7   2XIN                  10     001011   027540   000012
002 0012  UVWX8   1ADX                  11     001012   017700   000013
                                                                          OVERFLOW INDICATORS AND 17 BIT
                                                                          ADDRESS CLASS INSTRUCTIONS
002 0013  YZ9     TADA                         001013   001113   000000   SET INDIC NO SUPPRESS
002 0014          SUBAL                        001014   001351   000000   SET INDIC SUPPRESS LEFT ALARM
002 0015          ADRAR                        001015   003452   000000   SET INDIC SUPPRESS RIGHT ALARM
002 0016          ADDAB                        001016   001050   000000   SET INDIC SUPPRESS BOTH  ALARMS
002 0017          TSUAE                        001017   001414   000000   SUPPRESS EVERYTHING ALARMS  INOT
002 0020          CSLA                         001020   000210   000000
002 0021          RSTA                         001021   003350   000000
002 0022          STAA                         001022   003410   000000
002 0023          4XACA                        001023   047650   000000
002 0024          ADBA                         001024   001153   000000
002 0025          FSTA                         001025   003240   000000   NOT 17 BIT ADDR BIT 12 NOT EQ 1
                                                                          OCTAL AUXILIARY INSTRUCTIONS
002 0026  2A      BSN14            2A          001026   005214   001026
002 0027  22A     SEL11            22A         001027   006211   001027
002 0030  222A    PER01            222A        001030   000101   001030
002 0031  2222A   SDR47            040020      001031   006147   040020   NOTE NO LEGALITY CHECK NO AUX
                                                                          DRUM 47 IS ON Q-7
002 0032          SDX10                 10     001032   006110   040010   AUX DRUM
                  DRM02                 50                                  OCTAL AUX NOTE ONLY DRUM LOC IS
                                                                          CHANGED NOT CORE LOC IN ABSOLUTE
                                                                          DECIMAL AUXILIARY INSTRUCTIONS
                                                                          NOTE DEC DIRECTORS ON NEXT 3 INS
002 0050          RDS01                 13     001033   006701   000015   INTERLEAVE EVERY 8TH REGISTER
002 0051          WRT02                  8     001034   006742   000010   INTERLEAVE EVERY 16TH REGISTER
002 0052          WRT04                 24     001035   006744   000030   INTERLEAVE EVERY 64TH REGISTER
002 0053         4BPX24            YZ9         001036   045130   001013
002 0054          POS31            ITEM        001037   004700   000005   THE ITEM I T E M DEFINED BY A
002 0055          RES26            ITEM        001040   004700   000026    CPO CARD FURTHER DOWN IN PROG.
002 0056          TOB29            ABCD3       001041   000535   001005    NEW 65K INSTRUCTIONS
002 0057          TTB17            70A         001042   000561   001070
                                                                          GENERAL INPUT AND GENERAL OUTPUT
                                                                          CALLING SEQUENCES
                                                                            NOTE THE AUX OF THE FIRST WORD
                                                                          AND THE THREE PARAMETERS IN THE
                                                                          SECOND WORD ARE CONSIDERED DEC.
002 0060          RTP04                        001043   007204   000000   READ TAPE ON UNIT 14-HOLLERITH
002 0061               0                 0     001044   000000   000000
002 0062          RCD                          001045   007240   000000   READ A CARD-NO BINARY CARDS
002 0063               0                 0     001046   000000   000000
002 0064          WPR29            70A         001047   007275
```

2

```
                                                              001070   CONVERT THE 6 BIT HOL INFO START
002 0065            1 15                     001050   000417   000000   ING IN 70A BIT 29 AND EXTENDING
                                                                        FOR 15 CHARACTERS TO CARD IMAGE
                                                                        A PRINT ON LINE PRINTER IN PRINT
                                                                        WHEEL POSITIONS 1-15
002 0066            APU29                    001051   007335   000000   WRITE PRINTER- CONVERT FIRST 64
002 0067            1 64                     001052   000500   000000   CHARACTERS OF OUTPUT 6 BIT IMAGE
                                                                        TO CARD IMAGE   PUNCH A CARD
                                                                        17-80
002 0070            MOV23          70A       001053   007427   001070   MOVE HOLLERITH CHARACTERS
002 0071            1  5                     001054   000405   000000
002 0072            UDI07          ACC       001055   007047   000000   UNSIGNED DEC INTEGER ACC MEANS
002 0073            1  2             6       001056   000402   000006   ACCUMULATOR
002 0074            SDI31          CONST     001057   007037   001001   SIGNED DEC INTEGER
002 0075            1  5            15       001060   000405   000020
002 0076            SDF31          FRACT     001061   007537   001004   SIGNED DEC FRACTION
002 0077            9  6            16       001062   004406   000020
002 0100            OCF15          CONST     001063   007157   001001   OCTAL FRACTION
002 0101            17  6           15       001064   010406   000020
002 0102            CLR                      001065   007340   000000
002 0103            UDF15          FRACT     001066   000000   000000   UNSIGNED DECIMAL FRACTILLEGAL        INDETR LHW
002 0104            1  5            15       001067   001005   000015   HERE SINCE SECOND WORD OF CLR
                                                                        CALLING SEQUENCE WAS OMITTED
002 0105   70A      HOL            TAPE      001070   001531   170514   HOLLERITH FEATURE THE WORDS IN
002 0106            HOL            NOT R     001071   122004   153224   THE DIRECTORS OF THESE THREE INS
002 0107            HOL            EADY      001072   000615   030724   T. ARE CONV TO 6 BIT INVERTED
                                                                        FORMAT AND PUT INTO BITS 0-29 OF
                                                                        EACH WORD
002 0110            CAD            70A       001073   001000   001070   HOLLERITH RC WORD THE WORD TAPE
002 0111            SUB     HOL    TAPE      001074   001343   001520   IS SET UP AS AN RC WORD IN 6 BIT
002 0112            BFZ            UVWX8     001075   005400   001012
                                                                        VARIOUS OTHER RC WORDS
002 0113            CAD     ADDA   ITEM      001076   001000   001521   NOTE 17 BIT ADDR IN RC WORD INST
002 0114   A        ADD      3456            001077   001043   001522   OCTAL LHN RC
002 0115            ADD    + 2460        3   001100   001043   001523   SIGNED DEC INTEGER LHW RC.
002 0116            CAD    +.7213  +.1981    001101   001000   001524   SIGNED DEC FRACTION LHW RC
002 0117            CAD     ETR    ITEM      001102   001000   001526   THIS CARD CAUSES TWO RC WORDS TO
                                                                        BE GENERATED
                                                                           1. A MASK FOR ITEM
                                                                           2. AN RC WORD CONTAINING.
                                                                              A. ETR IN LHW
                                                                              B. ADDR OF ITEM MASK RHW
002 0120                  0      0      77   001103   000000   000000   NO RC WORDS WITH CONST INSTR LHW     ILL USE RC
002 0121            CAD     DIT          3   001104   001000   000000   ASSEMBLY INSTR NOT ALLOWED RC        ILL USE RC
002 0122            FST     CAD    TAG7      001105   003240   000000   STORE CLASS DOES NOT TAKE RC         ILL USE RC
002 0123            DEL    +    0  +     0   001106   004600   000000   SAME FOR SHIFT CLASS INSTRUCTION     ILL USE RC
002 0124   RC1      CAD    +   16  +    16   001107   001000   001527   NO DUPLICATIONS IN RC WORDS OR
002 0125   RC2      ADD    000020  000020    001110   001043   001527   ITEM MASKS BUT ——
002 0126            ETR    +    0  +     0   001111   000040   001530    Z E R O  IS A SPECIAL CASE
002 0127            ETR    -    0  +     0   001112   000040   001531   BECAUSE OF MASKS.
002 0130   AB*C     ETR    +    0  -     0   001113   000040   001532    NOTE ILLEGAL TAG PRINTOUT AT
002 0131            ETR    -    0  -     0   001114   000040   001533   BEGINING OF PROGRAM
           GGK      SKP                200   001115                     DIRECTOR IS DEC NOTE BOTH DRUM
           HIR      SKP                 10   001425                     AND CORE LOCATIONS STEPPED 210
                                                                        SKIP CAN BE TAGGED
002 0454            CAD            HIR       001437   001000   001425   FIRST REGISTER OF SKIPPED REGION
002 0455            CAD            GGK       001440   001000   001115   MAY BE ADDRESSED.
002 0456            HOL            TAXI      001441   001625   170514
002 0502            DIT                 20   001465   001625   170514   DIRECTOR DECIMAL
002 0503            ADD                 10*  001466   001043   000000                                       UNDEFINED
                    SKP                20/
```

```
                                                                                                                 INDSTR RHW
002 0504   A       TAG              A             001467   001103   001470   NOTE FIRST OF THE DUPLICATE TAG     DUPL TAG
002 0505   A       ADD              +   20        001470   001043   000024   ADDRESSES IS USED FOR BOTH TAGS
002 0506           SUB              A             001471   001343   001470   ERROR PRINTOUT IS ON LINE WHERE     DUPL TAG
                                                                             DUPLIC TAG IS USED IN DIRECTOR
002 0507   COB     MUL               B            001472   002500   000000   UNDEFINED MEANS I CANNOT FIND       UNDEFINED
                                                                             THE EXPRESSION IN THE DIRECTOR
                                                                             IN THE TAG TABLE THE COMPOOL
                                                                             OVERRIDE TABLE OR FOR DCA PROGS
                                                                             THE COMPOOL NOTE BLANKS ARE NOT
                                                                             ZEROS
                                                                             *** CPO CARDS-     MAY BE INSERT
                                                                             ED ANYWHERE IN THE DECK AND IN
                                                                             ANY ORDER THEY DO NOT APPEAR IN
                                                                             THE BINARY DECK OR ABSOLUTE LIST
                                                                             AS REGISTERS OF THE PROGRAM
           OKAY    TCPO                2114                                  TABLE CARD-TABLE OKAY IS LOCATED
                                                                             IN REG 2114 OF CORE
           ITEM    ICP005          OKAY                                      ITEM CARD-ITEM I T E M IS 5 BITS
                                                                             LONG ITS MOST SIGNIFICANT BIT
                                                                             IS IN POSITION 7 AND IT IS LOCA-
                                                                             TED IN TABLE OKAY
           BOOK    ICP001     27   OKAY                                       ANOTHER EXAMPLE.
           TSTS    ICP010     19   TDTO
           TDTO    TCPO              1750
002 0510           5CAD            TSTS           001473   051000   001750
002 0511           ETR             TSTS           001474   000040   001534
002 0512           POS31           TSTS           001475   004700   000035
002 0513           ADD         0        1         001476   001043   001535
002 0514           REG31           TSTS           001477   004700   000003
002 0515           LDB             TSTS           001500   000300   001534
002 0516           5DEP            TDTO           001501   053600   001750
002 0517   RC3     ETR             BOOK           001502   000040   001536   NOTICE SAME RC WORD FOR RC1,2,3
002 0520           TOB             BOOK           001503   000533   002114   FOR AN ITEM TAG THE AUX IS RELA-
002 0521           TTR04           TSTS           001504   000570   001750   TIVE TO MOST SIGNIFICANT BIT
002 0522           APC                            001505   000000   000000   ILLEGAL INSTRUCTIONS                INDETR LHW
002 0523           CAD                            001506   000000   000000                                       INDETR LHW
                                                                             ***EXAMPLES OF RELATIVE ADDRESS
002 0524           CMF             ITEM   +  1    001507   000460   002115   ALSO THE REST OF THE NEW 65K
002 0525           CML             ITEM   + 10    001510   000440   002126   INSTRUCTIONS
002 0526           CMR             ITEM   +100    001511   000420   002260
002 0527           CMM             TDTO   -  2    001512   000400   001746
002 0530           CDF             TDTO   - 20    001513   000473   001724
002 0531           CDL             TDTO   -100    001514   000453   001604
002 0532           CDR         0        4         001515   000433   001537
002 0533           CDM       HOL   DRUM           001516   000413   001540
002 0534           CAC             ANYTHING G     001517   101700   177777   OES HERE THE FULL 32 BITS ARE
                                                                             FILLED IN BY PROG. TRANSLATOR
                   END             ONE
002 0535                                          001520   001531   170514
002 0536                                          001521   001053   002114
002 0537                                          001522   003456   000000
002 0540                                          001523   004634   000003
002 0541                                          001524   121653   014533
002 0542                                          001525   174000   000000
002 0543                                          001526   000040   001525
002 0544                                          001527   000020   000020
002 0545                                          001530   000000   000000
002 0546                                          001531   177777   000000
002 0547                                          001532   000000   177777
002 0550                                          001533   177777
```

5

```
                                                                177777
002 0551                                      001534   000000   017770
002 0552                                      001535   000000   000001
002 0553                                      001536   000000   000020
002 0554                                      001537   000000   000004
002 0555                                      001540   001105   024720
READY READER
```

5.0 THE COMPASS COMPOOL

5.1 INTRODUCTION

The binary Compool used in COMPASS is called the COMPASS Compool, to distinguish it from the slightly different Compool in the Lincoln Utility System. In terms of information contained, the Lincoln Compool and the COMPASS Compool are essentially identical. Because the translation processes which use the Compool are different in the two utility systems, the structure of the information is different. The COMPASS Compool is composed of five sections, containing Tables, Programs, Items, Scaling and Environment. These are described in detail below.

5.2 COMPASS COMPOOL SECTIONS

A table tag is made up of three letters and a digit. The digit is called the "block number." Each block of a table contains the same number of registers, and the blocks occupy consecutive and adjoining blocks of core and drum storage.

The table's definition consists of its initial core location, its drum location (if any), its block length and the number of blocks it contains. LLLØ, for instance, might be defined as being at core location n, and as having m registers per block. LLL1, therefore, is understood to mean core location n + m, and LLL2, n + 2 m.

1. Section T - Table Information

Section T of the Compool contains 751 registers. It can contain a maximum of 250 table tags. In COMPASS, it is stored on drum field 14, registers 0000 to 1356 .C2T and C3T are indexed by the same increment value as C1T.

		16 — 31
CØT (1 word)		No. Table Tags in Section T

	0 — 17	18 — 31
C1T (250 words)	Three-letter table tags in 6-bit Hollerith-Characters read from right to left. Tags are sorted in ascending algebraic order.	Total No. words in table (givenfor tables where the no. words per block times no. of blocks does not equal total no. of words in table)

	0	1 — 15	16 — 31
C2T (250 words)	Core Address	No. words per block, for each tag in C1T.	Core Address

	0 — 9	10 — 15	16 — 31
C3T (250 words)	No. blocks in this table.	Drum Field	Drum Address

2. Section P (Programs)

A Program tag is made up of three letters. Since programs of a system are sometimes also identified by a four digit tag, with the first digit constant for that system, the last three digits of this tag may also be included in the Compool. The program's Compool definition consists of its core and drum location, and the allocated drum length.

Section P of the Compool is 451 registers long, containing a maximum of 150 program tags. As a part of COMPASS, it is stored on drum field 14, registers 1357 - 2261.

C2P and C3P are indexed by the same increment value as ClP.

		16 31
CØP (1 word)		No. of Program Tags in Section P

	0 17	18 29	
ClP (150 words)	Three-letter program tag, in 6-bit Hollerith, reading from right to left	Program No. (if any). Second digit in bits 18-21; third digit in bits 22-25; fourth digit in bits 26-29. Tags are sorted in ascending algebraic order.	

		1 15	16 31
C2P (150 words)		Max. no. of words in each program listed in ClP	Core address of each program in ClP

		10 15	16 31
C3P (150 words)		Drum Field of each program in ClP	Drum address of each program in ClP.

3. Section I (Item Information)

Any part of a table word may be defined as an item. The format of these tags is four letters. The definition consists of the table block in which this item is located, its bit position within words of that block, and the type of information this item is designed to contain.

All items must be one of seven types:

1. Binary
2. Charactron
3. Floating
4. Logical
5. Track Number
6. Binary-coded decimal
7. Alphabetic

If the type is "floating", a reference to section S is given for scaling information on this item.

Section I will contain a maximum of 850 items. It is broken down into three blocks: CØI (1 register), C1I (850 registers) and C2I (850 registers). Its total length, therefore, is 1701 registers. In COMPASS, it is located on drum field 13, registers ØØØØ to 3244.

		16 31
CØI (1 Word)		No. of Item Tags in Section I

	0 23	24 26	27 31
C1I (850 words)	Four-letter item tag, reading right to left, in 6-bit Hollerith. Tags are sorted in ascending algebraic order.	Type of Coding of Item (7 possible types listed above.)	Address, relative to CØI, of scaling info. for item. Given only for "Floating"coding

	0 17	18 21	22 26	27 31
C2I (850 words)	Three-letter table name for each item in 6-bit Hollerith, right to left.	Table Block No.	Most significant bit of item in table word.	No. Bits in item.

4. Section S (Scaling Information)

Section S, or CØS, is 32 registers long and is stored on drum field 13 in registers 3736 - 3775. It is indexed by a value given for floating type items in C1I. The three character ident given on the IDT card is stored on drum field 13, register 3776.

	0 15	16 31
CØS (32 words)	Exponent	Coding type

5. Section E (Environment Information)

For each program of the system using the Compool, a list of the table and program tags available to it is included in Section E. This section has been designed to provide the Program Translator with all the information (except Section I) which it requires to assemble a given program. Since it consists of a catalogue of available tags for many programs, its structure is somewhat complex. Basically, it consists of three sections: CØE (1 word), C1E (150 words), and C2E (2742 words). C1E is an index to C2E. It should be noted that C1E is not sorted in any individual manner; the entries are in the same order as the programs are listed in C1P. Thus, if a program has been found by a binary search of C1P with the use of index register#1,the relevant portion of C1E can be obtained by a "1 CAD C1E."

CØE and C1E are stored on drum field 14, 2262 - 251Ø. C2E is stored on drum field 14, registers 2511 to 3777 and drum field 15, registers ØØØØ to 3777.

CØE (1 word)

	16 — 31
	No. Entries in C1E (always 150)

C1E (150 words)

0 — 11	12 — 18	19 — 31
Prog. Ident.	No. Entries in C2E	Location of First Entry in C2E

C2E (3072 words)

0 — 17	18 — 21		28 — 31
Program or Table Tag	Block (Blank for Prog. Tags)		No. Blocks (For Tables Only)

	1 — 15	16 — 31
	No. words (For Programs) or Words/Block (for tables) (This block should be blank)	Core Address

5.3 Subroutines

In the hope that all programs using tne COMPASS Compool will do so in a consistent fashion, a set of subroutines is available which will perform the necessary search, and which will exit with either an indication that the tag is not in the Compool or with the relative location of the tag in the Compool in Index Register 1. This is presently available as a manuscript subroutine, and should prove useful to those planning to use the COMPASS Compool.

6.0 PROGRAM Assemble Compool

6.1 IDENT 0363

6.2 FUNCTIONS Constructs (on either cards or tape) a binary COMPASS Compool, of the format described in section 5.0 of this document, from symbolic input information on either cards or tape, and upon request produces a symbolic and octal listing in side by side format.

6.3 OPERATION See Section 2.0 of this document

6.4 INPUTS

6.4.1 Cards

6.4.1.1 COMPASS Control Card

cols. 18 - 23 = "COMPOL" (to indicate Assemble Compool Program)

cols. 25 - 28 = "TAPE" or "CARD" (symbolic input)

cols. 30 - 33 = "TAPE" or "CARD" (binary output)

cols. 36 - 39 = "TAPE" or "DIR" or "NONE" (listings to tape drive number 4, DIRECT, or NONE)

Special control card for COMPASS Compool listing assembled on master tape.

cols. 18 - 23 = "COMPOL" (initiates load COMPASS from TAPE drive number 1 to AM drums)

cols 36 - 39 = "TAPE" or "DIR" (listing, delayed or direct)

cols. 47 - 49 = "DCP" (list Compool from master)

6.4.1.2 Assemble Compool Deck - the deck must be in the order shown below:

1. Identification Card

cols. 25 - 27 = "IDT"

cols. 36 - 41 = Compool identification characters (chosen by user of program); these identifying characters appear in Hollerity in the Compool.

2. Table Input Cards

col. 18 = "I" (for indexable)

cols. 19 - 21 = three-letter table tag

col. 51 = number of blocks

cols. 52 - 54 = number words per block, in decimal

cols. 59 - 62 = total number of words if not equal to number of blocks X words per block; otherwise blank.

cols. 63 - 68 = beginning core location of table, in octal

cols. 69 - 71 = drum field

cols. 72 - 75 = drum location

3. Program Input Cards

col. 18 = "N" (for non-indexable)

cols. 19 - 21 = three-letter program tag

cols. 55 - 58 = four-digit ident. number

cols. 59-62 = number words in program, in decimal
cols. 63-68 = beginning core location of program, in octal
cols. 69-71 = durm field
cols. 72-75 = beginning drum location of program, in octal

4. Item Input Cards

col. 18 = "T" (for item)
cols. 19-22 = four-letter item tag
cols. 27-29 = three-letter table tag
col. 30 = number of table block in which item is located
cols. 31-32 = position of first (most significant) bit, in decimal
cols. 33-34 = number of bits which item occupies, in decimal
col. 35 = type of coding of item. "F" (floating point decimal) or "L" (logical) or "B" (binary) or "C" (charactron) or "T" (track number) or X (binary coded decimal) or "A" (alphabetic).
cols. 37-39 = signed exponent, col. 37 = ±; cols. 38 - 39 $\leq 15_{10}$.
cols. 76-79 = multiplier

5. Environment Input Cards

col. 18 = "E" (for environment)
cols. 19-21 = three-letter table or program tag
col. 22 = table block number (blank, if cols. 19-21 contain a program tag)
cols. 23-26 = four-digit program ident. code (blank if cols. 19-21 contain a table tag)
col. 51 = number blocks in table (blank if cols. 19-21 contain a table tag), in decimal
cols. 63-68 = beginning core location of table or program (if not punched here, program will take this information from either Section P, for programs, or Section T, for tables of the compool), in octal

6. End Card

Cols. 25-27 = "END"

6.4.2 Tapes

The Assemble Compool deck described above may, if desired, be prestored on tape. The order and contents of cards will be exactly as described above.

6.5 OUTPUTS

6.5.1 Cards

A binary Compool deck will be punched out on the card punch. The format of these cards is shown in Section 3.0 of this document.

6.5.2 Tape

The binary output of the program may be, if desired, written out on tape unit number 3. The format is shown in Section 3.0 of this document.

6.6 SPECIAL POINTS:

1. The program has been designed to use input cards of the Lincoln Utility Assemble Compool (UAC) Program. However, only such cards with an I, N, T or E in col. 18 will be accepted; others will be logged as illegal, but will not interfere with operation of the program.

2. Cards which are illegal for any reason are logged and then ignored.

3. If there is an excessive number of input cards for any compool section the program halts. Further operation may be risky and is left to the operator's discretion.

4. Because of the sorting techniques used, there must be at least two input cards for each compool section.

5. If the four input decks are not in the proper sequence, chaos results, with all cards being logged as illegal. Further operation is impossible.

6. If duplicate tags are discovered the tag will be logged and the program will halt. Pushing "Program Continue" will cause one of the tags to be ignored (which tag is ignored is a random decision of the program); the program will then continue operating.

7. It should be noted that the new Compool is assembled using aux drums only, for temporary storage. The new Compool does not replace the Compool version which is a part of the operating system. In no way will any COMPASS program alter the system which was read from tape and is stored on MD drums during operation.

7.0 PROGRAM Assemble Sequence Parameters (ASP)

7.1 IDENT 0364

7.2 FUNCTIONS Constructs a Sequence Parameter Table (SPT) and an SEQ table from symbolic input information contained on either cards or on tape, produces (if desired) a symbolic listing of the tables, direct or delayed, and also outputs a binary deck or tape of the SPT and SEQ tables.

7.3 OPERATION See Section 2.0 of this document.

7.4 INPUTS

7.4.1 Cards

7.4.1.1 COMPASS Control Card

cols. 18-23 = "SEQPAR" (to indicate Assemble Sequence Parameters Program)

cols. 25-28 = "TAPE" or "CARD" (symbolic input)

cols. 30-33 = "TAPE" or "CARD" (binary output)

cols. 36-39 = "TAPE" (symbolic listing on tape) or "DIR" (cols. 36-38, symbolic listing on direct printer) or "NONE" (no symbolic listing)

7.4.1.2 Assemble Sequence Parameters Deck (The SPT table will be loaded in the same order as the cards in the deck).

1. Comment cards - Any number of these, with any combination of characters, may be put in the deck preceding the Ident. card.

2. Ident. Card

   cols. 25-27 = "IDT"

   cols. 40-41 = (Numeric Mod) this information will be gang punched on every binary card, preceded by "SPTØ." (Sequence Parameter Table.)

   cols. 42-80 = any entry

3. Transfer ("T") Cards - Contain all the information required by Program PEC to initiate an in-out transfer. Several items on "T" cards may be specified by tag; the values for such items will be taken from the COMPASS Compool; these items are: RHSD, PTCL, NWDS, and CBRA.

(* = required items; others optional)

Column	Item	Description	
17	TYPE*	Card Type (T)	(See
19-21	SFTI*	Subframe in which operating-	Note d,
23-24	PTSK	Conditionality bit in PTC (See note a, below)	(below
26	VLTI	Variable length transfer indicator	

Column	Item	Description
28	SSDI	SEL/SDR indicator
30-32	SDSC*	SEL/SDR index interval
34-37	RHSD*	Drum Address
39-44	PTCL*	Core location
46	ROWI*	Read/Write Indicator
48-51	NWDS*	Number of words
53-55	EOWC	Entry on which conditional (see note e, below)
57	RTAI	Record transfer address indicator channel number (see note b, below)
59	LTAI	Look up transfer address indicator
61	VACN	Variable address channel number (see note b, below)
63	CLFK	Clear field key (see note c, below)
65-67	CBRA	Conditional branch address
69	LACI	Lock address counter indicator
71-73	REID	Required entry identity (see note e, below)
75-78	PROG ID	DCA Program number (see note e, below)

Notes: a. PTSK ≤ 63

b. VACN ≤ 3

c. CLFK has the following values:
Blank - off
Ø - clear field A (CFAI)
1 - clear field B (CFBI)

d. SFTI must contain at least one subframe indicator, or blockette will be left blank.

e. EOWC and REID are related items and occur in pairs. They are of the form letter-digit-digit. The digit-digit parts must be unique without duplicates. The letter is used only for program function reference, e.g., K for switches, B for Bomarc, etc.

4. <u>Program</u> ("P") Cards - specifies the core location to which PEC will branch to operate a particular program.

a. TYPE (P)*
b. SFTI *
c. PTSK
d. PTCL *
e. EOWC
f. REID

5. <u>End Card</u> - last card in deck.

cols. 25-27 = "END"

7.4.2 <u>Tapes</u>

The Assemble Sequence Parameters Program input cards may, if desired, be prestored on tape. The order and contents of cards is as described above.

7.5 OUTPUTS

7.5.1 Cards

1. Sequence Parameter Table (SPT) binary deck, with drum locations. The Sequence Parameter Table contains three words per P or T card and is set up as follows:

	0 1	2 7	8 15	16 31
SPT 0	SPARE	PTSK	SDSC	PTCL

	0	1-2	3-4	5	6	7	8 15	16	17 31
SPT 1	ROWI	CLFK	VACN	LACI	LTAI	RTAI	EOWC	VLTI	RHSD

	0 2	3 15	16 31
SPT 2	SFTT	NWDS	CBRA$_{(T)}$ PENC$_{(P)}$

2. SEQ binary deck
   SEQ. The Sequence Code Table is set up as follows:

Word	Ø	RHW	Number SEQ entries
Word	1-N	L12-R13	Three characters of DCA Program code of P card in 6-bit Hollerith.

7.5.2 Tapes

The SPT and SEQ binary tables may be recorded on tape rather than on cards.

7.5.3 Printouts

A symbolic listing of the SPT table may be requested.

7.6 SPECIAL POINTS

1. The program does not stop when it encounters an illegal item on a card. The card is logged on the direct printer along with the reason for its being illegal. The bits in the SPT table where the item would have gone are left blank, and the program continues operating.

2. ASP uses the COMPASS Compool to look up the current locations of programs and tables involved in constructing the SPT table. The cards are read in, processed, and the output is arranged using the General Input-General Output subroutine features of the COMPASS system.

8.0 PROGRAM Environment Simulation

8.1 IDENT 0366

8.2 FUNCTIONS

The Environment Simulation program is an adaptation and extension of the Lincoln Table Simulation program. In Environment Simulation the generation of output is treated as the assembly of input data read in from cards or tapes coded in alphanumeric form. Two kinds of output can be generated through the operation of this program, binary cards or tape records. The nature of the output is described below. Unlike Table Simulation, this program does not store the information in its Compool-assigned location. Thus, Environment Simulation is designed to generate data which can be used to check out the master program or to evaluate the transfer function of any program. The programmer exercises direct control over the input data.

8.3 OPERATION See Section 2.0 of this document.

8.4 INPUTS

8.4.1 Cards

8.4.1.1 COMPASS Control Card

cols. 18 - 23 = "ENVISM" (to indicate Environment Simulation Program)
cols. 25 - 28 = "TAPE" or "CARD" (symbolic input)
cols. 30 - 33 = "TAPE" or "CARD" (binary output)
cols. 36 - 39 = "TAPE" (symbolic listing on tape) or
"DIR" (columns 36 - 38, symbolic listing on direct printer) or
"NONE" (no symbolic listing)
cols. 42 - 45 = "DRUM" or "CORE" (type of storage)

8.4.1.2 Environment Simulation Deck

1. Ident. Card

cols. 25 - 27 = "IDT"
cols. 36 - 41 = test no. (optional); first digit must be a "5"; if core environment is to be simulated, first four digits should be DCA number of program using the information.

2. Tagged Items

cols. 18 - 21 = index value or channel number in decimal; indicates relative location in table block in which item is stored; can be any value ≦ (n-1), where u = number of words per block.
cols. 23 - 26 = four-letter item tag
cols. 28 - 39 = item value

a. Binary or logical items - unsigned decimal integers.

b. Floating-point Decimal - All items specified in the Compool as having F (Floating-point decimal) coding are expressed as either signed, unsigned or dual signed decimal numbers. Fractional parts of a unit must be proceeded by a decimal point, and a comma will be used to separate the two values of a dual item.

c. Alphabetic (see note below) - All items which are expressed as having C (Charactron) or A (Alphabetic) coding in the Compool are expressed in octal fraction form.

d. Binary-Coded Decimal (see note below) - All items which are expressed as having X (Binary-coded Decimal) coding in the Compool are expressed in octal fraction form.

e. Track Number - All items which are specified in the Compool as having T (Track Number) coding are expressed in the form LLDD or LDDD where D is a decimal digit and L is a letter.

3. Tagless Items

cols. 18 - 21 - index value of channel, in decimal
cols. 23 - 26 - tag of table block in which item belongs
cols. 28 - 39 - item value in octal fraction form, full word

4. End Card

cols. 25 - 27 = "END"

8.4.2 Tapes

The Environment Simulation deck described above may, if desired, be prestored on tape.

NOTE: These will involve precoding as prescribed and then resolving that result to octal fraction form to render the correct input expression. If the final octal fraction expression is equivalent in magnitude to a half-word, it may be entered in either columns 28 - 33 or columns 34 - 39. In the former case, columns 34 - 39 must be left blank. In the latter case, columns 28 - 33 must be coded plus zero.

8.5 OUTPUTS

8.5.1 Cards

Environment Simulation will generate simulated data on either punched cards or tapes. In either case, the output is derived from a table comprised of addresses (core or drum) alternating with corresponding contents. This table is formed in core memory from the input using the Compool to determine the address for the individual items simulated as well as to establish the coding type, bit position, number of bits and conversion factor. The cards produced by Environment Simulation preserve the format and order of the table of alternating addresses and contents generated in core. No duplications exist in either of these modes.

The data is presented in the same order as that given in the input. That is, the first data word corresponds to the first card input, etc. The IDENT is gangpunched in columns 1 - 6.

8.5.1.1 Table Card Format

1st word: 9's Row, LHW: Ø1ØØØØ (Col. 20 identifies card type)
9's Row, RHW: ØØØØ26 (No. of data words on card, cols. 44 - 48)

2nd word: 9's Row, Full Word: Checksum

3rd word: 8's Row, LHW: Drum Field (Cols. 27 - 32 blank for core locations)
8's Row, RHW: Address (Cols. 33 - 48)

4th word: 8's Row, Full Word: Item Value (Cols. 49 - 80)

The rest of the card is filled to a maximum of 22 entries of alternating addresses and contents.

8.5.1.2 The End Card

1st word: 9's Row, LHW: Ø5ØØØØ
9's Row, RHW: ØØØØØØ

2nd word: 9's Row, Full Word: Checksum

8.5.2 Tape

The tape output is preceded by the usual Storage Ident. record. The first word of the Storage Record contains the number of data words in the record (left-half word) and the tape record identity bits (ØØØØ11) right-half word. The second word is blank. Following this is the series of alternating addresses and contents. The record is terminated by the complement checksum. The Tape Record cannot exceed a length of $2,503_{10}$ registers of $1,250_{10}$ card inputs. (Also see the section directly above for more information.)

8.6 SPECIAL POINTS

1. Error Print-Out
   Should the input cards be improperly prepared or contain a non-existent tag (from the current Compool) the card will be printed out with an appropriate message indicating where, on the card, the error was found. In addition, should the number of entries to the table of alternating addresses and contents exceed 1250, the cards exceeding that limit will be printed out with an appropriate message. In all cases, after the print-out, the program will continue operation on the data successfully read in.

2. When the program ident. is not found after searching Section E of the Compool, the program prints out an appropriate message and halts.

3. Provision has been made for permitting a Compool override action. In order to initiate this function, the Environment Simulation Deck must be supplemented by Table Input and/or Item Input cards in the format prescribed for the Assemble Compool Deck. Any table or item feature can be given alternative designation in this supplementary input. It is also possible to submit definitions for non-existent tables and items.

   These supplements must precede the Environment Simulation Deck and follow the IDT card if it is desired that all of the succeeding input be affected. In order to exclude certain input from involvement, the Compool Deck Supplement must be entered after the input. In addition, when it is desired to override the table designation or allocation for an item, both the respective Table Input Override and the Item Input Override must be given, the one immediately following the other.

   The program can handle 150_{10} item override entries and 50_{10} table override entries.

4. An item or table value can be repeated up to a maximum of 1000_{10} times. The input whose value is to be repeated is indicated by □ $DDDD_{10}$ (digit-digit-digit-digit) in columns 67 - 71, inclusive.

8.7 LIMITATIONS OF PROGRAM

Certain items in the Compool are affected by changes in the scaling for F-type coding. Environment Simulation assumes that the exponent represents a power of 2; the multiplier is not used. Therefore, with their presently specified scaling, these items cannot presently be simulated by Environment Simulation: CSXX, CSYY, HALT.

9.0 PROGRAM ASSEMBLE GEOGRAPHY

9.1 IDENT 0365

9.2 FUNCTIONS Converts fixed geography display information in Hollerith form which is on cards or tape into display format; output is either on cards or tape.

9.3 OPERATION See Section 2.0 of this document

9.4 INPUTS

9.4.1 Cards

9.4.1.1. COMPASS Control Card

cols. 18-23 = "GEOGR" (to indicate Assembly Geography Program)
cols. 25-28 = "TAPE" or "CARD" (symbolic input)
cols. 30-33 = "TAPE" or "CARD" (binary output)

9.4.1.2 Assemble Geography Deck

The input deck has the following order: Reference card, data cards, end card. Each vector message is made up of a V card followed by an X card.

cols. 1-17 of all of the following cards have the same format:
cols. 1-7 = card room no.
cols. 8-9 = subsector no.
cols. 10-13 - category name
cols. 14-16 = no. cards in category
col. 17 = blank

1. Reference

Column	Information	Card Contents
1-17	(see above)	
18	Card Identification	"R"
19-20	Subsector Number	Unsigned Decimal Integer (UDI)
21-26	System Center-Longitude	Unsigned Decimal Integers
	Cols. 21-23	Degrees (UDI)
	Cols. 24-26	1/10 Minutes (UDI)
27-31	System Center-Latitude	Unsigned Decimal Integer
	Cols. 27-28	Degrees (UDI)
	Cols. 29-31	1/10 Minutes (UDI)
32-37	Display Center-Longitude	Unsigned Decimal Integer
	Cols. 32-34	Degrees (UDI)
	Cols. 35-37	1/10 Minutes (UDI)
38-42	Display Center-Latitude	Unsigned Decimal Integer
	Cols. 38-39	Degrees (UDI)
	Cols. 40-42	1/10 Minutes (UDI)
43-47	Display Vector Multiplier	Unsigned Decimal Fraction (UDF)
48-52	Display Position Multiplier	Unsigned Decimal Fraction
53-80	----------------------	Blank

2. Tabular Message Card

Information stored on tabular message type cards may be either track (T) or information (I) messages. The tabular message card consists of 28 fields of information.

Column	Information	Card Contents
1-17	(See above)	
18	Card Identification	"T" or "I"
19	Display Drum Field	UDI (1-6)
20-22	Drum Slot Number	UDI (1-240)
23-24	Display Message Category	UDI (Ø-31)
25	E Position	"A","B","V","R","I" or "H"
26-36	E Location	
	Display Coordinates	
	Cols. 26-30 X	Signed Decimal Integer
	Cols 32-36 Y	Signed Decimal Integer
	Degrees Longitude	
	Cols 26-28	Degrees
	Cols 29-31	1/10 Minutes
	Degrees Latitude	
	Cols 32-33	Degrees
	Cols 34-36	1/10 Minutes
37-38	A_1	Charactron Code or Character
39-40	A_2	"
41-42	A_3	"
43-44	A_4	"
45-46	A_5	"
47-48	A_6	"
49-50	B_1	"
51-52	B_2	"
53-54	B_3	"
55-56	B_4	"
57-58	B_5	"
59-60	C_1	"
61-62	C_2	"
63-64	C_3	"
65-66	C_4	"
67-68	D_1	"
69-70	D_2	"
71-72	E	"
73	Light Gun Control	Blank or "L"
74-76	Vector Magnitude	Unsigned Decimal Integer
77-80	Vector Heading	Unsigned Decimal Integer

3. Vector Message Card - First Card

The first card for a vector message contains 16 fields of information.

Columns	Information	Card Contents
1-17	(see above	
18	Card Identification	"V"
19	Display Drum Field	UDI (1-6)
20-22	Drum Slot Number	UDI (1-256)
23-24	Display Message Category	UDI (Ø-31)
25-35	Origin Vector 4	
	Display Coordinates	
	Cols. 25-29 X	Signed Decimal Integer
	Cols. 31-35 Y	Signed Decimal Integer
	Degrees Longitude	
	Cols 25-27	Degrees
	Cols 28-30	1/10 Minutes
	Degrees Latitude	
	Cols. 31-32	Degrees
	Cols. 33-35	1/10 Minutes
36-38	Vector 4 Magnitude	Unsigned Decimal Integer
39-42	Vector 4 Heading	Unsigned Decimal Integer
43-53	Origin Vector 3	
	Display Coordinates	
	Cols. 43-47	Signed Decimal Integer
	Cols 49-53	Signed Decimal Integer
	Degrees Longitude	
	Cols 43-45	Degrees
	Cols. 46-48	1/10 Minutes
	Degrees Latitude	
	Cols. 49-50	Degrees
	Cols. 51-53	1/10 Minutes
54-56	Vector 3 Magnitude	Unsigned Decimal Integer
57-60	Vector 3 Heading	Unsigned Decimal Integer
61-62	G_1	Charactron Code or Character
63-64	G_2	"
65-66	G_3	"
67-68	G_4	"
69-80		Blank

4. **Vector Message Card - Second Card**

The second card of the vector message has the same format as the first card with the following exceptions:

Columns	Information	Card Contents
1-17	(see above)	
18	Card Identification	"X"
23-24	----------	Blank
25-42	Vector 2	
43-60	Vector 1	
61-80	----------	Blank

5. End Card

The last card of the fixed geography display deck contains a "Z" in column 18.

9.4.2 Tapes

The Assemble Geography deck described above may, if desired, be prestored on tape. The order and contents of cards will be the same.

9.5 OUTPUTS

9.5.1 Cards

A binary deck will be produced; the format is shown in Section 3.0 of this document.

9.5.2 Tape

If specified, a binary tape will be produced; the format is shown in Section 3.0 of this document.

9.6 SPECIAL POINTS

1. The following legality checks are performed on all cards:

   a. Proper format in each item
   b. Duplication of drum slot assignments
   c. Proper limits in each card item.

   For any of the above errors, or if the first card read is not a reference card, the contents of the card and the column number of the first column of the field in which the error occurred are printed on the line printer. The program will process the rest of the cards and check for further errors.

2. All message cards are processed to convert vector positions to x,y in display units, convert vector magnitudes and headings to $\dot{x}$, $\dot{y}$ in display units, convert characters to charactron code, and set necessary indicator bits.

3. The UFGØ table is set from the first 43 slots of drum field 43.

10.0 PROGRAM Memory Print

10.1 IDENT 0352

10.2 FUNCTIONS

Prints out on the direct or delayed printer, in octal or instruction form, all or parts of core, drum or tape. The program is controlled by the "A" and "B" and Sense Switches, the use of which is described below.

10.3 OPERATION

10.3.1 COMPASS Programs Already On Drums

1. Use the following binary card to get Memory Print into core:

```
SDR   26          0
LDC               0
WRT           4000_8
SDR   11          0
LDC              60
RDS           1720_8
HLT               0
BPX              60
```

(The read-in card stores the contents of core memory locations Ø through 3777 onto drum 26 and then reads the Memory Print program into registers 6Ø through 1777, and halts with the program counter at 7. Upo completion of the read-in, drum 26 contains the read-in program in locations Ø-27 and the contents of core memory in 3Ø through 3777.)

2. Assign "Test Memory".

3. Press "Load From Card Reader."

4. (Go to Section 10.3.3)

10.3.2 COMPASS Programs Not Already on Drums

A Memory Print Deck can be obtained and run by itself, apart from COMPASS system. The deck comes with a special read-in card.

1. Read Memory Print deck into core.

2. (Go to section 10.3.3)

10.3.3 Control of Memory Print Program

1. Set Sense Switches:

Sense Switch No. 1: ON = Pressing "Program Continue" causes contents of drum 26 to be read into core, thus restoring core. Operates after one pass of program.
OFF = Pressing "Program Continue" causes Memory Print Program to be rerun.

Sense Switch No. 2: ON = Direct Output on Line Printer
OFF = Delayed (Tape Unit No. 4) Output

Sense Switch No. 3: ON = Octal Constants Printout
OFF = Instruction Printout

Sense Switch No. 4: ON = Right "A" and "B" Switches determine portion of tape record to be printed.
OFF = Entire tape record will be printed

2. Set "A" and "B" Switches:

Core

"A": LS and RS - R15 = Starting Address
L1 - L15 = zero

"B": LS and RS - R15 = Final Address
L1 - L15 = Zero

Drums

"A": L4 - L9 = SDR Code (61)
L1Ø - L15 = Drum Code
R1 = "1" For an aux. drum
RHW = Starting Address
"B": LHW = Zero
RHW = Final Address

Tapes

"A": L4 - L9 = SEL code (62)
L1Ø - L15 = Tape Code
RHW = Starting Address (zero for full record)
"B" LHW = Zero
L1 = Files may be printed from tape by depressing L1 of the B switches. Records will be printed until the "end of file" mark is reached or L1 is raised. Only one end of file will be written on the delayed output tape. The number of words in the tape record is printed.
RHW = Final Address

Examples:

Print all of core delayed as instructions

Sense Switches: 2-OFF 3-OFF

A: ØØØØØØ ØØØØØØ
B: ØØØØØØ Ø2ØØØØ

Print 165Ø$_8$ to 273Ø$_8$ of auxiliary drum 2 direct as constants

Sense Switches: 2 - ON 3- ON

A ØØ61Ø2 Ø4165Ø
B ØØØØØØ ØØ237Ø

Print one record of tape 3 direct as instructions

Sense Switches: 2 - ON 3 - OFF 4 - OFF
A: ØØ6213 ØØØØØØ
B: ØØØØØØ ØØ4ØØØ

Print 6Ø$_8$ to 3ØØ$_8$ of the next record on tape 2 direct as constants

Sense Switches: 2 - ON 3 - ON 4 - ON

A: ØØ6212 ØØØØ6Ø
B: ØØØØØØ ØØØ3ØØ

3. Press "Program Continue."

4. Program may be re-run as often as desired by resetting "A" and "B" switches and pressing "Program Continue". Changing the "A" and "B" switches will not affect the operating of Memory Print once it has begun. Sense switches 2 and 3 are examined at each pass.

   To restore core after completion, depress SENSE SWITCH I and press "Program Continue".

10.4 OUTPUTS

The output of the program is printed four registers to a line with the location of the first register of the line specified at the left. Only the first register of a consecutive string of identical registers is printed, the rest are suppressed. Each block of memory printed is preceded by a line specifying the block requested.

10.5 SPECIAL POINTS

1. Memory Print makes several legality checks and prints "ERROR" on the line printer in the following cases: Test memory unassigned; starting address greater than the final address; left "A" switches (L1-L15) not zero, SDR, or SEL; tape unit 4 requested to be printed.

2. The large memory addresses (up to 377,777$_8$) and the new instructions are included. Instructions using L12 for control are printed with a "1" following the instruction code (e.g., STA1, ADD1).

3. Since test memory is used for control, a special Memory Print must be used when operating on a computer having the full 65K memory. This program operates in the same fashion as Ø352 and is available as a binary deck. The maximum record length is restricted to $200,000_8$.

11.0 PROGRAM File Maintenance

11.1 IDENT 0367

11.2 FUNCTIONS

This program is a group of subroutines designed to perform various handling and processing functions on tapes produced by the COMPASS System. The functions performed are divided into three broad classifications: Functions involving Hollerith tapes only; functions involving binary tapes only; functions involving Hollerith or binary tapes.

11.3 OPERATION See Section 2.0 of this document.

11.4 INPUTS

11.4.1 Functions Involving Hollerith Tapes Only

1. Load

Control Card:	Cols. 18-21 = "LOAD"
Input:	Any Hollerith punched cards
Output:	Hollerith (cols 17-80)on tape 2
Character Type:	Hollerith

The LOAD subroutine provides a means for loading tapes with Hollerith data where card-to-tape equipment is not available. The subroutine writes 13-word records containing the Hollerith coding of columns 17 through 80 of the cards on tape 2. The process operates until end-of-file is reached in the card reader. At this point an end-of-file record is written on tape 2 followed by a branch to COMPASS Control. The input deck is set up with the LOAD control card followed by the cards to be loaded on the tape.

2. Update

Control Card:	Cols 18-23 = "UPDATE" Cols 25-28 = "CARDS" or "TAPE"
Input:	Master on tape 4 Changes on tape 3 or cards
Output:	New master on tape 2
Character Type:	Hollerith

The UPDATE subroutine provides a means for changing tapes loaded with instruction decks of programs. The subroutine changes, deletes or inserts whole programs or cards within a program.

To insert, change, or delete an entire program: Punch an "IDT" card identical to the one on the master tape, and in columns 30-35 punch "CHANGE," "INSERT" or "DELETE." If the program is to be deleted, this is all that is needed. If change or insert, this card must be followed by the new program, including an "IDT" card and an "END" card. For insert, the new program will be inserted before the program indicated on the "INSERT" card.

To insert, change or delete a card within a program: Punch an "IDT" card identical to the "IDT" of the program on the master tape, columns 30-35 blank. For each change within a program, punch a control card with "C" in column 17, previous symbolic location in columns 18-22, increment in 24-27 (increment must be positive) and "CHANGE," "INSERT" or "DELETE" in columns 30-35. If "DELETE," punch in cols. 36-41 the number of cards to be deleted. If "CHANGE" or "INSERT," it must be followed by the cards to be inserted or changed.

Only one "IDT" card is needed for a program, regardless of how many changes are made in it. A control card is needed for each non-consecutive location affected, or for each change of control. For example, five consecutive cards may be changed after one control card, but if a card is to be inserted before the sixth card, another control card is needed.

To change an "IDT" card, the following cards are necessary:

1. An "IDT" card like the one on tape.
2. A card with "C" in column 17 and "CHANGE" in columns 30-35.
3. The new "IDT" card.

In the same way, cards may be inserted, deleted or changed before the first location tag by leaving columns 18-23 blank on the control card but putting an increment in columns 24-27.

If the change cards for updating are in the card reader, follow them with a card containing IDT in columns 25-27 and FINISH in columns 3Ø-35. This may also be done if the change cards are on tape. This card tells UPDATE that this has read the last card.

If UPDATE cannot find the symbolic tag in a program which is referred to by a change card it will print out this fact on the line printer and go on to make changes to the next program for which changes have been specified.

Since update provides the facility whereby an instruction card file may be maintained on tape, a method is provided to make the backup card files reflect the changes to the tape files. The same deck used to update the master tape files may be processed using EAM procedures through the card room.

The only type of change card which cannot be directly processed by EAM procedures is the DELETE X. For this reason the programmer should, for example, follow a DELETE 3 cards in the change deck with 3 cards containing just the program identification and card numbers to be deleted in columns 1-6 and 8-13 of the card. The update program will bypass these cards if they are in the deck.

11.4.2 Functions Involving Binary Tapes Only

1. Punch

Control Card:	Cols. 18 - 22 = "PUNCH"
Input:	Tape 3
Output:	Binary Deck

The "PUNCH" subroutine generates binary decks from tape storage records on tape 3. The tape storage identification is gang-punched on each card. The subroutine will punch the storage record at which the tape is located. To punch a specific record, the tape must be positioned (see below) to that record.

2. Merge

Control Card:	Cols. 18 - 22 = "MERGE"
Input:	Master on tape 2 Change on tape 4
Output:	New master on tape 3

The "MERGE" subroutine will change, delete or insert a block of records corresponding to the records associated with a storage identification record. These functions are specified on merge control cards. Each control card is punched with "IDT" in columns 25 - 27.

To change a record block, the six character Hollerith identification of that block, as it appears on the master tape, is punched in columns 36 - 41 with the word "CHANGE" in columns 30 - 35. The record block on tape 4 whose first four identification characters agree with the identification on the control card is substituted for the existing block.

To delete a record block, the six character Hollerith identification of that block, as it appears on the master tape, is punched in columns 36 - 41 with the word "DELETE" in columns 30 - 35.

To insert one or more record blocks, the six character Hollerith identification of the block after which the insertions are to be made is punched in columns 36 - 41 with the word "INSERT" in columns 30 - 35. For each record block to be inserted, punch an IDT card with the record identification in columns 36 - 41. Columns 30 - 35 of these cards are blank. These cards are placed directly following the insert card.

To change a File or End Identification record, regular change cards are punched with the file or end identification in columns 36 - 41 and the new file or end identification in columns 43 - 48.

The "MERGE" control cards must be in the order of the records on the master tape (2). An end card ("END" in cols. 25 - 27) is the last card in the deck. The records on the change tape may be in any order provided an end-of-file is written on the tape.

3. File

Control Card:	Cols. 18 - 21 = "FILE" Cols. 25 - 30 = Identification
Output:	File identification record on tape 3.
Character Type:	Binary

The FILE subroutine writes a file identification record on tape 3 with the six-character identification specified in columns 25 - 30 of the control card.

4. End

Control Card:	Cols. 18-20 = "END"
	Cols. 25-30 = Identification
Output:	End Identification record on tape 3
Character Type:	Binary

The END subroutine writes an end identification record on tape 3 with the six character identification specified in columns 25-30 of the control card. An end-of-file is written following the end identification record.

11.4.3 Functions Involving Hollerith or Binary Tapes

1. Position

Control Card:	Cols. 18-20 = "POS"
	Col. 22 = Tape Number
	Col. 23 = Rewind Indication
	Cols. 25-30 = Program Identification
Character Type:	Hollerith and/or Binary

The "POSITION" subroutine places the tape unit specified in column 22 or the control card to a point just before the record block identification in columns 25-30. For Hollerith tapes this point is before the "IDT" card with the specified identification. For binary tapes this point is before the storage identification record with the specified identification. Column 23 of the control card contains either an "R" or is left blank signifying position with or without rewind, respectively.

2. Duplicate

Control Card:	Cols. 18-23 = "DUPLIC"
	Col. 25 = Number of copies
Input:	Tape 2
Output:	Copy 1 - tape 3
	Copy 2 - tape 4
	Copy 3 - tape 1
Character Type:	Hollerith and/or Binary

The duplicate subroutine transfers the contents of tape #2 outs tapes #3 and 4 and 1, respectively, as determined by the number of copies specified in column 25 of the control card.

3. Compare

Control Card:	Cols. 18-23 = "CMPARE"
Input:	Tapes 2 and 3
Output:	Difference logged on line printer
Character Type:	Hollerith and/or Binary

The "COMPARE" subroutine compares tapes 2 and 3 record for record.

If the record lengths are not the same, no comparison is made and the record number with the lengths of the records is printed.

```
Record          X          Tape 2    XXX  Words   Tape 3     XXX  Words
```

Records of the same length are compared word for word and differences are printed.

```
Record          X          Word XXX        Tape 2 Word      Tape 3 word
```

The words are printed as octal fractions. Records are numbered deimally from 1. Words within records are in octal with the position first word of the record assumed at Ø.

Comparison continues until an End of File is reached on either or both tapes. The appropriate message is printed.

4. Catalog

Control Card:	Cols. 18 - 20="LOG" Col. 22 = Tape to be logged
Output:	Listing on printer
Character Type:	Hollerith and/or Binary

The CATALOG subroutine reads one file from the tape unit specified in column 22 of the control card, printing out the type of information in the file. For binary tapes this consists of the types of records, the identifications of the records, the number of data words in the record, and the starting address of program records. For Hollerith tapes this consists of the "IDT" cards, the number of cards between the "IDT" card and the "END" card, and the "END" card.

12.0 PROGRAM General Input

12.1 IDENT 0360

12.2 FUNCTIONS

General Input is a closed subroutine allowing the programmer to specify, by means of calling sequences, various input operations and conversions. It will read cards or Hollerith tape records, and convert any field from octal or decimal to binary. It will read, as one operation, one card or up to 20 words from tape. If tape records are longer than this, information in the later words will be lost. Two calling sequence words are needed to create a full binary word. General Input will interpret a series of calling sequences, and branch back to the first register which is not interpretable as a calling sequence. If the requested operation is not possible, Condition Light No. 2 will be turned on.

The two-word calling sequence specifies the operation to be performed, where to get the input information, and what to do with the output. It contains: The action to be performed, the location in the input image to start the conversion, the number of characters to be converted, the address (which can be modified by a specified index register) or the accumulator, in which to store the result. * A tape number may also be specified.

12.3 OPERATION

This program, which is actually a closed subroutine, is automatically used by the program translator. Anyone wishing to use General Input by itself must incorporate it into his own program as with any other closed subroutine.

12.4 INPUTS

Two-word calling sequence:

Word 1:

Col. 24 = index register
The "Index Register" and "Address" together specify where the output of a conversion is to be placed. If the "Address" contains a zero, (on the symbolic card) and "ACC" in column 36-39, the result will be left in the right accumulator. Not used with RCD or RTP.

Cols. 25-27 = operation code (numbers in parenthesis, below, are octal codes)

1. RCD. (724) Reads one card, converts to six-bit Hollerith and stores it in pre-stored tape format. If card reader is not ready, Sense Light No. 2 is turned on.

   Parameters needed: none.

2. SDI. (700) The input columns specified (in "Location" and "How Many") are assumed to be a signed decimal integer. This is converted to binary, and stored in the right-half word of the specified address. If the first character is not "/" or "-", or if any of the following characters are not digits or blanks, Sense Light No. 2 is turned on. Parameters needed: relation location, "How Many", address.

3. UDI. (704) The specified field is converted as an unsigned decimal integer, and stored, as with SDI. If any of the characters specified are not digits or blanks, Sense Light No. 2 is turned on. Parameters needed: relative location, "How Many," address.

4. OCI. (710) The specified field is converted from an octal integer to binary and stored, as with SDI. If any of the characters are not digits (0 to 7) or blanks, Sense Light No. 2 is turned on. Parameters needed: relative location, "How Many," address.

5. MOV (740) The Hollerith characters in the columns specified by "Locations' and "How Many" are moved to the address specified, or to the accumulator if specified. A maximum of 5 characters may be moved with one calling sequence. The characters are stored in normal sequence (not inverted) and all blank bits are left at the beginning of the word. For example if the calling sequence requests that three characters be moved, starting with the character in position ten, character twelve will be stored in bits 26-31 of the specified location; character eleven will be stored in bits 20-25; character ten will be stored in bits 14-19; and bits 0-18 will be 0. If only one character is to be moved, it will be stored in bits 26-31 of the specified location. Parameters needed; relative location, "How Many," address.

6. UDF (744) The columns specified must contain a decimal point followed by four decimal digits. This field will be converted as an unsigned decimal fraction, and stored, as in SDI. If the first character is not a decimal point, or one of the following four columns is not a decimal digit, Sense Light No. 2 is turned on. Number of columns need not be specified, since the field is always 5 digits. Parameters needed: relative location, address.

7. RTP (720) Reads one record from a Hollerith tape with a PER07 (Assign Parity) and waits for the I/O Interlock to go off. Parameters needed: tape number. If tape is not ready, or not prepared, condition light #2 is turned on.

8. SDF (750) The columns specified must contain a "/" or "-" followed by a decimal point and 4 decimal digits. This field is converted as a signed decimal fraction and stored, as in SDI. If any illegal characters are found, Sense Light No. 2 is turned on. The number of columns need not be specified, since the field is always 6 characters. Parameters needed: relative location, address.

9. OCA (760) The specified code is converted from an octal address to a 17 bit binary number and is stored in bits ∅ and 16-31 of the specified address. If any of the characters are not digits ∅-7 or blank, Sense Light No. 2 is turned on.

cols. 28-29 = tape no. ; must be specified only with the RTP instruction; legal range is from 1-4.
cols. 36-41 = address

Second Word:

Cols. 24-26 = relative location

"Relative Location" refers to the position in the input image of the initial character to be converted. t is used only with the conversion routines, and not with RCD or RTP, in which cases it must be zero. It is a decimal integer with the range 1-100.

cols. 27-29 = "How Many."

"How Many" specifies in decimal the number of characters, starting at "Relative Location," which are to be converted. This number must not exceed the number of characters that can be fitted into half-word after conversion, except for MOV which requires no conversion. The field must be zero for RTP and RCD.

12.5 SPECIAL POINTS

1. It should be noted that the output of all the conversion routines, except MOVE, is a binary number of up to 16 bits. This is always stored in the right half of the specified register; the left-half of the word is cleared to ∅.

2. RCD and RTP do not require information in the second word of the calling sequence. However, this word cannot be omitted, since General Input assumes that all calling sequences are two words long.

3. Sense Light No. 2 is used to indicate illegal request. The sense light should, therefore, be turned off before branching to General Input.

4. Calling Sequence:

Word 1:	1 3	4 10	11 15	16 31
	Index Reg.	Operation	Tape	Address

Word 2:	0 7	8 15	
	Relative Location	"How Many"	

13.0 PROGRAM General Output

13.1 IDENT 0361

13.2 FUNCTIONS

General Output is a closed subroutine enabling the programmer to produce the equivalent of a 120 character print line in any format desired. The program, under the direction of the programmer, will convert binary information to Hollerith characters and store them in a 24 word "tape image". It will write the record on tape or convert and print the 24 words as one line on the line printer, or convert and punch the first 64 characters on a card.

The operation of General Output is governed by a BPX to the output program, followed by a two-word calling sequence. Where more than one action is desired of the output program, a series of calling sequences may be written with only one branch. The output program examines the calling sequence(s), performs the actions specified, and returns control to the main program at the first instruction not distinguishable as a calling sequence.

13.3 OPERATION

This program, which is actually a closed subroutine, is automatically used by the Program Translator. Anyone wishing to use General Output by itself must incorporate it into his own program, as with any other closed subroutine.

13.4 INPUTS

Two-word calling sequence:

13.4.1 Word 1:

col. 24 = Index register
The "Index Register" and "Address" together specify the address of the first register to be converted. Processing the contents of the accumulator is specified by a "Ø" in the "Address."

cols. 25-27 = operation code (numbers in parenthesis, below, are octal codes; the illustration referred to is at the end of this chapter.)

1. SDI: (700) Signed Decimal Integer

The SDI routine converts a 16 bit half-word to a signed decimal integer with leading zeros suppressed. The output of this routine is a sign followed by 5 integer characters placed in the "Relative Location" specified and the character positions following.

The first calling sequence in the illustration shown converts the right-half word of 40Z, placing the result in positions 1-6 of the tape image. The sequence at 10B converts the contents of the right accumulator and places the results in position 9-14. The sequence at 10C converts the left-half word of 40Z and places the result in 17-22. The contents of the tape image (equivalent to the printed line)is now:

```
∤ ----0 -- ∤ ---- 0 --  -31273  -- ....
```

2. UDI: (704) Unsigned Decimal Integer

The UDI routine converts up to 15 bits of binary data into an unsigned decimal integer with the number of digits specified in "How Many". Leading zeros will be suppressed.

The calling sequence at 10D converts bits R7-R15 of register 40Z to an unsigned decimal integer and places the result in positions 25-27 of the tape image. The sequence at 10E converts bits L1-L15 of register 40Z and places the result in positions 30-32. Note that the contents of the register was a negative number. The contents of the tape image are now:

```
∤ ---- 0 -- ∤ ---- 0 --  -31273 ---- 0 -- 494 -- ....
```

3. OCI: (710) Octal Integer

The OCI routine converts up to 15 bits of binary data into an octal integer. Leading zeros are suppressed. The converted octal integer is placed in the tape image starting at the "Relative Location" specified and the character position following. The OCI routine does not interpret signs.

The calling sequence at 10F converts L1-L12 of register 40W to an octal integer and places the result in positions 35-37. The contents of the tape image are now:

```
∤ ---- 0 -- ∤ ---- ∅ --  -31273 ----- 0 -- 494 -- 323..
```

4. OCF: (714) Octal Fraction

The OCF routine converts the contents of the half-word specified to a 6-character octal fraction. The converted octal number is placed in the tape image starting at the "Relative Location" specified and the five character positions following. Leading zeros are suppressed.

The calling sequence at 10G converts the right-half word of register 40W to an octal integer and places the result in position 1-6. Note that the contents of the tape image have not been erased and that the octal number generated is placed into the positions previously occupied by the characters generated by sequence 10A. The contents of the tape image are now:

```
140000 -- ∤ ---- 0 --  -31273 ----- 0 -- 494 -- 323 -- ....
```

5. WTP: (720) Write Tape

The WTP routine writes the contents of the tape image onto the tape unit selected. The WTP routine does not check for tape legalities nor write end of file.

The calling sequence at 10H writes the contents of the tape image onto tape unit 3. The contents of the tape image remain unchanged by this operation.

6. WPR: (724) Write Printer

The WPR routine prints the contents of the tape image on the line printer. The routine converts and prints a 120 character line. It also is possible to print from a core location other than the tape image. To do this the Hollerith characters must be in sequential core registers, positioned from right to left. The least significant bit of the first character must be in one of the following bit positions: 5, 11, 17, 23, or 29 of the first register. If the information extends beyond the first register, all the following registers must have the information starting in position 29. The "How Many" determines the number of characters to be printed.

The calling sequence at 10 J prints the contents of the tape image. The sequence at 10K prints the 9 characters starting in bit 17 of register 40W in the relative location specified. The contents of the original tape image are unaffected by the second printing from core. The two lines printed out are :

```
140000 -- / ---- 0 --  -31273 ---- 0 -- 494 -- 323 -- ....
--------- PRINT - OUT - ..
```

7. WPU: (730) Write Punch

The WPU routine punches the contents of the first 64 characters of the tape image on the card punch. It is possible to punch directly from core memory, as in printing from core. To punch in cols. 1-16 of the card, the data to be punched in these columns must be placed in "Relative Location" positions 1-16 and Sense Light No. 3 turned on prior to branching to General Output. Note that Sense Light No. 3 must be off for regular punching.

The calling sequence at 10L punches the contents of the tape image. The sequence at 10M punches out the 9 characters starting at bit 17 of 40W in columns 25-34 (Relative Location). The contents of the cards is the same as the output printed by the WPR instructions.

8. CLR: (734) Clear Tape Image

The CLR routine clears the tape image or any block of the tape image as specified by the "Relative Location" and "How Many."

The calling sequence at 10N clears the contents of the tape image.

9. MOV: Move Hollerith

The MOV routine moves Hollerith characters set up in core memory and places them in the tape image at the "Relative Location" specified. The characters to be moved must be set up as for printing from core.

The calling sequence at 10P moves the characters stored in 40W into the tape image. The sequence in 10Q prints out the tape image. The contents of the tape image are now:

```
_ _ _ _ _ _ _ _ _ PRINT _ OUT _ _ _
```

10. UDF: Unsigned Decimal Fraction

The UDF routine converts 15 bits of binary data into a 4 digit unsigned decimal fraction. The "Relative Location" specifies the position of the decimal point in the tape image.

The calling sequence at 10R converts the right half of register 40T into an unsigned decimal fraction and places the result in positions 21-25. The contents of the tape image are now:

```
_ _ _ _ _ _ _ _ _ PRINT _ OUT _ _ _ .5625 _ _ . . . .
```

11. SDF: Signed Decimal Fraction

The SDF routine converts a 16 bit half-word into a 4-digit signed decimal fraction. The "Relative Location" specifies the position of the sign in the tape image.

The calling sequence at 10S converts the left-half of register 40T into a signed decimal fraction and places the result in positions 28-33. The sign of the fraction is in 28 and the decimal point in 29. The contents of the tape image are now:

```
_ _ _ _ _ _ _ _ _ PRINT _ OUT _ _ _ .5625 _ _ -.5000 _ _ _ . . .
```

12. OCA: Octal Address

The OCA routine converts 17 bits consisting of the LS and the RHW of the register specified into a six-digit octal number starting in the relative location specified.

The routine at 11A contains calling sequences which will convert the 17 bit addresses in 41X, 41Y and 41Z. The contents of the tape image are now:

```
300000 _ _ _ 200000 _ _ _ _ 154763 _ _ . . .
```

cols. 28-29 = first bit; position of the least significant bit of the data
to be processed. (See coding and legal values).

cols. 36-41 = address

The "Address" (modified by the "IR" if designated) refers to the location of the first word to be processed by the output program. Where it is possible to process more than one register at one time, the registers must be in sequence.

13.4.2 Word 2:

cols. 24-26 = "relative location"
"Relative Location" refers to the position in the tape image of the first character converted. It is a decimal integer with the range 1-12Ø.

Cols. 27-29 = "How Many"
"How Many" specifies in decimal the number of characters, starting at "Relative Location," into which the data is to be converted.

Cols. 36-41 = no. bits (no. of binary digits to be processed.)

13.5 SPECIAL POINTS

1. CODING AND LEGAL VALUES

OPERATION	CODING		LEGAL VALUES			
NAME	SYMBOLIC	OCTAL	1st BIT	HOW MANY	NO. OF BITS	REL. LOC
Signed Decimal Integer	SDI	700	15 or 31	6	16	1 - 115
Unsigned Decimal Integer	UDI	704	0 - 31	1 - 5	1 - 15	1 - 120
Octal Integer	OCI	710	0 - 31	1 - 5	1 - 15	1 - 120
Octal Fraction	OCF	714	15 or 31	6	16	1 - 115
Write Tape	WTP	720	Tape Unit	120	Blank	1
Write Printer	WPR	724	5,11,17,23 or 29	1 - 120	Blank	1 - 120
Write Punch	WPU	730	5,11,17,23 or 29	1 - 64	Blank	1 - 64
Clear Tape Image	CLR	734	Blank	1- 120	Blank	1 - 120
Move Hollerith	MOV	740	5,11,17,23 or 29	1 - 120	Blank	1-120
Unsigned Decimal Fraction	UDF	744	0 - 31	5	15	1 - 116
Signed Decimal	SDF	750	15 or 31	6	16	1 - 115
Octal Address	OCA	760	31	6	16	1 - 115

2. Illustration of General Output

```
                    BPX             8ØAGO
1ØA                 SDI31           4ØZ
                     1  6              16
                    CAD             4ØZ
                    BPX             8ØAGO
1ØB                 SDI31           ACC
                     9  6              16
1ØC                 SDI15           4ØZ
                    17  6              16
1ØD                 UDI31           4ØZ
                    25  3               9
1ØE                 UDI15           4ØZ
                    3Ø  3              15
1ØF                 OCI12           4ØW
                    35  3              12
1ØG                 OCF31           4ØW
                     1  6              16
1ØH                 WTP 3               Ø
                     112Ø               Ø
1ØJ                 WPR29               Ø
                     112Ø               Ø
1ØK                 WPR17           4ØW
                     9  9               Ø
1ØL                 WPU29               Ø
                     1  64
1ØM                 WPU17           4ØW
                     9  9               Ø
1ØN                 CLR                 Ø
                     112Ø               Ø
1ØP                 MOV17           4ØW
                     9  9               Ø
1ØQ                 WPR29               Ø
                     112Ø               Ø
1ØR                 UOF31           4ØT
                     21 5              15
1ØS                 SDF15           4ØT
                    28  6              16
                    HLT
4ØY                137777          044000
4ØZ                -31273          000000
4ØW                 HOL             FRI
                    HOL             NT-OU
                    HOL             T----
4ØT                 .5625          -.5000
11A                 BPX             8ØAGO
                    OCA31           41X
                     1  6              16
                    OCA31           41Y
                    10  6              16
                    OCA31           41Z
                    20  6              16
41X                101000          100000
41Y                101000          000000
41Z                001000          154763
```

14.0 PROGRAM: Link

14.1 IDENT 0372

14.2 FUNCTIONS

Translates certain Lincoln Utility System terms into COMPASS language. The terms translated are

Lincoln	COMPASS
IDENTC	IDT
TERM START	END
SKIP	SKP
DITTO	DIT
CORE	(deleted)
DRUM	(deleted)
TØØ2	TPYØ + ØØ2
T--3	TPYØ + --3
A CAD	ØØA CAD
B ADD	ØØB ADD
1B FST	Ø1B FST
26A LDB	26A LDB
B DEP	26B DEP

Suppressed digits in tags in the director are filled in with zeros.

14.3 OPERATION

1. DCA Program to be assembled should be loaded on prestored tape and put on tape unit No. 3.

2. COMPASS Control Card, with "LINK" in cols. 18-21, should be put into card reader.

3. A COMPASS Control Card, with "POS" in cols. 18-20, and "2R" in cols. 22-23 should follow the "LINK" card in the reader.

4. Press "Load From AM Drums."
The Lincoln deck on Tape No. 3 will be translated to a COMPASS language deck on Tape No. 2. The tape on drive No. 2 will be rewound to the load point, ready for assembly.

14.4 SPECIAL POINTS

A Compool Override Card (see Program Translator) should be in the instruction deck (preferably following the "Ident" card) to define the location of the artificial table TPYØ, unless table TPYØ is defined in the environment section of the compool for this program.

APPENDIX A. Changes to Program Translator Program (4.0)

The following changes have been incorporated to date into the 65K version of the Program Translator, as differentiated from the 8K version.

1. Capacities have been increased as follows:

a)	Maximum number of cards	7650
b)	Maximum number of tags	2000
c)	Maximum number of RC words	750
d)	Maximum number of CPO cards	300

2. When the maximum number of cards is exceeded, the direct printer will log, "TOO MANY CARDS".

3. When the maximum number of CPO cards is exceeded, the direct printer will log, "EXCESSIVE" to the right of each additional card.

4. The program will assemble programs located anywhere in the 65K (large) memory. It will also accommodate programs in the 4K (small) memory, except for the following case:

It will not assign an address above 177777 to a word containing a constant (i.e., not in instruction) in the LHW and a location tag in the RHW. To do this would prohibit the use of the subroutine tape.

5. All assemblies, DCA or otherwise, will log on the direct printer all IDT, LOC, DRM and END cards.

6. DCA assemblies will, in addition to (5) above, log on the direct printer all CPO cards; and the number of spare registers, as determined from the Compool allocation minus the program length, will be indicated after the END card.

7. In case an END card contains an undefined tag on the director, it will be so indicated on the direct printer.

8. Instructions or tables located at zero may now be tagged, and absolute references to register zero will be printed (and punched) as 000000, rather than 377777, as in previous versions.

9. All programs, including DCA, will be accepted with 5 character alpha-numeric tags. The tag table search sequence is as follows:

a) CPO assignments
b) Location tags
c) Compool (if Compool sensitive)

Thus, a program may assign for its own use a new item to an existing table in the Compool, or assign a Compool item to a table defined by the program.

10. The program will now distinguish between an EOF and an EOT, thus very large files of successive programs may be loaded onto a tape without fear of running off the end. If an EOT is sensed before the END card of an assembly, the translator will halt while a new tape containing the remainder of the program is readied on TD-2. A CONTINUE action will allow it proceed assembling the program from the new tape.

11. A program which does not include a LOC card to define the starting location will automatically be assembled starting in location 300.

J. R. Tobey

J. R. Tobey

JRT/gw

Mitre Corp.

D-12

THE MITRE CORPORATION
Lexington 73, Massachusetts

Technical Memorandum TM—15 #4

Sheet 1 of 5 Sheets

TITLE:

Subject: Basic XD-1 Information: COMPASS UTILITY PROGRAMS

To: J. D. Porter

From: J. R. Tobey
J. R. Tobey

Date: 3 March 1959

Approved: J H Burrows
J. H. Burrows

ABSTRACT

This memo contains brief descriptions of seven (7) programs designed to aid in the operation of post-assemblies of COMPASS programs. In the case of two one card programs, Core Read-In, and Drum Read-In, a program listing is supplied. All programs, however, do have operating descriptions as well as program descriptions included in this memorandum.

DISTRIBUTION LIST.

J. R. Tobey (25 copies)

1. This document has been prepared for internal use only. It has not been reviewed by Office of Security Review, Department of Defense, and therefore, is not intended for public release. Further dissemination or reproduction in whole or in part of the material within this document shall not be made without the express written approval of The MITRE Corporation (Technical Information Department).

2. The research reported in this document was supported by the Department of the Air Force under Air Force Contract AF-33 (600) 37882.

TM-15 #14

TABLE OF CONTENTS

1.0 TAPE READ-IN

1.1 DESCRIPTION

TAPE READ-IN is a four (4) card "Load from Card Reader" program which will perform the following functions:

1. Read into core a binary tape (Tape Unit 3) containing the following records:

    a. File ID (Type 1)
    b. Storage ID (Tape 2)
    c. Drum Storage Record (Type 3)
    d. Core Storage Record (Type 6)
    e. Operate Record (Type 7)
    f. End ID (Type 5)

2. Place the contents of "d" (Core Storage Record) into its properly assigned core location.

3. Place the contents of "c" (Drum Storage Record) onto its properly assigned drum field(s).

4. Provide an option (Sense Switch 1) for operating "d" (Core Stoarage Record).

1.2 OPERATION

If the binary tape to be read in contains more than one (1) identification group, it must first be positioned, using a COMPASS Control Card.

TAPE READ-IN is loaded by the "Load from Card Reader" action, and will halt under the following conditions:

1. Program Counter at thirty-two (32):

    a. Condition Light 4 "OFF" -- A checksum error has occurred.
    b. Condition Light 4 "ON" -- An END ID record has just been read.

    A "Program Continue" action with the PC at 32 will cause Tape 3 to backspace and re-read the same record.

2. Program Counter at one hundred ten (11Ø)

    a. Condition Light 4 "OFF" - An OPERATE record has just been read. A "Program Continue" action will:

        1. With S.S. 1 "ON" -- operate the last Core Record that has been read in.

2. With S.S. 1 "OFF" -- read the next record.

b. Condition Light 4 "ON" -- A STORAGE ID record has just been read. A "Program Continue" action will read the next record.

NOTE: TAPE READ-IN is designed for use with COMPASS binary output tape (on Tape Unit 3).

2.0 CORE READ-IN

2.1 Purpose

To read into core storage COMPASS binary decks assembled for CORE (decks assembled for drums will be read into core at the drum addresses -- the program does not check type of assembly).

2.2 Loading and Control

This card is entered by a LOAD FROM CARD READER action and occupies Regs. 0-57. It will read in the binary cards following it until a branch card is sensed (determined by +0 in bits R. 11-15 of word zero). It will then halt with the PC=27. A CONTINUE action will bypass this condition. The sum check will not be effective when word one is full zero, either plus or minus.

2.3 Program Description

```
LOC   0        RDS        30      CORE READ IN-READ ONE CARD.
LOC   1        LDC        30      WAIT FOR 1/0 LOCK AND SET CTR
LOC   2        CAD        30      FOR NEXT CARD
LOC   3        RSTA       27      SET INITIAL CORE ADDRESS
LOC   4        RSTA       23
LOC   5       1XIN        26      SUM CHECK CARD
LOC   6       1ADD        31
LOC   7       1BPX01       6
LOC  10        BFZ        14      BRANCH IF OK
LOC  11        CAD        31      IF NOT OK, CHECK BYPASS
LOC  12        BFZ        14
LOC  13        HLT        37      CHECKSUM HALT
LOC  14        CAD        30      GET NR OF WORDS.
LOC  15        FCL        20
LOC  16        ETR        13
LOC  17        BFZ        26      IF ZERO - BPX CARD
LOC  20       1XAC                MOVE WDS TO CORE LOC
LOC  21       1BPX01      22
LOC  22       1CAD        32
LOC  23       1FST
LOC  24       1BPX01      22
LOC  25        BPX         0      RETURN FOR NEXT CARD
LOC  26        HLT                ALL CARDS LOADED
LOC  27        BPX                BRANCH TO OPERATE
```

3.0 DRUM READ-IN (0045)

3.1 Purpose

To read-in COMPASS binary cards assembled for DRUMS and store their contents on drums.

3.2 Loading and Control

This card is entered by a LOAD FROM CARD READER action and utilizes registers Ø - 57. It will read-in the binary cards following it until an End card is sensed. (Determined by a 9's row punch in column 18). It will then halt with the PC = 31. When a check sum error is encountered, it will halt with the PC = 25. A CONTINUE action will bypass this condition. When the checksum word (9's row right) is full ± Ø, the checksum will not be performed.

NOTE: If an attempt is made to read in with this card a COMPASS deck assembled for CORE, the computer will hang up selecting drum Ø after the first card is read.

3.3 Program Description

```
 0    SDROO          0    PERFORM XFER TO DRUM
 1    WRT            0    DISCONNECTS ON FIRST PASS
 2    SEL01               READ A CARD
 3    LDC           30
 4    RDS           24
 5    LDC           32    PAUSE AND SET FOR DRUM XFER
 6    CAD           30
 7    RST            0    SET DRUM ADDRESS
10    FCL           26
11    RSR           10
12    RST            1    SET NUMBER OF WORDS
13   1XIN           22    SET IX FOR CHECKSUM
14    LDB           25
15    DEP            0
16    CAD           31    CHECKSUM BY-PASS
17    BFZ           26
20    CAD           30    CALC CHECKSUM
21   1ADD           31
22   1BPX01         21
23    BFZ           26
24    HLT                 CHECKSUM ERROR HALT
25        77         0    MASK - ETR INSTRUCTION
26    TOB01         30    TEST FOR END CARD
27    BPX            0
```

4.0 PUNCH 24 WORD CARD

If it is desired to write short, self loading programs, the use of this one card program takes all the drudgery out of it.

Write the program in symbolic and assemble it with tape specified as the binary output, and core as to the type of storage. Symbolic input and output are left to the programmer's choice. If symbolic input is cards, a WAIT card follows the instruction deck; if tape, the WAIT card follows the ASSEMB card. This one card program follows in the reader. When the assembly is complete, COMPASS will halt upon reading the WAIT card. Then "Assign Test Memory" and put the number of cards in the right A switch and "Load from Card Reader". The deck will then be punched, 24 data words to the card and two extra cards are fed through automatically. If more copies are desired, press "Program Continue".

The LOC card should contain zero in the address field. The program may not be over a drum field in length nor may it contain any SKP instructions. The first instruction of the program to be assembled will usually contain a RDS instruction. Do not branch to this instruction. If this instruction has to be modified, or branched to, address it in absolute.

5.0 Q-7 UNIVERSAL CARD LOAD

The Universal Card Read-In Program (UNCDRD) is a six card, self-loading deck. It is designed to accept the following types of cards and store them in their specified core or drum locations:

1. COMPASS binary cards to be stored in core.

2. COMPASS binary cards to be stored on drum.

3. Environment Sim Load cards to be stored in core or drums.

4. Octal correction cards to be stored in core or drums.

An incorrect check sum will halt the computer with the program counter equal to 20. If "Continue" is pressed, the card will be stored in its proper location. To bypass the check sum verification (on card types 1-2-2 above) column 27 in the 9's row now should be punched.

A punch in column 18, row 9 signifies an END card. When this card is reached the computer will halt. Upon pushing "Continue" a branch to the core location specified in the End card will occur. This core location is also stored in the live register.

The UNCDRD occupies and uses registers 0-237. Therefore, special caution should be observed to avoid reading in data below core location 240.

6.0 Q-7 3-CARD OCTAL LOAD (0359)

6.1 Purpose

3 Card Octal Load is designed to simplify the storage of information into core or drum memory.

6.2 Operation

The program accepts cards of the following format:

col. 18-22	location
23-24	drum
25-36	1st word
37-48	2nd word
49-60	3rd word
61-72	4th word

The parameters specified in "1st word" will be stored in the location specified. The "2nd word", if not blank, will be stored in the following register; similarly with "3rd word" and "4th word". If the drum field is blank or contains zeros, it is understood that core is desired. In the case of aux drums, column 18 will contain a 4; i.e., 18 to 22 will be translated as the complete right half of the SDR instruction.

Cards must contain only octal numbers. No checks will be made for illegal punches. The first "word" field which is completely blank signifies the end of the card; no further information will be read from it. Zeros (except all zeros) may be supressed in any field.

An "E" in column 17 will signify an END card; when this is read, the program will halt.

The program is available as a Three-card self loading deck. It occupies and uses registers 0-137.

7.0 BOLD - A Binary and Octal Loader for the AN/FSQ-7

7.1 Purpose

BOLD is designed to load cards produced by the COMPASS system (cards with drum locations, cards with core locations and Environment Simulation cards) and octal correction cards in both UNCDRD format and OTL format.

7.2 Card Formats

Card formats are as specified in TM-15 #3 except that OTL format may be usef for octal correction cards -- i.e., mnemonic codes may be used for instructions and an "x" in column 18 may be used to denote auxiliary drum storage. In addition, one extra column, column 17, is added to the location field of octal correction cards to permit the expression of addresses in excess of 77777. BOLD will store corrections into any core location.

It is not permissible to introduce meaningless punches into the zones of end cards or binary cards with a word count of 16. Doing so may cause the card to be erroneously considered as an octal correction.

7.3 Special Features

a. Sum Checks

All methods for suppressing the sum check used by other loaders in use are also applicable with BOLD. Either a 9 punch in column 27 or a full zero check sum word will suffice.

b. Octal Cards

1. In addition to the instructions recognized by OTL, BOLD recognizes the new compare, test bits and CAC instructions. A test memory address, however, is not automatically provided for CAC and thus must be supplied by the programmer. In addition, the mnemonic "STZ" will be interpreted as the ops code 300.

2. As in coding for the Program Translator, the 17-bit option may be indicated by following the mnemonic instruction code with the letter "A".

3. If a right half word requiring 17 bits is specified, it will be so stored; but if the right half word requires less than 17 bits, the sign specified for the left half word will be stored in left sign.

   As an example, the following are representations of the same word and would be interpreted as being identical:

```
4ADDA 200371
4ADD10200371
041050200371
141050200371
141050200371
```

c. End Cards

When BOLD encounters an end card, a holt will occur. Pushing "PROGRAM CONTINUE" will cause a branch to the location specified on the end card. If an uninterrupted branch to this location is desired, a 9 punch should be placed in column 19 of the end card. Since this will invalidate the check sum, one of the sum check inhibition described in "a" should also be employed.

7.4 STORAGE AND ASSEMBLY INFORMATION

BOLD currently requires 245 (decimal) registers of permanent storage plus 27 registers of temporary storage. In the standard version currently available, this storage is located consecutively in registers 177360 to 177777.

BOLD may of course be assembled at other locations. The symbolic deck has 0700 as an Ident. and is on file in the Building F card room. To assemble BOLD at a different location, it will usually be desirable to change the TCPO cards defining temporary storage as well as the LOC card. There are four of these defining the symbols IM, DPI, VAL, and WORD. Twenty- four registers must be reserved for IM; the other symbols refer to single registers. To place the temporary storage immediately after the program, add 365 (octal) to the value on the LOC card to get the value for the first TCPO card.

7.5 USE

a. Operating Deck

1. One card loader (B' for example)
2. Binary deck of BOLD (with end card)
3. Cards to be loaded
4. End card

Copies of the standard version come with a copy of B' attached.

b. Logging

Logging of octal cards is controlled by sense switch four (4). If sense switch 4 is in the inactive (up) position, all octal correction cards will be listed on the direct printer. If sense switch 4 is in the active (down) position, only error cards will be so listed.

c. Halts

BOLD contains three halts as follows:

Cause of Halt	P.C. contents (standard version)	P.C. contents (relative	Effect of pushing "PROGRAM CONTINUE"
An end card has been encountered.	177403	23	Branch to core location on end card.
An illegal mnemonic code has been detected (most, but not all, such errors will be detected).	177564	204	Read next card.
Check sum error	177642	262	Check sum will be ignored.

NOTE: a) A check sum error in loading BOLD itself will cause B' to come to a halt at 14.

b) If BOLD is not destroyed in the machine, it can be re-entered by a BPX to 17736∅.

c) No 12 punch is allowed in col. 79 or 80 of an octal card.

JRT/hht

Mitre Co.
W-19

THE MITRE CORPORATION
Lexington 73, Massachusetts

Technical Memorandum TM—15 #5
Sheet 1 of 7 Sheets

TITLE:

Subject: Basic XD-1 Information: DEBUGGING PACKAGE

To: J. D. Porter

From: J. R. Tobey

Date: 4 March 1959

Approved: J H Burrows
J. H. Burrows

ABSTRACT

With acknowledgement to SDC for information contained within, this memo contains the workings of four programs designed as a diagnostic aid in program checkout:

1) Memory Print program to print out selected portions of core memory, direct or delayed.

2) Storage Dump program to read-in Memory Print from drums and operate it when the COMPASS programs are on drums.

3) Q-7 Trap and Print to interrupt the operation of a program at specified points and print the contents of the machine registers and any portion of core memory.

4) Loop Trace program design to aid in unpredictable problem areas of program checkout.

DISTRIBUTION LIST

J. R. Tobey (25 copies)

1. This document has been prepared for internal use only. It has not been reviewed by Office of Security Review, Department of Defense, and therefore, is not intended for public release. Further dissemination or reproduction in whole or in part of the material within this document shall not be made without the express written approval of The MITRE Corporation (Technical Information Department).

2. The research reported in this document was supported by the Department of the Air Force under Air Force Contract AF-33 (600) 37882.

TM-15 #5

TABLE OF CONTENTS

1.0 Q-7 Memory Print (0352)

1.1 Purpose

The Memory Print program (0352) will print out selected portions of core memory, drums, or binary tape records as constants or instructions either directly on the line printer or as delayed output on tape unit 4. The program will operate in either core memory, and with the new instructions. The program will destroy itself and restore rare memory (with the exception of locations Ø-27) if desired when printing core or drums. When printing the contents of binary tapes, however, locations 2000 thru 6004 or core memory are used as temporary storage and part of this region may be destroyed depending on the length of the tape record. None of the control or computing registers are retained.

1.2 Operation

Memory print is available as a self loading binary deck, or on the COMPASS system tape. This description of the program is concerned with the binary deck.

The program is read-in with test memory assigned. The read-in stores the contents of core memory Ø thru 3777 onto drums 26 and then reads in the Memory Print program into registers 60 thru 1777 and halts with the program counter at 6Ø. Upon completion of read-in drum 26 contains the read-in program in locations Ø-27 and the contents of core memory in 3Ø thru 3777. The program operates on information in the A and B switches using the sense switches for control and is operated by pressing the "Program Continue" button.

The A and B switches provide the parameters required by Memory Print. The left A switches (not including LS and R1 for aux. drums) determine whether core, drums, or tapes are to be printed and also which drum or tape. The right A switches and the LS (for the 17 bit large memory addresses) specify the starting address of the region to be printed. The right B switches and the LS specify the final address of the region to be printed. The A and B switches do not affect the operation of the program once it has begun.

The four sense switches control the operation of the program. Sense switch 1 controls the re-run of the program. In the off (up) postion pressing "program continue" will re-run the program. In the on (down) position pressing "program continue" will cause the program to read in the contents of drum 26 into core memory, thus restoring core. Sense switch 2 in the off (up) position will give a delayed output on tape 4. In the on (down) position the output will be on the line printer.

Sense switch 3 in the off position will provide output in the form of instructions. In the on position the output is in the form of octal constants. Sense switch 4 controls the tape printing. In the off position an entire record will be printed. In the on position the right A and B switches determine the portion of the tape record to be printed.

The output of the program is printed four registers to a line with the location of the first register of the line specified at the left. Only the first register of a consecutive string of identical registers is printed, the rest being suppressed. Each block of memory printed is preceded by a line specifying the block requested.

1.3 Operation Features

Memory Print makes several legality checks and prints ERROR on the line printer in the following cases: test memory unassigned; starting address greater than the final address; left A switches (L1-L15) not zero, SDR or SEL; tape unit 4 requested to be printed.

Sense switches 2 and 3 are examined continuously. The A and B switches are examined initially and may be changed while the program is in operation without affecting the output.

In printing tapes the addresses specified begin with the value in the right A switches. When printing whole records this should be zero. When printing a part of a tape record the addresses in the A and B switches are interpreted on the basis of the record starting at zero. Maximum record length is 2Ø51 words.

The large memory addresses (up to $377,777_8$) and the new instructions are included. Instructions using L12 for control are printed with a "1" following the instruction code (e.g., STA1, ADD1).

Examples

1) Print all of core delayed as instructions

Same Switches: 2-up 3-up
A: 000000 000000
B: 000000 020000

2) Print 1650_8 to 2730_8 of auxiliary drum 2 direct as constants.

Same Switches: 2-down 3-down
A: 006102 041650
B: 000000 002370

3) Print one record of tape 3 direct as instructions

Sense Switches: 2-down 3-up 4-up
A: 006213 000000
B: 000000 00400

4) Print 60_8 for 300_8 of the next record on tape 2 direct as constants.

Sense Switches: 2-down 3-down 4-down
A: 006212 000000
B: 000000 000300

Restore core
Sense Switch 1-down

1.4 Summary of Operation

Load deck in Card Reader with Test Memory Assigned Press.
"Load from Card Reader".
Set A and B switches for desired output.
Press "Program Continue".

Program may be re-run as often as desired by resetting A and B switches and pressing "Program Continue". Changing the A and B switches will not affect the operating of the Memory Print once it has begun. Sense switches 2 and 3 are examined at each pass.

To restore core after completion, depress sense switch 1 and "Program Continue".

1.5 Switch Operation

SENSE SWITCH	2	ON:	DIRECT OUTPUT
		OFF:	DELAYED OUTPUT (TAPE 4)
SENSE SWITCH	3	ON:	OCTAL CONSTANTS
		OFF:	INSTRUCTIONS
SENSE SWITCH	4	ON:	PART OF TAPE RECORD
		OFF:	TAPE RECORD

A and B SWITCHES - CORE

A:	L1 - L15 - ZERO	LS and RHW	: STARTING ADDRESS
B:	L1 - L15 - ZERO	LS and RHW	: FINAL ADDRESS

A and B SWITCHES - DRUM

A:	LHW - SDR and DRUM CODE*	RHW	: STARTING ADDRESS
B:	LHW - ZERO	RHW	: FINAL ADDRESS

* also R1 for auxiliary

A and B SWITCHES - TAPE

```
A:  LHW - SEL and TAPE CODE    RHW: STARTING ADDRESS*
B:  LHW - ZERO                 RHW: FINAL ADDRESS

                         * zero for full record
```

This program is further described in TM-15 #3

2.0 Storage Dump

2.1 Purpose

To read-in Memory Print from drums and operate it, when the COMPASS programs are on drums.

2.2 Loading and Control

A "Load From Card Reader" action will enter this one card program and cause the Memory Print program to be read into core beginning at location 60. After the read-in, a halt will occur with PC=7. A "Continue" will operate Memory Print.

```
0      SDR26
1      LDC          0
2      WRT       2048
3      SDR11
4      LDC         60
5      RDS        976
6      HLT
7      BPX         60
```

3.0 Q-7 Trap and Print

3.1 Purpose

The Trap and Print program (0355) is intended to interrupt the operation of a program at specified points and print the contents of the machine registers and any portion of core memory. The original program will then continue at the point where it was interrupted. Printing may be direct or delayed. It will handle locations in core through 177777.

Trap and Print is controlled by means of cards which specify the location at which the trap is to occur, the region of memory to be printed, and how often the trap is to be executed, if the program goes through it more than once.

3.2 Restrictions

The following restrictions govern the use of the trap and print program:

1. No more than 25 traps may be used at one time.

2. Only one contiguous portion of memory may be printed at each trap.

3. Although all other registers and triggers are saved and restored, the IO word counter and A register are not saved.

4. Instructions which the program modified cannot be trapped.

5. Since the A register is not saved, neither an STA instruction nor BPX to an STA instruction can be trapped.

3.3 Use

The program to be trapped, and any octal corrections, should be read in with a read-in program which stores the starting location in the live register. The following deck is then read in by pressing the Load From Card Reader button:

Core Read In - Single card which does not store the starting location in the live register.

Trap and Print Binary deck

Trap Print control cards
End Card - 9 in column 17
Any other cards
If no other cards follow the end card, it may be omitted.

Trap and Print starts at 16700_8 but can be relocated for convenience.

If a program stops unexpectedly when Trap and Print is in memory, any portion of memory can be dumped by manually branching to Trap and Print with the address of the first register to be dumped in the left B switches and the address of the last register to be dumped in the right B switches.

3.4 Control Cards

The cards used to control Trap and Print have the following format:

Columns 17 - 22: L (Octal location of instructions to be trapped).
Columns 25 - 30: M (Octal number of times to bypass trap).
Columns 31 - 36: N (Octal number of times to execute trap).
Columns 37 - 42: A (Octal location of first register to be printed).
Columns 43 - 48: B (Octal location of last register to be printed).

End card:

Column 17: 9

Trap and Print will do nothing the first M times the instruction at L is executed; it will print the first N times after that. If M = N = 0, the trap will be executed each time the instruction is encountered.

The registers from A to B will be printed each time the trap is executed; if A = B = 0, no memory registers, but only the machine registers will be printed.

3.5 Output

Output will be on tape 4, unless sense switch 2 is depressed, in which case printing is direct on the line printer.

No headings are provided on the output. Each trap prints the following information in octal:

> A line (indented one space) containing, in this order, the location of the trap, contents of accumulator, contents of B register, contents of index registers 1, 2, 4 and 5, the numbers of the condition lights which were on, and L or R to indicate left or right overflow indicators.

This line if followed by the contents of memory registers requested (if any). A location and four full words are printed per line. Printing is non-repititive; i.e., only the first of a block of registers with the same contents will be printed, but -0 is distinguished from 0. For example:

```
 110440  013642  177546  027453  002310  177777  000144  003216  000001  R12
112100  014637  021301  177510  036142  000000  000000  177777  177777
112104                                  150437  000324  000277  000234
```

Here the instruction at core location 110440 was trapped. At that point the accumulator contained 013642, 177546; the B register contained 027453, 002310; index registers 1, 2, 4 and 5 contained 177777, 000144, 003216 and 000001; the right overflow indicator and sense lights 1 and 2 were on. The contents of registers 112100-112107 are printed. Since 112104 and 112105 are blank, they had the same value as 112103.

If output is on tape, Trap and Print does not write end of file or rewind the tape. A one-card program is available to perform this function.

4.0 Loop Trace (0296 and 0297)

4.1 Purpose

Loop Trace is a twenty-eight card self loading program which has been designed as a diagnostic aid in program checkout and also as an emergency aid in unpredictable problem areas. It functions as a branch trap, printing each active branch encountered by the program being traced, the location and director of the branch, the contents of the accumulator, the B register and four index registers just after the branch was executed. It is equipped to handly any combination of IO transfers in the program being traced. Two versions are available: LOW (core memory Ø to 1267) and HIGH (Ø to 27 and 2ØØØØØ to 377777). Binary decks are available from the card room by request. Requests for HIGH should be for deck Ø296. Requests for LOW should be for Ø297. A sample printout is included as appendix number 1.

4.2 Operation

To trace a loop...

Whenever any program unexpectedly loops and a record of the branches being executed would be helpful,

1. Stop the program using "PROGRAM STOP".
2. Place the contents of the program counter in the right "A" switches (and the left sign if necessary in the large memory).
3. Assign test memory.

4. Depress sense switch number 2 if output is desired on the line printer.

5. Place the deck of "Loop Trace" in the card reader.

6. Ready reader.

7. Push "LOAD FROM CARD READER".

If everything has been done properly, the loop will be traced and control return to the looping program. The work "COMPLETE" will be logged on the printer at the conclusion of the trace. At this point a core dump may be taken.

To use Loop Trace as a branch trap program,

1. Read in the program with its environment into core.

2. Place the starting address of the program in the right "A" switches (and the LS if needed).

3. Assign test memory.

4. Set sense switch number 2 up for delayed or down for direct output.

5. Place the deck of "Loop Trace" (HIGH or LOW) in the reader.

6. Ready reader.

7. Push "LOAD FROM CARD READER".

The printer will restore the page if selected initially. The trace will continue until either a program halt or an attempted branch to any area occupied by "Loop Trace". In either case, "I AM FINISHED..." will be logged on the printer and the trace is complete. No halt will ever be made by "Loop Trace" unless it first logs the reason on the printer.

4.3 <u>Printouts</u>

ASSIGN TEST MEMORY --- self explanatory. Assign test memory and "CONTINUE".

READY TAPE 4 --- DLO tape has been selected and either no tape 4 is on line or it is physically not ready (not prepared does not stop "Loop Trace"). Either depress sense switch number 2 for direct output or ready tape number 4. Push "CONTINUE".

COMPLETE --- "Loop Trace" has completed tracing a loop. The next line to be printed would be identical to the first line. The traced program will be in control (and will be looping, of course).

I AM FINISHED --- The last active branch encountered was an attempt to enter that portion of core occupied by "Loop Trace" (∅ to 1267 for LOW, ∅ to 27 and 2∅∅∅∅∅ to 377777 for HIGH) or the traced program was preparing to halt. End of file will be written on the DLO tape if sense switch number 2 is up at the moment this branch is encountered. Any attempt to press "CONTINUE" will do nothing.

4.4 Use of Sense Switches 2 and 3

It is not necessary to get all your output on the printer of the DLO tape. Each line is sent to the IO unit selected by the sense switch. This means you can start direct and finish delayed, or any combination of printer and tape. "Loop Trace" will not lose a line regardless of how much the sense switch setting is changed. The delayed output tape is never rewound. This means that Compass listings, core dumps and the "Loop Trace" output may all be placed on the same tape. "Loop Trace" will write end of file at the completion of the trace if tape is selected by the sense switch at that moment. If it is wished to send output to both the printer and the tape, SS number 2 should be up, and SS number 3 down. This, in effect, allows the operator to monitor the DLO tape. SS number 3 may be raised or lowered at will without affecting the DLO.

4.5 The "B" Switches

Occasionally it is more desirable to have one's output direct and immediately available, yet a great many active branches at the beginning of the program must first be executed and in being logged, slow down the program with unnecessary information. In such cases, the address in the "A" switches should be the location where logging is desired. The starting location of the program itself should be placed in the "B" switches. The program will then be operated from the point specified in the "B" switches to the point specified in the "A" switches where printing (if sense switch number 2 is down) will begin. No check on the comparative size of the values in the "A" and "B" switches is made. They are checked for reference to areas of core occupied by "Loop Trace" however (see note under "I AM FINISHED..." above). If no printing is obtained using this "B" switches feature, it means that the address specified in the "A" switches was never finished.

4.6 Restrictions

1. "Loop Trace" uses six of the new computer instructions and hence cannot be used on any Q-7 not recognizing them.

2. Although one may switch between direct and DLO at will, no indication is made on the output that this has been done and it may be difficult to tell when the output was sent to DLO and when to the printer. Sense switch number 3 should be used for both outputs, if desired.

3. When using the "B" switches feature described above, be careful to select the address of the instruction where printing is to start such that this instruction is never modified or referred to by any other instruction. Usually any non tagged instruction in your program will be satisfactory. This is necessary because the instruction specified in the "A" switches is replaced with a BPX to "Loop Trace" and should any other instruction subsequently refer to this location, it will not contain what it expects to find.

4. "Loop Trace" is unable to distinguish an actual halt from a +Ø, +Ø register which will be filled with a FST before operation. As an example:

```
              3 CAD    6ØA
                FST    1ØA
       1ØA      + Ø    + Ø
```

Assume that 6ØA is a table of branch instructions. "Loop Trace" uses a type of look ahead routine rather than an interpretive one, consequently it will assume that the program would halt if allowed to, and will itself halt with the "I AM FINISHED..." printout. There is not solution to this problem except coding a dummy branch (if that is what is expected) or other class instruction (if a branch is not expected) into that register.

5. Any branch to a subroutine which does not begin with an STA(A), but with a FST, say, then an STA(A), will result in "Loop Trace" ending the trace at the exit of the subroutine. No solution or recovery possible.

6. Indexing into a table which immediately follows a BPX without an intervening branch class instruction or TTB will result in erroneous results, the exact nature of which is completely unpredictable. If any programmer's program contains either type of coding described under this or the preceding point, he should use a different trace program.

J. R. Tobey

J. R. Tobey

JRT/hht

```
 LOC.      INST.   DIR.          LAC     RAC      LBR     RBR        IX 1    IX 2    IX 4    IX 5   LOW  --  200000

200002    BSN13   200004        040000  100000   177777  177777     000027  200000  200000  000003
200006    BPX     200007        040000  100000   177777  177777     000027  200000  200000  000003
200012    BSN14   200012        040000  100000   177777  177777     000027  200000  200000  000003
200013    BSN04   200066        040000  100000   177777  177777     000027  200000  200000  000003
200071    BSN11   200073        177776  177777   177775  177777     000027  200000  200000  000003
200101    BRM     200103        177776  177777   177775  177777     000027  200000  200000  000003
200104   1BPX01   200105        177776  177777   177775  177777     000027  200000  200000  000003
200110   2BPX12   200106        177777  177777   177777  177777     000027  000005  200000  000003
200110   2BPX12   200106        177777  177777   177777  177777     000027  200000  200000  000003
200111    BSN22   200136        177777  177777   177777  177777     000027  200000  200000  000003
200143    BLM     200145        104312  175000   177777  177777     000027  200000  200000  000003
200145    BFM     200153        104312  175000   177777  177777     000027  200000  200000  000003
200157   5BPX01   200155        177777  002503   177777  177777     000027  200000  200000  000005
200157   5BPX01   200155        177777  001567   177777  177777     000027  200000  200000  000004
200157   5BPX01   200155        177777  000653   177777  177777     000027  200000  200000  000003
200156    BFM     200161        177777  177736   177777  177777     000027  200000  200000  000003
200166    BFZ     200000        000000  000000   177777  177777     000027  200000  200000  000003

COMPLETE
```

THE MITRE CORPORATION
Lexington 73, Massachusetts

Technical Memorandum TM—15 #6
Sheet 1 of 11 Sheets

TITLE:

Subject: Basic XD-1 Information: Scaling for the Fixed Point Computer

To: J. P. Porter

From: J. R. Tobey

Date: 15 April 1959

Approved: A. R. Chandler
A. R. Chandler

ABSTRACT

In the process of digital computation, three things must be known about each quantity used: 1. The digits in the numerical expression of the quantity, 2. The position of the radix, or digital point, with respect to the digits, and 3. The units in which the quantity is expressed.

Automatic digital computers, in general, keep no record of the units associated with the numbers they process, and the record of this information is therefore, the responsibility of the programmer.

Some digital computers are built to keep an automatic record of the position of the point in every number taking part in the computation. A computer that does this is called a floating-point computer as opposed to a fixed-point computer which keeps no record of the position of the point with respect to the digits. When programming a fixed point computer, the programmer must not only remember the units in which the quantities are expressed, he must also keep a record of the position of the point. The technique by which he does this is called scaling.

DISTRIBUTION LIST.
J. R. Tobey (25 copies)

1. This document has been prepared for internal use only. It has not been reviewed by Office of Security Review, Department of Defense, and therefore, is not intended for public release. Further dissemination or reproduction in whole or in part of the material within this document shall not be made without the express written approval of The MITRE Corporation (Technical Information Department).

2. The research reported in this document was supported by the Department of the Air Force under Air Force Contract AF-33 (600) 37882.

1. Introduction

In the process of digital computation, three things must be known about each quantity used: 1. the digits in the numerical expression of the quantity, 2. the position of the radix, or digital point, with respect to the digits, and 3. the units in which the quantity is expressed, such as inches, seconds miles per hour, or bits per time at bat. Unless all three of these are known, the quantity is incompletely expressed, and is meaningless.

Automatic digital computers, in general, keep no record of the units associated with the numbers they process, and the record of this information is therefore the responsibility of the programmer. For example, if the computer is programmed to compute a velocity, the programmer must know that the numbers expressing distance and time are expressed in miles and hours, respectively, in order to know that the result of the computation is a number expressing velocity in miles per hour.

Some digital computers are built to keep an automatic record of the position of the point in every number taking part in the computation. In such a computer, the point will automatically assume a position that depends on the result of the computation. A computer that does this is called a floating point computer. For reasons that will be evident below, a computer that keeps no record of the position of the point with respect to the digits may be called a fixed point computer. When programming a fixed point computer, the programmer must not only remember the units in which the quantities are expressed, he must also keep a record of the position of the point. The technique by which he does this is called scaling.

2. Scaling for a Desk Calculator

Because of the size of the keyboard, only a limited number of digits may be used to express a number in a desk calculator. For instance, a calculator with only five columns of keys can handle only five of the digits of the quantity: $23,962.17. In order to store this number in the keyboard, the five most significant digits of the number are punched in the five columns. If the dials are then cleared to zero and the add button is pressed, the digits will appear in the dials as 23962. The operator remembers that the units are dollars and that the decimal point is after the fifth digit. If he then wishes to add the quantity \$402.23 to the previous quantity, he will punch the number 00402 into the keyboard and press the add button. Again the decimal point is after the fifth digit, and the sum, 24364, appears in the dials, where it is understood to mean \$24,364.00. This is the correct value of the sum to the nearest dollar.

If, in another problem, the operator knows in advance that the quantities he will add and the sum he computes will never exceed \$999.99, he can decide to place the decimal point after the third digit and perform his computation with greater precision. He would then add the quantities \$170.18 and \$73.46 by clearing the dials, punching 17018 into the keyboard, pressing the add button, punching 07346, and pressing the add button again. The sum 24364 again will appear in the dials, but this time it is understood to mean \$243.64, since the decimal point is after the third digit.

At this point a notation will be introduced. In order to provide a reference point from which to count digit positions to the decimal point, it will be convenient to assume that there is a marker just to the left of the first keyboard column. This marker can be thought of as another decimal point, always to be associated with the numbers in the machine. Thus the quantity \$243.64 appears in the machine as, .24364, or a number 1/1000 as large as the real number that is meant. Using the letter d to represent the quantity in dollars and the symbol $\bar{d}$ (read d bar) to represent the corresponding number in the machine, the operator of the machine writes the equation $d=1000\ \bar{d}$, which means that the decimal point in d is three places to the right of the point in the corresponding number $\bar{d}$. If the real point in x, (that is, the point in the real number x), is understood to be 9 places to the right of the machine point in $\bar{x}$, he writes $x=1,000,000,000\ \bar{x}$, or $x = 10^9\ \bar{x}$. Since the machine point is understood to remain always at the same position, the machine may be called a fixed point computer.

It is conventional to assume, for scaling purposes, that the fixed point is to the left of the first digit, and that therefore the numbers in the machine are always less than one. It is this convention that gives rise to the belief on the part of some programmers that a fixed point computer can handle only fractions. To consider the numbers in the machine to be greater than one, it is only necessary to forget the convention and adopt a different one. For instance, the convention that memory register addresses are all whole numbers, with the point to the right of the last digit, is often convenient.

3. Scaling for AN/FSQ-7

The basic information unit it the AN/FSQ-7 Central Computer System is a 16-digit binary number called a half word. The first binary digit is reserved for the sign of the number, and is called the sign bit. The other 15 bits in the 16-bit half word are called magnitude bits and represent the magnitude of the actual number stored. If the number is negative, the sign bit is a 1 and the magnitude bits are in complement form.

For purposes of scaling, it is conventional to assume the fixed binary point to be just to the left of the first magnitude bit in the machine half word, or to the right of the sign bit. To load the binary number $x = 101110.1011$ into a register in the computer, the programmer must know the position of the real point in x with respect to the fixed point in $\bar{x}$. For this case, suppose the real point is to be nine places to the right of the fixed point. If $\bar{x}$ is loaded this way, it appears in the register as 0.000 101 110 101 100, which is a number much smaller than x.

Just as a decimal number is multiplied by the base of the decimal system, namely ten, by moving the decimal point one decimal place to the right, a binary number is multiplied by the base of the binary system, namely two, by moving the binary point one binary place to the right. For example, the binary number 0.000 101 110 101 100, multipled by two 9 times (or by 2^9), is equal to 101110,1011, and the equation, $x = 2^9 \bar{x}$, is true. Here x is the actual value of the quantity, $\bar{x}$ is the number in the register, and 2^9 is the scale factor that expresses the relationship between x and $\bar{x}$.

4. Determination of Scale Factors for Programming

The programmer may be required to write a program to compute the value of f, where $f = x(y+z)$. He is given the information that 25 values that the variable x has assumed are stored in Core Memory registers starting at 100, and in the 24 registers following, and that x is caled 2^7.

He writes on his worksheet:

$$4.1)\ x = 2^7\,\bar{x}.$$

Similarly, y is in the 25-register block starting at 200, scaled 2^3, and z is in the 25-register block at 300, scaled 2^5. He writes

$$4.2)\ y = 2^3\,\bar{y},$$

and

$$4.3)\ z = 2^5\,\bar{z}.$$

Since the largest possible value of $\bar{x}$ is 0.111 111 111 111 111, he knows that x itself cannot exceed 2^7 time this value, or 1111111.11111111, a binary number just less than 10000000, or 2^7.

He writes

$$/x/ < 2^7$$

which means "the positive magnitude of x is less than 2^7." Similarly, he knows that

$$/x/ < 2^3$$

and that

$$/z/ < 2^5.$$

He still must determine the scale factor for the sum, (y+z), since this number must appear in the accumulator some time during the computation. Since (y+z) will be largest when both y and z are at their maximum values and have the same sign, he writes:

$$/y+z/ < 2^3 + 2^5.$$

This number is less than 2^6, since $2^5 + 2^5 = 2^6$, and so he writes:

$$/y+z/ < 2^6$$

and decides that 2^6 is a safe scale factor for (y + z). Any larger scale factor would also be safe, but in order to keep as many significant bits as possible, the binary point must be as far to the left as is safe, so he writes:

$$4.4)\ (y+z) = 2^6\,(\overline{y+z}).$$

Finally, by similar methods, he determines that 2^{13} or anything larger will be a safe scaling factor for f. If the programmer's assignment is to store the results, f, in the 25 registers beginning at 400, scaled 2^{15}, he writes on the worksheet:

$$4.5)\ f = 2^{15}\,\bar{f},$$

and is now ready to use the scale factors.

Use of the Scale Facors in Programming

The first step in the computation is to compute the quantity $\overline{(y+z)}$.

The programmer rewrites the scaling equation 4.4): 5.1) $2^6 \overline{(y+z)} = y+z$

Substituting the scaled equivalent of y, 4.2) $y = 2^3 \bar{y}$, and of z, 4.3) $z = 2^5 \bar{z}$: 5.2) $2^6(y+z) = 2^3 \bar{y} + 2^5 \bar{z}$,

Dividing all terms by 2^6, 5.3) $\overline{(y+z)} = 2^{-3} \bar{y}+2^{-1} \bar{z}$,

he now has the equation that tells him how to compute $\overline{(y+z)}$. Equation 5.3) says that in order to compute $\overline{(y+z)}$, the computer

must obtain $\bar{y}$. CAD 200

Multiply it by 2^{-3}, by shifting it three places to the right: ASR 3

and store $2^{-3} \bar{y}$ for future use: FST 1000

Then it must obtain $\bar{z}$: CAD 300

and multiply it by 2^{-1}: ASR 1

Since $2^{-1} \bar{z}$ is now in the accumulator, $2^{-3} \bar{y}$ may be added: ADD 1000

and the first part of the program is finished.

In order to complete the program, the programmer writes the original equation: 5.4) $f = x \cdot (y + Z)$.

Again substituting the scaled equivalents;

4.5) $f = 2^{15} \bar{f}$, 4.1) $x = 2^7 \bar{x}$, and 4.4) $y+z = 2^6 \overline{(y+z)}$: 5.5) $2^{15}\bar{f} = 2^7\bar{x} \cdot 2^6\overline{(y+z)}$.

collecting factors: 5.6) $2^{15}\bar{f} = 2^{13}\bar{x} \cdot \overline{(y+z)}$

and dividing through by 2^{15}: 5.7) $\bar{f} = 2^{-2}\bar{x} \cdot \overline{(y+z)}$.

he can complete the computation. Equation 5.7) says "multiply $\overline{(y+z)}$ by $\bar{x}$, and multiply the result by 2^{-2}".

Accordingly, since $\overline{(y+z)}$ is now in the accumulator, multiply it by $\bar{x}$,	MUL	100
shift the product 2 places to the right,	DSR	2
round off the result,	SLR	0
and store the answer, $\bar{f}$, into 400.	FST	400

After indexing, the completed program is as follows:

```
 0          1  XIN       30
 1          1  CAD      200
 2             ASR        3
 3             FST     1000
 4          1  CAD      300
 5             ASR        1
 6             ADD     1000
 7          1  MUL      100
10             DSR        2
11             SLR        0
12          1  FST      400
13          1  BPXØ1      1
14             HLT        0
```

Thus the scaling equations have not only provided a convenient way to record scale factors, they have also made it easy for the programmer to decide in which direction and how far to shift, to obtain the desired results

6. Choice of Scale Factors according to Upper Limits

In the example above, the programmer was given the scale factors for almost all of the quantities to be handled in the computation. In the following example, the scale factors are not given, and the programmer is free to choose them. Here it can be assumed that he has control over the loading of the data, and can store the bits into the registers in any bit position he wishes. In order to choose the best scale factors, he must know the upper limits, (and for divisors, the lower limits), on each factor taking part in the computation.

Suppose the following computation is to be performed: $g = \frac{(u-v)}{w} + x$, and the following limits are given $|u| < 49$, $|v| < 21$, $|x| < 18$, and $2 \leq |w| < 158$. (This last statement means "The magnitude of w is equal to or greater than 2 and less than 158"). Since u can assume its largest positive value at the same time that v is at its largest negative value, the upper limit on $|u - v|$ is the sum of the separate upper limits on u and v. Hence, $|u - v| < 49 + 21$ or 70. Since 70 is less than 2^7 or 128, the programmer chooses the scale factor 2^7 for (u - v). 6.1) $(u - v) = 2^7 \overline{(u - v)}$

This is the best scale factor to choose, since it is the smallest safe value.

To save machine time, he chooses 2^7 for the scale factor for each of the two quantities u and v. 6.2) $u = 2^7 \bar{u}$

6.3) $v = 2^7 \bar{v}$

This is safe, since $|u|$ and $|v|$ are each less than 2^7. He loses precision this way, since the binary points in these quantities are too far to the right, but if he chose smaller scale factors, he would have to shift $\bar{u}$ and $\bar{v}$ to the right before subtracting, losing the same precision as before, and wasting computer time in shifting as well.

Since $|w| < 158$, he scales 6.4) $w = 2^8 \bar{w}$

He must now choose a scaling factor for the final result, since, as shown above,

$\frac{u - v}{w}$ and the quantity x should have the same scaling factor as g, their sum.

To find the upper limit on g, the programmer assumes that all the variables conspire to make g large. In this case, x will be nearly 18, (u - v) will be nearly 70, and to make g large, the divisor, w, will assume its smallest value, 2, so that

$\frac{u - v}{w}$ will be as large as possible. Altogeter, the limits combine to make the upper limit on g equal to

$\frac{70}{2} + 18$, or 53.

Hence the scale factor for g is 2^6: 6.5) $g = 2^6 \bar{g}$, and accordingly he chooses: 6.6) $x = 2^{6}\,\bar{x}$,

and for the quotient: 6.7) $\frac{u-v}{w} = 2^{6}\left(\overline{\frac{u-v}{w}}\right)$

Rewriting equation 6.1): 6.8) $2^{7}(\overline{u-v}) = (u-v)$

Substituting for u and v using equations 6.2) and 6.3): 6.9) $2^{7}\;\overline{(u-v)} = 2^{7}\bar{u} - 2^{7}\bar{v}.$

Cancelling out 2^{7}, 6.10) $\overline{(u-v)} = \bar{u} - \bar{v}$

From 6.10) the first steps in the program may be written: (where $L(\bar{u})$ means the memory location of $\bar{u}$). and no preliminary shifting is necessary.

```
CAD     L(u)
SUB     L(v)
```

Now, rewriting equation 6.7): 6.11) $2^{6}\left(\overline{\frac{u-v}{w}}\right) = \frac{u-v}{w}$

Substituting in the right side of equation 6.11), using 6.1 and 6.4): 6.12) $2^{6}\left(\overline{\frac{u-v}{w}}\right) = \frac{2^{7}\;\overline{u-v}}{2^{8}\,\bar{w}}$

Dividing through by 2^{6} and collecting the power of 2: 6.13) $\left(\overline{\frac{u-v}{w}}\right) = \frac{2^{-7}\;\overline{(u-v)}}{\bar{w}}$

Since $\overline{(u-v)}$ is now in the accumulator, the progmammer, using 6.13), may now write:

```
DSR     7
```

to multiply by 2^{-7}. Then, he writes:

```
DVD     L(w̄)
```

and

```
SLR     17
```

The last step, SLR 17, has nothing to do with scaling. After a division, the first magnitude bit of the resulting quotient, unlike the results of MUL, ADD, or SUB, is in the B register sign position. The "fixed point" must still be just to the left of the first magnitude bit, and hence must be thought of as being at the left end of the B register. In order to restore the fixed point and the magnitude bits to their normal positions, the instruction SLR 17 is given. Since the "fixed point" and the magnitude bits move together, the scale factor for the quotient is not affected.

The quotient, $\left(\overline{\frac{u - v}{w}}\right)$, is now in the accumulator. Rewriting equation 6.5):

$$2^6 \bar{g} = g, \tag{6.14}$$

and substituting from the formula for g:

$$2^6 \bar{g} = \frac{u - v}{w} + x, \tag{6.15}$$

Substituting from equations 6.6) and 6.7):

$$2^6 \bar{g} = 2^6\left(\overline{\frac{u - v}{w}}\right) + 2^6 \bar{x} . \tag{6.16}$$

Cancelling out 2^6:

$$\bar{g} = \left(\overline{\frac{u - v}{w}}\right) + \bar{x} . \tag{6.17}$$

Accordingly, without further shifting, $\bar{x}$ is added: ADD L($\bar{x}$)

The result, $\bar{g}$, is stored: FST L($\bar{g}$)

and the program is finished. HLT 0

Below is the complete program.

```
CAD     L(ū)
SUB     L(v̄)
DSR      7
DVD     L(w̄)
SLR     17
ADD     L(x̄)
FST     L(ḡ)
HLT      0
```

The program contains a minimum number of shift instructions because with the aid of the scaling technique, the proper scale factors for maximum precision and minimum shifting were chosen. This is the most fruitful application of the technique, but even where the choice of scale factors is not left to the programmer, as in the first problem, the technique is a useful aid in writing programs.

J. R. Tobey
James R. Tobey

JRT/ht/ar

Technical Memorandum TM-15 #7

Sheet 1 of 4 Sheets

[redacted]

Contract No. AF 33(600) 39852

SUBJECT: BASIC XD-1 INFORMATION:
SAGE 1401 LOG PROGRAM (LOCOT)

Project 910

TO: Distribution List

FROM: T. Connors, J. J. Robinson

DEPT.: D-19 (Computer Applications)

DATE: 20 June 1962

APPROVED: J. H. Burrows

THE MITRE CORPORATION
Bedford, Massachusetts

ISSUED AT: Bedford

ABSTRACT

The 1401 Program LOCOT, which logs a COSEAL-format SAGE-mode tape, has been released for use. This program essentially duplicates the LOG function of COSEAL; it is good to use because logging tapes on the relatively inexpensive 1401 can save relatively expensive XD-1 time. Copies of the LOCOT Program deck are available at the 1401 in Building F.

TM-15 #7

SAGE 1401 LOG PROGRAM

I. **Purpose:** LOCOT - (LOg a COseal Tape) is a Sage 14Ø1 Program designed to log binary tapes produced by the Coseal System. This program logs the identification and other pertinent information of each tape record. Ten (10) types of binary records may be found on a Coseal Master; a listing and sample of the output for each is contained in Section IV of this memorandum. The tape record formats are described in detail in FN-6179 COSEAL UTILITY SYSTEM FOR THE AN/FSQ-7 COMPUTER, System Development Corporation.

II. **Restrictions:**

A 14Ø1 System modified to read and write 728 mode (Sage mode) tapes, 4 K, one tape drive, advanced programming, special features (High-Low-Eq Compare, Multiply/Divide, Print Storage, Column Binary) and 132 character print line, are necessary for the operation of this program.

III. **Description of Records:**

Each record produced by Coseal contains the record identification bits in bit positions 28-31 of word one. In addition, some records contain further descriptive material. LOCOT distinguishes records from one another by the record identification bits, and prints the record type and whatever descriptive material it can. The following paragraphs describe each of the ten records. A sample print follows. For records whose identification bits are not one of the ten legal types, LOCOT prints the comment "Unknown Record Type" (except combined tape records, see below).

1. Storage Identification Record	Ident Bits 28 31 (ØØØ1)	Record Length (3 words)

The Storage Identification Record precedes one or more storage records and contains identifications of them. This record and the storage record or records it identifies are called an identification group.

2. Core Storage Record	Ident Bits 28 31 (Ø11Ø)	Record Length (3 words + number of data words)

The Core Storage Record contains actual binary data, the number of data words, and the starting core location of the data.

3. Core Operate Record

Ident Bits 28 31 (Ø111)
Record Length (3 words)

The Core Operate Record follows one or more storage records for drum or core containing a program. This record contains the core address specified in the End Card of the program when it was assembled.

4. Tag Table Record

Ident Bits 28 31 (1Ø11)
Record Length (8 words + the number of DDL & non-DDL tags)

The Tag Table Record information printed by LOCOT consists only of the Ident and number of Digit-Digit-Letter Tags. For a more complete description of what is contained in a particular tag table record, a tape dump is suggested.

5. Drum Storage Record

Ident Bits 28 31 (ØØ11)
Record Length (3 words + number of data words)

The Drum Storage Record contains actual binary data, the number of data words, the drum field and starting drum location of the data.

6. File Identification Record

Ident Bits 28 31 (ØØØ1)
Record Length (3 words)

The File Identification Record is the first record of a file. A "file" is a programming system (e.g. DCA, Coseal). A file is structually made up of identification groups, and is usually ended by an End Ident Record.

7. End Identification Record

Ident Bits 28 29 (Ø1Ø1)
Record Length (3 words)

The End Identification Record is the last record of a file and is normally followed by an end-of-file mark.

8. Combined Tape Identification Record

Ident Bits 28 31 (1Ø1Ø)
Record Length (3 words)

A Combined Tape Record is a record of not more than 3900 words produced as output from a Coseal Assembly. Each set of combined tape records associated with a particular Coseal IDT card is preceded by a combined tape ident record.

The Combined Tape Ident Record contains the Ident and Mod of the program which follows in the next record(s). It also contains the compool ident of the compool used. LOCOT prints information from the combined tape ident record and passes over the following combined tape records until it has passed a record containing the indicator for "last record of a combined tape program". This indicator is a 1 Bit in the 31st Bit position of the first word of that record. After passing the record containing that indicator, LOCOT resumes logging records of any type.

	Ident Bits	Record Length
9. Environment Simulation Record	28 31 (1ØØ1)	(3 words & number of data words)

The Environment Simulation Record (which is produced by the Environment Simulation Program) is designed to generate data which can be used to check out the master program or to evaluate the transfer function of any program.

10. Correction Inventory Record

The Correction Inventory Records are produced by DCA tape load and are located between the End Identification Record and the End-Of-File mark of a DCA master tape. At present this record has not been generated and therefore the section of the program which handles this hasn't been checked out.

IV. Sample Print of Each Record Type (variable information is underlined)

STORAGE IDENTIFICATION RECORD	IDENT PLT AH COMPOOL R19
CORE STORAGE RECORD	0153 WORDS START CORE 000006
CORE OPERATE RECORD	START CORE 204000
TAG TABLE RECORD	0019 DDL
FILE IDENTIFICATION RECORD	IDENT COSUT COMPOOL Ø1B
DRUM STORAGE RECORD	0460 WORDS START DRUM Ø2 2Ø5Ø
END IDENTIFICATION RECORD	IDENT PILOT COMPOOL Ø1B
ENVIRONMENT SIMULATION RECORD	1Ø44 WORDS
COMBINED TAPE PROGRAM	IDENT PLT MOD AH COMPOOL R19 RECORDS 0004

V. Operating Instructions

When starting, set tapes to load point, Load LOCOT and press start. LOCOT will check tape records for parities. If a record contains a tape parity LOCOT will try ten (10) times before printing out a comment. (See number 4 of this section.) When an End-Of-Tape has been read LOCOT will print out (END-OF-FILE). (PRESS START FOR NEXT FILE.)

1. Deck Make-up

   LOCOT Program

2. Tapes

   Select 3
   Tape to be logged

3. Sense Switches

   None

4. Error Printout

   Monitor the printer for the following printouts:

Comment	Recovery and Reason
Redundant Record Press. Start to print it.	Record read was redundant. Pressing start button will print out appropriate areas and continue on to next record.
Unknown Record Type	When this comment is printed a record read from tape was unable to be identified by the program. The printer will not stop.

5 Stops

1265	Redundant Record. Press start to print it.
1311	An End-Of-File on Tape 3. Press Start for next file.
2147	An indication of machine trouble, please take a core dump. Pressing start will continue program ignoring record just read.

TC:JJR/nep
Attachment: Distribution
List

T Connors
T. Connors
J. J. Robinson
J. J. Robinson

DISTRIBUTION LIST

Department D-1

C. A. Zraket

Department D-11

W. Amory (4)

Department D-12

L. V. Giusti (2)
E. P. Gill

Department D-16

H. J. Kirshner
D. L. Bailey
P. R. Simons
J. Morey
R. Larson (ESD)
J. Crouse (16)
D-16 Data File (M.Mulry)

Department D-17

Dr. Stanely Frank
S. Rosenfeld

Department D-19

ATC Programmers (33)
J. Burrows
J. Ishihara
V. Wilkie
D-19 ATC Data File (10)
All D-19 Sub-Department Heads

Technical Memorandum TM-15 #8

Sheet 1 of 4 Sheets

Contract No. AF-19 (628) 2390

SUBJECT: BASIC XD-1 INFORMATION - SAGE MULTIPLE UTILITY PROGRAM (SMU-3A) (U) Project 220

TO: Distribution List

FROM: XD-1 Facility Office

DEPT.: D-19

DATE: 5 April 1963

APPROVED: J H Burrows
J. H. Burrows

THE MITRE CORPORATION
Bedford, Massachusetts

ISSUED AT: Bedford, Massachusetts

This Technical Memorandum is prepared for Corporation use and for controlled external distribution. Reproduction or further dissemination is not authorized. It has not been reviewed by Office of Security Review, Department of Defense, and therefore is not for public release.

ABSTRACT

The SAGE Multiple Utility Program (SMU-3A) is a 1401 program consisting of seven utility routines which make possible off-line processing of XD-1 (or AN/FSQ-7) computer inputs and outputs.

Brief descriptions of the SMU program functions and operating procedures are contained in this document.

1.0 INTRODUCTION

The SAGE Multiple Utility Program (SMU-3A) is a modified version of the SAGE Multiple Utility Program described in TM-77#5,S1. Differences between Version 3A and previous versions are:

1. The original PEST-coded instructions have been converted to 1401 Autocoder language.

2. A "batched record" option has been added to the Card-to-Tape function.

3. The Card-to-Card and Card-to-Printer functions have been removed. (One-card 1401 programs are available for these functions, if required.)

For the convenience of XD-1 users a complete description of the SAGE Multiple Utility Program (SMU-3A) is contained in this document.

2.0 GENERAL DESCRIPTION

The SAGE Multiple Utility Program - Version 3A (SMU) is a 1401 program consisting of seven utility routines, which make possible off-line processing of XD-1 (or AN/FSQ-7) Computer inputs and outputs. These routines include: (1) Tape-to-Printer, (2) Card-to-Tape, including COMPASS feature and BATCH feature, (3) Tape-to-Card, (4) Tape-to-Tape, (5) Space Back, (6) Space Forward, and (7) Stop. Particular functions to be performed are controlled by sense switch settings and/or control cards.

3.0 PROGRAM FUNCTIONS

3.1 Tape-to-Printer

Prints a BCD tape with a maximum record length of 1794 characters or 299 SAGE words. Through the use of sense switches, the user has the option of selecting Program Control, Single Spacing or Double Spacing.

All switches are off for Program Control. Single spacing is forced by using sense switch "D"; double spacing is obtained by use of sense switch "E". Both single and double spacing (Sense Switches "D" or "E") override Program Control.

On reading an end-of-file mark from the tape unit selected, the comment "END OF FILE-PRESS START FOR NEXT FILE." is printed. When the START button is pressed, the next file will be printed.

3.2 Card-to-Tape

Prepares a BCD SAGE mode tape of 13 words (one card image, Columns 17-80) per record. The first 16 columns of every card are ignored because they are not read by the XD-1 Computer.

A SAGE format end-of-file mark is written on the tape when the last card is read. The tape unit used for writing is select #1. Sense switch "B" controls this "prestore" function. The card-to-tape function includes two optional features: COMPASS and BATCH.

3.2.1 COMPASS Feature

Use of the COMPASS feature causes BCD information in columns 8-13 of the cards being prestored to be moved to columns 72-77. A COMPASS control card is required and must be placed in front of the deck to be prestored. Control card format is shown below:

1-7	8-79	80
COMPASS	╳	*

The COMPASS control card needs only to be read in once for subsequent COMPASS prestoring. The SMU deck must be read in again to return control back to normal prestore.

3.2.2 BATCH Feature

The BATCH feature permits two to a maximum of nineteen card images (Columns 17-80) to be batched on one tape record, resulting in faster assembly input to the XD-1.

A BATCH control card must be placed in front of the deck to be prestored. Control card format is shown below:

1-5	6-15	16-17	18-79	80
BATCH	╳	(No. of cards to be Batched)	╳	*

The BATCH feature can only be used to prestore special assembly input decks: Assemble Compool, Assemble Initial Conditions, Assemble Sequence Parameters, Assemble MORØ and Assemble Geography. Input to put, the Storage Allocation Program, may also be batched. BATCH cannot be used to prestore XD-1 program assembly input decks.

The BATCH control card needs only to be read in once for subsequent BATCH prestoring. The SMU deck must be read in again to return control to normal prestore.

3.3 Tape-to-Card

Punches out a deck of cards from a variable length tape record that does not exceed 1794 characters or 299 SAGE words. The tape record must be in BCD format and written in the SAGE mode.

All punching begins in column 17, in order to be compatible with the SAGE (XD-1) computer. If a tape record is greater than 78 characters or 13 SAGE words, every 78th character will be dropped.

All SAGE words consist of six (6) characters on tape, but only five (5) characters on an IBM card. The first character of every word contains the parity bit. Thus, in transferring a tape word for punching, only characters 2-6 of a word are moved to the punch image.

The tape-to-card function is controlled by sense switch "C". An end-of-file mark on the tape terminates this operation.

3.4 Tape-to-Tape

Duplicates a tape of variable length records that do not exceed 1794 characters or 299 SAGE words. The tape-to-tape operation will read BCD or Binary records from the input tape on select #3 and write the same record on a blank tape on select #1.

An end-of-file mark on select #3 will be written on select #1. Sense switches "B" and "C" control this operation.

3.5 Space Back

Allows the back-spacing of tape files. The START button is pushed for each file to be back-spaced.

Sense switch "F" controls this operation.

3.6 Space Forward

Provides the facility to space the tape forward one file at a time. The START button is pushed for each file to be skipped.

Sense switch "G" controls this operation.

3.7 Stop

Terminates any of the foregoing functions through the use of sense switches "F" and "G".

4.0 OPERATING INSTRUCTIONS

4.1 Sense Switches and Tape Set-Up

Sense switch settings and tape set-ups for each SMU function are summarized below:

Function	Sense Switches	Input Tapes	Output Tapes
Tape to Printer			
Program Control	No switches	TD#3	
Single Space	D	TD#3	
Double Space	E	TD#3	
Card to Tape			
Normal Prestore	B		TD#1
COMPASS Feature	B		TD#1
BATCH Feature	B		TD#1
Tape to Card	C	TD#3	
Tape to Tape	B, C	TD#3	TD#1
Space Back	F		
Space Forward	G		
Stop	F, G		

4.2 Error Printouts

The following error printouts are logged on the line printer:

a. "CONTROL CARD IN ERROR" (Either the BATCH or the COMPASS control card has been punched improperly.) Check and correct control card. Reload deck to be prestored using corrected control card.

b. "REDUNDANT RECORD" (A redundancy has occurred in reading a tape record.) Press START to continue.

4.3 Special Points

When performing tape functions the tape units must be set to low density for compatibility with the Model 728 Tape Units used with the XD-1 Computer.

Once SMU has been read in and one function has been performed, resetting the sense switches and pushing the START button will again operate the program. SMU does not have to be read in each time a different utility function is desired.

/mg
Attachment: Distribution List

R. A. Holt
R. A. Holt

S. Gross
Approved by: S. Gross

DISTRIBUTION LIST

D-1

D. R. Brown
R. R. Everett
C. A. Zraket

D-11

W. Amory
R. C. Davis
E. P. Petersen
J. V. Sullivan

D-12

J. P. Hanks
W. R. Hayne
R. N. Mechlin
A. Werlin

D-16

M. F. Coffin (7)
M. Hazle
W. R. Kreiser
J. P. Locher, III
A. Melly
W. R. Milliken
O. L. Morgenstern
A. D. McKersie
L. E. Sanford
P. R. Simons
G. K. Sioras
J. A. Varela
Data File (2)
G. T. Sabol
G. Newall

D-17

Dr. Stanley Frank
S. Rosenfeld

D-19

W. Bernikowich
F. Cataldo
T. L. Connors
C. J. DeSalvo
R. C. Fish
M. T. Gattozzi
L. V. Gera (7)
M. Gerrin (10)
S. Gross
R. A. Holt
J. A. Ishihara
D. L. James
O. R. Kinney
J. P. Kintz (50)
M. Lancelotta (30)
D. G. Miller
M. J. Peskin
M. R. Pew
C. E. Putnam
E. R. Robertson
J. A. Singer
D. Sweetser
C. L. Tausch
J. A. Terrasi
L. D. Vass
J. Burrows
V. Wilkie

C-33

F. P. Hazel, Jr.
D. Karkota

C-37

O. O. Sulkala

SDC

J. Carey
W. F. Polak

PHILCO

J. McPhail