IB59-1N

MOBIDIC PROGRAMMING AIDS INSTRUCTION MANUAL

VOLUME I

SYLVANIA ELECTRONIC SYSTEMS
Government Systems Management
for GENERAL TELEPHONE & ELECTRONICS

SALERNO

MOBIDIC PROGRAMMING AIDS
INSTRUCTION MANUAL

VOLUME I
PROGRAM WRITE-UPS

Signal Corps
Technical Requirements
SCL 1866

Contract No. DA-36-039-sc-78111

Submitted to

U.S. Army
Signal Research and Development Laboratories
Fort Monmouth, New Jersey

SYLVANIA ELECTRONIC SYSTEMS
A Division of Sylvania Electric Products Inc.
DATA SYSTEMS OPERATIONS
189 B Street - Needham Heights 94, Massachusetts

LIST OF VOLUMES

Volume	
I	PROGRAM WRITE-UPS
II	PROGRAM FLOW CHARTS
III	PROGRAM LISTINGS
IV	PROGRAM LISTINGS
V	REAL-TIME SYSTEM
VI	MOBIDIC B PROGRAMS AND ADDENDA (To Be Supplied)

VOLUME I

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

TABLE OF CONTENTS (Cont.)

LIST OF TABLES

CONTENTS OF OTHER VOLUMES

Directly following this page are the Contents Listings of:

VOLUME II - PROGRAM FLOW CHARTS

VOLUME III - PROGRAM LISTINGS

VOLUME IV - PROGRAM LISTINGS

VOLUME V - REAL-TIME SYSTEM

TABLE OF CONTENTS

VOLUME II FLOW DIAGRAMS

TABLE OF CONTENTS (Cont.)

VOLUME II FLOW DIAGRAMS

TABLE OF CONTENTS

VOLUME III PROGRAM LISTINGS

TABLE OF CONTENTS (Cont.)

VOLUME III PROGRAM LISTINGS

TABLE OF CONTENTS

VOLUME IV PROGRAM LISTINGS

TABLE OF CONTENTS

VOLUME V MOBIDIC REAL TIME SYSTEM

LIST OF ILLUSTRATIONS

VOLUME V MOBIDIC REAL TIME SYSTEM

VOLUME I

PROGRAM WRITE-UPS

PREFACE

This instruction manual has been written as an aid to Signal Corps programming personnel in performing programming duties with the MOBIDIC computer. Four volumes are contained in the manual. Along with introductory material, Volume I contains the necessary program write-ups to inform the reader of the purpose of a given program, the approach taken in its formulation and the special steps required of programming personnel in undertaking its use.

Embodied in Volume II are the flow diagrams for the various routines, subroutines, etc., whose write-ups are contained in Volume I. Therefore, while reading a given program write-up in Volume I the reader may reference the appropriate flow diagram for an aid in understanding the mechanics of the program.

Volumes III and IV contain the hard copy of the coded programs explained in Volume I.

The governing attitude in writing this manual has been one that presupposes a fundamental knowledge of programming on the part of the reader. It is not a training manual. Therefore, a reference to the PROGRAMMING MANUAL FOR THE MOBIDIC COMPUTER is suggested for a more basic understanding of MOBIDIC programming features. Information relating to the MOBIDIC A Input-Output System may be found in the document titled "MOBIDIC A Input-Output System and Micro Flow Charts".

SECTION I

INTRODUCTION

1.1 PURPOSE OF MINIMAL PROGRAMMING AIDS

The purpose of the Minimal Programming Aids (MPA) is to meet the initial, minimum data-processing requirements of MOBIDIC A to facilitate computer operation in the areas of Combat Support Data Processing, Combat Control Data Processing, and Combat Computations.

The Minimal Programming Aids consist of a library of utility programs. They may be broken down into the following five groups:

1. MOBIDIC Assembly Program (MAP)
2. Input-Output Routines
3. Malfunction Control Routines
4. Generalized Data Handling Routines
5. Mathematical Subroutines

1.2 MOBIDIC A COMPUTER

The MOBIDIC A Computer is a large-scale, general-purpose, mobile digital computer developed by Sylvania Electronic Systems for the U.S. Army Signal Research and Development Laboratories. The machine operates in the parallel mode with an internally-stored program. Logical organization within the machine is in the binary system. The machine is equipped with high-speed random access memories and a complete family of input-output devices. The entire equipment is mounted in a trailer van to enable operation in the field, but its advanced concepts of reliability and flexibility are equally advantageous for fixed plant or strategic installation.

Because of its operational and design characteristics, MOBIDIC offers many advantages over other existing commercial and military computers in most

data processing applications. Features of particular advantage in MOBIDIC are:

Computation Speed

The basic clock rate for MOBIDIC is one megacycle. Computation times for the four basic fixed point arithmetic operations, including memory, are as follows:

Add	16 μsec
Subtract	16 μsec
Multiply	86 μsec
Divide	88 μsec

The great majority of the instructions in the MOBIDIC repertoire are performed in 16 μseconds.

Single Address

MOBIDIC is designed primarily as a single-address machine. The standard instruction word, however, in addition to a full 15-bit address portion, also contains a 12-bit portion referred to as the β portion. The different uses of the β portion make a number of the MOBIDIC instructions resemble those of a two-address machine. This arrangement combines the afficiency of a single-address machine with some of the versatility of a double-address machine.

Number System

MOBIDIC is built to operate internally in the binary number system. This is due to the greater efficiency, economy and simplicity of the binary system over coded decimal or alphanumeric systems for computational purposes. However, there are orders which make it possible to handle alphanumeric information with reasonable efficiency.

Expandability

In order to adapt MOBIDIC to possible future requirements of increased scope, the computer has been made logically expandable in several ways.

Addressable Registers

In order to maintain a high degree of versatility, many internal MOBIDIC registers have been made addressable. Those registers may be addressed by the program and used for storage of data as if they were locations in the high-speed memory.

Simultaneous Read-Write-Compute

The in-out converters are sub-processing units which control all of the in-out devices (other than real-time registers). Use of the in-out converters enables computation in the central machine to proceed independently of in-out operation. Either converter, by itself, can control any of the in-out devices provided. The number of devices which can operate simultaneously is equal to the number of converters. A maximum of two additional input-output converters may be connected to the MOBIDIC if desired.

1.2.1 General Organization

The MOBIDIC System is composed of the following machine units:

Central Computer
Memory Units
In-Out Converters
Flexowriter
Paper Tape Readers
Paper Tape Punches
Magnetic Tape Units
Communications Equipment
Power Supplies and Air Conditioning Units

In addition to these units, other devices such as card equipment, high-speed printers, and displays are available.

TABLE 1-1. MOBIDIC COMPUTER CHARACTERISTICS

Mode of Operation	Basic	Expanded
Word Structure	Binary-Fixed Point Fractional-Magnitude and Sign	Binary-Fixed Point Fractional-Magnitude and Sign
Word Length (including parity and sign bits)	38 Bits	50 Bits
Core Memory (number and capacity)	Two-4,096 words each. Total 8,192 words.	Seven-4,096 words each. Total 28,672 words.
Magnetic Tape Units	2	63*
Simultaneous Tapes	2	4**
Other In-Out Devices		
Paper Tape Readers		
8-Channel	1	
5-Channel	1	
Paper Tape Punches		
8-Channel	1	
5-Channel	1	
Flexowriter	1	
System Features		
In-Out Converters	2	4
Index Registers	4	7
Real-Time Input Registers	1	No Practical Limit
Real-Time Output Registers	1	No Practical Limit

*The addressing format of MOBIDIC allows a total of up to 63 In-Out Devices.

**The number of In-Out Devices that can be operated simultaneously depends upon the number of In-Out Converters incorporated in the system. For example: If there are 2 In-Out Converters in the system, then 2 and only 2 In-Out Devices can be operated at the same time. If, however, the number of In-Out Converters were increased to 3, then 3 In-Out Devices could be operated simultaneously.

TABLE 1-1. MOBIDIC COMPUTER CHARACTERISTICS (Cont.)

Mode of Operation	Basic	Expanded
Operating Speeds (including Memory Access)		
Addition	16 μsec	
Subtraction	16 μsec	
Multiplication	86 μsec	
Division	88 μsec	

SECTION II

FACTUAL DATA

2.1 COMPUTER APPLICATIONS

The military applications of the computer are grouped into four major categories: logistics, scientific computation, battle-area surveillance, and real-time operations. Emphasis is placed on speed, input-output capability, versatility, and reliability.

The computer is a parallel-binary machine with a word lingth of 38 bits including parity. Operation is synchronous; arithmetic is performed using the fixed-point, magnitude, and sign convention. The clock rate is one megacycle per second.

2.2 INSTRUCTIONS AND TIMING

Speeds for the four basic (fixed point) arithmetic instructions are 16 microseconds for Add and Subtract, 86 microseconds for Multiply, and 88 microseconds for Divide. Sixteen microseconds is the minimum time for a basic instruction and 88 microseconds is the maximum. Currently there are 52 instructions mechanized in MOBIDIC "A".

MOBIDIC is provided with high-speed magnetic core memories in modules of 4096 words each. Repetitive random-access time is 8 microseconds. A total of 7 memories can be used, giving a maximum capacity of 28,672 words.

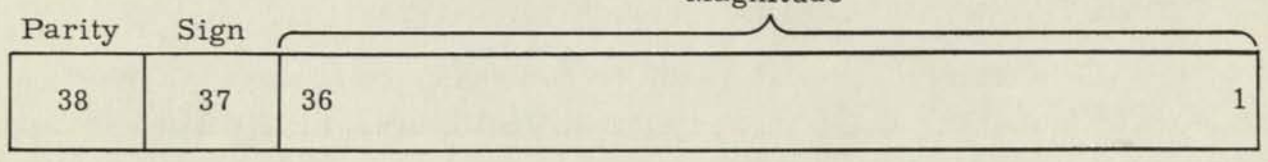
The computer can be equipped with high-speed magnetic tapes, high speed paper-tape reader, paper-tape punches, Flexowriter, card reader and punch, high-speed line printer, and a disc memory that is capable of storing 50×10^6 bits. It also contains real-time registers which permit data to be transferred to and from external devices or systems in real-time.

2.3 WORD FORMAT

The word format used in MOBIDIC is as follows:

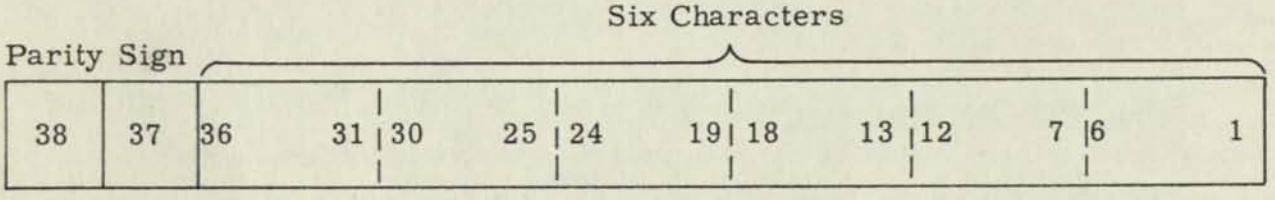
2.3.1 Binary Data

From left to right in the format for binary numerical data, bits 38 and 37 are the parity and sign bits, respectively. Bits 36 through 1 specify magnitude.



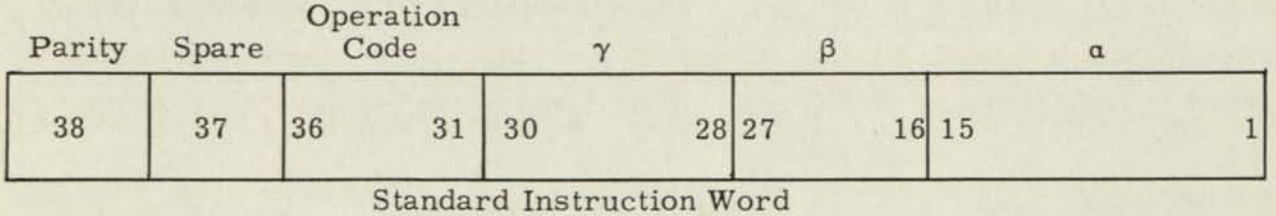
2.3.2 Alphanumeric Data

The format is the same as that for binary data, except that the sign bit is not significant and six alphanumeric characters occupy thelocation previously used for magnitude.



2.3.3 Standard Instruction

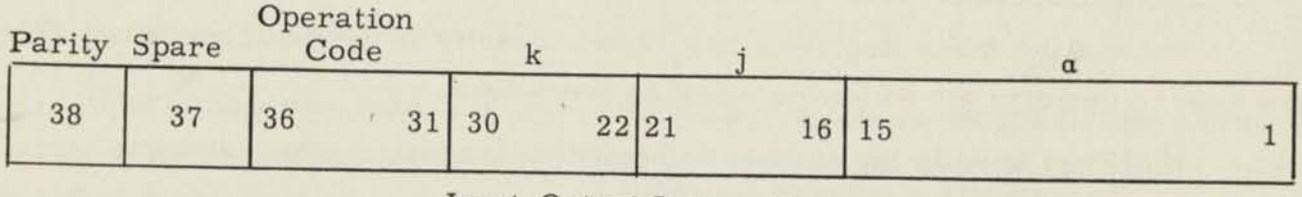
In this format bit 38 is parity, as before. Bit 37 is a spare. The next six bits, 31 through 36, constitute the operation code, which specifies the operation to be performed. Bits 28 through 30, the gamma bits, are used for indexing. Bits 16 through 27, the beta bits, are loaded into an index register depending on the type of instruction or added to the contents of an index register. They may also be used in conjunction with the gamma bits to specify a second address. The alpha bits, 1 through 15, constitute the address portion of an instruction.



Standard Instruction Word

2.3.4 Input-Output Instructions

The format is identical with that of standard instructions except for the assignments made to bits 16 through 30. Bits 22 through 30 specify the number of words or blocks to be processed; bits 16 through 21 specify the particular in-out device addressed.



Input-Output Instruction

2.4 TRANSFER BUS AND REGISTERS

2.4.1 Transfer Bus

All major elements of the computer are tied to the main transfer bus, through which all information flows within the computer system. The transfer bus greatly facilitates the flexibility and expandability of the machine, since additional components such as memories, display devices, and in-out converters can be added with little or no modification to the central computer.

2.4.2 Memory Address Register and Memory In-Out Register

The memory address register holds the address of the location in memory from or into which information is being transferred, and the Memory In-Out Register holds the data that either have been read out of memory or are about to be written into memory. These are not program addressable.

2.4.3 Program Counter (PC) and Program Counter Store (PCS)

The Program Counter is a 15-bit register that holds the address of the next instruction. It is arranged to step automatically through all of the memory cells in sequence.

The Program Counter Store (PCS) provides temporary storage for the contents of the Program Counter when control is transferred to a subroutine with

a "Transfer and Load PCS" instruction. At the end of the subroutine, a "Transfer to PC" instruction can be used to return control automatically to the main program.

2.4.4 Index Registers

Each of the Index Registers has 12-bit capacity in the basic machine. They are used to perform the following indexing operations:

1. They specify the number to be added to a programmed address to generate a relative address.
2. They specify the number of additional times that a particular programming routine is to be repeated.

There are four Index Registers in the basic machine. This number can be increased to a maximum of seven, and the capacity of an index register can be increased to a maximum of 15 bits. In addition to the four functioning index registers, MOBIDIC is equipped with a fictitious index register whose function is to store a permanent zero.

2.4.5 Arithmetic Unit

This unit is composed of three special storage registers (Accumulator or A-register, B-register, and Q-register) and the control circuitry necessary to perform the arithmetic and logic operations specified by the instruction. They are all individually addressable.

2.4.5.1 A-Register

The A-register, or Accumulator, is the main register in the arithmetic unit. Most arithmetic and comparison operations involve the contents of the Accumulator. The Accumulator can be reset to zero before entering new information, or else new information can be added to the contents of the Accumulator without prior resetting. It also holds the sum, difference, product, and remainser after addition, subtraction, multiplication, and division, respectively. In addition, it contains the augend or minuend at the start of an addition or subtraction, and the multiplicand before multiplication.

2.4.5.2 B-Register

The B-Register is used to hold one of the operands of an arithmetic operation. It stores the addend, subtrahend, multiplier, and divisor for addition, subtraction, multiplication and division, respectively.

2.4.5.3 Q-Register

The Q-Register holds the low-order bits of a double-length dividend (that is, the 36 least significant bits of a 72-bit dividend) at the start of a division. At the end of multiplication and division, it contains the low-order bits of a double-length product and the quotient, respectively. The Q-register is also used during the "mask" instruction*. Finally, the Q-register can be joined with the A-register to perform "shift" and "cycle" operations. The Q-register will contain the low-order bits in each case.

2.4.6 Converter Instruction Register

When an In-Out instruction is sent to the selected converter it is placed in the converter instruction register: an addressable register containing the current status of the instruction being processed.

2.4.6.1 Address Counter

Bits 1-15 (a) of the Converter Instruction Register specify the memory address in the central processor to or from which the data is being transferred. This is the address counter and will be incremented by one during each memory access. It will not be incremented if it contains a register address.

*The "mask" instruction (MSK) compares each bit in the A-register with its corresponding bit in the Q-register. If the bit in the Q-register is 0, nothing is done; if the bit in the Q-register is a 1, the corresponding bit in the A-register either 1 or 0 is placed into the corresponding bit of the word specified in the a portion of the MSK instruction.

2.4.6.2 Device Address Register

Bits 16-21 (j) of the Converter Instruction Register specify the device being addressed by the instruction. This is the device address register and its contents are not changed during the processing of an in-out instruction.

2.4.6.3 Word Block Flip-Flop and Word Block Counter

Bits 22-30 (K) of the Converter Instruction Register indicate the number of words or blocks to be processed during the operation. The exact mode of operation depends on the interplay of the word block flip-flop, bit 30, and the word block counter, bits 22-29. Generally, K will be decreased by one, depending on the in-out instruction, after a word or block is processed by the converter.

2.4.6.4 Instruction Register

Bits 31-36 of the Converter Instruction Register specify the type of in-out instruction being processed. This is the part of the converter instruction register that controls the device specified by the device address register.

2.4.7 Real-Time Registers

2.4.7.1 Real-Time Output Register

This register is connected to the main transfer bus and is addressable. When a word is assembled in the register, it is released to the output medium connected to that real-time register.

2.4.7.2 Real-Time Input Register

The information loaded into the real-time input register is transferred either to memory or to an addressable register upon completion of the currently executed instruction. If information arrives during an instruction which requires no access to memory at that particular time, such as during multiplication or division, the transfer occurs while the instruction is executed without delay in the main program.

2.4.7.3 Real-Time Address Register

This register stores the 15-bits (a) which are necessary to specify the memory location or the location of the addressable register which is to receive the information arriving at the real-time input register. The real-time address register is addressable and its contents is automatically incremented after each memory access, so that incoming data is stored in consecutive memory locations..

SECTION III

PROGRAMMING

3.1 MINIMAL PROGRAMMING AIDS

In writing a library of utility routines for use on a new computer, it is necessary to decide on certain basic objectives and to have a clearly understood philosophy of the basic requirements. The Minimal Programming Aids written by Sylvania Electronic Systems Inc. are intended to provide basic routines of a general utility nature for immediate use on MOBIDIC.

3.2 PROGRAMS, ROUTINES, AND SUBROUTINES

This manual contains specific programs consisting of the following: symbolic assembly programs, input-output routines, malfunction control routines, generalized data handling routines, and mathematical subroutines.

3.2.1 Symbolic Assembly Program

This series of programs is referred to as the MOBIDIC Assembly Program (MAP). The assembly routine itself is a fairly basic type of one-to-one symbolic assembly program. It has a reasonable list of pseudo operations which include the ability to handle various types of decimal, octal, and alphanumeric data, library routines, reservation of storage blocks, remarks, origins, and equating of absolute and symbolic names. Also included are pseudo instructions which permit the programmer to code in relative rather than symbolic form.

MAP permits two types of binary outputs, one absolute and the other relocatable to any area designated by a programmer. The relocation may be specified at the time of running. Each binary tape contains its own loading block.

3.2.2 Mathematical Subroutines

3.2.2.1 Functions

There are seven (7) basic mathematical functions: square root, exponential, logarithm, and four trigonometric routines--written in both fixed and floating point.

All routines which involve angles will contain entrances and exits for numbers expressed either as degrees or as radians.

3.2.2.2 Complex Numbers and Double Precision Routines

The complex numbers and double precision routines for the four arithmetic operations have been written in both fixed and floating point.

3.2.2.3 Floating Point Routines

In the case of the floating point, the representation is a one-word packed, excess 256 system, where the first nine bits represent the exponent. In floating point double precision, the second word is used as the low order part of the mantissa.

3.2.3 Input-Output and Format Control Routines

These programs are capable of handling both numeric and alphanumeric information. They have been designed for primary use with paper tape but can be easily adapted to other media. The input-output routines allow the programmer maximum flexibility in the specification of the type and mode of data he wishes to operate upon. As far as input is concerned, the routines allow both numeric and alphanumeric information which may be put into one or several storage areas. The numerical information may be a fixed integer or fraction, a mixed number, a floating point number, or a scaled number. The conversion to the proper binary form is handled automatically.

The output allows the programmer full control over format on the output device and allows him the option of collecting his numbers in any of several ways. Conversion of binary integers, fractions, floating point or mixed numbers can be requested for any number of words. It is also possible for alphanumeric information to be controlled in output format.

3.2.4 Malfunction Control Routines

These are "saving" type programs capable of efficiently coping with interruptions necessitating the stopping of the computer in the middle of a program.

The program, which preserves the contents of the addressable registers, the flip-flops, and the core memory, is written in two forms. The first is for complete operator control and the second is for program control as part of the normal processing.

3.2.5 Generalized Data Handling Routines

The primary difficulty in designing these routines was that the exact details of the applications were not clearly spelled out. They may, in fact, vary widely from one user to another or from one installation to another.

3.2.5.1 Purpose

The programs are designed to sort and/or merge a wide variety of information. They allow for both fixed and variable size items where size may extend beyond 1100 words.

3.2.5.2 Objectives

The objectives of these routines are as follows:

a. Error indication and correction should be handled by the program as much as possible.

b. The need for human intervention and the resultant error should be minimized.

c. The system involving both the tapes and the programs should be self-contained and consistent without any exceptions.

SECTION IV

MOBIDIC SYMBOLIC ASSEMBLY PROGRAM

4.1 MOBIDIC ASSEMBLY PROGRAM

The symbolic label for the MOBIDIC Assembly Program is MAP.

4.2 PURPOSE

The MOBIDIC Assembly Program translates MOBIDIC programs which are written in a convenient symbolic coding language into explicit binary language which MOBIDIC can use directly, in the form of a "Binary Output Tape". An "Edited Assembly Listing" is also produced.

4.3 USAGE

4.3.1 Preparation of Cards or Tapes

The input to the MOBIDIC Assembly Program is prepared on punched paper tape, either 8-hole paper tape in FIELDATA code prepared on a Flexowriter, or 5-hole paper tape in Baudot code prepared on a Teletypewriter. When the Flexowriter is used, the left margin must be set at column 45, with tabs set at columns 56, 64, 72, and 97. The right margin is set at the far right. The symbolic MOBIDIC program is then typed on the Flexowriter following a prescribed format (given below) with the punch "on", producing a paper tape containing the symbolic MOBIDIC program. A "carriage return" must precede the first line of coding and a carriage return followed by two stop codes must be punched at the end of the program. When the paper tape has been finished, it is fed through the "read latch" of the Flexowriter and a "read copy" of the paper tape is produced. This read copy is then proof-read by the programmer to ensure that the paper tape has been prepared correctly and is ready for assembly.

4. 3. 2 Input Format

The input format consists of several fields, as follows:

Control Character Field (column 45)

Dewey Decimal Field (columns 46-53)

Symbolic Location Field (columns 56-61)

Operation Code Field (columns 64-69)

Variable Field (columns 72-94)

Remarks Field (columns 97-130)

Starting with the Control Character Field, the characters are punched forming the various fields and ending in a carriage return. This sequence of characters (ending in a carriage return) is referred to as a "line of coding". Columns 54, 55, 62, 63, 70, 71, 95, 96 are used for tab purposes.

The function of the various fields is described in some detail in a later section on "Format for a Line of Coding".

4. 3. 3 Output from MAP

The output from the MOBIDIC Assembly Program is a Binary Output Tape and an Edited Assembly Listing. The Binary Output Tape may be produced on either the 8-hole paper tape punch, or a magnetic tape (Magnetic Tape No. 3), or both if desired. The Edited Assembly Listing may be produced on either the 8-hole paper tape punch or the on-line Flexowriter. More detail on the Binary Output Tape and the Edited Assembly Listing will be found in later sections of this writeup.

4. 3. 4 Coding Information

4. 3. 4. 1 Sense Flip-Flop Settings

As one of the input conditions for assembling a program using MAP, it is necessary that the console switches for the general sense flip-flops 1 through 6 be

positioned in a specific way to indicate certain conditions to MAP. Each sense flip-flop switch, from SFF1 through SFF6, must be either manually set, manually reset, or neutral in accordance with the following scheme.

SENSE FLIP-FLOP SETTINGS

FOR MOBIDIC ASSEMBLY PROGRAM

SFF No.	Manually Set	Manually Reset	Neutral
1	Binary Output on Magnetic Tape Only	Binary Output on Both Paper Tape and Magnetic Tape	Binary Output on Paper Tape Only
2	No Library Routines on Listing	No Listing at all	Normal Listing, with Library Routines
3	Listing on Paper Tape in Pass 2 - Stop After Pass 2	Listing on Flexowriter in Pass 2 - Stop After Pass 2	Listing on Magnetic Tape Only in Pass 2
4	Copy Magnetic Tape Listing From Pass 2 Onto Paper Tape (Pass 3)	Copy Magnetic Tape Listing From Pass 2 Onto Flexowriter (Pass 3)	Stop After Pass 2; If Desired to Get Printable Listing, Manually Set or Reset SFF 4 and Start at PC
5	Magnetic Tape Input to Pass 1 (Fieldata)	Paper Tape Input to Pass 1 (Baudot)	Paper Tape Input to Pass 1 (Fieldata)
6	Binary Output RELOCATABLE	Binary Output FULL (Self Loading) - for Paper Tape Only	Binary Output ABSOLUTE

Note 1: SFF 1 must be compatible with SFF3. That is, SFF3 must not be manually set if SFF1 is either manually reset or neutral, because there is only one 8-hole paper tape punch available for use when the Binary Output Tape and the Edited Assembly are produced during Pass 2. If SFF1 and SFF3 are not compatible, a printout occurs at the beginning of Pass 2, the two flip-flop switches must be positioned appropriately, and the "Start at PC Switch" used.

Note 2: SFF6 must be compatible with SFF1. That is, if SFF6 is manually reset, SFF1 must be neutral. If an incompatible condition occurs, a printout occurs at the beginning of Pass 2 and appropriate action must be taken as described in Note 1 above.

Note 3: SFF7 through SFF16 must be in a neutral condition at the beginning of each pass. If they are not, a printout occurs and the switches must be placed in the neutral position and the "Start at PC Switch" used.

4.3.4.2 Calling Sequence for MAP

To use the MOBIDIC Assembly Program to assemble a symbolic MOBIDIC program, the following equipment is necessary:

MOBIDIC Computer with at least two memories,

8-hole paper tape reader (for Fieldata input),

5-hole paper tape reader (if input is in Baudot code),

8-hole paper tape punch, and

at least two magnetic tape units; if Dewey Decimal is used, 6 tapes are required.

The following procedure must be used:

1. The System Tape containing the MOBIDIC Assembly Program must be mounted on Magnetic Tape No. 1.

2. A blank magnetic tape (fitted with a "ring" for writing) must be mounted on Magnetic Tape No. 2.

3. A blank magnetic tape must be mounted on Magnetic Tape No. 3, if the Binary Output Tape is to be on magnetic tape. If Dewey Decimal is used, this will also be used for the unsorted correction lines.

4. A blank magnetic tape must be mounted on Magnetic Tape No. 4, if the Edited Assembly Listing is to be on magnetic tape during Pass 2. If Dewey Decimal prepass is used, this will be used for the sorted correction lines.

5. A blank magnetic tape must be mounted on Magnetic Tape No. 5 to be used as output for the Dewey Decimal prepass. This will also be used as input to Pass 1.

6. The input to the Dewey Decimal prepass, if on magnetic tape, will be on Magnetic Tape No. 6.

7. The 8-hole paper tape punch must be turned on and a few feet of blank tape produced, if the paper tape punch is to be used for either the Binary Output Tape or the Edited Assembly Listing.

8. The Flexowriter must be turned on, with the "Local-Receive" switch in the "Receive" position, and an ample supply of paper available and positioned properly in the platen.

9. Sense flip-flop switches 1 through 6 must be set in accordance with the procedure described above under "Sense Flip-Flop Settings" and Sense flip-flop switches 7 through 16 must be in the neutral position.

10. The program to be assembled must be mounted on the appropriate paper tape reader and threaded properly.

11. The First Pass of MAP (or the Dewey Decimal Prepass) is called into memory from the System Tape and execution of the assembly is initiated by a "Start at ASR". The beginning line has a symbolic location MCLPS1, and in the version of MAP, at this writing, has an octal location of 3551. This location must be loaded into ASR prior to the initiation of the program.

Other Subroutines Required

None (contains many subroutines itself)

Error Halts

No Error Halts - See "Error Detection Features"

Sense Flip-Flops Used

SFF1 through SFF6 for parameters; SFF7 through SFF16 for internal processing.

Number of Storage Locations for Symbol Table

2046 (sufficient for 1023 symbols)

Approximate Time

Varies with input and output devices used and with the length of the program being assembled.

4.3.5 Accuracy Information

Average Error: 2^{-37} in DEC and FLT pseudo-ops

Maximum Error: 2^{-36} in DEC and FLT pseudo-ops

All other calculations use integral arithmetic and are therefore exact.

4.4 ERROR DETECTION FEATURES

The MOBIDIC Assembly Program checks the symbolic input program for various kinds of format errors and also for apparent logical errors in the variable field. For example, if a 7-character symbol is used in the symbol field, this will be detected as an error; or, if an apparently incorrect order is given (such as TRU QRG), an error indication is given, although the binary equivalent is still produced.

The format errors and apparent logical errors which have been detected by MAP are shown as "flags" on the Edited Assembly Listing. A maximum of 5 such flags may appear on the listing (to the left of the octal notations) for one instruction line. The letters used for these flags are as follows:

Flag	Significance = Possible error in:
C	Control Character Field
D	Dewey Decimal Field
S	Symbol Field
$\overline{O}$	Operation Code Field
A	Address part of variable field
G	Gamma part of variable field
B	Beta part of variable field
J	J-part of variable field
K	K-part of variable field
V	Variable Field (either error in Pseudo-op, or multiple errors for Machine instruction)

4.4.1 Format for a Line of Coding

4.4.1.1 Control Character Field (column 45)

The Control Character Field may contain any of four characters:

(1) Space

(2) Tab

(3) Carriage Return

(4) Asterisk (*)

(1) Space

A space should be used to reach the Dewey Decimal Field, if there is to be an explicit Dewey Decimal Number with the line of coding.

(2) Tab

A tab may be used to reach the Symbolic Location Field, if there is not to be any explicit Dewey Decimal with the line of coding.

(3) Carriage Return

A carriage return may occur in the Control Character Field, if the Symbolic Input Program, at any time, has two or more carriage returns consecutively, for double or triple-spacing etc.

(4) Asterisk

An asterisk (*) in the Control Character Field signifies that the entire line of coding is to be interpreted as "Remarks". An asterisk, being an upper case character, must actually be preceded by an "upper case" and followed by a "lower case" FIELDATA character.

Any other character appearing in the Control Character Field will be interpreted as an error, resulting in an error printout. The whole line will be interpreted as a line of remarks; no processing of the other fields will occur.

4.4.1.2 Dewey Decimal Field (columns 46-53)

A Dewey Decimal System may be used with the MOBIDIC Assembly Program for ease of incorporating corrections and for certain sequencing applications. This system is described later in some detail under section 4.7.

If the Dewey Decimal System is used, a Dewey Decimal number must appear on some lines of coding. If a specific Dewey Decimal number does appear on a specific line of coding, it must be in the form of a lower case letter of the alphabet followed by one, two, or three pairs of decimal digits (00 to 99) followed by a period (.); or a lower case letter simply followed by a period with no digits at all.

Examples of acceptable Dewey Decimal numbers are as follows:

a.
b01.
c2350.
d456789.

A tab or spaces should follow any Dewey Decimal number such as the above in order to skip to the next field. A tab is recommended.

If there is no Dewey Decimal number associated with the line of coding, a tab should normally appear in the Control Character Field in order to skip over the Dewey Decimal Field to the Symbolic Location Field. If, instead, a space is used in the Control Character Field and there is no Dewey Decimal Number, either a tab or the correct number of spaces may still be used to reach the Symbolic Location Field (column 56).

The meaning of the Dewey Decimal numbers and instructions for their use will be given in more detail in the section on the "Dewey Decimal Prepass".

4.4.1.3 Symbolic Location Field (columns 56-61)

In the Symbolic Location Field, a symbol may appear which may serve either as a mnemonic label for the position of an instruction or data, or as a mnemonic label for a parameter to be defined. In any event, the symbol, if present, must consist of from one to six alphanumeric characters (letters of the alphabet and/or digits from 0 to 9) of which at least one must be a letter. Examples of symbols are given below:

A

CHECK

EXIT

1HALF

BTS23

99999Z

ADJUST

In the implementation of the MOBIDIC Assembly Program, certain standard symbols are used to refer to addressable registers, input-output devices, and sense flip-flops (e.g., ACC for the Accumulator, FLX for the Flexowriter, and ISN for the Interpret Sign Flip-Flop). These standard symbols, which are listed in Appendix A should never be used as a symbolic location in a program.

In using symbols in the Symbolic Location Field, it is possible for several sections of a program, or for several programs being assembled together, to contain the same symbols for different locations or parameters causing "multiply-defined" symbols. Although this unhappy situation may be avoided by specific ground rules for coding, it is convenient to be able to "head" symbols so that two or three indentical symbols (such as "EXIT") can be headed by different characters, producing symbols which are now distinguishable from each other.

The MOBIDIC Assembly Program has the capability of heading symbols using a system which will be described later. However, only symbols of 5 characters or less can be headed. A six-character symbol (such as ADJUST) cannot be headed.

After the symbol appearing in the Symbolic Location Field, a tab must follow to skip to the next field, or, alternatively, the correct number of spaces to reach column 64. A tab is recommended.

If there is no symbol in the Symbolic Location Field, which is permissible, a tab or the correct number of spaces should be used to skip to the Operation Code Field (column 64).

4.4.1.4 Operation Code Field (columns 64-69)

In the Operation Code Field, a mnemonic code consisting of from three to six alphabetic characters must appear. This code may represent a MOBIDIC

Machine Instruction, such as CLA; or a Simulated Machine Instruction (to be described later), such as LDQ; or a Pseudo-Operation (Pseudo-Op), such as ORG, DEC, or END. A specific list of MOBIDIC Machine Instructions, Simulated Machine Instructions, and Pseudo-ops will be given later with a description of how they are handled by MAP.

In the case of machine instructions, any of three characters may appear after the Operation Code. These characters are as follows:

Character	Significance
- (Minus Sign)	Attach a minus sign to this instruction when it has been fully processed, i.e., make bit 37 a "one".
' (Apostrophe)	For an order in a region of relative coding (see 'Relative Start' pseudo-op), do not increment the alpha portion of the variable field.
; (Semicolon)	For a MOV order in a region of relative coding (see 'Relative Start' pseudo-op), do not increment the gamma-beta portion of the variable field.

If these characters do appear in the Operation Code Field (i.e., any one, any two, or all three), they may be in any order, but they must immediately follow the Operation Code, itself.

After the Operation Code and the special characters described above (if present), a carriage return is permissible if there are no remarks and, if the machine instruction or pseudo-operation requires no information in the Variable Field. If information appears in the Variable Field, which is usually the case, a tab must be used to skip to the Variable Field, or the correct number of spaces to reach column 72. A tab is recommended.

If the Operation Code is missing, of if it is anything other than three to six consecutive alphabetic characters, MAP will produce an error printout and will treat the Operation Code Field as if it were a HLT instruction. This procedure will also occur if the Operation Code is ostensibly all right, but does not appear in the Operation Code Table which contains a list of all the permissible MOBIDIC machine instructions, Simulated Machine Instructions, and Pseudo-ops.

4.4.1.5 Variable Field (columns 72-94)

The Variable Field contains the various fields which need to be processed for the completion of an instruction or for the processing of a pseudo-op. In some cases, the variable field need not contain anything. In these cases, there may be a tab to reach the remarks field, or a carriage return if there are to be no remarks (another option will be mentioned later-an exception to the rule).

Assuming that meaningful data is needed in the variable field, for machine instructions, the field required must be as follows:

1. Internal machine instructions other than the MOV order: alpha, gamma, beta (if all three fields are needed) or alpha, gamma (if beta is not needed) or alpha (if neither gamma nor beta is needed)

2. For the MOV order, the fields must be: alpha, gamma-beta (where gamma-beta is one entity and, when processed, is placed in bits 16-30)

3. For the input output order, other than RWD, SKP, and BSP: alpha, j-field, k-field

4. For the SKP and BSP orders: j-field, k-field (no alpha field at all)

5. For the RWD order: j-field (no alpha field or k-field)

In the composition of the various subfields, there must be no spaces. Commas are used to separate the various fields. After the last subfield, there should be a tab to reach the remarks field or a carriage return if there are no remarks.

There is an exception to this. The Variable Field can run beyond column 94 (its normal right limit), if necessary. If so, it must be ended by a carriage return if there are no remarks, or by a space if there are to be remarks. It should not be ended by a tab.

As well as the possibility of having an unusually long variable field ended by a space instead of a tab to reach the remarks, it is also possible to have a short variable field also ended by a space instead of a tab, in which case, the remarks start immediately after the space. This can be done if desired where the Variable Field is short and the remarks are long; thus the Variable Field can overflow into the left end of the Remarks Field, and the Remarks Field can overflow into the

right end of the Variable Field, if necessary. This exception to the rule of the various fields being a fixed size is the reason why the "MAP CODING FORM" has a dotted line at the right end of the Variable Field.

Further rules for what quantities are allowable or necessary in the various subfields for machine instructions are given in a later section on "Processing MOBIDIC Machine Instructions", and for pseudo-ops in the section on Pseudo-ops.

4.4.1.6 Remarks Field (Columns 97-130)

The Remarks Field contains the remarks which help to explain and clarify the coding, if any remarks are present. The remarks can contain any characters whatsoever, but must be ended by a carriage return. Normally, the remarks start in column 97, but they can start either before or after that column, if the Variable Field is ended by a space instead of a tab. There must be no more than 34 characters in the Remarks Field (not including FIELDATA characters for "upper case" and "lower case", which take no space on the paper) assuming the Remarks Field starts in column 97. If there are any more than 34 characters, the Flexowriter carriage is likely to hit the right margin.

4.5 METHOD OF PROCESSING

4.5.1 First Pass of MAP

Input to Pass 1

1. Symbolic Program on Punched Paper Tape.
2. Sense flip-flop settings for SFF1 through SFF6: each one either manually set, manually reset, or "neutral". SFF7 through SFF16 must be in "neutral" position.

Actions Performed by Pass 1

1. Copies Symbolic Program (on paper tape) onto magnetic tape, one block per line of coding.
2. Checks Control Character, Dewey Decimal, and Symbol Fields for format errors. Sets error flags for listing for each possible error discovered.
3. Assembles Op Code and checks Space Field for format errors; sets error flags for listing for each possible error discovered.
4. Determines whether Op Code is Regular or Simulated Machine Instruction, or Pseudo-op.
5. Processes Pseudo-op depending upon the functions required by the particular pseudo-op.

MAP CODING FORM

PROBLEM: JOB NO.

PROGRAMMER: DATE: PAGE OF ☐ PAPER TAPE ☐ CARDS

45 CONTROL 1	46 DEWEY DECIMAL FIELD 53 / 2 9	56 SYMBOLIC LOCATION 61 / 12 17	64 OP. CODE 69 / 20 25	72 VARIABLE FIELD (α, γ, β) or (α, j, k) 94 / 28 50	97 REMARKS FIELD (for Paper Tape) 130 / REMARKS (for Cards) 53 72	IDENTIFICATION (for Cards) 73 80

N-1286(12-59)

Note: For Flexowriter, left margin at 45; tabs at 56, 64, 72, 97. For cards, skips at 12, 20, 28, 53, 73

6. Forms Symbol Table in memory, sorted by symbol. The Symbol Table also contains binary equivalent or location for each Symbol.

7. When END pseudo-op is encountered, finishes writing the magnetic tape copy of original Symbolic Program, rewinds this magnetic tape, reads in Pass 2 of MAP, and transfers control to Pass 2.

Output from Pass 1

1. Magnetic Tape Copy of Symbolic Program.

2. Symbolic Table (stored in memory).

4.5.2 Second Pass of MAP

Input to Pass 2

1. Magnetic Tape Copy of Symbolic Program (from Pass 1).

2. Symbol Table (from Pass 1).

3. Sense Flip-Flop Settings (same as for Pass 1).

Actions Performed by Pass 2

1. Checks Control Character, Dewey Decimal, and Symbolic Fields for format errors; sets appropriate flags if any errors.

2. Assembles Op Code and checks Space Field for format errors; sets flags if any errors.

3. Determines whether Op Code is Regular or Simulated Machine Instruction, or Pseudo-op.

4. For Regular or Simulated Machine Instruction, gets binary equivalent of Op Code and processes Variable Field; checks Variable Field for format errors.

5. For Pseudo-ops, completes the processing of the Variable Field according to the particular function of the pseudo-op.

6. From binary equivalents obtained above, prepares Binary Output Program, and writes out on paper tape, magnetic tape, or both, as specified by Sense Flip-Flop settings.

7. Prepares Edited Assembly Listing, and writes out on Flexowriter, paper tape, or magnetic tape. Edited Assembly Listing includes flags for any format errors discovered. The binary Output and Edited Assembly output can be processed during this pass only if there is no conflict between output devices required.

8. When END pseudo-op is encountered, finishes writing Binary Output Program and Edited Assembly Listing, and rewinds magnetic tapes. If Edited Assembly Listing was on Magnetic tape, reads in Pass 3 of MAP, and transfers control to Pass 3.

Output from Pass 2

1. Binary Output Program, on either magnetic tape, paper tape, or both.

2. Edited Assembly Listing of Program, either on magnetic tape, paper tape, or Flexowriter.

4.5.3 Third Pass of MAP

Input to Pass 3

1. Edited Assembly Listing on magnetic tape (from Pass 2).

2. Sense Flip-Flop Settings (same as for Pass 1).

Actions Performed by Pass 3

1. Note: Pass 3 is used only when Pass 2 has written the Edited Assembly Listing on Magnetic Tape.

2. Pass 3 copies the magnetic tape Edited Assembly Listing onto either the on-line Flexowriter or directly onto paper tape using the 8-hole paper tape punch (suitable for typing on off-line Flexowriter).

3. When finished, rewinds all magnetic tapes not already rewound.

Output from Pass 3

1. Edited Assembly Listing on either paper tape or Flexowriter.

4.5.4 Processing MOBIDIC Machine Instructions

In Pass 2, the Variable Field of the MOBIDIC Machine Instruction is processed to obtain a binary equivalent for the various subfields involved. In this process, different orders naturally fall into different classes. For example, a SKP order would not be processed in exactly the same way as a RPT order. In implementing the processing of the 52 MOBIDIC Machine Instructions, the MOBIDIC Assembly Program handles them in convenient groups. The following chart gives these groups and describes what information must be present, what information can be present, and what information is interpreted as being ostensibly correct.

Each field may consist of a symbol, a numeric quantity, or any meaningful algebraic combination of symbol (s) and numeric (s). In the computation of the variable field, for algebraic combinations, the standard algebraic rules are used i.e. that is, multiplications and divisions are performed before additions and subtractions (in order from left to right).

The chart gives the subfields which may or must be present for each order. Note that for the RWD, BSP, and SKP orders, the alphafield is omitted entirely, and must not be present.

Group Number	OP CODEs and Octal Equivalent	Subfields Permitted	Importance of this Subfield	Code number for Allowable Contents of Subfield (see Explanatory Table)
1	RWD - 77	j-field	Required	227
2	SKP - 66 BSP - 67	j-field	Required	227
		k-field	Required	203
3	RAN - 70 WAN - 74 ROK - 72 WOK - 76 RRV - 71	alpha	Required	747
		j-field	Required	227
		k-field	Required	203
4	SEN - 05 SNS - 06 SNR - 07	alpha	Required	703
		gamma	Required	247
		beta	Required	237
5	LOD - 51	alpha	Required	747
		gamma	Required	247
		beta	Required	247
6	MOV - 52	alpha	Required	747
		gamma-beta (one item)	Required	747
7	TRU - 40 TRL - 41	alpha	Required	703
		gamma	Optional	247
		beta	Optional	203
8	SHR - 32 SRL - 33 CYS - 34 CYL - 35	alpha	Required	203
		gamma	Optional	247

Group Number	OP CODEs and Octal Equivalent	Subfields Permitted	Importance of this Subfield	Code number for Allowable Contents of Subfield (see Explanatory Table)
9	SHL - 30 SLL - 31	alpha	Required	203
		gamma	Optional	247
		beta	Optional*	203
10	HLT - 00 TRS - 42	alpha	Optional	777
		gamma	Optional	777
		beta	Optional	777
11	ADD - 12 ADM - 13 SUB - 16 SBM - 17 DVD - 22 DVL - 23	alpha	Required	747
		gamma	Optional	247
		beta	Optional*	203
12	ADB - 24 SBB - 25	alpha	Required	747
		gamma	Required	247
		beta	Required	203
13	LDX - 53 RPT - 01	alpha	Required	203
		gamma	Required	247
		beta	Required	203
14	TRP - 44 TRZ - 45 TRN - 46	alpha	Required	703
		gamma	Optional	247

*Note: If the beta field for group 9 or 11, although "optional" is omitted, a "b" flag is produced on the listing to remind the programmer that overflow is possible.

Group Number	OP CODEs and Octal Equivalent	Subfields Permitted	Importance of this Subfield	Code number for Allowable Contents of Subfield (see Explanatory Table)
15	CLA - 10 CAM - 11 CLS - 14 CSM - 15 LGM - 02 LGA - 03 LGN - 04 MLY - 20 MLR - 21 TRC - 47 STR - 50 NRM - 37 RPA - 54 MSK - 55	alpha	Required	747
		gamma	Optional	247
16	TRX - 43	alpha	Required	703
		gamma	Required	247
		beta	Required	203

EXPLANATORY TABLE FOR "ALLOWABLE CONTENTS OF SUBFIELD"

Subfield Code No.	Allowable Contents of Subfield (An algebraic combination of these items is also permitted)
203	Numeric, Numeric-type Symbol
227	Numeric, Input-Output Device, Defined Symbol, Numeric-type Symbol
237	Numeric, Input-Output Device, Flip-Flop, Defined Symbol, or Numeric-type Symbol
247	Numeric, Addressable Register, Defined Symbol, Numeric-type Symbol
703	Numeric, Location-type Symbol, Numeric-type Symbol
747	Numeric, Location-type Symbol, Addressable Register, Defined Symbol, Numeric-type Symbol
777	Numeric, Location-type Symbol, Addressable Register, Input-Output Device, Flip-Flop, Defined Symbol, Numeric-type Symbol (in other words, all of the above)

4.5.5 Use of the Asterisk

The asterisk (*) may be used to indicate the line on which the asterisk appears (i.e., the location counter), or a multiplication sign when appearing between two quantities.

For example:

TRU *+5 means "Transfer control to 5 lines down in the program" but

SHR 2*2 means "Shift right" "2 x 3", or 6.

The MOBIDIC Assembly Program determines which meaning the asterisk has by examining its position relative to what precedes it.

4.5.6 Use of a Double Asterisk

A "double asterisk" (**) implies that a subfield is to be filled in later by the program itself and should be initially zero. The symbol ** is processed as a numeric zero and satisfies any subfield which is considered "required" in the processing of machine instructions. For example, if the alpha-field and k-field are each to be filled in by the program for a WAN order using magnetic tape no. 2, the order should be written:

WAN **, mt2, **

4.5.7 Binary Output Program

1. The most important output of the MOBIDIC Assembly Program is the Binary Output Program which is produced during Pass 2. This Binary Output Program is produced on either the 8-hole paper tape punch, magnetic tape, (magnetic tape no. 3), or both, if desired. Which of these options is desired is determined by the setting of Sense Flip-Flop 1, as shown in the Table of Sense Flip-Flops for MAP, given earlier in this writeup.

2. The binary output may be in one of three forms: "absolute", "relocatable", or "full". The choice is determined by the setting of SFF6. The "full" option must be used for paper tape only.

Pass 2 of MAP fills a binary output buffer with fully processed binary words (up to a maximum of 510 words). When a binary output buffer has been filled, or when an org, bss, or bes pseudo-op has been detected, the binary output is completed (including a "hash sum"), a new binary output buffer is started (there are two buffers available), and the completed one is written out.

3. When the end pseudo-op has been detected in Pass 2, the binary output buffer currently being filled is completed and written out, and the variable field is processed to produce a Transfer Line, which is then written out; if paper tape is used, a stop code is written out after the Transfer Line.

4. The absolute format is provided for programs which are to be stored in a certain place in memory. The binary output tape for the absolute format (either paper or magnetic tape) is read into memory using a simple binary loader which reads the words into the desired place in core memory, computes and checks a "hash sum" to indicate that the program has been read in correctly without any errors. Control is transferred to the starting line of the program after stopping, if desired. An absolute program cannot be relocated by the loader; if it is desired to store it elsewhere, the program must be reassembled.

5. The relocatable format is provided for programs which are to be stored nominally at a certain place in the memory, but which may be relocated elsewhere. When this format is used, quantities which are considered to be independent of location within memory are still considered relocatable. In this scheme, the following criteria are used:

Absolute	Relocatable
Pure Numeric	Location-type Symbol
Addressable Register	
Input-Output Device	
Sense Flip-Flop	
Defined Symbol	
Numeric-type (equ) Symbol	

For algebraic combinations of absolute and relocatable quantities, the following ground rules are used:

(1) Any combination of an absolute quantity with another absolute quantity produces an absolute quantity.

(2) For combinations of absolute and relocatable quantities, the following relationships are used, where abs and rel are convenient abbreviations:

(a) rel + abs or abs + rel xx = rel

(b) rel - abs or abs - rel = rel

(c) rel * abs or abs * rel = rel

(d) rel ÷ abs = rel (Note: abs ÷ rel is not very meaningful by itself.)

(e) rel + rel = rel

(f) rel - rel = xxx abs

(g) rel ÷ rel = abs (Note: rel * rel is not very meaningful by itself.)

Note that most combinations of absolute and relocatable quantities are relocatable, except the difference or quotient of two relocatable quantities.

Most relocatable quantities are "relocatable direct". However, if a quantity in a field is relocatable but negative, it is processed module 2^{36} and considered "relocatable complement".

To read in a Binary Output Tape, a relocatable loader is used which stores the program where desired, adjusts relocatable fields by the proper amount, computes and checks a "hash sum", and transfers control to a starting line after stopping, if desired.

6. The "full" option is used for short programs ($\leq$ 508 words) containing no more than one origin, and no bss or bes pseudo-ops, which are designed to be read directly into the computer from the 8-hole paper tape reader using the Program Read-in button on the MOBIDIC Console. When the full option is used, the entire program is punched out in octal without any address words, hash sums, or counts. To use the "full" option, the programmer must supply as the very first two orders in the program, the folflowing orders (no origin appears before them as these orders must not be stored in memory):

```
ROK     (LOC1), ptr8, (NUMB)

TRU     (LOC2)
```

where (LOC1) is the starting position in which the program following the two orders is to be stored, (NUMB) is the number of words required for the program which follows, and (LOC2) is the location at which the program is to start once it has been loaded into memory.

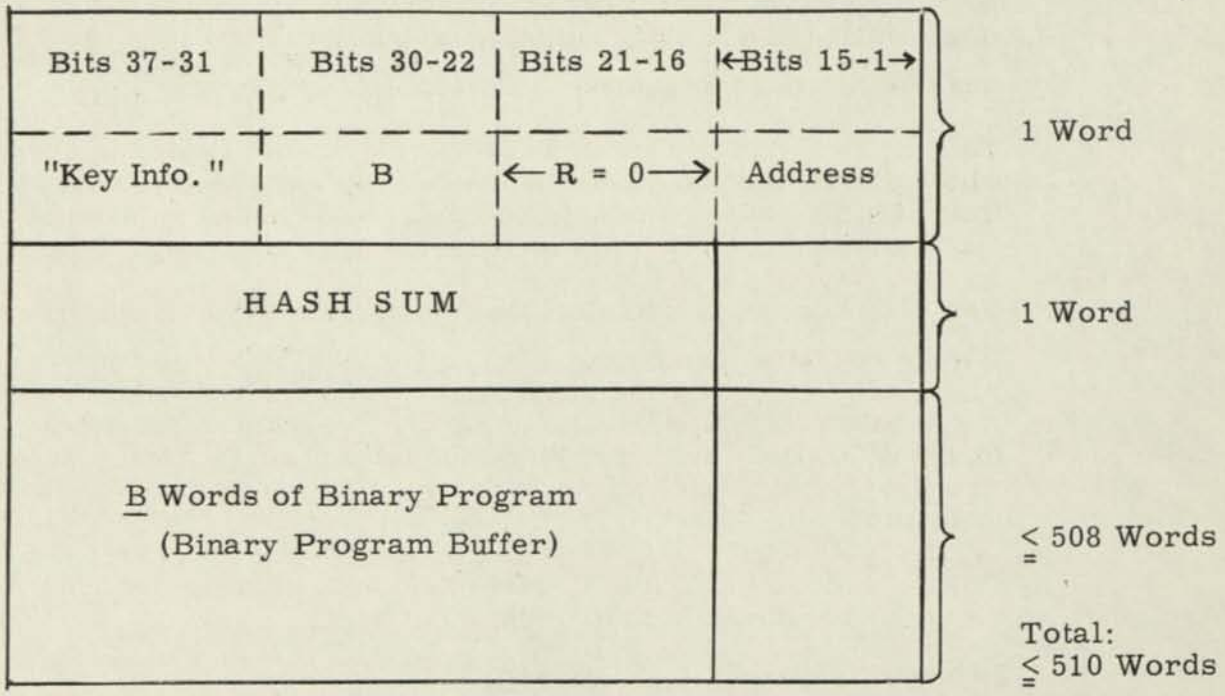
For example, if a program containing 40 words, which starts in position 95 (containing no bss or bes), were desired to be written using the "full" option, the program should start as follows (where execution is to start in 98):

```
ROK     95,ptr8,40
TRU     98
ORG     95
```

(Forty words of program)

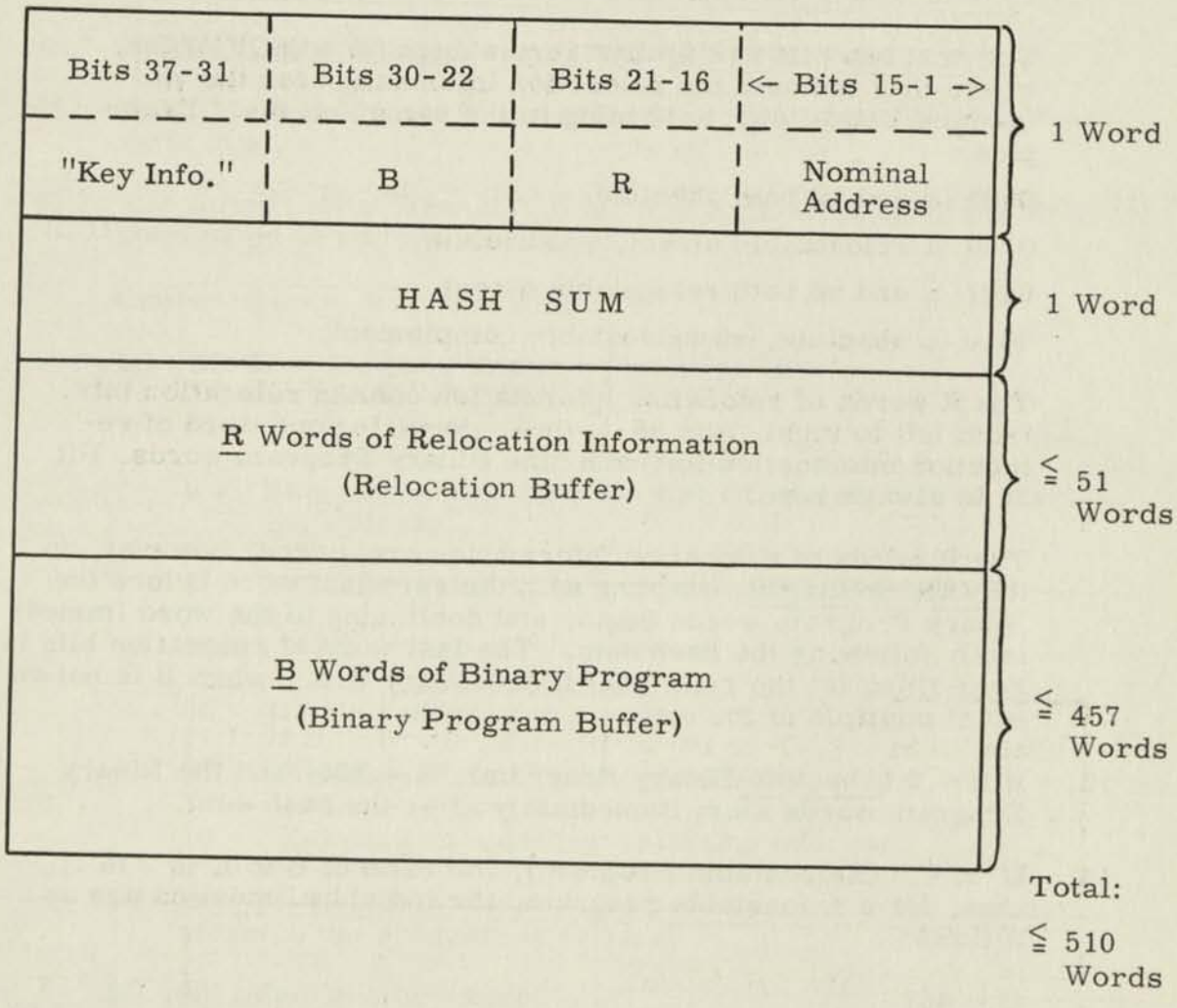
In the above example, the first two words are not counted as part of the 40; they are for use by the Program Read-in button only to load in the rest of the program.

7. The format of the absolute, full, and relocatable blocks of information which comprise a binary output tape are shown in the folflowing diagrams, followed by some explanatory remarks.



NOTE: The Binary Output Format for the FULL option is identical to the Binary Program Buffer by itself, with the address word and hash sum ignored.

9. Binary Output Format for Relocatable Option



10. Nominal Address: This is the starting address at which the binary program would be stored if it were not relocated.

11. R will be zero if the program is absolute (i.e., not relocatable at all). Otherwise, R is the number of words containing relocatable bit information. Bit 37 is not used and is always zero in the words containing relocation bit information.

12. In a relocatable program, four bits of relocation information are used for each word of the Binary Program. The first two bits (reading from left to right) give the relocation information for the α (alpha) field of the word, according to the following scheme:

00 Absolute

01 Relocatable Direct

10 Relocatable Complement

The next two bits are always zero except for a MOV order, where they indicate the relocation information for the $\gamma\beta$ (gamma-beta) field, according to the same scheme. Examples:

0000 α and $\gamma\beta$ both absolute.

0100 α relocatable direct, $\gamma\beta$ absolute.

0101 α and $\gamma\beta$ both relocatable direct.

0010 α absolute, $\gamma\beta$ relocatable complement.

The R words of relocation information contain relocation bits from left to right, bits 36-1; thus, requiring one word of relocation information for each nine Binary Program words. Bit 37 is always zero.

The R words of relocation information are stored, however, in REVERSE ORDER, starting with the very last word before the Binary Program words begin, and continuing to the word immediately following the hash sum. The last word of relocation bits is zero-filled (at the right end) if necessary (i.e., when B is not an exact multiple of 9).

13. If R = 0 (Absolute Binary Program), B = 508, and the Binary Program words start immediately after the hash sum.

14. If R $\neq$ 0 (Relocatable Program), the ratio of B to R is 9 to 1; thus, for a relocatable program, the individual maxima are as follows:

    B = 457

    R = 51

15. For the sake of flexibility, the MOBIDIC Assembly Program uses a symbolic location BNBFSZ (Binary Buffer Size) which contains the number 457, and another location TŌBFSZ (Total Buffer Size) which contains the number 508; thus, the numbers may be changed to any smaller constants simply by changing the contents of BNBFSZ and TŌBFSZ.

16. "Key" information will contain a 1 in bit 37 if this block represents a "transfer line" (see Item 14). Also, bit 35 = 1 implies that the Hash Sum is to be ignored; otherwise, bits 37-31 will be zero.

17. The hash sum will be a sum of all the words in R and B, plus the first word. As these words are added up, the one's complement is taken of all negative words (producing a positive number), and all overflows are ignored.

18. Note that each block never exceeds 510 words.

19. Note that this format is the same for either paper tape or magnetic tape.

20. The format for a "transfer line" (the equivalent of "transfer to start of program") is as follows:

    One word only, with this format:*

    (a) Bit 37 = 1 means this is a "transfer line".

    (b) Bit 36:

        0 = "Stop computer, then start at PC to transfer to starting address"

        1 = "Transfer automatically to starting address".

    (c) Bits 17-16:

        00 = Starting address is absolute.

        01 = "Relocate direct" starting address.

        10 = "Relocate complement" starting address.

    (d) Bits 15-1 contain the starting address, i.e., the location at which the program is to start.

    (e) All other bits are zero.

21. For paper tape, a stop code follows this "transfer line".

---

*On magnetic tape, the transfer line will be a one (1) word block.

4.5.8 Simulated Machine Instructions

In addition to the 52 standard machine instructions provided in MOBIDIC A, the following 11 "Simulated Machine Instructions" are provided with MAP for the convenience of the programmer. Note that these Simulated Machine Instructions are really equivalent to some of the 52 standard machine instructions, and accordingly, should be distinguished from pseudo-ops. The 11 Simulated Machine Instructions listed below constitute the complete list available in MAP. Their use is encouraged; for example, LDQ MASKER is much more convenient to write then LOD MASKER, 0, QRG.

Simulated Machine Instruction	MOBIDIC Equivalent	Purpose of Instruction
CHS	CLS ACC	Change Sign
CLZ	CLA 70000_8	Clear and Add Zero (Gets plus zero in accumulator)
CSZ	CSM 70000_8	Clear and Subtract Zero (Gets minus zero in accumulator)
LDQα,γ	LODa, γ, QRG	Load Q-Register
MVQ$\gamma\beta$	MOV QRG, $\gamma\beta$	Move Q-Register to a storage location
MVZ$\gamma\beta$	MOV 70000_8, γ, β	Move Zero to a storage location
NOP	CLA ACC	No Operation
SSM	CSM ACC	Set Sign Minus
SSP	CAM ACC	Set Sign Plus
TRI	LODa, γ, PCT	Transfer Indirect (Transfer to the contents of a, γ)
XAQ	CYL 37	Exchange Accumulator & Q-Register

4.5.9 Edited Assembly Listing

The Edited Assembly Listing shows the programmer how MAP has assembled the program. The Edited Assembly Listing may be produced during Pass 2 on the on-line Flexowriter, or on the 8-hole paper tape punch if this is not being used for the Binary Output Program. If it is desired to produce an Edited Assembly Listing on the 8-hole paper tape punch but this is being used for the Binary Output Program, the Edited Assembly Listing may be put temporarily on magnetic tape No. 4, and Pass 3 may then be used to copy the magnetic tape onto the 8-hole paper tape punch for printing. The Edited Assembly Listing can be omitted entirely if desired, or library routines may be omitted from the listing; all options are fixed by the use of Sense Flip-Flops 3 and 4, as given earlier in this writeup. When an Edited Assembly Listing is to be produced, the left margin of the Flexowriter is set at position 15, with tabs at positions 45, 56, 64, 72, and 97. Columns 15-44 are used for the information produced during assembly; columns 45-130 contain the original line of coding, except in some cases where a line of coding produces several words where it is blank. For an Edited Assembly Listing the fields are as follows:

col. 15 - 19 - Error Indication Field

20 - Space

21 - 44 - Octal Address and Instruction Field

45 - Control Field

46 - 53 - Dewey Decimal Field

56 - 61 - Symbol Field

64 - 69 - Operation Code Field

72 - 94 - Variable Field

97 - 130 - Remarks Field

15 - 19 - Error Indication Field - will contain 1 to 5 characters to indicate the type of error (or errors) occurring on this instruction line. If no errors are found, then this

field will be blank. One or more spaces will be used to reach the octal address field. The flags which are used are listed under "Error Detection Features".

21 - 44 - octal address and octal instruction field - will contain the octal address and the octal equivalent of the instructions located on this line. There are 8 different formats depending on the type of instruction located on this instruction line.

1) For machine instructions - other than in-out instructions - the format is as follows:

octal address	space	sign	op code	space	γ	space	β	space	α	space
XXXXX	X	X	XX	X	X	X	XXXX	X	XXXXX	XX
21-25	26	27	28-29	30	31	32	33-36	37	38-42	43-44

2) For in-out instructions, the format is as follows:

octal address	space	sign	op code	space	K	space	J	space	α	space
XXXXX	X	X	XX	X	XXX	X	XX	X	XXXXX	XX
21-25	26	27	28-29	30	31-33	34	35-36	37	38-42	43-44

3) For the "move" instruction the format is as follows:

octal address	space	sign	op code	space	$\gamma\beta$	space	α	space
XXXXX	X	X	XX	X	XXXXX	XX	XXXXX	XX
21-25	26	27	28-29	30	31-35	36-37	38-42	43-44

4) For the pseudo-op DEC, - the format is as follows:

octal address	space	sign	12 octal digits	space
XXXXX	X	X	XXXXXXXXXXXX	XXXXX
21-25	26	27	28-39	40-44

5) For pseudo-ops ORG - BES - BSS - SYN - EQU - DEF - RLS - RLE - LIB - END the format is as follows:

space	octal address	space
XXXXXXXXXXXXXXXXX	XXXXX	XX
21-37	38-42	43-44

6) For the pseudo-op OCT, the format is as follows:

octal address	space	sign	OCT	space	AL	space	NUM	space	BER	space
XXXXX	X	X	XXX	X	XXX	X	XXX	X	XXX	XX
21-25	26	27	28-30	31	32-34	35	36-38	39	40-42	43-44

7) For the pseudo-op FLT, the format is as follows:

octal address	space	sign	characteristic	space	mantissa	space
XXXXX	X	X	XXX	X	XXXXXXXXXX	XXXX
21-25	26	27	28-30	31	32-40	41-44

8) For the pseudo-op ALF or ALF or ALZ, the format is as follows:

octal address	space	octal char.	space	octal char.	space	octal char.	space	octal char.	space	octal char.	space	octal char.	space
XXXXX	X	XX	X	XX	X	XX	X	XX	X	XX	X	XX	X
21-25	26	27-28	29	30-31	32	33-34	35	36-37	38	39-40	41	42-43	44

Spaces will always be used to reach the control field (col. 45) from the octal instruction field. The spacing of the fields of the assembly listing, from the control field (col. 45) to the remarks field (col. 97-130), is identical to that of the symbolic program listing.

4.5.10 Supplementary Data Standard Symbolism

This section contains four tables for referencing purposes. They are as follows:

1. Table 4-1 - Addressable Registers
2. Table 4-2 - Input-Output Devices
3. Table 4-3 - Flip-Flops
4. Table 4-4 - Octal Equivalents of Fieldata, Baudot and Mnemonic Operation Codes

TABLE 4-1. ADDRESSABLE REGISTERS

Symbol	Octal Equivalent	Significance
zero	70000	Nonexistent memory location - contains zero
ir0	70000	Index Register zero (nonexistent) - contains zero
ir1	70001	Index Register one
ir2	70002	Index Register two
ir3	70003	Index Register three
ir4	70004	Index Register four
ir5	70005	Index Register five
ir6	70006	Index Register six
ir7	70007	Index Register seven
acc	70010	Accumulator
qrg	70011	Q-Register
brg	70012	B-Register
pct	70013	Program Counter
pcs	70014	Program Counter Store
cio	70015	Instruction Register of Converter Receiving Input order
wsr	70020	Word Switch Register
rar	70021	Real Time Address Register
ror	70022	Real Time Output Register
ci1	70030	Instruction Register of First In-Out Converter
ci2	70031	Instruction Register of Second In-Out Converter

TABLE 4-2. INPUT-OUTPUT DEVICES

Symbol	Octal Equivalent	Significance
mt0	37	Magnetic Tape Number 0 (fictitious magnetic tape)
mt1	40	Magnetic Tape Number 1
:	:	:
.	.	.
mt8	47	Magnetic Tape Number 8
flx	26	Flexowriter
lpr	16	On-Line Printer
ptr5	24	Paper Tape Reader (5-hole)
ptr8	20	Paper Tape Reader (8-hole)
ptp5	25	Paper Tape Punch (5-hole)
ptp8	22	Paper Tape Punch (8-hole)
cdr	14	Card Reader
cdp	15	Card Punch

TABLE 4-3. FLIP-FLOPS

Symbol	Octal Equivalent	Significance
ova	100	Overflow Alarm
ropi	101	Real Time Output Program Interrupt
isn	102	Interpret Sign
nhc	103	No Halt on Converter Error
rpe	104	Real Time Parity Error
robb	105	Real Time Output Register Busy Bit
ror38	106	Bit Position 38 of Real Time Output Register
sff1	110	Sense Flip-Flop 1
:	:	:
.	.	.
sff8	117	Sense Flip-Flop 8
sff9	120	Sense Flip-Flop 9
:	:	:
.	.	.
sff16	127	Sense Flip-Flop 16
ioa1	130	In-Out Alarm Converter 1
ioa2	131	In-Out Alarm Converter 2
tpe	135	Tape Erase

TABLE 4-4. OCTAL EQUIVALENTS OF FIELDATA, BAUDOT AND MNEMONIC OPERATION CODES

Octal	Mnemonic Operation Code	Fieldata Code	Baudot Code
00	HLT	MASTER SPACE	U, L 00
01	RPT	UPPER CASE	U, L 33
02	LGM	LOWER CASE	U, L 37
03	LGA	TAB	N.E.
04	LGN	CARRIAGE RETURN	U, L 10
05	SEN	SPACE	U, L 04
06	SNS	A	L 03
07	SNR	B	L 31
10	CLA	C	L 16
11	CAM	D	L 11
12	ADD	E	L 01
13	ADM	F	L 15
14	CLS	G	L 32
15	CSM	H	L 24
16	SUB	I	L 06
17	SBM	J	L 13
20	MLY	K	L 17
21	MLR	L	L 22
22	DVD	M	L 34
23	DVL	N	L 14
24	ADB	O	L 30
25	SBB	P	L 26
26		Q	L 27
27		R	L 12

Notes: 1. N.E. - No Equivalent
2. Fieldata characters in parenthesis indicate that they are typed in the lower case mode.

TABLE 4-4. OCTAL EQUIVALENTS OF FIELDATA, BAUDOT AND MNEMONIC OPERATION CODES (CONT.)

Octal	Mnemonic Operation Code	Fieldata Code	Baudot Code
30	SHL	S	L 05
31	SLL	T	L 20
32	SHR	U	L 07
33	SRL	V	L 36
34	CYS	W	L 25
35	CYL	X	L 35
36		Y	L 25
37	NRM	Z	L 21
40	TRU	)	U 22
41	TRL	-	U 05
42	TRS	+	U 32
43	TRX		N.E.
44	TRP	=	N.E.
45	TRZ		N.E.
46	TRN		N.E.
47	TRC	$	U 11
50	STR	*	N.E.
51	LOD	(	U 17
52	MOV	"	U 21
53	LDX	:	U 16
54	RPA	?	U 35
55	MSK	!	U 15
56		,	U 14
57		STOP CODE	N.E.

Notes: 1. N.E. - No Equivalent
2. Fieldata characters in parenthesis indicate that they are typed in the lower case mode.

TABLE 4-4. OCTAL EQUIVALENTS OF FIELDATA, BAUDOT AND MNEMONIC OPERATION CODES (CONT.)

Octal	Mnemonic Operation Code	Fieldata Code	Baudot Code
60		(0)	U 26
61		(1)	U 27
62		(2)	U 23
63		(3)	U 01
64		(4)	U 12
65		(5)	U 20
66	SKP	(6)	U 25
67	BSP	(7)	U 07
70	RAN	(8)	U 06
71	RRV	(9)	U 30
72	ROK	(')	U 13
73		(;)	U 36
74	WAN	(/)	U 31
75	WWA	(·)	U 34
76	WOK		N.E.
77	RWD	CODE DELETE	N.E.

Notes: 1. N.E. - No Equivalent
2. Fieldata characters in parenthesis indicate that they are typed in the lower case mode.

4.6 PSEUDO-OPS

4.6.1 Origin: org

The org pseudo-op sets the location counter to the value of the expression appearing in the variable field.

The org pseudo-op, in the variable field, should have a decimal number, a previously defined symbol, or any meaningful algebraic combination thereof. The numeric value of this expression is assigned to the location counter. If MAP cannot obtain a binary equivalent, a flag is set for the listing, and the location counter is set to zero.

After this expression, there should be a tab, if remarks are to follow, or a carriage return, if there are no remarks.

When an org is encountered in Pass 2, the binary output block currently being filled is completed and a new one is started with the address of the org.

Normally, an org pseudo-op should appear at the beginning of a program. If it does not, an origin of zero is automatically assigned by MAP.

4.6.2 Block Started by Symbol: bss

The bss pseudo-op reserves a block of core storage extending from L to L + n - 1, where L equals the present value of the location counter and n is the numerical equivalent of the expression appearing in the variable field. The expression n may be a decimal number, a previously defined symbol, or any meaningful algebraic combination thereof. If MAP cannot obtain a binary equivalent of this expression, a flag is set for the listing, and n is set to zero.

Normally, a symbol appears in the symbolic location field for a line of coding containing a bss pseudo-op. This symbol is assigned to the first word of the block of core storage locations reserved.

When a bss is encountered in Pass 2, the binary output block currently being filled is completed and a new one is started with the address equal to L + n (terms defined above).

4.6.3 Block Ended by Symbol: bes

The bes pseudo-op reserves a block of core storage extending from L to L + n - 1, and in fact, behaves exactly as the bss pseudo-op except for the treatment of the symbol appearing in the Symbolic Location Field. This symbol is assigned to location L + n, corresponding to the location of the first word following the block reserved.

4.6.4 Equals: equ

The equ pseudo-op equates the symbol appearing in the symbolic location field to the numerical value of the expression appearing in the variable field. The expression in the variable field may be a decimal number or a meaningful algebraic expression containing decimal number(s) and/or previously defined symbol(s). It should, however, be equivalent to a number, not an address or storage location. The equ pseudo-op should be used, as a rule, only in those cases where the symbol appearing in the symbolic location field implies a preset parameter such as the order of a matrix, the degree of a polynomial, the number of items in a group, or any other quantity which is invariant with respect to the location of the program in storage. (If the symbol refers to the location of either a piece of data or an instruction, the syn pseudo-op should be used.) If MAP cannot obtain a binary equivalent for the expression appearing in the variable field, a flag is set for the listing and the binary equivalent is set to zero. If the binary equivalent is obtained but is representative of a location, the binary equivalent obtained is used as is, but an error flag does not occur indicating an apparent error in the use of the equ pseudo-op.

Examples of equ pseudo-ops follow:

```
shftn     equ    27              number of shifts
matord    equ    a*b + c/d - 2   matrix order
numb      equ    start-begin     number of words
```

4.6.5 Define: def

The symbol appearing in the symbolic location field is assigned the binary equivalent of the number in the variable field, which must be a single octal number. The def pseudo-op is used for defining a symbol for an addressable register, in-out device, or sense flip-flop which is not one of the "standard symbols" used in MAP. It may also be used for assigning an octal equivalent to a symbol referring to a location whose octal position is known. If the variable field does not contain a valid single octal number, flag is set for the listing and zero is used for the binary equivalent. If the octal number exceeds 77777, the 15 low-order bits (the number module 2^{15}) are used.

Examples of def pseudo-op follow:

```
spi     def     160     stop program interrupt
wcc     def     167     write control characters
loder   def     17725   location of loader routine
```

4.6.6 Synonym: syn

The syn pseudo-op is used to relate the symbol in the symbolic location field to the expression in the variable field, which may be another symbol or a meaningful algebraic combination of previously defined symbol(s) and number(s). The syn pseudo-op should be used in those cases where the symbol appearing in the location field specifies the location of a piece of data, the location of an instruction, or any other quantity whose value depends upon the location of the program in storage; it may also be used to equate a symbol used by the programmer to a "standard symbol". If MAP cannot obtain a binary equivalent for the expression in the variable field, a flag is set for the listing, and the binary equivalent is set to zero. If the binary equivalent is obtained but is representative of an apparent error (such as a single decimal number, where an equ should be used), the binary equivalent is used as is, but an error flag does occur.

Examples of syn pseudo-ops follow:

```
com       syn     common + 12
sffz      syn     sff13           sffz is the same as sff13
adjust    syn     fix-18
```

4.6.7 Octal Data: oct

The oct pseudo-op takes the octal integer(s) in the variable field, converts them to binary, and assigns them (if more than one) to successive locations beginning with the current value of the location counter. If there is a symbol in the symbolic location field, the symbol is entered in the table and is assigned the current value of the location counter. More than one octal number may be written in the variable field; if this is done, a comma must be used to separate the various octal numbers; the last octal number must be followed by a tab, space, or carriage return (not a comma). If there are several octal numbers, it is permissible for the last one to extend beyond column 94, the normal end of the variable field, but it must be followed by either a space (for following remarks) or a carriage return, not a tab.

Each octal number must consist of from one to twelve valid octal digits; each negative number must be preceded by a minus sign; each positive number may be preceded by a plus sign if desired, but this is not required. If any octal number is in error (i.e., contains non-octal characters or is more than 12 digits), a binary equivalent of zero is substituted, a flag is set for the listing and MAP scans to a comma or other ending character to look for the next octal number or the end of the date. Each octal number is treated as being "right-justified", i.e., ending in bit 1 (not "left-justified" or in bit 36).

Examples of a single octal number and multiple numbers follow:

```
jmask     oct     7700000                         mask for "j" bits
octdat    oct     -3, +4, -77, +37, 40, 41, 42    octal data
```

4.6.8 Fixed Point Decimal Data: dec

The dec pseudo-op takes the fixed point decimal number(s) in the variable field, converts them to binary, and assigns them (if more than one) to successive locations beginning with the current value of the location counter. If there is a symbol in the symbolic location field, the symbol is entered in the table and is assigned the current value of the location counter. More than one decimal number may be written in the variable field; if this technique is used, a comma must be used to separate the various decimal numbers; the last decimal number must be followed by a tab, space, or carriage return (not a comma). If there are several decimal numbers, it is permissible for the last one to extend beyond column 94, the normal end of the variable field, but it must be followed by either a space (for following remarks) of a carriage return, not a tab.

Each negative decimal number must be preceded by a minus sign; a positive number may be preceded by a plus sign if desired, but this is not required. Decimal numbers (ignoring the sign) may be any of the following classes:

(1) An integer, which may range from 0 to 68719476735 ($= 2^{36} - 1$); the integer should not be followed by a decimal point.

(2) A fixed point fraction, preceded by a decimal point, and consisting of up to eleven digits.

(3) A scaled fixed point number, with an attached binary scale factor; this may be in the form of a mixed number with a sufficiently large negative binary scale factor to make it less than 1.0, or a fixed point fraction with either a negative binary scale factor or a positive binary scale factor sufficiently small so that the resultant number is less than 1.0. In each case, the number is in the form of up to eleven significant figures, followed by the letter "b", followed by the binary scale factor, orpower of two.

Examples of scaled fixed point numbers are as follows:

3.1415926535b-2	("pi" scaled by 1/4, or 2^{-2})
85.1234567b-9	(a number, such as degrees, scaled by 2^{-9})
0.11234b3 or .11234b3	(a small number scaled upward by $8 = 2^{+3}$, actually equivalent to .89872)
0.66666666667b-2 or .66666666667b-2	(a fractional number scaled downward by 1/4, or 2^{-2}, in this case the fractional value of 2/3 scaled downward by 1/4)

Examples of the use of the dec pseudo-op follow:

piovr4	dec	3.1415926536b-2	pi over 4
degdat	dec	-45.8b-9, +22.125b-9	scaled degree data
decdat	dec	-99, -372, 0, +82, +.75, .4	decimal integers and fractions

If any decimal number is in error (e.g., the binary equivalent is equal to or greater than 1.0 in absolute value, or extraneous characters appear in the middle of a number), a binary equivalent of zero is substituted, MAP scans to a comma or other ending character to look for the next decimal number or the end of the data, and a flag is set for the listing.

4.6.9 Floating Point Data: flt

The flt pseudo-op takes the floating point number(s) in the variable field, converts them to binary, and assigns them (if more than one) to successive locations beginning with the current value of the location counter. If there is a symbol in the symbolic location field, the symbol is entered in the table and is assigned the current value of the location counter. More than one floating point number may be written in the variable field; if this technique is used, a comma must be used to separate the various floating point numbers; the last floating point number must be

followed by a tab, space, or carriage return (not a comma). If there are several floating point numbers, it is permissible for the last one to extend beyond column 94, the normal end of the variable field, but it must be followed by either a space (for following remarks) or a carriage return, not a tab.

Each negative floating point number must be preceded by a minus sign; a positive floating point number may be preceded by a plus sign if desired, but this is not required. Floating point numbers must have no more than ten significant figures. Any more digits contribute nothing to the accuracy of a MOBIDIC floating point number, and are ignored by MAP (except where they might affect the placement of the decimal point in determining the size of the number).

A floating point number may be in either of two forms:

(1) An ordinary number, either an integer, fraction, or mixed number, such as 98370, 9837., 9837.0, 9837.45, 3.141592654, 0.75, 0.00123, .345678, or 00012345, or the like. Any of these is acceptable although there should be no more than 10 significant figures.

(2) Exponential form, in which the number (as above) is followed by the letter "e" and the power of ten by which the number is to be multiplied. For example, the first and last numbers given above might be written as 9.8370e4 or 0.9837e5 (or some such), and .12345e-3 or 1.2345e-4 (or some such).

In any event, the exponent is a base 10 exponent. The letter "b" must not be used.

Each floating point number to be represented is converted to the binary form of a "Normalized MOBIDIC floating point number", as defined in the write-up of the "Floating Point Arithmetic Operations" subroutine, to be found elsewhere in the MOBIDIC Instruction Manual.

If any floating point number is in error (e.g., the binary equivalent is equal to or greater than 2^{256} in absolute value, which is too big to represent as a MOBIDIC floating point number, or extraneous characters appear in the middle of a number), a binary equivalent of zero is substituted, MAP scans to a comma or

other ending character to look for the next floating point number or the end of the data, and a flag is set for the listing. (A number whose binary equivalent is less than 2^{-256} in absolute value is set to zero without any error indication.)

4.6.10 Alphanumeric Data: alf and alz

1. The alf pseudo-op is used for storing alphanumeric data in the computer. It retains its alphanumeric nature rather than getting converted to some other form such as binary. There is another pseudo-op alz, available, if it is desired to replace FIELDATA spaces (000101) by binary zeroes (000000).

The following paragraphs explain the format and give the rules for use of the alf and alz pseudo-ops.

2. The alf or alz pseudo-ops require that the variable field contain an expression equivalent to the number of lines of alphanumeric data to be processed. After this expression, there may be a Remarks Field if desired, followed by a carriage return, or simply a carriage return if there are no remarks. Immediately following this carriage return, the specified number of lines of alphanumeric data must follow.

3. The expression in the variable field may be an actual decimal integral number, or a predefined symbol, or any meaningful algebraic combination thereof; but in any event, the binary equivalent must equal the number of lines of alphanumeric data. This number should always be = 1 in value. In the great majority of cases, it is expected a regular decimal integer (e.g., 12) would be used.

4. Following this expression in the variable field, there may be:

(a) Carriage return (if no remarks).

(b) Tab, then remarks, then carriage return, or

(c) Space, then remarks, then carriage return.

In any event, the alphanumeric data follows the carriage return.

5. The specified number of lines of alphanumeric information which follow must obey the following rules:

(a) Each line may be no more than 86 characters including the carriage return at the end, except that upper case and lower case characters do not count as part of the 86.

(b) Each line must end either in "Carriage Return" or in the sequence "Tab-Carriage Return." A line of alphanumeric data is, in fact, distinguished by the carriage return at the end; thus, the "number of lines of alphanumeric data" is actually equal to the number of carriage returns.

(c) A carriage return all by itself on a line constitutes a valid line of alphanumeric data and may be used for double spacing, etc., if desired.

(d) The rules for ending a line of alphanumeric data are as follows:

(1) End a line simply with a carriage return if that carriage return is an integral part of the alphanumeric data.

(2) End a line with the sequence "tab-carriage return" if a carriage return, although not an integral part of the alphanumeric data, is required to end the line on the MAP coding form. For example, if a line of alphanumeric data consisting of 120 characters is desired, the first 80 characters could be followed by a "tab-carriage return" then, the remaining 40 characters are followed simply by a carriage return. The "tab-carriage return" does not get converted to binary information by Pass 2, but is merely a necessary artifice to obviate the restriction of 86 characters. Both lines, however, count as a line of alphanumeric information as far as the count is concerned.

6. It is suggested that the following convention be adopted by MOBIDIC programmers for indicating that a "tab" character is to be punched:

[t] means "tab"

Similarly, if there is any doubt whether a carriage return is wanted, the programmer could write:

[cr] for "carriage return"

7. The alz pseudo-op is identical to the alf pseudo-op, except that (in Pass 2) MAP converts each FIELDATA Space (05) to a binary zero. The alz pseudo-op will be used for storing alphanumeric information in the memory for masking (rather than printing) purposes.

8. An example of the use of the alf pseudo-op is included below. Note that the number of words taken in the memory by the alphanumeric data need not necessarily be known as an exact number by the programmer. In this example, the alphanumeric data starts in Symbolic Location title[1]: the next order in the program starts in Symbolic Location contin; thus, the programmer may write an output order as follows:

WAN title[1], flx, kl

where

kl equ contin-title[1]

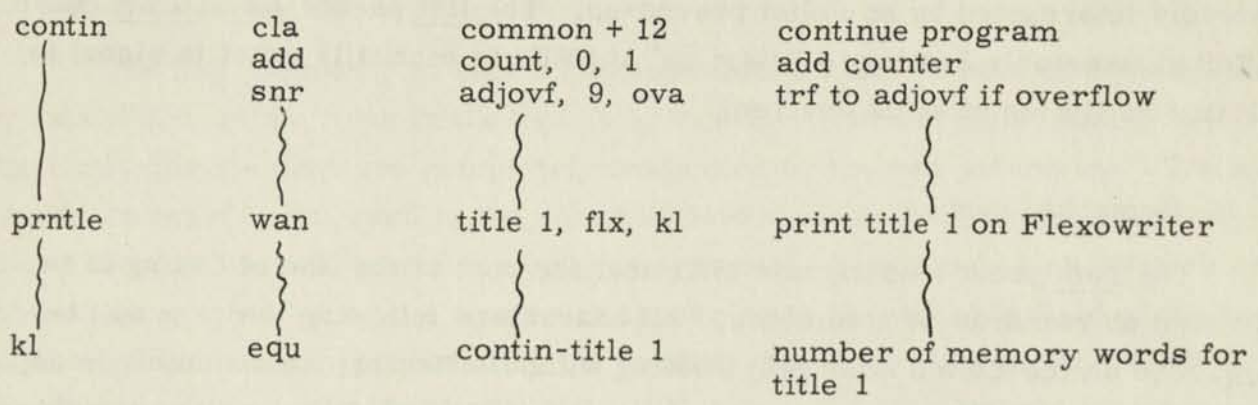
The only restriction would be, naturally, that the numeric value of contin-title 1 would not exceed 511.

Example:

Control	Dewey Decimal Field	Symbolic Location	OP Code	Variable Field	Remarks Field
		skip	tru	contin	skip over title to contin
		title 1	alf	12	12 lines of title 1 follow

This is an example of the alf pseudo-op where the variable field contains an actual decimal integral number, which would be the usual case. The number 12 means that twelve lines of alphanumeric information are to follow. For example, we are on the fourth line of those 12 right now.

Each line normally ends in a carriage return, which is part of the alphanumeric information itself. However, if it is desired to have an unusually long line, which will not physically fit on the MAP coding form because the character count exceeds 86, a "tab-carriage return" combination may be used, indicating that the carriage return is NOT part of the alphanumeric data, but necessary to bring us to the start of the next line on the MAP coding form. Note, however, that a line counts as a line of alphanumeric data as far as the count (in this case 12) is concerned, regardless of which option it ends in.



4.6.11 Alphanumeric Data with Zeros instead of Spaces: alz

The alz pseudo-op is provided to permit the storage of alphanumeric data separated by zero-bit characters (binary 000000) instead of FIELDATA spaces. It is identical to the alf pseudo-op except that each FIELDATA space (05_8) encountered is transformed to a six-bit "binary zero." The alz pseudo-op is useful for storing alphanumeric data for masking or searching (rather than printing) purposes.

The description of the alf pseudo-op should be consulted for further information.

4.6.12 Discontinue Edited Assembly Listing Temporarily: unlist

The unlist pseudo-op is a signal to MAP that the Edited Assembly Listing is to be temporarily discontinued. The binary output tape will continue to be produced as usual. The Edited Assembly Listing is stopped after the unlist itself has been produced and is resumed only upon discovery of a list pseudo-op.

The unlist and list pseudo-ops are useful for omitting time-consuming portions from an Edited Assembly Listing which are known to be correct from previous assemblies or other sources (possibly including divine inspiration or other inside information).

4.6.13 Resume Edited Assembly Listing: List

The list pseudo-op causes the Edited Assembly Listing to be resumed immediately, including the printing of the list line itself, after it has been

previously interrupted by an unlist pseudo-op. The list pseudo-op will not cause the Edited Assembly Listing to "start up" if SFF2 is manually reset (a signal to MAP that no listing at all is desired).

4.6.14 Remarks: rem

The rem pseudo-op signals MAP that the rest of the line of coding is to be treated as remarks or comments. All characters following the rem will be reproduced on the Edited Assembly Listing without affecting the assembly in any way. A maximum of 59 characters (columns 72-130) may follow a rem pseudo-op (including spaces and punctuation); a carriage return must always follow the last remark character.

4.6.15 End of Program: end

The end pseudo-op indicates to MAP the end of the program to be assembled. The end also may be used to indicate where the program being assembled is to start when it is loaded into memory to be run. The alpha portion of the variable field (whether numeric, symbolic, or any meaningful algebraic combination thereof) designates the starting line for execution of the running program. If the gamma part of the variable field contains anything other than a 1 (such as a zero or nothing at all), the computer will halt when the program has been loaded into memory, and the program may then be started at its starting location by the operator depressing the "Start at PC" switch on the console. The latter method is frequently used so that the operator has time to set the Flexowriter to the left margin, turn on the paper tape punch, etc.

The end pseudo-op must follow the last line of information to be processed; the line must be ended (with or without remarks) by a carriage return, and this in turn must be followed by two stop codes. Examples of an end pseudo-op are as follows:

```
end          begin, 1      start program at "begin"
end          195           start program at 195
```

4.6.16 Head: hed

The hed pseudo-op causes a heading effect on all symbols of 5 characters or less which follow, both in the symbolic location field and the variable field. Six-character symbols are completely unaffected by the hed pseudo-op. The hed pseudo-op must be followed in the variable field by a single letter or numeric, giving 36 possible characters for heading symbols. Symbols of 5 characters or less which are headed by the character appearing in the variable field of the hed pseudo-op are actually preceded by the heading character; that is, the heading character occupies bits 36-31 of a MOBIDIC word, and the symbol being headed occupies the remainder of the word, starting in bits 30-25, etc. Also, a minus sign is attached to the headed symbol. For example, the symbol "check" when headed by the character "a" becomes "-acheck" in the memory, and is uniquely distinguished from any other symbol "check," whether headed or not, and also from a 6-character symbol "acheck," which would not, of course, have the minus sign attached.

The heading of all symbols of 5 characters or less, which is started by the hed pseudo-op, is ended when the unh ("unhead") pseudo-op is encountered. The "hed" and "unh" pseudo-ops are used in pairs, as the "list" and "unlist" pseudo-ops.

During Pass 2, when MAP is trying to obtain a binary equivalent from the symbol table for all symbols encountered in the variable field, if a symbol is not found, it is examined to see if it is headed. If it is, the minus sign and heading character are stripped off, and the resultant unheaded symbol is looked up again. This procedure assures that symbols in the variable field which refer to addressable registers, in-out devices, sense flip-flops, symbols outside the headed portion of coding, and other bonafide symbols are still processed satisfactorily.

The heading procedure is useful for assembling together small programs which contain symbols within such as "check," "exit," "test," "fixit," etc., which should be different for each program to avoid multiple definition of the same symbol, but happen to be the same because of their obvious mnemonic value. Nothing can be done to head 6-character symbols such as "adjust."

If the first character of the variable field is not a letter of the alphabet or a numeric, no heading takes place at all and a flag is set in the listing.

The heading system provided in MAP also allows reference to a specific headed symbol from a region of coding not headed by the same character as the one in current use. This region of coding need not be headed at all. To accomplish this cross-reference to a headed symbol, the programmer must write the heading character of that symbol, followed by a dollar sign ($), followed by the symbol. For example, to write a "trn" order referring to the locat on "check" which is within a region of coding headed by the letter "a," the programmer must write:

```
trn a$check
```

This procedure, of course, is not necessary if the "trn" order itself also happens to be within the region headed by "a," although no error indication would be given if this were done.

4.6.17 Unhead: unh

The unh ("unhead") pseudo-op causes the heading of symbols to cease immediately and thus nullifies, for the lines of coding which follow, the heading process previously initiated by a hed (head) pseudo-op. See the discussion of the hed pseudo-op for a more complete description of the heading process available in MAP.

4.6.18 Relative Start: rls

The rls pseudo-op precedes a section of relative coding as described below. When the rls pseudo-op is detected, the value of the location counter is noted and preserved for use in processing the relative coding; hereafter, it shall be referred to as the "value of the location counter at time of rls."

The rules for this scheme are as follows:

1. Only the alpha field, and the gamma-beta field (for a MOV order), are affected; all other fields (i.e., gamma, beta, "j", and "k") are handled in the usual way.

2. If the alpha or gamma-beta field is a single numeric (e.g. 8191) it is affected by the rls pseudo-op as described in 4. below. If it is anything other than a single numeric, the preceding rls has no affect and it is handled in the usual way.

3. In the "Operation Code Field," columns 64-69, an apostrophe immediately following the Operation Code (e.g., cla') signifies that the alpha field of this order is not to be affected by the preceding rls. For a MOV order, in addition to the possible presence of an apostrophe, a semicolon immediately following the Operation Code (e.g., mov;) signifies that the gamma-beta field of this order is not to be affected by the preceding rls. If both an apostrophe and a semicolon are needed, they should be placed immediately after the MOV order (e.g., mov';).

4. Except for a certain class of orders where the meaning of the alpha field is never a storage location (listed below), rls pseudo-op has the following effect on the alpha field and on the gamma-beta field (for a MOV order) of the line of coding being processed.

   If the alpha or gamma-beta field is a single numeric, and if the Op Code does not have an apostrophe or semicolon for gamma-beta, the "value of the location counter at time of rls" is added to the numeric value of the alpha or gamma-beta field in processing the order. If the apostrophe or semicolon is present, this process is inhibited; thus the relative fields are incremented and the fields which are not relative are unaffected.

5. For certain orders, the meaning of the alpha field is independent of memory location; for these Op Codes, the alpha field is never incremented by the "value of the location counter at time of rls," even though the apostrophe is not present. These orders are as follows: SHL, SLL, SHR, SRL, CYS, CYL, RPT, LDX, TRS, and HLT. With these orders, an apostrophe is not necessary.

The relative coding process continues, according to the rules given above, until a "relative end" (rle) pseudo-op is encountered.

4.6.19 Relative End: rle

The rle pseudo-op indicates to MAP that a section of relative coding has been terminated and that all subsequent orders should be processed in the usual way.

4.6.20 Insert Library Routine: lib

The lib pseudo-op is used to indicate that a routine from the library tape is to be inserted in the program at the point where the lib pseudo-op appears. The variable field of the lib pseudo-op must contain the Symbolic Label of the library routine desired. For example, if the programmer wishes to insert the library routine for Floating Point Arithmetic Operations (FLTPT1), he would write:

```
lib     fltpt1     Insert floating point here
```

When lib pseudo-op is encountered, the desired routine is called in from the library tape and assembled, starting with the value of the location counter at the time the lib pseudo-op is encountered. Each successive library routine called is internally assigned a 6-bit heading character by which the symbols in that library routine, exclusive of callwords, are headed. This procedure is provided to prevent duplicate symbols amont the library routines, which are coded sumbolically, and the program itself. The 6-bit heading characters are assigned starting with the number 63 (778) and working downwards. If the programmer assigns his heading symbols (if any) from a to z, the chance that any two symbols will have the same heading character is extremely small.

In the processing of the library routine being assembled, the callwords of the routine (for this example, flpadd, flpsub, flpmul, flpdiv, and flpnrm) are added to the symbol table, and thus, any "trl" orders to the callwords of the routine used by the program will be filled in by MAP without any special effort by the programmer.

When the library routine has been completely assembled, MAP reverts to assembling the regular program, starting with the location counter at the value where the library routine left off. For example, if the location counter is 920 when a library routine of 60 orders is called with a lib pseudo-op, it will be 980 when the library routine has been completely assembled. Any new origin specified by the program using the routine should be at least 980 so as not to conflict with the library routine.

A special feature of MAP is that it keeps track of the implied routines which should be included in the memory for each routine being assembled. For example, the Fixed Point Arcsine-Arccosine Routine (ARCSC1) requires that the Fixed Point Square Root (SQRT2) be included somewhere in the program being assembled. It is the programmer's responsibility to insure that this other routine is inserted someplace (with an lib pseudo-op) in the program. However, if this is not done before the end of the program, MAP will recognize the omission of the implied SQRT2 routine, and will give a printout at the end of Pass 1 to the effect that the SQRT2 routine has been omitted from the program.

This procedure is used, instead of automatically inserting an implied routine right there on the spot, for the following reasons:

1. Each routine used may be included in the memory once and only once, instead of being included every time another routine specifies it as an implied routine.

2. Each routine may be located where the programmer wished to put it; thus, aiding in a memory layout conforming to the programmer's plans, rather than restricting his choice of memory layout.

3. If, for some reason, a programmer wished to substitute a routine of his own for one on the library tape, he is permitted to do this without being forced to use an implied routine on the library tape.

It should be noted that the program writeup of each library routine clearly indicates "Other Subroutines Required," so that the programmer should have no difficulty in making sure that he has not omitted a needed subroutine.

One important thing that the programmer must do is to include a reference to "common" someplace in his program for library routines which use "common" locations for their temporary and working storage. For example, if the programmer were using several of the mathematical routines, he could include a block of 40 storage locations somewhere in his program by writing:

```
common bss  40  common locations for lib rtns
```

If desired, the Edited Assembly Listing of library routines may be inhibited by placing SFF2 in the manually set position at the start of assembly.

Each library routine ends with a pseudo-op endlib (end library routine), which signals MAP to revert to assembling the regular program. The endlib pseudo-op, as well as the lib pseudo-op, does appear on the Edited Assembly Listing even if "No Library on Listing" is specified by SFF2, in which case, however, there will be nothing in between.

4.6.21 End of Library Routine: endlib

The "endlib" pseudo-op appears on the Edited Assembly Listing to bracket the library routine called, but originates from the library and not from the symbolic input. It is therefore, a pseudo-op for the library routine, but not for the symbolic input to MAP. In other words, it is never written in the symbolic input in conjunction with the lib pseudo-op, but is printed on the listing to denote the end of the coding for a particular library routine.

4.6.22 Dewey Decimal Correction Start: crs

The crs pseudo-op is used in conjunction with the Dewey Decimal System to denote the beginning of a band of Dewey Decimal corrections. See write-up of Dewey Decimal System for full details.

4.6.23 Dewey Decimal Corrections End: cre

The cre pseudo-op is used with the Dewey Decimal System to signify the end of a band of Dewey Decimal Corrections. See write-up of Dewey Decimal System for full detail.

4.6.24 Dewey Decimal Deletion: del

The del pseudo-op is used in conjunction with the Dewey Decimal System and contains a Dewey Decimal number (explicit or implied) on that line of coding. See write-up of Dewey Decimal System for full detail.

4.6.25 Dewey Decimal Origin Number: orgnum

The orgnum pseudo-op is used in conjunction with Dewey Decimal corrections and usually appears just after a crs pseudo-op. See write-up of Dewey Decimal System for full detail.

4.7 DEWEY DECIMAL SYSTEM

4.7.1 General

Programs stored on cards can often be corrected with little difficulty. Obvious problems arise, however, when it is necessary to correct a program that has been stored on tape or to effect structural changes in a program. Many devices suitable for particular conditions may not be feasible, efficient, or even practicable for a given program. A general type of solution is therefore desirable, one that transfers the bulk of corrective labor from the programmer to the computer and at the same time preserves the continuity of processing. Such a solution is the DEWEY DECIMAL SYSTEM.

The DEWEY DECIMAL SYSTEM is a programming device that gives the computer the ability to effect programmer-designated changes during assembly. The changes may involve substitutions, deletions, and insertions. They are open-ended, in the sense that they can be successively updated. The system allows the computer to incorporate designated changes and corrections by the use of ordering tags or sequencing labels associated with program elements. These tags have been dubbed Dewey Decimals or Dewey Decimal Numbers because of their resemblance to the numbers used in library catalogues. Dewey Decimals establish relative priorities of program elements, and assign proper sequential locations in the program to corrections, deletions, and insertions. If the Dewey Decimal System is used, one physical program store contains all programmer-designated changes as well as the program itself. Such a store also contains sufficient information to give both the composition of the program and its changes, and how the changes are to be applied.

Under the present system a Dewey Decimal Number may be composed of from two to five alphanumeric character subfields. The first or letter subfield is always a single lower case letter of the English alphabet. The last or terminal point subfield is always a single alphanumeric decimal point. Each subfield, if any, between the initial letter and the terminal point is a pair of decimal digits from 00 to 99. A Dewey Decimal is thus a string of two, or four, or six, or eight characters chosen from the 26 lower case letters, 10 decimal digits, and decimal

point. It is also a string of subfields of these characters in any of four specific configurations. These configurations are the four forms or levels allowable under the present system for Dewey Decimals, and are:

Level	Subfield 1	Subfield 2	Subfield 3	Subfield 4	Subfield 5
1	a	.			
2	a	xx	.		
3	a	xx	xx	.	
4	a	xx	xx	xx	.

where "a" stands for any one of the 26 lower case English letters, "xx" is a two-character number from 00 to 99, and "." is the alphanumeric decimal point.

To illustrate the mechanism of ordering, consider the following set of Dewey Decimals:

m000001., q01., b00., a000002., z0000., m000000., a000001., z., q05., b0000., z0001., q0501., m01., m., q02., m0001., b01., q., a000000., b02., m0000., z0100., q04., a0000., q0500., a0001., q03., m00., b0002., a01., m000100., q00., b0001., z00., b., a00., z010001., a., z010000., z01.

If a Dewey Decimal less its terminal point is given in its octal representation, with a pair of octal zeros (equivalent to six binary zeros) used for each digit in an unused subfield, the above set can be considered equivalent to:

22606060606061, 26606160606060, 07606000000000, 06606060606062, 37606060600000, 22606060606060, 06606060606061, 37000000000000, 26606500000000, . . . , 37606100000000

The numerical order of the numbers in this equivalent set is the correct order of the given set of Dewey Decimals. The correct order of the given set of Dewey Decimals is then:

a., a00., a0000., a000000., a000001., a000002., a0001., a01., b., b00., b0000., b0001., b0002., b01., b02., m., m00., m0000., m000000., m000001., m0001., m000100., m01., q., q00., q01., q02., q03., q04., q05., q0500., q0501., z., z00., z0000., z0001., z01., z0100., z010000., z010001.

In every subfield a blank takes priority; that is, a decimal point in any number subfield takes precedence over any numerical entry in that subfield.

As will be seen later (see 9.7.2 USAGE) the above set of Dewey Decimals cannot occur in the main coding of a program in the order given. The set was used only to illustrate relative order among Dewey Decimals of all possible types.

The system is of course expandable, but at present is restricted to the number and types of subfields and characters given above. Within the present restrictions, however, the system offers a large number of possibilities for tags in any one program. Level 1 has 26 possible choices of lower case letters; level 2 has 100 possible choices of number for each choice of initial letter, or 2,600 in all; level 3 has 260,000; and level 4 has 26,000,000. Since all levels can be used, as many as 26,262,626 Dewey Decimals are available for any one program.

It should be noted that deviation from the four allowable forms or levels of Dewey Decimal will generate an error in the present system. When a Dewey Decimal is used, both an initial lower case letter and a terminal decimal point must appear. The initial letter subfield can contain no character other than a single lower case English letter. An intervening number subfield can contain only a pair of decimal digits. If a number subfield contains only one decimal digit or any character but a decimal digit, or if the initial letter subfield contains anything but a single lower case English letter, an error is generated. An error will also be generated by the use of more than five subfields. The program scans for initial lower case letter, then for pairs of decimal digits up to three pairs until the terminal point is reached. The decimal point signals the end of the Dewey Decimal number. If any of these conditions is violated, assembly is not halted but a Dewey Decimal flag is caused to appear on the coding line affected.

4.7.2 Usage

When the Dewey Decimal System is used, each line of coding in the program is identified by a unique Dewey Decimal if an ordering tag is applicable. All correction lines are similarly identified. The Dewey Decimal tag assigned to a correction line identifies the main program coding line to which the correction applies. It thus establishes a clear connection between original and correction coding lines.

A band of corrections applies to particular lines of coding in the main program. Since a band of corrections can consist of only one line of coding, it is possible to have a separate correction band for each main program coding line to be changed, and for every coding line to be inserted. However, it is advisable and recommended that as many corrections and insertions as possible be consolidated into a single band, thus minimizing the total number of correction bands for a given program. This would also be helpful to the programmer, since all corrections would be contained in the minimum number of bands.

It is not necessary for the programmer to assign a Dewey Decimal to every coding line. Within limits, the Dewey Decimal Program will tag all coding lines not tagged by the programmer. The Dewey Decimals specifically assigned by the programmer are called explicit, while those assigned internally by the program are called implicit.

Explicit Dewey Decimals must be "proper" in the main program; that is, they must be in proper serial order. Violation of this condition will cause the Dewey Decimal Program to suspend itself until it meets a new and proper Dewey Decimal.

Whenever the program encounters a proper Dewey Decimal in a sequence of coding lines, it establishes an increment in the last subfield of that Dewey Decimal. To every succeeding line not explicitly tagged, the program assigns a number which is the last explicit tag incremented by the succession number of the new line relative to the explicitly tagged line. This process continues until either of two things happens:

1. The last subfield of the last explicit Dewey Decimal is exhausted before a new explicit number occurs.

2. A new and proper explicit Dewey Decimal occurs before the last subfield of the last explicit number is exhausted.

If a new and proper explicit Dewey Decimal occurs before the last subfield of the previous explicit number is exhausted by encrementation, the Dewey Decimal Program reinitializes itself relative to the new explicit number.

If the last subfield of the previous explicit number is exhausted before a new and proper explicit number occurs, the Dewey Decimal Program suspends itself until it meets a new and proper (relative to the last number assigned) explicit number. In this case, no error halt occurs; but the printout will show both a Dewey Decimal flag and no Dewey Decimal tag for every line affected.

In any case, whenever an improper Dewey Decimal occurs the program suspends itself until a new and proper number is encountered. All lines affected will have no Dewey Decimal tags and will display a Dewey Decimal flag in the printout. No error halt occurs.

The Dewey Decimal Program will in no case increment a subfield beyond its limit, or transcend the level of any Dewey Decimal. Incrementation of a subfield beyond its limit and transcendence of the level of a Dewey Decimal are equivalent to generation of an improper number and will cause suspense of the Dewey Decimal Program until a new and proper number is encountered.

To illustrate: Suppose a program contains just ten lines of main coding. If the programmer assigns "a." to line 1, "d." to line 5, and "g." to line 8, the program first assigns "b.", "c.", and "d." to lines 2, 3, and 4. It then encounters "d." on line 5. Since "d." is improper with respect to the last Dewey Decimal ("d.") assigned, the Dewey Decimal Program goes into suspense and assigns no tags to lines 5, 6, and 7. When it scans line 8, it finds that the explicit "g." is proper with respect to "d." on line 4, the last Dewey Decimal assigned. It then reinitializes itself with respect to "g." and assigns "h." and "i." to lines 9 and 10. No error halt occurs at any time. The program printout shows "a." through "d." for lines 1 through 4, "g." through "i." for lines 8 through 10, and Dewey Decimal flags but no Dewey Decimal tags for lines 5 through 7.

In general, the discretion of the programmer must be governed by reasonable precautions in the use of the Dewey Decimal Program. Care must also be used in the use of Dewey Decimal levels. Carelessness can result in the assignment of Dewey Decimals that do not permit the program to interpolate as required. No hard and fast rule can be given here. The only guides are experience, prudence, and the ability to assess contingencies.

All correction lines must be tagged consistently with the main program coding lines to which they refer. Within a correction band defined by the "crs" and "cre" codes (see 9.7.3 below), lines may be tagged in any order that preserves both correct sequence and consistency with the main program coding lines. The programmer must also assign explicit Dewey Decimals sufficient to permit the program to assign implicit numbers properly. For example, if four consecutive lines are to be inserted between "g2345." and "g2346." one between "k6702." and "k6703.", and two between "t4152." and "t4153.", all seven correction lines may be included in a single correction band defined by the codes "crs" and "cre". An acceptable set of Dewey Decimal tags for the seven lines within the band would be:

t415202., t415203., k670210., g234531., g234532., g234533., g234534.

Another acceptable set would be:

g234541., g234542., g234543., g234544., t415222., t415223., k670250.

In either case, the underlined tags are the only ones that need to be explicitly assigned by the programmer. If the lines within each "g", "k", or "t" block are in proper sequence, the blocks themselves may be in any order whatsoever within the correction band. If, however, the programmer elects to assign an explicit tag to every correction line, the correction lines within a band may be in any order. The "crs" and "cre" codes suspend, within linits, the requirement for serial order.

4.7.3 CRS and CRE

A "crs" operation code signals the beginning of a band of corrections, and a "cre" operation code signals its end. These two operation codes bracket a correction band and must be used in pairs. All coding lines within a "crs - cre" pair are processed as corrections to the main body of programming. A "crs" must not be followed by another "crs" until a "cre" has been used. Correction bands may not be nested. If a "crs" or a "cre" appears on a coding line, no Dewey Decimal number may appear on that line.

4.7.4 Orgnum

The function of the "orgnum" operation code is to identify an origin in the main body of programming. If used, it must appear between a "crs" and its following "cre". The address field of the "orgnum" operation code must contain a number which is the decimal count of that origin with which a sequence of corrections is associated. Every origin must be referenced by an "orgnum". For example, "orgnum 1" refers to the first origin in the program, "orgnum 2" refers to the second origin, etc.

If no origins are used in the main program, then no "orgnum" need appear within a correction band.

If an "orgnum" appears on a coding line, no Dewey Decimal number may appear on that line.

SECTION V

MATHEMATICAL SUBROUTINES AND DETAILED FLOW CHARTS

5.1 SPECIFICATIONS FOR MATHEMATICAL SUBROUTINES

The following information specifies the requirements which must be observed when using the mathematical subroutines.

1. All MPA subroutines are entered through the use of a calling sequence located in the main program. A subroutine calling sequence is defined as a series of computer instructions and storage locations in the main program which provides: for transfer of program control to the subroutine, storage locations for all required parameters in excess of those contained in the accumulator and the Q-register, storage locations for results computed by the subroutine, and a normal return location to which control is transferred by the subroutine upon completion of the assigned function. The first instruction in the calling sequence will be a Load PCS and Transfer (TRL) to the address of the subroutine entrance instruction. The address stored in the PCS is used by the subroutine in executing a transfer to the normal return location in the main program. The starting address of a subroutine may be specified by a mnemonic representing a relative address or an octal number representing as absolute address.

2. All one word input and/or output is in the Accumulator. Return is always to the line following the one containing the "trl" order.

3. All two-word inputs and/or outputs are in the Accumulator and the Q-Register, with the second number (c. g., divisor in floating point division) in the Q-Register. Return is always to the line following the one containing the "trl" order.

4. All n word inputs and/or outputs will have the first two words in the Accumulator and Q-Register, and the rest under the "trl" order. Thus, the lines under the "trl" order may contain the inputs, or be reserved for the outputs, or both. Return is to the line following the last one containing data.

5. All error halts will occur in the subroutine. In no case, will the control be returned to the main program.

6. There will be an error halt for each violation of the specifications. In the a portion of the "hlt" will be the current contents of the PCS so that the program entry will be completely determined. The beta bits are used to indicate the number of the error halt for subroutines containing more than one.

7. All subroutines will use the same temporary storage area, referred to as "common". In general, the input variables will be found in common, common + 1, etc.

8. Certain sense flip-flops will be used by the subroutines and therefore may not be locked from the console. They will not be stored to their original state at the time of exit from the subroutine.

9. The state of the OVA may be changed by the subroutine and will not be restored.

The following mathematical subroutine descriptions are written in a definite format for the sake of consistency.

5.1.1 Fixed Point Square Root

Symbolic Label: SQRT2

PURPOSE:

Computes the square root of a fixed point number x, $0 \leq x < 1$

USAGE:

A. Preparation of Cards or Tapes
None

B. Input Format
Fixed Point Fractional Number

C. Output Format
Fixed Point Fractional Number

D. Coding Information

Calling Sequence

A) trl sqrtfx

A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: x	ACC: $\sqrt{x}$

Other Subroutines Required

None

Error Halts

Input number negative ($x < 0$)

Sense Flip-Flops Used

None

Number of Storage Locations

51

"Common" Locations Used

3 (common thru common + 2)

Approximate Time

Average: 0.75 milliseconds

Maximum: 0.8 milliseconds

E. Accuracy Information

Average Error: Exact

Maximum Error: $\pm 2^{-36}$

RESTRICTIONS:

Input number x must be either zero or positive.

METHOD:

$$\left.\begin{array}{l} x = 2^n y,\ 1/2 \leq y < 1 \\ \qquad\quad 1/4 \leq y < 1/2 \end{array}\right\} \quad \text{where n is even}$$

$$y_o = -a_o y^2 + a_1 y + a_2 ,$$

$$y_{n+1} = \frac{y}{2y_n} + \frac{y_n}{2} ,$$

requiring only two iterations.

$$\left.\begin{array}{l} a_o = 12884901888 \times 2^{-36} \\ a_1 = 59497322958 \times 2^{-36} \\ a_2 = 22107055666 \times 2^{-36} \end{array}\right\} \quad 1/2 \leq y < 1$$

$$\left.\begin{array}{l} a_o = 9663676416 \times 2^{-36} \\ a_1 = 42880953483 \times 2^{-36} \\ a_2 = 15365674998 \times 2^{-36} \end{array}\right\} \quad 1/4 \leq y < 1/2$$

Remarks:

None

5.1.2 Fixed Point Sine, Cosine

Symbolic Label: SNCS 2

PURPOSE:

Computes the sine or cosine of a scaled fixed point angle which is expressed either in radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format (See Note 3)

Angle x expressed in radians scaled by $1/8(x/8)$, or in degrees scaled by $2^{-9}(x \cdot 2^{-9})$, i. e. binary point considered to be 9 places from the left.

C. Output Format

Fixed Point Fractional Number

D. Coding Information

1. Calling Sequence for sine, radians

A) trl sinrad

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: $\frac{x}{8}$ in radians	ACC: sine x

2. Calling Sequence for sine, degrees

A) trl sindeg

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Condition
ACC: $x \cdot 2^{-9}$ in degrees (Binary point considered to be 9 places from the left.)	ACC: sine x

3. Calling Sequence For cosine, radians

A) trl cosrad

A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: $\frac{x}{8}$ in radians	ACC: cosine x

4. Calling Sequence for cosine, degrees

A) trl cosdeg

A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: $x \cdot 2^{-9}$ in degrees (Binary point considered to be 9 places from the left.)	ACC: cosine x

Other Subroutines Required

None

Error Halts

See "Restrictions" and Note 2

Sense Flip-Flops Used

None

Number of Storage Locations

58

"Common" Locations Used

2 (common thru common + 1)

Approximate Time

Average: 1.2 milliseconds

Maximum: 1.3 milliseconds

E. Accuracy Information

Average Error: 2^{-36}

Maximum Error: 2^{-33} (See Note 1)

RESTRICTIONS:

$-2\pi \leq x \leq 2\pi$ (radians)

$-360 \leq x \leq 360$ (degrees) (See Note 2)

METHOD:

Form x' such that sin x' = sin x and $0 < x' < \pi/2$

Then cos x = sin ($\pi/2$ - x)

$$\sin x = \sin \pi/2\, y \qquad y = \frac{2x'}{\pi}$$

$$\text{Sin } \frac{\pi}{2}y = 2 \sum_{i=0}^{6} C_{2i+1} y^{2i+1}$$

The series is an economized Taylor series expansion for sin $\frac{\pi}{2}$y

$C_1 = \pi/4$

$C_3 = -22195157385 \times 2^{-36}$

$C_5 = 2738217778 \times 2^{-36}$

$C_7 = -160863808 \times 2^{-36}$

$C_9 = 5512621 \times 2^{-36}$

$C_{11} = -123530 \times 2^{-36}$

$C_{13} = 1871 \times 2^{-36}$

Note 1. Since degree input is converted to scaled radian input upon entry, accuracy for degree input is limited to 2^{-33}.

Note 2. The routine will accept some angles outside this range and produce the correct result. For sine output, any angle less than 8 radians in magnitude can be accomodated; for cosine output, the range is somewhat more restricted but larger than the stated range.

Note 3. The degree scaling allows three octal digits to the left of the binary point e.g. an input angle of 15^{o}_{10} would be represented as +017.00...0.

5.1.3 Fixed Point Tangent, Cotangent

Symbolic Label: TNCT2

PURPOSE:

Computes the tangent or cotangent of a scaled fixed point angle which is expressed either in radians or degrees. The tangent or cotangent will be less than 1 in absolute value.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

(See Note 3)

Angle x expressed in radians for tangent, or scaled by $1/2\ \frac{(x)}{2}$ for cotangent; or in degrees scaled by $2^{-9}(x \cdot 2^{-9})$ for either function (binary point considered to be 9 places from the left)

C. Output Format

Fixed Point Fractional Number

D. Coding Information

1. Calling Sequence for tangent, radians

A) trl tanrad

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x in radians	ACC: tangent x

2. Calling Sequence for tangent, degrees

A) trl tandeg

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Condition
ACC: $x \cdot 2^{-9}$ in degrees (binary point considered to be 9 places from the left)	ACC: tangent x

3. Calling Sequence for Cotangent, Radians

   A) trl cotrad

A + 1) Returns control here

Entry Conditions and Exit Conditions

Entry Condition	Exit Condition
ACC: x/2 in radians	ACC: cotangent x

4. Calling Sequence for cotangent, degrees

   A) trl cotdeg

A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: $x \cdot 2^{-9}$ in degrees (binary point considered to be 9 places from the left)	ACC: cotangent x

Other Subroutines Required

None

Error Halts

1. For tangent entry: $|x| \leq \frac{\pi}{4}$ (radians) or $|x \cdot 2^{-9}| \leq 45.2^{-9}$ (degrees)
2. For cotangent entry: $\left|\frac{x}{2}\right| > \frac{\pi}{8}$ or $\left|\frac{x}{2}\right| > \frac{\pi}{4}$ (radians); or $|x \cdot 2^{-9}| > 45.2^{-9}$ or $|x \cdot 2^{-9}| > 90.2^{-9}$ (degrees).

Sense Flip-Flops Used

None

Number of storage locations

63

"Common" Locations Used

4 (common thru common + 3)

Approximate Time

Minimum: .9 milliseconds (for tan x in radians)

Maximum: 1.0 milliseconds (for cot x in degrees)

E. Accuracy Information

Average Error: 2^{-37}

Maximum Error: 2^{-36} (See Note 2)

RESTRICTIONS:

1. For tangent in radians, $0 \leqq |x| \leqq \frac{\pi}{4}$
2. For tangent in degrees, $0 \leqq |x \cdot 2^{-9}| \leqq 45.2^{-9}$
3. For cotangent in radians, $\frac{\pi}{4} \leqq |\frac{x}{2}| \leqq \frac{\pi}{2}$
4. For cotangent in degrees, $45.2^{-9} \leqq |x \cdot 2^{-9}| \leqq 90.2^{-9}$

METHOD:

$\cot |x| = \tan (\frac{\pi}{2} - |x|)$

$\tan (-x) = - \tan x$

$$\tan |x| = \frac{a_1 x - a_3 x^3 + a_5 x^5}{a_0 x - a_2 x^2 + a_4 x^4 - x^6}$$

(Rational polynomial approximation to tan x)

This approximation has an associated error

$\eta < 3 \times 10^{-13}$, $|x| < \frac{\pi}{4}$

$a_0 = 665280$

$a_1 = 332640 \times 2^1$

$a_2 = 75600 \times 2^2$

$a_3 = 10080 \times 2^3$

$a_4 = 840 \times 2^4$

$a_5 = 42 \times 2^5$

Note 1. For an input number $x = \pm 45^o$ (or $\pm\frac{\pi}{4}$), tan x (cot x) is given as $1-2^{-36}$ (777777777777_8)

Note 2. The error figures assume exact input; when degree input is not exact, accuracy is limited to the number of significant bits in the argument.

Note 3. The degree scaling allows three octal digits to the left of the binary point; e.g., an input angle of 15^o_{10} would be represented as +017.00...0

5.1.4. Fixed Point Arscine - Arccosine

Symbolic Label: ARCSC2

PURPOSE:

To compute arcsin (x) or arccos (x), for a fixed point number x, and to give the result expressed in either radians or degrees. Values are in the first or third quadrant for arcsin (x) and in the first or second quadrant for arccos (x). Positive angles are given. The degree answers are scaled by 2^{-9}. The radian answers are scaled by 2^{-3}.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

Fixed Point Fractional Number

C. Output Format (See Note 2)

Angle y expressed in radians scaled by 2^{-3}; or in degrees by 2^{-9}

D. Coding Information

1. Calling Sequence for arcsine, radians

A) trl arsinr

A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: x	ACC: 2^{-3} arcsin x

2. Calling Sequence for arcsine, degrees

A) trl arsind

A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: x	ACC: 2^{-9} arcsin x

3. Calling Sequence for arccosine, radians

A) trl arcosr
A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: x	ACC: 2^{-3} arccos x

4. Calling sequence for arccosine, degrees

A) trl arcosd
A + 1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Condition
ACC: x	ACC: 2^{-9} arccos x

Other Subroutines Required

For all entries: None

Error Halts

For all entries: None

Sense Flip-Flop Used

None

Number of Storage Locations

127

"Common" Locations Used

5 (Common thru common + 4)

Approximate Time

Minimum: 2.3 milliseconds

Maximum: 2.3 milliseconds

E. Accuracy Information

Average Error: 10^{-9}

Maximum Error: 10^{-9}

RESTRICTIONS:

None

METHOD:

Evaluate

$$\psi(x) = \frac{\arccos |x|}{\sqrt{1 - |x|}}$$ by means of a polynomial approximation

$$\psi^*(x) = a_o + \sum_{i=1}^{8} a_i |x|^i$$

Then

$$1/8 \arccos |x| = \sqrt{1 - |x|} \quad \psi^*(x)$$

$$1/8 \arcsin |x| = 1/8 \left(\frac{\pi}{2} - \arccos |x|\right)$$

$$\arcsin (-|x|) = \pi + \arcsin |x|$$

$$\arccos (-|x|) = \pi - \arccos |x|$$

Note 1. The Output is scaled by 2^{-3} (or 2^{-9}); unscaled accuracy of the result is limited therefore to the number of bits retained after scaling.

Note 2. The degree output scaling allows three octal digits to the left of the binary point, e.g., an output angle of 15^{o}_{10} would be represented as +017.00...0.

5.1.5 Fixed Point Arctangent, Arccotangent (ARCTC2)

PURPOSE:

To compute arctangent $\frac{y}{x}$ or arccotangent $\frac{y}{x}$ and give result in degrees or radians in the proper quadrant. Answer for arctan is in the quadrant such that sine and cosine have the sign of y and x respectively. The degree answers are scaled by 2^{-9}. The radian answers are scaled by 2^{-3}.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

x: Fixed Point Fractional Number
y: Fixed Point Fractional Number

C. Output Format (See Note 2)

Angle x expressed in radians and scaled by $1/8(\frac{x}{8})$; or in degrees scaled by $2^{-9}(x \cdot 2^{-9})$.

D. Coding Information

1. Calling Sequence for arctangent, radians

A) trl artanr

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Condition
ACC: y	ACC: $\frac{1}{8}\arctan\frac{y}{x}$
QRG: x	

2. Calling Sequence for arccotangent degrees

A) trl artand

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Condition
ACC: y	ACC 2^{-9} arctan $\frac{y}{x}$
QRG: x	

3. Calling Sequence for arccotangent radians

A) trl arcotr

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Condition
ACC: y	ACC $\frac{1}{8}$ arccot $\frac{y}{x}$
QRG: x	

4. Calling Sequence for arccotangent degrees

A) trl arcotd

A + 1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Condition
ACC: y	ACC 2^{-9} arccot $\frac{y}{x}$
QRG: x	

Other Subroutines Required

For all entries: None

Error Halts

For all entries: x = 0 and y = 0

Sense Flip-Flops Used

None

Number of Storage Locations

"Common" Locations Used

4 (common thru common + 3)

Approximate Time

Minimum 1.6 milliseconds

Maximum 1.7 milliseconds

E. Accuracy Information

Average Error: 2^{-36}

Maximum Error: 2^{-33} (See Note 1)

RESTRICTIONS:

1. At least one of x, y must not be equal to zero.

METHOD:

If $y < x$, form arctan $\frac{y}{x}$

If $y > x$, form arctan $\frac{x}{y}$

The routine uses a continued fraction approximation to arctan z:

$$\text{Arctan } z = za_1\left[z^2 + b_1 + a_2\left[z^2 + b_2 + a_3\left[z^2 + b_3 + a_4\left[z^2 + b_4\right]^{-1}\right]^{-1}\right]^{-1}\right]^{-1}$$

$0 \leq z < 1$

$a_1 = 15.19704838$ $b_1 = 37.741540761$

$a_2 = -181.10084458$ $b_2 = 9.073054714$

$a_3 = -2.624415818$ $b_3 = 2.6456857766$

$a_4 = -0.16853569334$ $b_4 = 1.3787177232$

Then $\text{arccot}\ \frac{y}{x} = \arctan\ \frac{x}{y} = \frac{\pi}{2} - \arctan\ \frac{y}{x}$ if $y < x$

$\text{arccot}\ \frac{x}{y} = \arctan\ \frac{y}{x} = \frac{\pi}{2} - \arctan\ \frac{x}{y}$ if $y > xx$

Note 1. The output is scaled by 2^{-3} (or 2^{-9}); unscaled accuracy of the result is limited therefore to the number of bits retained after scaling.

Note 2. The degree output scaling allows three octal digits to the left of the binary point, e.g., an output angle of 15°_{10} would be represented as + 017.00...0

5.1.6 Fixed Point Exponential

Symbolic Label: EXP2

PURPOSE:

Computes e^x for fixed point number x zero or negative $(-1 < x \leq 0)$.

USAGE:

A. Preparation of Cards or tapes

None

B. Input Format

Fixed Point Fractional Number

C. Output Format

Fixed Point Fractional Number

D. Coding Information

Calling Sequence

A) trl expofx

A+1) Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: e^x

Other Subroutines Required

None

Error Halts

Input number positive, non-zero $(x > 0)$

Sense Flip-Flops Used

None

Number of Storage Locations

39

"Common" Locations Used

5 (common thru common + 4)

Approximate Time

Average: 0.9 milliseconds

Maximum: 0.9 milliseconds

E. Accuracy Information

Average Error: $\pm 2^{-36}$

Maximum Error: $\pm 2^{-35}$

RESTRICTIONS:

Input number x must be either zero or negative.

METHOD:

$$e^x = \frac{A + B}{A - B}$$

where

$$A = a_o + a_2 x^2 + a_4 x^4 + x^6$$

$$B = a_1 x + a_3 x^3 + a_5 x^5$$

(Rational polynomial approximation to e^x) See Note 2

This approximation has an associated error

$$\eta < 4 \times 10^{-13}, \quad |x| < 1$$

$a_o = 665280$

$a_1 = 332640$

$a_2 = 75600$

$a_3 = 10080$

$a_4 = 840$

$a_5 = 42$

Remarks:

Note 1. For an input number $x = \pm 0$, e^x is given as $1 - 2^{-36}$ $(.777777777777_8)$ elapsed time is 48 μ sec.

Note 2. Lanczos, Applied Analysis (1956).

5.1.7 Fixed Point Natural Logarithm

Symbolic Label: LOG2

PURPOSE:

Computes 1/32 ln x for a fixed point number x where $0 < x < 1$

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

Fixed Point Fractional Number

C. Output Format

Fixed Point Fractional Number, scaled by 2^{-5}

D. Coding Information

1. Calling Sequence for ln x

A) trl natlog

A+1) Returns control here

Entry and Exit Conditions

Entry Condition	Exit Conditions
ACC: x	ACC: $\frac{\ln x}{32}$
	COM: ln x (if $1/2 \leq x < 1$)

Other Subroutines Required

None

Error Halts

Input number negative

Sense Flip-Flops Used

None

Number of Storage Locations

43

"Common" Locations Used

5 (common thru common +4)

Approximate Time

Maximum: 1.1 milliseconds

E. Accuracy Information

Average Error: 2^{-36}

Maximum Error: 2^{-35}

RESTRICTIONS:

1. x must be greater than zero.

METHOD:

Set $x = 2^n z$ where $1/2 \leq z < 1$

then form $y = \frac{2(z - 1/2)}{(z + 1/2)}$

Then $2z = \frac{1 + \frac{y}{2}}{1 - \frac{y}{2}}$ is an algebraic identity

Expand $\ln \frac{1+\frac{y}{2}}{1+\frac{y}{2}} = P(y) = (y + a_3y^3 + a_5y^5 + a_7y^7 + a_9y^9)$

This polynomial has an associated error $\eta < 10^{-11}$ for $1/2 \leq z < 1$

then $\ln z = P(y) - \ln\sqrt{2}$

and $\ln x = P(y) - \ln\sqrt{2} - n \ln 2$

$a_3 = 0.083333329444$

$a_5 = 0.012500185911$

$a_7 = 0.002228558603$

$a_9 = 0.000464044457$

Remarks:

Note 1. If the input number satisfies $1/2 \leq x < 1$, ln x (unscaled) is left in location COM.

5.1.8 Floating Point Arithmetic Operations

Symbolic Label: FLTPT1

PURPOSE:

To perform the basic floating point arithmetic operations of addition, subtraction, multiplication, division, or normalization upon MOBIDIC Floating Point Numbers.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC Floating Point Numbers (X and X')(not necessarily normalized)

C. Output Format

Normalized MOBIDIC Floating Point Number (X'')

D. Coding Information

1. Calling Sequence for floating point addition

A) trl flpadd

A+1) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X	ACC: X'' = X + X'
QRG: X'	QRG: (Transient Quantity)

2. Calling Sequence for floating point subtraction

A) trl flpsub

A+1) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X	ACC: X'' = X - X'
QRG: X'	QRG: (Transient Quantity)

3. Calling Sequence for floating point multiplication

A) trl flpmul

A+1) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X	ACC: X'' = X * X'
QRG: X'	QRG: (Transient Quantity)

4. Calling Sequence for floating point division

A) trl flpdiv

A+1) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X	ACC: $X'' = X \div X'$
QRG: X'	QRG: (Transient Quantity)

5. Calling Sequence for floating point normalization

A) trl flpnrm

A+1) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X	ACC: Normalized X
QRG: (Irrevelant)	QRG: (Transient Quantity)

Other Subroutines Required

None

Error Halts

1. Characteristic too large during or after an arithmetic operation
2. Attempted division by zero

Sense Flip-Flops Used

None

Number of Storage Locations

220

"Common" Locations Used

13 (common thru common + 12)

Approximate Time

Average:	Addition:	0.928 millisecond
	Subtraction:	0.944 millisecond
	Multiplication:	0.754 millisecond
	Division:	0.749 millisecond
	Normalization:	0.080 millisecond if input number needs to is already normalized.
		0.368 millisecond if input number be normalized.

E. Accuracy Information

Average Error: 2^{-28}

Maximum Error: 2^{-27}

RESTRICTIONS:

1. Input numbers must be of such a size that the specified arithmetic operation produces a number less than 2^{256} in absolute value.
2. Division by zero must not be attempted.

METHOD:

1. Definition of Floating Point Terminology

   In general, a "floating point number" as used in digital computer applications consists of three quantities, two explicit and one implicit, which are representative of a single number. Any number can be expressed as the product of two numbers, one of which is an implied integral number to an explicit, positive or negative integral power. The number which is raised to a power is called the "base"; the power is called the "exponent" or "characteristic". The explicit number which the base to the power multiplies is called the "mantissa" or "fractional part". The format of a floating point number then somewhat resembles the classic one of a logarithm. For practical purposes the base is chosen to be positive, and most frequently either 10 or 2. Floating point systems based on 10 may be called "Base 10 Floating Point" or "Decimal Floating Point", while those based on 2 may be called "Base 2 Floating Point" or "Binary Floating Point".

Examples of these floating point systems are as follows:

A. Floating Point:

+48.293 = +.48293 x 100 = +.48293 x 10 = +.48293(2), where

1) +48.293 is the number to be represented,
2) +.48293 is the fractional part, and
3) 2 is the exponent (associated with the implied base 10).

B. Binary Floating Point:

+7.2 = +.9 x 8 = +.9 x 2 = +.9(3), where

1) +7.2 is the number to be represented,
2) +.9 is the fractional part, and
3) 3 is the exponent (associated with the implied base 2)

Floating point numbers, as described above, may be represented in actual computer words in several different ways. One method is to use two machine words to represent each floating point number, one for the fractional part and one for the exponent, and to allow independent algebraic signs for the words. This system can be somewhat inconvenient because it requires two machine words for each floating point number. Another method is to use a "packed" one-word floating point system, where quantities representing the fractional part and the exponent are "packed" into one word length and the exponent is always positive. This system allows fewer significant figures for each part and a smaller range of representation than the two-word system mentioned above; but is more convenient in that it requires only one machine word for each floating point number.

2. Definition of a "MOBIDIC Floating Point Number"

In MOBIDIC, a packed one-word binary floating point system is used, where quantities representing the fractional part and the exponent are packed into one MOBIDIC word. In this system, the following rules apply:

(a) The base is the number 2.

(b) The exponent is of the representation (not necessarily of the represented number) positive.

(c) The sign bit of the MOBIDIC word (bit 37) represents the algebraic sign of the floating point number being represented.

(d) The "characteristic", located in the most significant 9 bits of the MOBIDIC word (bits 36-28), represents the exponent of the number according to an "excess 256" system. In this system, the characteristic equals the exponent plus 256, and (as mentioned above) the characteristic is located in bits 36-28 of a MOBIDIC word.

Since only 9 bits are available for the characteristic, a non-zero number being represented must be of such size that the true exponent of the representation lies in the range between -256 and +255 inclusive:

$$-256 \leqq \text{exponent} \leqq +255.$$

In order to avoid negative exponents, the true exponent must be increased by 256. The characteristic of a MOBIDIC Floating Point Number then lies in the range between 0 and 511 inclusive:

$$0 \leqq \text{characteristic} \leqq +511,$$

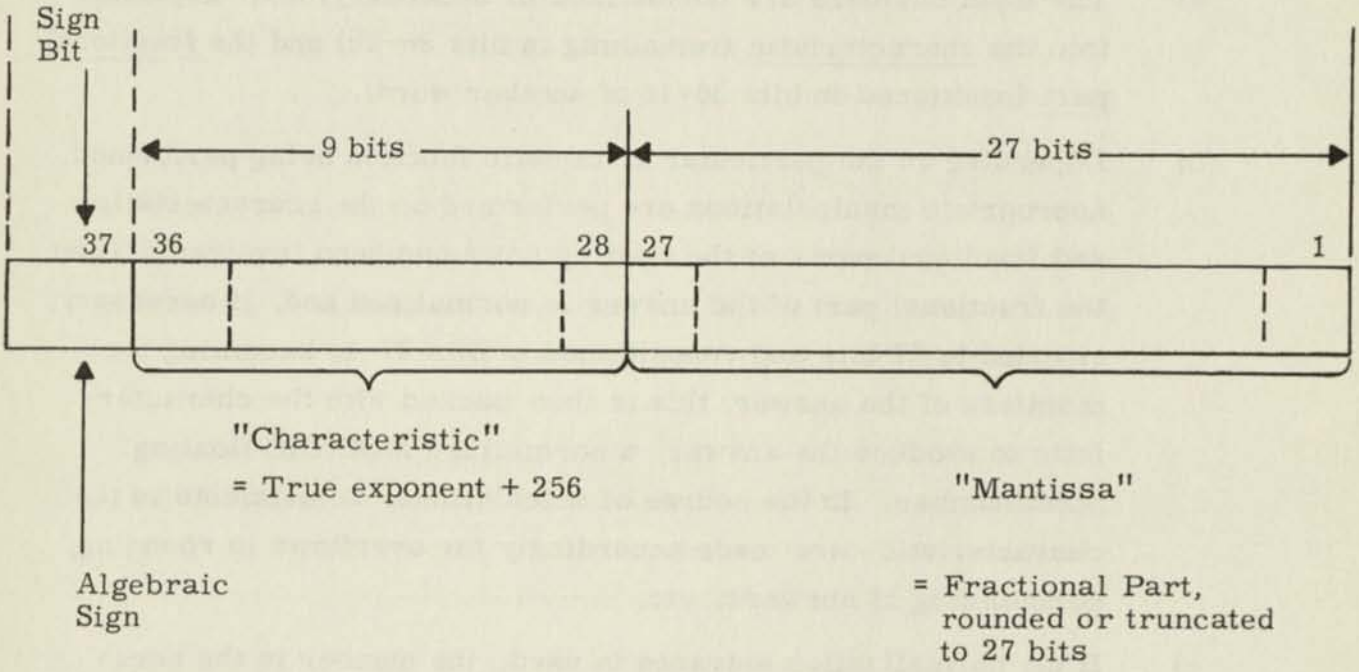
Where 0 means a true exponent of -256 and +511 means a true exponent of +255. (Note that the binary equivalent of 511 is 111111111, which just fits into 9 bits).

(e) The "mantissa", located in the remainder of the packed MOBIDIC word (bits 27-1), represents the fractional part of the number, and is less than 1. The mantissa, in fact, may be considered to be equal to the fractional part, except that it starts in bit 27 (binary point considered to be between bits 28 and 27) instead of bit 36, and is limited to 27 bits instead of 36. A fractional part

of 28 or more significant bits is necessarily truncated or rounded (usually rounded) to 27 bits in the composition of a MOBIDIC floating point number.

(f) The number zero is represented by 0's in all 36 bits of the MOBIDIC word, and usually with a 0 in the sign bit (thus +0).

(g) The system for representing MOBIDIC floating point numbers, as described above, may be shown as follows:



3. Definition of a "Normalized" MOBIDIC Floating Point Number

A "normalized" MOBIDIC Floating Point Number is one where the fractional part (if non-zero) is equal to or greater than .5 in absolute value. Thus the most significant bit (bit 27) of the mantissa of a non-zero normalized floating point number is necessarily a "1". The number zero is said to be normalized when bits 36-1 are all zero, whereas an unnormalized zero would contain 0's in the mantissa but not the characteristic. Normalized numbers are usually used because the significant bits available for the mantissa are maximized, and

normalized MOBIDIC floating point numbers may conveniently be compared for algebraic size directly, without regard to the fact that they happen to be floating point numbers.

4. Method for Floating Point Arithmetic Operations

The floating point arithmetic operations of addition, subtraction, multiplication, division, or normalization are performed by this routine as follows:

a) The input numbers are normalized (if necessary) and "unpacked" into the characteristic (remaining in bits 36-28) and the fractional part (positioned in bits 36-10 of another word).

b) Depending on the particular arithmetic function being performed, appropriate manipulations are performed on the characteristics and fractional parts of the floating point numbers involved. Then the fractional part of the answer is normalized and, if necessary, rounded to 27 bits and repositioned in bits 27-1, becoming the mantissa of the answer; this is then packed with the characteristic to produce the answer, a normalized MOBIDIC floating point number. In the course of calculations, adjustments in the characteristic are made accordingly for overflows in rounding, normalizing of answers, etc.

c) If the normalization entrance is used, the number in the accumulator is normalized, without regard to the possible contents of the Q-Register, which are not preserved.

d) If a characteristic greater than + 511 (including an exponent greater than + 255) arises in the course of the calculations, an error halt occurs indicating the characteristic, and thus the floating point number itself, is too large.

e) If a negative characteristic (indicating a true exponent algebraically less than - 256) arises in the course of the calculations, the answer is set to zero, represented by a MOBIDIC word of all Q's including the sign bit (thus +0).

REMARKS:

Input numbers X and X' need not necessarily be normalized; however, the routine is faster if they are. Except for the normalization entrance, the times given above are based on normalized input numbers. In any event, the answer X'' is always normalized.

5.1.9 "Unfloat" Conversion Routine

Symbolic Label: UNFLT1

PURPOSE:

To compute the normalized fraction and exponent equivalent for a MOBIDIC floating point number.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number

C. Output Format

1. Normalized fraction (fractional part)
2. Fixed point integer (exponent)

D. Coding Information

1. Calling Sequence

A) TRL UNFLTN

A + 1) ◄——— Normal Return

2. Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: MOBIDIC Floating Point No.	ACC: Fractional Part
QRG: (Irrelevent)	QRG: Exponent

3. Other Sub routines Required

None

4. Error Halts

None

5. Sense Flip Flops Used

None

6. Storage Requirements

a. Subroutine: 13 locations

b. Common Storage: 2 (Common, Common + 1)

7. Approximate Time

a. Maximum: .230 milliseconds

b. Average: .198 milliseconds

E. Accuracy Information

This subroutine unfloats a MOBIDIC floating point number without introduction of error.

RESTRICTIONS:

None

METHOD:

1. The exponent is shifted into bits 36-27 of the Q Register by a cycle short 9 and shift long right 9.

2. The accumulator (fractional part) is normalized and stored. The number of shifts for normalizing is stored.

3. |Exponent|—> Bits 9-1 of the accumulator. 247 (256 modulus -9) and the normalizing count are subtracted from the fraction.

4. The computed exponent is stored in the Q Register and normalized fraction stored in the accumulator

REMARKS:

Note 1. If a zero value floating point number is used as an argument, the accumulator will contain zero and the Q Register the reset of the input value of $|C(AC)_{36-27}|$ -27.

Note 2. See the writeup of "Floating Point Arithmetic Operations for a description of MOBIDIC floating point numbers".

5.1.10 "Float" Conversion Routine

Symbolic Label: FLOAT1

PURPOSE:

To produce a normalized MOBIDIC floating point number, given a fixed point "fractional part" and "exponent". (The exponent has an implied base of 2.)

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

Fixed point "fractional part" and "exponent"

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

Calling Sequence

A) trl floatn

A+1) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: Fractional Part	ACC: Normalized MOBIDIC Floating Point Number
ORG: Exponent	ORG: (Transient Quantity)

Other Subroutines Required

None

Error Halts

Too large a characteristic (>511) in the resultant normalized MOBIDIC floating point number.

Sense Flip-Flops Used

None

Number of Storage Locations

28

"Common" Locations Used

2 (common and common + 1)

Approximate Time

Average: 0.27 millisecond

Maximum: 0.31 millisecond

E. Accuracy Information

Average Error: 2^{-28} in mantissa

Maximum Error: 2^{-27} in mantissa (from roundoff)

RESTRICTIONS:

The fractional part and exponent must be of such a size (e.g. exponent < + 256) that the characteristic of the resultant normalized MOBIDIC floating point number is no greater than 511.

METHOD:

1. The fractional part is normalized and rounded to 27 bits, becoming the mantissa of the MOBIDIC floating point number, retaining the algebraic sign.
2. The number 256 minus the number of shifts from normalizing, plus the given exponent, produces the characteristic of the MOBIDIC floating point number which is packed with the mantissa to produce the desired answer.
3. If the fractional part is zero, the answer is zero with the same sign.
4. If the characteristic is negative, the answer is set to +0.

5. If it is desired to float a regular fixed point fractional number, this number should be in the Accumulator at entry, with the Q-Register containing zero.

REMARKS:

1. If overflow occurs in rounding, the mantissa is set to 777777777_8 with the proper sign.

2. If the characteristic is greater than 511, the answer is set to 777777777777_8 with the proper sign, and an error halt occurs. If "Start at PC" is activated, control is returned to the using program with this answer in ACC.

3. See the writeup of "Floating Point Arithmetic Operations" for a full description of a "normalized MOBIDIC floating point number", and of the terms "fractional part", "exponent", "mantissa", and "characteristic".

5.1.11 Floating Point Square Root

Symbolic Label: FSQRT1

PURPOSE:

To compute the square root of a floating point number N, where $N \geqq 0$.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

Calling Sequence

A) trl SQRTFL

A + 1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: N	ACC: $\sqrt{N}$

Other Subroutines Required

None

Error Halts

N<0

Sense Flip-Flops Used

None

Number of Storage Locations

68

"Common" Locations Used

5 (common thru common + 4)

Approximate Time

Average: 1.18 milliseconds

Maximum: 1.22 milliseconds

E. Accuracy Information

Average Error: 2^{-28} in mantissa

Maximum Error: 2^{-27} in mantissa

RESTRICTIONS:

N must be positive or zero.

METHOD:

1. The floating point number is unfloated into its fractional part and exponent. The fractional part is shifted right 0 or 1 place to make the exponent even.

2. The exponent of the answer is half of the exponent of the input number (adjusted by 1 bit if necessary).

3. The square root of the fractional part is computed using the method of the fixed point square root routine.

4. The square root of the fraction and the exponent of the answer are "floated" to obtain the answer.

5. If the input number is zero, the answer is zero.

REMARKS:

See the writeup of "Floating Point Arithmetic Operations" for a full description of "MOBIDIC floating point number", "fractional part", "exponent", and "mantissa".

5.1.12 Title: Floating Point Sine-Cosine

Both arg & fnc'tn in FL PT

Symbolic Label: FSNCS1

PURPOSE:

To compute the sine or cosine of a floating point angle X expressed in radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

1. Calling Sequence for floating point sine, radians

A) trl flsinr

A+1)◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in radians	ACC: sin X

2. Calling Sequence for floating point sine, degrees

A) trl flsind

A+1)◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in degrees	ACC: sin X

3. Calling Sequence for floating point cosine, radians

A) trl flcosr

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in radians	ACC: cos X

4. Calling Sequence for floating point cosine, degrees

A) trl flcosd

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in degrees	ACC: cos X

Other Subroutines Required

Label	Title
FLOAT1	"FLOAT" Conversion Routine

→ UNFLOAT?

Error Halts

$|X|$ (in circles) $\geq N_{max}$; $N_{max} = 2^{18}$ circles (see RESTRICTIONS)

Sense Flip-Flops Used

None

Number of Storage Locations

158

"Common" Locations Used

8 (common thru common + 7)

Approximate Time

Average: 2.15 milliseconds

Maximum: 2.35 milliseconds

E. Accuracy Information

Average Error: 2^{-27} in mantissa

Maximum Error: 2^{-26} in mantissa

RESTRICTIONS:

The input number X must be less than N_{max} circles in absolute value, for the result to have significant accuracy. N_{max} may be changed by the programmer if desired. Its value is 2^{18} circles as given in the routine.

METHOD:

1. The input number is unfloated, then converted from radians or degrees to "circles", using a double precision constant for the conversion, to avoid introducing any appreciable error in the conversion.
2. Any extra circles (corresponding to an input number larger than 360 degrees or 2π radians) are eliminated.
3. For the cosine entrances, 1/4 of a circle is added, corresponding to the identity: $\cos \theta = \sin (\theta + \pi/2.)$
4. If the number is very small, the answer is obtained by converting to radians and using a 2-term Taylor series for the sine.
5. Otherwise, the number is scaled to a fixed point fraction, and the method of the fixed point sine routine used.
6. The answer is "floated" to produce a normalized MOBIDIC floating point number.

REMARKS:

See writeup of "Floating Point Arithmetic Operations" for a full description of the format of a "MOBIDIC floating point number".

5.1.13 Title: Floating Point Tangent-Cotangent

Symbolic Label: FTNCT1

Both arg & fnctn in FL PT

PURPOSE:

To compute the tangent or cotangent in floating point of an angle X expressed in radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

1. Calling Sequence for floating point tangent, radians

A) trl fltanr

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in radians	ACC: tan X

2. Calling Sequence for floating point tangent, degrees

A) trl fltand

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in degrees	ACC: tan X

3. Calling Sequence for floating point cotangent, radians

A) trl flctnr

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in radians	ACC: ctn X

4. Calling Sequence for floating point cotangent, degrees

A) trl flctnd

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: X in degrees	ACC: ctn X

Other Subroutines Required

Label	Title
TNCT1	Fixed Point Tangent, Cotangent
FLOAT1	"Float" Conversion Routine

Error Halts

1. $|X|$ (in semicircles) $\geq N_{max}$; $N_{max} = 2^{18}$ semicircles (see RESTRICTIONS).
2. Tangent of $[\pm(2n + 1)\pi/2]$ or Cotangent of $[\pm\pi n]$, $n = 0, 1, \cdots$ or angles so close to these that the corresponding tangent or cotangent is $\geq 2^{256}$ in absolute value.

Sense Flip-Flops Used

None

Number of Storage Locations

183

"Common" Locations Used

13 (common thru common + 12)

Approximate Time

Average: 2.86 milliseconds

Maximum: 3.23 milliseconds

E. Accuracy Information

Average Error: 2^{-27} in mantissa

Maximum Error: 2^{-26} in mantissa, except for X near 90^{o}, 180^{o}, 270^{o}, and 360^{o} (or radian equivalents)

RESTRICTIONS:

The input number x must be less than N_{max} semicircles in absolute value, for the result to have the significant accuracy. N_{max} may be changed by the programmer if desired. Its value is 2^{18} semicircles as given in the routine.

The input number x must not be so close to odd or even multiples of $\frac{\pi}{2}$ that the corresponding tangent or cotangent is an extremely large number, viz equal to or greater than 2^{256} in absolute value.

METHOD:

1. The input number X is unfloated, then converted from radians or degrees to semicircles, using a double precision constant for the conversion, to avoid introducing any appreciable error in the conversion.

2. Any extra semicircles (corresponding to an input number larger than 180 degrees or π radians) are eliminated.

3. The tangent or cotangent of $|X|$ is obtained in accordance with the following table, where $Y = |X/\pi|$, and $Z = \pi Y$:

Y	Z	Tangent X	Cotangent X
= 0	0	0	Undefined (Error Halt)
0 —> .25	0 —> $\pi/4$	tan Z	1/tan Z
= .25	= $\pi/4$	1.0	1.0
.25 —> .5	$\pi/4$ —> $\pi/2$	1/ctn Z	ctn Z
= .5	= $\pi/2$	Undefined (Error Halt)	0
.5 —> .75	$\pi/2$ —> $3\pi/4$	-1/tan (Z - $\pi/2$)	-tan (Z - $\pi/2$)
= .75	= $3\pi/4$	-1.0	-1.0
.75 —> 1.0	$3\pi/4$ —> π	-ctn (Z - $\pi/2$)	-1/ctn (Z - $\pi/2$)

4. The fixed point tangent-cotangent routine is used for computing tan Z or ctn Z, except for tan Z where Z is very small, where a 2-term Taylor series is used instead.

5. If X is negative, the sign of the answer is changed.

6. The answer is "floated" to produce a normalized MOBIDIC floating point number. If the characteristic is too large, an error halt occurs (see Restrictions and Error Halts).

REMARKS:

See writeup of "Floating Point Arithmetic Operations" for a full description of the format of "MOBIDIC floating point number".

5.1.14 Title: Floating Point Arcsine-Arccosine

Symbolic Label: FLASC1

PURPOSE:

To compute the arcsine or arccosine of a floating point number x, and give the result in radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

1. Calling Sequence for arcsine, radians

A) trl flasnr

A+1) <———————— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: arcsin x in radians

2. Calling Sequence for arcsine, degrees

A) trl flasnd

A+1) <———————— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: arcsin x in degrees

3. Calling Sequence for Arccosine, radians

A) trl flacsr

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: arccos x in radians

4. Calling Sequence for arccosine, degrees

A) trl flacsd

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: arccos x in degrees

Other Subroutines Required

Label	Title
ARCSC1	Fixed Point Arcsine, Arccosine
FLOAT1	"Float" Conversion Routine

Error Halts

$|x| > 1$

Sense Flip-Flops Used

None

Number of Storage Locations

133

"Common" Locations Used

9 (common through common + 8)

Approximate Time

Average: 3.05 milliseconds

Maximum: 3.21 milliseconds

E. Accuracy Information

Average Error: 2^{-25}

Maximum Error: 2^{-21}

RESTRICTIONS:

The input number x must be less than or equal to one (1.0) in absolute value.

METHOD:

1. For Arccosine, the number X is converted to a fixed point number, the arccosine is computed using the fixed point arcsine-arccosine routine, and the answer is "floated".
2. For Arcsine, for numbers other than very small ones, the number X is converted to a fixed point number, the arcsine is computed using the fixed point arcsine-arccosine routine, and the answer is "floated".
3. For the Arcsine of very small numbers, a 2-term Taylor series is used to compute the arcsine in radians; this number is converted to degrees if necessary, and "floated" to produce the desired answer.

REMARKS:

Values of the answer are in the first or third quadrant (as positive numbers) for the arcsine, and in the first or second quadrants (as positive numbers) for the arccosine.

See the writeup of "Floating Point Arithmetic Operations" for a full description of "MOBIDIC floating point numbers".

5.1.15 Title: Floating Point Arctangent-Arccotangent

Symbolic Label: FLATC1

PURPOSE:

To compute the arctangent y/x or the arccotangent y/x in floating point and to give the result in either radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

1. Calling Sequence for floating point arctangent, radians

A) trl flatnr

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: arctan $\frac{y}{x}$
ORG: x	ORG: (Transient Quantity)

2. Calling Sequence for floating point arctangent, degrees

A) trl flatnd

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: arctan $\frac{y}{x}$ in degrees
ORG: x	ORG: (Transient Quantity)

3. Calling Sequence for floating point arccotangent, radians

A) trl flactr

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: arccotan $\frac{y}{x}$ in radians
ORG: x	ORG: (Transient Quantity)

4. Calling Sequence for floating point arccotangent, degrees

A) trl flactd

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: arccotan $\frac{y}{x}$ in degrees
ORG: x	ORG: (Transient Quantity)

Other Subroutines Required

Label	Title
ARCTC1	Fixed Point Arctangent, Arccotangent
UNFLT1	"Unfloat" Conversion Routine
FLOAT1	"Float" Conversion Routine

Error Halts

y = x = 0

Sense Flip-Flops Used

None

Number of Storage Locations

55

"Common" Locations Used

8 (common thru common + 7)

Approximate Time

Average: 2.94 milliseconds

Maximum: 3.15 milliseconds

E. Accuracy Information

Average Error: Exact

Maximum Error: 2^{-27} in mantissa

RESTRICTIONS:

The input numbers y and x cannot both be zero.

METHOD:

1. Y and X are unfloated. The fractional part of the smaller of these is scaled by the difference in the exponents.
2. Then the fixed point arctangent routine is used to find the arctangent of the fractional parts of Y and X, in either degrees of radians.
3. The answer is then "floated" to produce a normalized MOBIDIC floating point number for Arctan Y/X.
4. The arccotangent is obtained by interchanging Y and X and proceeding as above.

REMARKS:

For either arctangent or arccotangent, the answer is in the proper quadrant, depending upon the signs of y and x. The angle will be expressed as a positive number from 0 to 2π radians or from 0 to 360 degrees.

See writeup of "Floating Point Arithmetic Operations" for a full description of the format of a "MOBIDIC floating point number".

5.1.16 Title: Floating Point Exponential

Symbolic Label: **FLEXP1**

PURPOSE:

To compute e^x in floating point

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

Calling Sequence

A) trl **FLEXPN**

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: e^x

Other Subroutines Required

Label	Title
FLOAT1	"Float" Conversion Routine

Error Halts

x > 176.752531

Sense Flip-Flops Used

None

Number of Storage Locations

226

"Common" Locations Used

17 (common through common + 16)

Approximate Time

Average: 5.80 milliseconds

Maximum: 6.02 milliseconds

E. Accuracy Information

Average Error: 2^{-28} in mantissa

Maximum Error: 2^{-27} in mantissa

RESTRICTIONS:

X must be no greater than +176.752531; otherwise e^x would be too large to be represented as a MOBIDIC floating point number.

METHOD:

$x = i \log_e 2 + f \log_e 2$ i is the integer, f is the fraction

$z = f \cdot \log_e 2$

$$e^z = 1 + \frac{2z}{2 - z + \frac{z2}{20} + \frac{98z2}{20(42 + z2)}}$$

$e^x = 2^i e^z$

$e^{-x} = \frac{1}{e^x}$

These calculations are performed in floating point.

For $x < -178.1388254$, e^x is set to +0.

REMARKS:

See writeup of "Floating Point Arithmetic Operations" for a full description of the format of a "MOBIDIC floating point number".

5.1.17 Title: Floating Point Natural Logarithm

Symbolic Label: FLLOG1

PURPOSE:

To compute $\log_e x$ for a floating point number $x > 0$.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point number (not necessarily normalized)

C. Output Format

Normalized MOBIDIC floating point number

D. Coding Information

Calling Sequence

A) trl fllogn

A+1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: x	ACC: $\log_e x$

Other Subroutines Required

Label	Title
LOG1	Fixed Point Natural Logarithm

UNFLT1 — "Unfloat" Conversion Routine

FLOAT1 — "Float" Conversion Routine

Error Halts

$x \leqq 0$

Sense Flip-Flops Used

None

Number of Storage Locations

94

"Common" Locations Used

9 (common thru common + 8)

Approximate Time

Average: 2.54 milliseconds

Maximum: 2.75 milliseconds

E. Accuracy Information

Average Error: 2^{-27}

Maximum Error: 2^{-24} (near x = 1.0)

RESTRICTIONS:

x must be positive and non-zero.

METHOD:

1. The input number X is unfloated, producing $F \cdot 2^N$.
2. LOG F is computed using the fixed point natural logarithm routine.
3. Then,

$$\text{LOG } X = \text{LOG } F + N \cdot \text{LOG } 2$$

4. The answer is "floated" to produce a normalized MOBIDIC floating point number.

5. If X = 1.0, the answer is set to zero.

REMARKS:

See writeup of "Floating Point Arithmetic Operations" for a full description of the format of a "MOBIDIC floating point number". If $\log_a x$ is desired, the answer should be multiplied by $\log_a e$.

5.1.18 Fixed Point Double Precision Arithmetic Operations

Symbolic Label: DBLPR1

PURPOSE:

To perform the basic arithmetic operations of addition, subtraction, multiplication, and division upon fixed point double precision numbers.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

The first operand $x = x_1 + x_2 \cdot 2^{-36}$ has the most significant part (x_1) in the Accumulator and the least significant part (x_2) in the Q-register at entry. The second operand $y = y_1 + y_2 \cdot 2^{-36}$ is stored under the "trl" order, with the most significant part (y_1) immediately beneath the "trl" order, and the least significant part (y_2) just below that. Both parts have the sign of the double precision number.

C. Output Format

The result (r) has the most significant part (r_1) placed in the Accumulater and the least significant part (r_2) in the Q-Register at exit. Both parts have the sign of the double precision number.

D. Coding Information

1. Calling Sequence for Double Precision Addition

A) trl dpadd

A+1) (y_1)

A+2) (y_2)

A+3) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions	
ACC: (x_1)	ACC: (r_1)	$r = x + y$
ORG: (x_2)	ORG: (r_2)	

2. Calling Sequence for Double Precision Subtraction

A) trl dpsub
A+1) (y_1)
A+2) (y_2)
A+3) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions	
ACC: (x_1)	ACC: (r_1)	$r = x - y$
ORG: (x_2)	ORG: (r_2)	

3. Calling Sequence for Double Precision Multiplication

A) trl dpmul
A+1) (y_1)
A+2) (y_2)
A+3) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions	
ACC: (x_1)	ACC: (r_1)	$r = x \cdot y$
ORG: (x_2)	ORG: (r_2)	

4. Calling Sequence for Double Precision Division

A) trl dpdiv
A+1) (y_1)
A+2) (y_2)
A+3) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: (x_1)	ACC: (r_1)
ORG: (x_2)	ORG: (r_2)

$r = x \div y$

Other Subroutines Required

None

Error Halts

Overflow in addition, subtraction, or division with no provisions for overflow made by programmer. (See REMARKS: "Provisions for Overflow")

Sense Flip-Flops Used

3 (SFF14 thru SFF16)

Number of Storage Locations

323

"Common" Locations Used

14 (common thru common + 13)

Approximate Time

0.57 millisecond for addition or subtraction
1.21 milliseconds for multiplication
3.16 milliseconds for division

E. Accuracy Information

Average Error: 2^{-72} for all entrances

Maximum Error: 1. 2^{-72} for addition, subtraction or multiplication
2. 2^{-71} for division

RESTRICTIONS:

Input numbers must have the same sign for both the most significant and the least significant parts of the double precision number. In case of overflow, an error halt will occur unless the programmer provides for such an eventuality - see **REMARKS**: "Provisions for Overflow".

METHOD:

Notation: First operand = $x = x_1 + x_2 \cdot 2^{-36}$

Second operand = $y = y_1 + y_2 \cdot 2^{-36}$

Result (answer) = $r_1 + r_2 \cdot 2^{-36}$

Addition: $r_2 = x_2 + y_2$

$r_1 = x_1 + y_1$ + (overflow from r_2, if any)

r_1 and r_2 are adjusted to have the same signs (if necessary)

Subtraction: y is replaced by -y and double precision addition is used.

Multiplication:

Subscript notation:

H = 36 High order bits of 72-bit product

L = 36 Low order bits of 72-bit product

"Rounder" = $(x_1y_2)_L + (x_2y_1)_L + (x_2y_2)_H$

$r_2 = (x_1y_1)_L + (x_1y_2)_H + (x_2y_1)_H$ + (overflows from "Rounder", if any) + 2^{-35} · (Rounder)

$r_1 = (x_1y_1)_H$ + (overflows from r_2, if any)

"Rounder" is lost during calculations; thus a 72-bit rounded product is the only result.

Division: The general case of division is handled by the following interpolation method:

$$r = \frac{x_1 + x_2 \cdot 2^{-36}}{y_1} - y_2 \left[\frac{x_1 + x_2 \cdot 2^{-36}}{y_1} - \frac{x_1 + x_2 \cdot 2^{-36}}{(y_1 + 2^{-36})} \right]$$

The various special cases which may occur are handled appropriately. In case of overflow in addition, subtraction, or division, the double precision operations subroutine handles the overflow as discussed under REMARKS: "Provisions for Overflow".

REMARKS:

PROVISIONS FOR OVERFLOW

The double precision operations subroutine performs its various operations upon fixed point numbers represented by 72 bits (plus sign) instead of 36 bits. Therefore, it is quite possible that overflow may occur, just as in using the basic machine capability of fixed point single precision. It is also quite likely that the programmer may wish to do something constructive in case of overflow in double precision addition, subtraction or division, rather than necessarily stopping on an error halt. For this reason, the double precision operations subroutine will take care of any overflow which may occur provided that the programmer senses the overflow alarm flip-flop with a SEN, SNR, or SNS order as the very next order in his program. If he does so, the overflow alarm flip-flop will be reset at exit if no overflow occurs, but will be set if overflow does occur. If overflow occurs in addition or subtraction, the operation is finished with the overflow bit lost, but with the result having the appropriate sign. If overflow occurs in division, the division does not take place and the original dividend (x) is in the Accumulator (x_1) and Q-Register (x_2) at exit. If overflow occurs and the programmer does not sense the overflow alarm flip-flop with the very next oder in his program, an error halt will occur.

5.1.19 Floating Point Double Precision Arithmetic Operations

Symbolic Label: FLPDP1

PURPOSE:

To perform the basic arithmetic operations of addition, subtraction, multiplication, division, and normalization upon MOBIDIC floating point double precision (69 bits) numbers.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

MOBIDIC floating point double precision numbers x and x' (not necessarily normalized) (See Method)

C. Output Format

Normalized MOBIDIC floating point double precision number x" (See Method)

D. Coding Information

1. Calling Sequence for Floating Point Double Precision Addition

A) trl fdpadd

A+1) }
A+2) } x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: }	x	ACC: }	x" = x +
ORG: }		ORG: }	

2. Calling Sequence for Floating Point Double Precision Subtraction

A) trl fdpsub

A+1) }
A+2) } x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: }	x	ACC: }	x'' = x - x'
ORG: }		ORG: }	

3. Calling Sequence for Floating Point Double Precision Multiplication

A) trl fdpmul

A+1) }
A+2) } x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: }	x	ACC: }	x'' = x . x'
ORG: }		ORG: }	

4. Calling Sequence for Floating Point Double Precision Division

A) trl fdpdiv

A+1) }
A+2) } x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC:	x	ACC:	$x'' = x \div x'$
ORG:		ORG:	

5. Calling Sequence for Floating Point Double Precision Normalization

A) trl fdpnrm

A+1) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC:	x	ACC:	x'' = Normalized x
ORG:		ORG:	

Other Subroutines Required

Label	Title
DBLPR1	Fixed Point Double Precision Arithmetic Operations

Error Halts

1. Characteristic too large during or after an arithmetic operation.
2. Attempted division by zero.
3. Input numbers have different signs for both parts.

Sense Flip-Flops Used

1 (SFF 13)

Number of Storage Locations

345

"Common" Locations Used

26 (COMMON thru COMMON + 25)
(includes common thru common + 13 for fixed point double precision arithmetic operations)

Approximate Time

2.8 milliseconds for addition or subtraction
2.9 milliseconds for multiplication
5.0 milliseconds for division
0.4 millisecond for normalization

E. Accuracy Information

Average Error: 2^{-63}

Maximum Error: 2^{-63}

RESTRICTIONS:

1. The input numbers must have the same sign for both parts of each number.
2. The input numbers must be of such a size that the specified arithmetic operation produces a number less than 2^{256} in absolute value.
3. Division by zero must not be attempted.

METHOD:

1. On MOBIDIC a "packed" floating point double precision system to the base 2 is used, where numbers representing the fractional part and the exponent are both "packed" into two MOBIDIC words. The rules by which each floating point precision number is represented in packed form are as follows:
   a) The sign bit of the MOBIDIC word (bit 37) represents the algebraic sign of the floating point double precision number. The signs of both words agree.
   b) The most significant 9 bits of the first MOBIDIC word (bits 36-28) are used for the "characteristic", which equals the exponent +256. The exponent must lie in the range between -256 and +255 inclusive; thus the characteristic must lie in the range from zero to +511.

c) The least significant 27 bits of the first MOBIDIC word (bits 27-1) plus the second word (bits 36-1) are used for the "mantissa", equivalent to the fractional part of the floating point double precision number.

d) The number zero is represented by 0's in all 36 bits of both MOBIDIC words and usually with a 0 in the sign bit (thus +0).

e) If a characteristic greater than +511 (indicating a true exponent greater than +255) arises in the course of calculations, an error halt occurs indicating that the characteristic and thus the floating point double precision number itself, is too large.

f) If a negative characteristic (indicating a true exponent algebraically less than -256) arises in the course of calculations, the answer is set to zero.

g) A MOBIDIC floating point double precision number (if non-zero) is said to be "normalized" if the fractional part $\geq .5$ in absolute value; this also implies that the most significant bit of the "mantissa" is a "one".

2. The method by which floating point double precision operations are performed by this subroutine is as follows:

a) The input numbers are normalized (if necessary) and unpacked into the characteristic and mantissa or fractional part.

b) The necessary computation on the characteristic is performed within this subroutine. The arithmetic operations carried out on the mantissa, or fractional part, are performed by the fixed point double precision subroutine.

c) The answer is normalized and packed according to the rules for a normalized MOBIDIC floating point double precision number, and is stored in the Accumulator and Q-Register before control is returned to the main program.

REMARKS:

Note 1. Note that when the normalization entrance is used, there is no second number x' involved, and control is returned to the line immediately below the "trl" order.

Note 2. See the writeup of "Floating Point Arithmetic Operations" for a discussion of floating point systems in general if desired.

5.1.20 Fixed Point Complex Number Operations

Symbolic Label: CMPLX1

PURPOSE:

To perform the basic operations of addition, subtraction, multiplication, and division upon fixed point complex numbers.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

The first operand x = a + bi has the real part (a) in the Accumulator and the imaginary part (b) in the Q-register at entry. The second operand x' = c + di is stored under the "trl" order, with the real part (c) immediately beneath the "trl" order, and the imaginary part (d) just below the real part.

C. Output Format

The complex answer x" = e + fi has the real part (e) in the Accumulator and the imaginary part (f) in the Q-register at exit.

D. Coding Information

1. Calling Sequence for Complex Number Addition

A) trl cpxadd

A+1) (c)
A+2) (d) } x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x + x'$
ORG: (b)		ORG: (f)	

2. Calling Sequence for Complex Number Subtraction

A) trl cpxsub

A+1) (c)
A+2) (d) x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x - x'$
ORG: (b)		ORG: (f)	

3. Calling Sequence for Complex Number Multiplication

A) trl cpxmul

A+1) (c)
A+2) (d) } x'

A+3) ◄——— Returns control here

Entry And Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x \cdot x'$
ORG: (b)		ORG: (f)	

4. Calling Sequence for Complex Number Division

A) trl cpxdiv

A+1) (c)
A+2) (d) } x'

A+3) ◄——— Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x \div x'$
ORG: (b)		ORG: (f)	

Other Subroutines Required

None

Error Halts:

1. Overflow in addition for either the real or the imaginary parts.
2. Overflow in subtraction for either the real or imaginary parts.
3. Overflow in multiplication for either the real or imaginary parts.
4. Overflow in division for either the real or the imaginary parts.

Sense Flip-Flops Used

2 (SFF15 and SFF16)

Number of Storage Locations

139

"Common" Locations Used

5 (COMMON thru COMMON +4)

Approximate Time

0.3 milliseconds for addition or subtraction
0.8 millisecond for multiplication
2.0 milliseconds for division

E. Accuracy Information

Average Error: 2^{-36} for all entrances

Maximum Error: 1. 2^{-36} for addition, subtraction and multiplication

2. 2^{-35} for division

RESTRICTIONS:

The numbers should be scaled so that an overflow will not occur.

METHOD:

Notation: First operand = $x = a + bi$

Second operand = $x' = c + di$

Answer = $x'' = e + fi$

Addition: $e = a + c$; $f = b + d$

Multiplication: $e = ac = bd$; $f = bc + ad$

Division: $e = \frac{ac + bd}{c^2 + d^2}$; $f = \frac{bc - ad}{c^2 + d^2}$

Note: if $c^2 + d^2 \geq 1.0$, the division is scaled as follows:

$$e = \frac{1}{2}\left\{\frac{ac + bd}{\frac{1}{2}(c^2 + c^2)}\right\} \text{ or } \left\{\frac{\frac{1}{2}(ac + bd}{\frac{1}{2}(c^2 + d^2)}\right\}$$

$$f = \frac{1}{2}\left\{\frac{bc - ad}{\frac{1}{2}(c^2 + d^2)}\right\} \text{ or } \left\{\frac{\frac{1}{2}(bc - ad)}{\frac{1}{2}(c^2 + d^2)}\right\}$$

Which of these methods above is used depends on whether or not $(ac + bd)$ or $(bc - ad)$ happens to be less than $\frac{1}{2}(c^2 + d^2)$ in absolute value.

REMARKS:

Note 1. If a problem involving complex number operations is too difficult to scale effectively, the routine for Floating Point Complex Number Arithmetic Operations (FCPLX1) may be used.

5.1.21 Title: Floating Point Complex Number Arithmetic Operations

Symbolic Label: FCPLX1

PURPOSE:

To perform the basic arithmetic operations of addition, subtraction, multiplication, and division upon single precision floating point complex numbers.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

Two floating point complex numbers of the form $x = a + bi$ and $x' = c + di$, where a, b, c, and d are MOBIDIC single precision floating point numbers (not necessarily normalized).

C. Output Format

A floating point complex number of the form $x'' = e + fi$, where e and f are normalized single precision MOBIDIC floating point numbers.

D. Coding Information

1. Calling Sequence for floating point complex number addition

A) trl fcxadd

A+1) (c)
A+2) (d) } x'

A+3) ← Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a) ORG: (b)	x	ACC: (e) QRG: (f)	$x'' = x + x'$

2. Calling Sequence for floating point complex number subtraction

A) trl fcxsub

A+1) (c)
A+2) (d) } x'

A+3) ← Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x - x'$
ORG: (b)		ORG: (f)	

3. Calling Sequence for floating point complex number multiplication

A) trl fcxmul

A+1) (c)
A+2) (d) } x'

A+3) ← Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x \cdot x'$
ORG: (b)		ORG: (f)	

4. Calling Sequence for floating point complex number division

A) trl fcxdiv

A+1) (c)
A+2) (d) } x'

A+3) ← Returns control here

Entry and Exit Conditions

Entry Conditions		Exit Conditions	
ACC: (a)	x	ACC: (e)	$x'' = x \div x'$
ORG: (b)		ORG: (f)	

Other Subroutines Required

Label	Title
FLTPT1	Floating Point Arithmetic Operations

Error Halts

1. Attempted division by zero ($c^2 + d^2 = 0$).
2. Characteristic of floating point number too large during or after an arithmetic operation, for either the real or imaginary parts. This error halt is located in the floating point operations subroutine (FLTPT1), which is used with this routine.

Sense Flip-flops Used

None

Number of Storage Locations

104

"Common" Locations Used

11 (common + 13 thru common + 23)

Approximate Time

Average:

Addition:	2.32 milliseconds
Subtraction:	2.34 milliseconds
Multiplication:	5.83 milliseconds
Division:	9.68 milliseconds

E. Accuracy Information

Average Error: 2^{-27}

Maximum Error: 2^{-26}

RESTRICTIONS:

1. Input numbers must be of such a size that the specified arithmetic operation produces a number, for each part, real and imaginary, which is less than 2^{256} in absolute value.
2. Division by zero (c and d both zero) must not be attempted.

METHOD:

Notation: First Operand = X = a + bi
Second Operand = X' = c + di
Answer = X''= e + fi

Addition: e = a + c; f = b + d

Subtraction: X' is replaced by -X' and floating point complex addition is used.

Multiplication: e = ac - bd; f = bc + ad

Division: $e = \frac{ac + bd}{c^2 + d^2}$; $f = \frac{bc - ad}{c^2 + d^2}$

Note: ac + bd and bc - ad are computed by replacing d by -d and then using part of the floating point complex multiplication.

REMARKS:

Note 1. See the writeup of "Floating Point Arithmetic Operations" for a full description of the format of a "MOBIDIC floating point number".

5.1.22 Fixed Point Polar-to-Cartesian Coordinate Conversion

Symbolic Label: PLCRT1

PURPOSE:

To compute fixed point binary fraction values for cartesian coordinates x and y given polar coordinate θ (expressed as radians or degrees) and r.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

1a. θ (degrees) - Fixed point binary mixed number (scale B-9)

1b. θ (Radians) - Fixed point binary mixed number (scale B-3)

2. r - Fixed point Fraction (Scale B0)

C. Output Format

x and y as fixed point binary fractions (Scale B0)

D. Coding Information

1a. Calling Sequence (θ input in degrees)

A) TRL PLCRTD

A+1) ◄——— Normal Return

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: θ (Scaled B-9)	ACC: y (Scale B0)
ORG: r (Scaled B0)	ORG: x (Scale B0)

1b. Calling Sequence (θ Input in Radians)

A) TRL PLCRTR

A+1) ◄——— Normal Return

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: θ (Scaled B-3)	ACC: y (Scaled B0)
ORG: r (Scaled B0)	ORG: x (Scaled B0)

2. Other Subroutines Required

Label	Title
SNCS1	Fixed Point Sine, Cosine

3. Error Halts

None

4. Sense Flip-flops Used

None

5. Number of Storage Locations

23

6. "Common" Locations Used

6 (common thru common +5)

7. Approximate Time

Average: 2.86 millisecond

Maximum: 3.13 millisecond

E. Accuracy Information

Average Error: 2^{-36}

Maximum Error: 2^{-34}

RESTRICTIONS:

θ must be $< 360^{o}$ or 2n radians (See "Error Halts")

METHOD:

1. Compute Sine θ and Cosine θ
2. y = r Sine θ
3. x = r cosine θ

REMARKS:

None

5.1.23 Title: Fixed Point Cartesian-to-Polar Coordinate Conversion

Symbolic Label: CRTPL1

PURPOSE:

To compute polar coordinates θ and r/2, given fixed point cartesian coordinates y and x; θ is to be expressed in radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

y and x fixed point fractional numbers

C. Output Format

r/2 a fixed point fractional number; θ expressed either as radians scaled by 1/8, or as degrees scaled by 2^{-9} (binary point considered to be 9 places from the left)

D. Coding Information

1. Calling Sequence for to be expressed in radians

A) trl crtplr

A + 1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: θ/8 in radians
ORG: x	ORG: r/2

2. Calling Sequence for θ to be expressed in degrees

A) trl crtpld

A + 1) <———— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: $2^{-9} \cdot \theta$ in degrees (binary point considered to be 9 places from the left)
ORG: x	ORG: r/2

Other Subroutines Required

Label	Title
SNCS1	Fixed Point Sine, Cosine
ARCTC1	Fixed Point Arctangent, Arcotangent

Error Halts

None

Sense Flip-Flops Used

None

Number of Storage Locations

46

"Common" Locations Used

10 (common thru common + 9)

Approximate Time

Average: 3.53 milliseconds
Maximum: 3.81 milliseconds

E. Accuracy Information

Average Error: 2^{-35}
Maximum Error: 2^{-34}

RESTRICTIONS:

None

METHOD:

$\theta/8 = 1/8 \arctan y/x$ (radians), or

$2^{-9} \cdot \theta$ in degrees $= \left(\frac{180}{512} \cdot \frac{8}{\pi}\right) \cdot (1/8 \arctan y/x$ in radians$)$

$r/2 = 1/2 \left|\frac{x}{\cos\theta}\right|$ if $|x| \geqq |y|$ or

$r/2 = 1/2 \left|\frac{y}{\sin\theta}\right|$ if $|y| > |x|$

if $y = x = 0$, θ and $r/2$ are set to +0.

REMARKS:

The values of θ are given as positive numbers from 0 to $2\pi/8$ in radians, or 0 to $2^{-9} \cdot 360$ degrees.

5.1.24 Floating Point Polar-to-Cartesian Coordinate Conversion

Symbolic Label: FPLCT1

PURPOSE:

To compute cartesian coordinates y and x given floating point polar coordinates θ and r; θ may be expressed in either radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

θ and r as MOBIDIC floating point numbers (not necessarily normalized).

C. Output Format

x and y as normalized MOBIDIC floating point numbers.

D. Coding Information

1a. Calling Sequence (θ in Radians)

A) TRL FPLCTR

A+1) ◄——— Normal Return

Entry Conditions	Exit Conditions
ACC: θ in Radians	ACC: y
ORG: r	ORG: x

1b. Calling Sequence (θ in Degrees)

A) TRL FPECTD

A+1) ◄——— Normal Return

Entry Conditions	Exit Conditions
ACC: θ in Degrees	ACC: r
ORG: r	ORG: x

2. Other Subroutines Required

Symbolic Label	Title
FSNCS1	Fixed Point Sine, Cosine
UNFLT1	"Unfloat" Conversion Routine
FLOAT1	"Float" Conversion Routine

3. Error Halts

None

4. Sense Flip-Flops Used

None

5. Number of Storage Locations

59

6. "Common" Locations Used

13 (Common thru common +12)

7. Approximate Time

Average: 6.51 milliseconds

Maximum: 6.95 milliseconds

E. Accuracy Information

Maximum Error:

Average Error:

RESTRICTIONS:

The input number θ must be less than N_{max} circles in absolute value for the result to have significant accuracy. N_{max} may be changed by the programmer if desired. Its value is 2^{18} circles as given in the Floating Point Sine, Cosine Routine.

METHODS:

1. Compute sin θ and cos θ
2. $y = r \sin \theta$
3. $x = r \cos \theta$

REMARKS:

Note 1. See "Floating Point Arithmetic Operations" for a description of "MOBIDIC Floating Point Numbers".

5.1.25 Title: Floating Point Cartesian-to-Polar Coordinate Conversion

Symbolic Label: FCTPL1

PURPOSE:

To compute polar coordinates θ and r, given floating point cartesian coordinates y and x; θ is to be expressed in radians or degrees.

USAGE:

A. Preparation of Cards or Tapes

None

B. Input Format

y and x MOBIDIC floating point numbers (not necessarily normalized)

C. Output Format

θ and r normalized MOBIDIC floating point numbers

D. Coding Information

1. Calling Sequence for θ to be expressed in radians

A) trl fctplr

A + 1) <———————— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: θ in radians
ORG: x	ORG: r

2. Calling Sequence for θ to be expressed in degrees

A) trl fctpld

A + 1) <———————— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: y	ACC: θ in degrees
ORG: x	ORG: r

Other Subroutines Required

Label	Title
CRTPL1	Fixed Point Cartesian-to-Polar Coordinate Conversion
ARCTC1	Fixed Point Arctangent, Arccotangent
SNCS1	Fixed Point Sine-cosine
UNFLT1	"Unfloat" Conversion Routine
FLOAT1	"Float" Conversion Routine

Error Halts

None

Sense Flip-Flops Used

None

Number of Storage Locations

59

"Common" Locations Used

17 (Common thru Common + 16)

Approximate Time

Average: 5.15 milliseconds

Maximum: 5.60 milliseconds

E. Accuracy Information

Average Error: 2^{-27}

Maximum Error: 2^{-26}

RESTRICTIONS:

x and y must not be so large that r is greater than the largest possible MOBIDIC floating point number which can be represented; that is, r must be less than 2^{256}. If r is too large, an error halt will occur in the "Float" conversion routine.

METHOD:

1) Y and x are unfloated. The fractional part of the smaller of these is scaled by the difference of the exponents. The exponent of the larger, increased by one, becomes the exponent for "floating" r in step 3.

2) The fixed point cartesian-to-polar routine is used to find θ and $r\theta 2$.

3) These are "floated" to produce normalized MOBIDIC floating point numbers for θ and r.

4) If y = x = 0, answers of 9 for both θ and r are given.

REMARKS:

See the writeup of "Floating Point Arithmetic Operations" for a full description of the format of a "MOBIDIC floating point number".

SECTION VI

INPUT ROUTINE WITH FORMAT CONTROL

6.1 MOBIDIC INPUT PROGRAM

Symbolic Label: MIP1

PURPOSE:

To select designated alphanumeric information from paper tape, in either Fieldata or Baudot code, convert it to binary, as indicated, and place it in core storage.

USAGE:

Two classes of information are possible; alphanumeric (titles) and decimal data.

A. Alphanumeric

A sequence of alphanumeric characters is called a title subfield which must begin and end with a lower case apostrophe.

B. Data

Decimal data may be represented in one of five acceptable forms. Depending on the form, the data word will be converted to either fixed or floating point and stored in the specified number memory location. Plus (+) and minus (-) signs are used to specify the sign. If no sign appears, the number is assumed to be positive.

"Decimal data subfields must be ended by either a tab or a carriage return. Only one character (i.e., tab or carriage return) may appear between subfields. If more than one character appears the second of these characters will be regarded as the first character of the following subfield. Similarly, if the very first character on the tape is a carriage return, it will result in an error since MIP will expect an apostrophe if a title is called for, or an upper case, or numeric if decimal data is called for."

"When a plus (+) or minus (-) sign is used in the input or in the calling sequence, it must be preceded by an upper case and followed by a lower case. If the upper case is omitted the sign will be regarded as an illegal character; if the lower case is omitted, the next character will be regarded as an illegal character since it will be still in upper case."

1. Fixed Point

   a. Fraction:

      Any number without a digit preceding the decimal point.

      Example: .3245

      This number is converted to a fixed point binary quantity with the binary point at the left of the word.

   b. Integer:

      A series of digits with no decimal point and no "e" or "b" characters.

      Example: -3245

      This number is converted to a binary integer with the binary point at the right of the word.

   c. Scaled Number:

      Any number with or without a decimal followed by the character "b" and an integer called the binary scale factor.

      Example: .3245b-3 Would produce the binary equivalent of this fraction with 3 leading binary zeros.

      This number is converted to a fixed point binary quantity. The binary scale factor specifies the implied number of places between the left of the storage cell and the binary point of this particular number.

2. Floating Point

a. Any number with digits before a decimal point and no "b".

Example: -29.5, 0.3245, 945.

Each number is converted to binary floating point. Note that the non-significant zero must appear. If the zero were omitted, the number .3245 would be converted to a fixed point binary quantity.

b. A number which contains a decimal point and is followed by the character "e" and a positive or negative integer.

Example: 32.45e-2, - 0.3245e8

The number following the "e" is the decimal exponent used in conversion to binary floating point.

Two methods for reading data into memory from paper tape are available:

A. Consecutive

1. BCONS (Input in Baudot Code)

2. FCONS (Input in Fieldata Code)

B. Single

1. BSING (Input in Baudot Code)

2. FSING (Input in Fieldata Code)

CONTROL SPECIFICATIONS:

A. Consecutive

k) tru fcons or bcons
dec a, b
alf C_1, C_2, ... ;
hlt y_1
. .
. .
. .
hlt y_b

a = Number of times a field is repeated.

A field is the sum of all the words in a specified number of subfields on paper tape, each subfield being composed of either title words or data words.

$$\therefore \text{Field} = \sum_{i=1}^{N} (C_i)$$

A file = specified number (namely "a") of fields

b = Number of storage areas

C_i = Number of words in each subfield

If "C" is negative, "C" words are skipped and not transferred to memory

If b > 1, then for every positive "c" there must be a corresponding "y"

If subfield is title information, C_i = t; (i.e., FIELDATA 31)

y_i = Starting location of each storage area

B. Single

k) tru fsing or bsing
dec a, b
alf C_1, C_2, $\cdots$ C_j; d;
hlt y_1
. .
. .
. .
hlt y_b

a = Number of fields

b = Number of storage areas

C_1 = Designated data words in each subfield which are to be stored

d = Length of a field

y_i = Starting location of each storage area

Return to the main program will be executed when either the total number of words read in equals the number of words in the file or if a stop code on tape is encountered.

EXAMPLES:

```
                                                    OCTAL
Example 1:    k)    tru   fcons      +40 0 0000 03726
                    dec   1,3        +000000 000001
                                     +000000 000003
                    alf   1
              45, 15, 50;            +646556616556
                                     +656073000000
                    hlt   xval       +00 0 0000 06000
                    hlt   yval       +00 0 0000 06100
                    hlt   zval       +00 0 0000 06200
                    .
                    .
                    .
                    .
              xval  def   6000
              yval  def   6100
              zval  def   6200
```

The first 45 numbers are converted as indicated and stored in memory cells "xval" to "xval + 44", the next 15 to into "yval" to "yval + 14" and the following 50 go into "zval" to "zval + 49".

Since a = 1, this pattern is not repeated and control is returned to the main program at A + 1; thus, a total of 110 words have been read.

Example 2: k) tru {bcons / fcons}

```
dec 1,4
alf t, 25, t, 15;
hlt hdg1
hlt v
hlt hdg2
hlt w
```

The first sequence of alphanumeric information, including the beginning and ending apostrophes, is stored in memory starting at location hdg1. The following 25 data words go into "v" to "v + 24"; the next set of title words go into "hdg2" and the next 15 numbers into "w" to "w + 14".

Since a = 1, a total of 2 alphanumeric sequence and 40 words have been read into memory.

Example 3: k) tru {bcons / fcons}

```
dec 1,1
alf 50, - 25, 30;
hlt w
```

The first 50 data words are stored in "w" to "w + 49". The next 25 are skipped over and the final 30 words in the field of 105 are stored in " w + 50" to "w + 70".

Since a = 1, a total of 105 numbers have been examined.

Example 4: k) tru {bcons / fcons}

```
dec 3,2
alf 25, - 10, 15;
hlt v
hlt w
```

The first 25 words are stored in "v" to "v + 24"; the next 10 words are skipped over, the next 15 words are stored in "w" to "w + 14".

Since a = 3, this cycle is repeated two more times so that the area "v" to "v + 74" contains 75 data words stored consecutively and area "w" to "w + 44" contains 45 words. The length of the file is 150.

Example 5:

```
k)  tru  { bsing
         { fsing
    dec  15, 1
    alf  1, 3, 5; 10;
    hlt  w
```

The first, third, and fifth words are stored consecutively in "w" to "w + 2". The eleventh, thirteenth, and fifteenth words are stored in "w + 3" to "w + 5". This pattern is repeated 15 times until 45 words have been selected from a file of 150, converted and stored starting in memory position "w".

Example 6:

```
k)  tru  { bsing
         { fsing
    dec  5, 3
    alf  1, 2, 3; 3;
    hlt  u
    hlt  v
    hlt  w
```

The first word is stored in "u", the second in "v", and the third in "w". This pattern is repeated with the fourth word in "u + 1", the fifth in "v + 1", the sixth in "w + 1", etc., until each of the three memory areas contains 5 data words.

CODING INFORMATION:

Calling Sequence

A) TRL K

A+1) ← Returns control here

K) TRU BSING
or
BCONS
or
FSING
or
FCONS

Where: BSING = Baudot input, single mode

BCONS = Baudot input, consec. mode

FSING = Fieldata input, single mode

FCONS = Fieldata input, consec. mode

Entry and Exit Conditions

None

Sense Flip-Flops Used

13, 14, 15, 16

Number of Storage Locations

72

"COMMON" Locations Used

91

Approximate Time

Maximum: < 3 milliseconds per character

Accuracy Information

Integer to integer Exact

Fraction
Scaled
Floating
} 10 decimal places (not rounded)

"If a stop code is read before the tab or carriage return of the last data subfield is read an exit from MIP1 will occur immediately, the remaining subfields will not be processed, and the current subfield will be lost. Two stop codes are recommended in this case, since before the first stop code has been detected by the program, the second stop code will have been read. If some character (preferably a stop code) does not follow the first stop code, the paper tape reader will read off the end of the paper tape looking for another character to read."

ERROR HALTS: (See LOCATION OF ERROR STOPS, below)

1. If any upper case digit or character other than + or - is encountered unless it is Baudot input.
2. If there are fewer words on tape than called for and no stop code.
3. If the beginning or ending title marks are omitted.
4. If any character in data subfield other than upper and lower case, e, b, ., +, -, or one of the digits 0-9 is encountered.

RESTRICTIONS:

1. Ten decimal places are allowed on fraction, scaled fixed point numbers, and floating point numbers.
2. Maximum integer allowed is $2^{36} - 1$.
3. Maximum range of floating point numbers is 0 to 10^{-76} to 10^{76}.
4. At present the program allows for 27 subfields. If a capacity for more than this is desired, an internal table would have to be lengthened.

METHOD:

The decimal data is inspected character by character.

1. The sign is stored as a word of zeros with a plus or minus sign. It it automatically set to plus before each field is begun.

2. The digits are converted to binary as they occur and the current value of the digits is stored in binary in a word (TEMP) originally set to plus zero.

3. The decimal point sets up a counter which will be stepped by one for each succeeding digit. The program contains a symbolic location called SWITCH which will transfer control to an appropriate subroutine when a terminating character (tab or carriage return) is encountered. A decimal point will cause the setting of SWITCH for transfer to a floating point conversion if a digit has occurred before the decimal point. If no digit has occurred before the decimal point SWITCH will be set for transfer to a fraction routine.

4. When a b or e is encountered, the decimal to binary routine is altered in such a way that succeeding digits will be converted and stored in a word (BN or EXP) which is set to zero at this point. The decimal place counter is turned off and SWITCH is further set to handle a scaled or floating point number.

The actions described above are independent of any prior action. Therefore, a decimal point following a "b" will have the same action as it would prior to the b. A b following an e will likewise be unaffected by the e (SWITCH will be reset accordingly, however). Prior to passing through SWITCH the program subtracts the decimal place count from EXP and places the difference in the decimal place count. Therefore, the following is possible, but not recommended;

```
.116 elb-3
```

This would yield the same result as;

1.116 b-.03 (recommended)

or

116 b-.03

The result would be an octal word in core of: +164000 00000

It is quite possible to scale a number completely out of the machine. If TEMP is zero when a terminating character is encountered, zero is stored regardless of BN, EXP, or the decimal place counter. Therefore, when

-b-3e.4

is encountered, the program will disregard BN, EXP and the decimal place counter, and will store a minus zero in the appropriate location.

The ALF pseudo-op now has a different form in MAP (see present MAP writeup in this manual). Therefore all calling sequences should be written as follows:

```
    k) tru   fcons or bcons
       dec   a,b
       alf   1
C1,C2,···;
       hlt   y1
       . .
       . .
       . .
       hlt   yb
```

LOCATION OF ERROR STOPS:

Contents of Instruction Register

HLT 1	Illegal input-look at last word stored to determine illegal word.
HLT 2	Error in Calling Sequence. Illegal character in Q-Register, and location of illegal word in accumulator.
HLT 3	Exponent in floating number is too large.
HLT 4	Characteristic in floating number is too large.

Binary Format Table

g		Field Count	
No. of sub-groups		No. of words in subfield	Subfield 1
"S" Tabs		No. of carriage returns at end of subgroup	Subgroup 1
I e r m	No. of deci-mal places	No. of words in subgroup	
etc.			Subgroup N
		n/(if any)	end of subfield
		n/	Title subfield

Note: "Subfields" are separated by semicolons. A "subgroup" is any particular mode specification. There may be many of these in a subfield.

SECTION VII

OUTPUT ROUTINE WITH FORMAT CONTROL

7.1 MOBIDIC OUTPUT PROGRAM

Symbolic Label: MOP1

PURPOSE:

To select designated words from storage, convert them to decimal, if needed, according to the mode specified, and transmit the results to paper tape in either Baudot or Fieldata code.

USAGE:

Information to be processed is treated as either title information or decimal data. Any number of title and/or data subfields may be printed by one application of the subroutine.

A. Title

The title subfield is printed out exactly as it appears in the designated memory area providing the first character is a Fieldata lower case apostrophe. Successive 6-bit characters are printed until a second apostrophe is encountered, the printing of which signifies the end of title.

B. Decimal Data

Each data word is converted according to one of the four following mode specifications:

1. i : Binary integer to decimal integer
2. r : Binary fraction to decimal fraction
3. e : Binary floating point to decimal floating point
4. m : Binary floating point to mixed decimal

The digit direction following each mode specification (except i) signifies the desired number of digits to be printed to the right of the decimal point. If more than one word is to be converted in the same fashion, a word count must precede the mode specification.

CONTROL AND FORMAT SPECIFICATIONS:

k) tru outf (Fieldata code) or outb (Baudot code)
dec a
alf C_1, C_2, C_3, ... C_i;
alf (Subfield Formats); g;

hlt y_1
. .
. .
. .
hlt y_i

a = Number of times the field is repeated

A field is the sum of all the words in the specified number of subfields in memory, each subfield being composed of either title words or data words.

. . Field $= \sum C_i$

C_i = Number of words in each subfield (if data) or the letter "t" (if title).

y_i = Starting location of each memory area. For every C_i there must be a corresponding y_i.

g = The number of carriage returns that will be executed after a file has been printed.

NOTE: If g = 0, there must be two (2) semicolons at the end of the subfield format. A file = specified number (namely "a") of fields.

SUBFIELD FORMATS:

Each data subfield format contains one or more line formats and the different mode specifications in a line format are separated by commas.

Example: A typical line format

1e4, 2r3, 1i, 2m2

would give a line which might look like

-.4075e-3 .055 -.67 25 29.25 1326.17

The end of each line format may be ended by one or more slashes, indicating carriage returns.

A semicolon indicates the end of a subfield format. If all the words of a subfield have not been printed by one application of a subfield format the format is repeated until the subfield is exhausted.

A. Carriage Returns

A slash anywhere in the format produces a single carriage return. Therefore, a double slash ;would cause one line to be skipped.

If C_i = t, indicating title information, the subfield format is merely n/;. Thus "n" carriage returns will be executed after the last line of the title is printed.

If C_i specifies a data subfield, then the number of carriage returns following the printing of the subfield is indicated at the end of the subfield format by n/;. A comma must separate the last line format from the carriage return indication.

Example: A format of alf t, 9

alf 2/; 3r4/, 3/;

would give

'x	y	z'
.3652	.6348	.0054
.4225	.5775	.1162
.7335	.2665	.3110

followed by four carriage returns (one supplied by the end of line format and 3 more supplied by the end of subfield indication).

If any carriage return specification is omitted; succeeding words will be printed on the same line. If the subroutine is called on to print more than eight data words per line, it will execute one carriage return automatically after printing the eighth word. Any overflow will be printed starting at column 1 of the next line.

B. Column Control

The column width is manually set on the Flexowriter at 15 character intervals thus limiting the output to a maximum of eight data words per line. The first character in each data word is printed in position 1 of each column. Immediately after the printing of the last character a tab is executed. In order to indent or to skip over one or more columns, the letter s is introduced preceded by the counter n.

Example: A typical output format alf t, 12;
alf 2/; 3r4, 1_i//2s1r4, 1_i//;
would produce a printout which might look like

'x	y	z	Case'
.0325	.4675	-.3960	1
		.0053	2
.0495	.3662	.0004	3
		-.1532	4

GENERAL REMARKS:

The number of words in the subfield need not be an integral multiple of the words in a field.

More than one data subfield may be printed on the same line. This is accomplished by omitting the slash character in all the formats except that of the last subfield to be printed on the line. (See Example 3).

EXAMPLES:

```
Example 1:    k)    tru     outf
                    dec     1
                    alf     t, 50
                    alf     2/; 3r2/2r2/;
                    hlt     hdgl
                    hlt     v
```

The above will produce as many lines of alphanumeric characters as are necessary to print the entire title field contained in memory area "hdgl", two carriage returns followed by the data (3 fixed point values on one line, 2 fixed point values on the second line, etc., repeated ten times because the subfield has 50 words).

```
Example 2:    k)    trl     outf
                    dec     2
                    alf     t, 25, t, 10;
                    alf     3/; 4e3, 1i/, 2/; 5/; 3r4/; 4/;
                    hlt     hdgl
                    hlt     u
                    hlt     hdg 2
                    hlt     v
```

The above will give the title field printed exactly as it appears in memory, 3 carriage returns, five lines (single spaced) composed of 4 floating point numbers and 1 integer, three carriage returns, title field, 5 carriage returns, 3 lines of 3 fixed point numbers with a tenth number alone on the fourth line. Five carriage returns (1 supplied by the line format plus 4 extras) will be provided after the tenth word from area "v" has been printed. Since "a" = 2, this entire output pattern would now be repeated with the same title headings but data from storage areas "u + 25 to u + 49 and v + 10 to v + 19".

```
Example 3:    k)    tru     outf
                    dec     5
                    alf     1, 1, 1;
                    alf     1r3; 1r3; 1r3/;3/;
                    hlt     xval
                    hlt     yval
                    hlt     zval
```

Since the slash is omitted in the first two data subfield formats and the subfield length in each case is 1, each line will contain three fixed point values (one from each of the memory areas - xval, yval, and zval). This pattern will be repeated four more times (single spaced). The 15 word file (composed of 5 three-word fields) will be followed by four carriage returns (one at the end of the fifth line plus three more specified by "g" = 3/).

CODING INFORMATION:

Calling Sequence

A) TRL K K) TRU { outb / or / outf

Out B = Baudot output desired

Out F = Fieldata output desired

A+1) ← Returns control here

Entry and Exit Conditions

None

Sense Flip-Flops Used

16, 15, 14, 13

Number of Storage Locations

821

"COMMON" Locations Used

201

Approximate Time

Maximum: Can process full buffer before alternate buffer is completely written. In effect, the program is "tape bound".

Accuracy Information

1. Average Error: Integer to integer => exact

2. Maximum Error: Fraction to fraction => accurate to "nth" decimal place (unrounded) where "n" is the number of decimal places specified by the programmer.

   Floating to floating => same as fraction to fraction
   Binary to mixed decimal => same as fraction to fraction.

ERROR HALTS: (See Location of Error Stops Below)

If the first character in the title subfield is not a Fieldata apostrophe

NOTE: If an illegitimate mode is specified, the designated word in memory will be converted to an integer with an identifying marker accompanying its print out. The routine will not stop. The identifying marker will be an "i" immediately preceding the number. There are error halts if any character is encountered other than those referred to, viz: i, e, r, m, digits, +, -, s, /, and decimal points. Of course, any character is allowed in a title field, provided that apostrophes are present in the correct places.

RESTRICTIONS:

If "M" conversion is specified and the binary number is > 68, 719, 476, 375 it is converted to floating decimal.

METHOD:

Subfield counts and formats are scanned and a binary format table is set up to be used for control throughout the program.

ON Conversions:

a. Binary Integer to Decimal Integer: successive division by 10, saving the remainder each time as a progressively more significant digit.

b. Binary Fraction to Decimal Fraction: successive multiplications by 10, saving the integer part of the answer as a progressively less significant digit.

c. Binary Floating to Decimal Floating: a polynomial approximation is used.

d. Binary Floating to Mixed Decimal: previously described integer and fractional conversions are used.

LOCATION OF ERROR STOPS:

Contents of Instruction Register	
HLT 5	Illegal mode or error in Calling Sequence

REMARKS

On Baudot Output: carriage returns are followed by line feeds, and since there is no teletype equivalent for a "Tab", the appropriate number of spaces is provided. An upper case is provided before title fields as a precautionary measure.

NOTE: Presently the program is set up to handle as much format as can be specified in 56 characters. If more characters are to be handled, the binary format table would have to be lengthened. Leading zeros are omitted on fixed point conversions. On integer to integer conversion, the least significant digit is printed in the 12th column of the field. On other conversions, spaces are not provided for leading zeros.

SECTION VIII

GENERALIZED DATA HANDLING ROUTINES

It is characteristic of data processing work that much of it involves the need for establishing and updating master files. These files contain information about a large number of individuals or items. The files are processed by sorting and merging with subsidiary files according to some control or key fields provided in each item. Frequently, files are generated according to artificial or fictitious keys, as in the case of program assemblies. Yet, it is a well-established fact that the majority of work done with such files, both in terms of number of items processed and time spent on such processing requires sorting and merging.

The generalized data handling routines provided with the Minimal Programming Aids are capable of sorting one input tape file or of merging several tapes into one string or of generating routines to handle a large variety of sorts and merges.

8.1 MOBIDIC SORT ROUTINE

Symbolic Label: SORT1

PURPOSE:

To sequence a file of fixed size items relative to a given amount of information contained in each item (called the control field). This program is designed to work on the basic machine. It will accept a maximum of one full reel (3600 feet) of magnetic tape input formed in the Hi-Reliability Mode and produce one fully sequenced and labeled standard tape in the Hi-Reliability mode. (See write-up entitled SPECIFICATIONS FOR STANDARD FIELDATA TAPES - revised 31 July 1959.)

This program may also be used for sorting multi-reel files by the use of several additional merge passes.

USAGE:

A. Preparation of Cards or Tapes

An input parameter tape must be used to give the necessary control information to the sort program.

The parameter tape requesting either a pure merge or sort or rerun shall be punched with the following lables and parameters. Each parameter will be followed immediately, and terminated by, a carriage return. This first parameter is preceded by a carriage return.

Parameter Number	Question or Statement	Number of Characters Required	Allowable Choice
1*	Is this a Sort or Merge or Rerun?	5	merg1, sort1, rern1
2	No. of Input reels	3	Xr1 where X = 1, 2
3	Today's Date (i.e., date of output reel)	9	Standard date/time group
4	Name of file	18	Exactly as it appears in File spec block
5	Number of file	4	Exactly as it appears in File spec plus a period
6	Format of file	3	rel - high reliability rex - relaxed
7**	Error checking desired?	6	fullck ignore ncxxxx

* If a rerun is required, parameter No. 1 is followed by a stop code.

** There are three sum checking modes. They are: full check (fullck) which sum checks the input blocks on their block hash total and reforms the block hash on the output data; ignore check (ignore) which designates that no sum-checking on the block hash is to be performed; and new check (ncxxxx) which checks the block hash, and the sumcheck word for the file and recomputes the sum check on the output tape using word xxx of each item. Results are carried in the trailer block.

Parameter Number	Question or Statement	Number of Characters Required	Allowable Choice
8	Number of allowable errors	3	xxe
9	Security classification	6	Same as file spec block
10	File password	6	Same as file spec block
11	Installation Code	6	Same as file spec block
12	Output File name	18	As you wish them to appear in the output file spec block
13	Output File number	4	As you wish them to appear in the output file spec block plus a period
14	Number of Keys	6	xxkeys where xx ± 08
15	Redundancy necessary for compatibility with general sort parameter tape format	2	00
16[T]	Key and word in item	10	xxkywdzzzz
17[T]	mask	12	12 octal digits
18	Stop Code	1	Stop Code

[T] These parameters are repeated for each key.

Sample Parameter Tape

sort1	We are to start by sorting and dispersing
2r1	We have two input reels
26062305e	The days date in standard date/time group
payroll a	The file name
001.	The file number
rel	The input has high reliability format
nc0241	We desire to sumcheck word 241 in the output file
03e	We will allow 3 errors before stopping sort
secret	The security classification is secret
xbaker	the password is XBAKER
washdc	The installation code is Washington D.C.
sorted payclass a	The output file name
001.	The output file number
04keys	The total number of keys is four
00	redundant characters
01kywd0240	The first key is contained in word 240
773000000000	bits 36-31, 29, 28 are to be used
02kywd0111	The second key is contained in word 111
000000000077	bits 6-1 are to be used
03kywd0002	The third key is contained in word 2
77000000000077	bits 36-31, 6-1 are to be used
04kywd0100	The last key is contained in word 100
7600000000007	bits 36-32, 3-1 are to be used
(stop code)	

B. Input Format

The input is composed of a file of items formed in the Hi-Reliability mode on magnetic tape.

C. Output Format

The output is composed of a tape file containing a fully sequenced and labeled standard tape in the Hi-Reliability mode. (See writeup entitled SPECIFICATION FOR STANDARD FIELDATA TAPES - revised 31 July 1959.)

Error Halts

In case of error, a message of the form ERRNN is printed on the Flexowriter where NN is the error number.

01 Unreadable Parameter Tape
02 Unreadable Tape Specification Block
03 Unreadable File Specification Block
04 Format Error in Parameter Tape
05 Unmatching Password
06 Unmatching Security Code
07 Unmatching File Name
08 Unmatching File Number
09 Number of Errors is greater than the number of allowable errors
10 Machine Error or Possible Bug in Program
11 Partial Item In Block (i.e., item split across blocks)
12 When the number of allowable errors = 0, and the item or block count in the trailer is unequal to the respective computed counts.
13 Unmatching Trailer Checksum and the number of Allowable Errors = 0
14 Over maximum number of data blocks

Sense Flip-Flops Used

1-16 They must be in the "neutral" position on the console.

Number of Storage Locations

The first phase program is 2500 orders. The second phase program is 1500 orders. In both cases, the rest of the memory is used for buffering. This means that 8192 storage locations are used.

APPROXIMATE TIMING FORMULA

N_B = No. of blocks

N_i = No. of items

W_i = words per item

W_B = words per block

N_B^* = No. blocks optimumly packed

T = time in microseconds

K = Intermediate merge phases

Sort and Dispersal Phase

$$T = 1.1 \left\{10^6 + 9.10^3 + N_i\right\} \frac{16500}{w_i} + 117W_i + 176 \left[\text{Log}_2 \left(\frac{1500}{w_i}\right) + 1\right] + 890 \left\{+ N_8 \left\{244w_B + 7.21 \cdot 10^3\right\}\right\}$$

First Merge Pass

$$T = 2.2 \left\{144 \left[125 + N_i \cdot W_i\right] 13.2 \cdot 10^3 \left[N_B + 2\right]\right\}$$

Intermediate Merge Passes

$$T = 2.2 \left\{144 \left[125 + N_i \cdot W_i\right] + 13.2 \cdot 10^3 \left[N_B^* + 2\right]\right\}$$

$$K = \text{Log}_2 \left[\frac{Ni}{\frac{3000}{wi}}\right] - 2$$

(Where bracketed quantities are rounded down to integers)
where K is rounded upward to the nearest integer

Collection Pass

$$T = N_i \left\{240 \cdot W_i + 800\right\} + N_B \left\{86WB + 6.6 \cdot 10^3 + 128\right\} + 34.2 \cdot 10^3$$

RESTRICTIONS:

1. The input file must be contained on one tape reel and must be in the Hi-Reliability format.
2. Fixed size items only.
3. No item may exceed 510 words.
4. Words containing control information must be positive.
5. Total control field size less than or equal to eight (8) words.
6. No item may be "split" across tape blocks.
7. 8192 words of core must be available to SORT1.
8. At least six (6) tape units and two (2) converters must be available.

METHODS AND DESCRIPTION OF SORT1

a) Reads itself into memory and transfers control to the starting line.

b) *Reads in the Specification block, and checks for labels, picks up item size, etc. initializes itself.

c) *Reads and checks the parameter tape, generates the coding necessary to set up the control field, performs label matching, and in case of inconsistency makes a descriptive error printout. Otherwise goes to (d).

d) *Starts the main read-in of the items to be sorted, strips their control field, placing it at the beginning of the item (without increasing size or item), and sets up an address table containing the address of each item.

e) *Sorts the address table by insertion.

f) *Disperses sorted items to an output area.

g) *Writes out the string of items in the output area.

h) Tests for converter error on new data that has been read in; if none returns to (d), otherwise continues to (i).

i) A subroutined re-read is performed three times. If after three reads the data is still in error, it will be labeled and entered on the reject tape. The next block will be read and control passes to (d).

*It should be noted here that those macro steps marked by an asterisk are performed with either partial or full overlapping.

j) Upon completion of (a-i) which is called the sort-augment-dispersal phase, the output tapes will contain a number of strings of sorted items. The trailer block item counts, etc., are checked at this time, and if there is no discrepancy, control now passes to the merging phase (k). This merging phase is part of the MOBIDIC Merge (MERG1) routine which is automatically called in by the completion of the sort phase.

k) The merging phase will read in strings, using the Von Neuman method to produce longer and longer strings. This phase may consist of several passes; however, at the end of some pass the output tapes will contain no more than one string each. At this time the final phase is entered.

l) The final phase of MERG1 is a single tape pass consisting of merging the final strings together into one output tape. This output tape will have as its header blocks the updated tape and file specification blocks which may, at the option of the operator, be printed out on the Flexowriter. It is during this pass that each item is reorganized into its original form and the Trailer block updated and written.

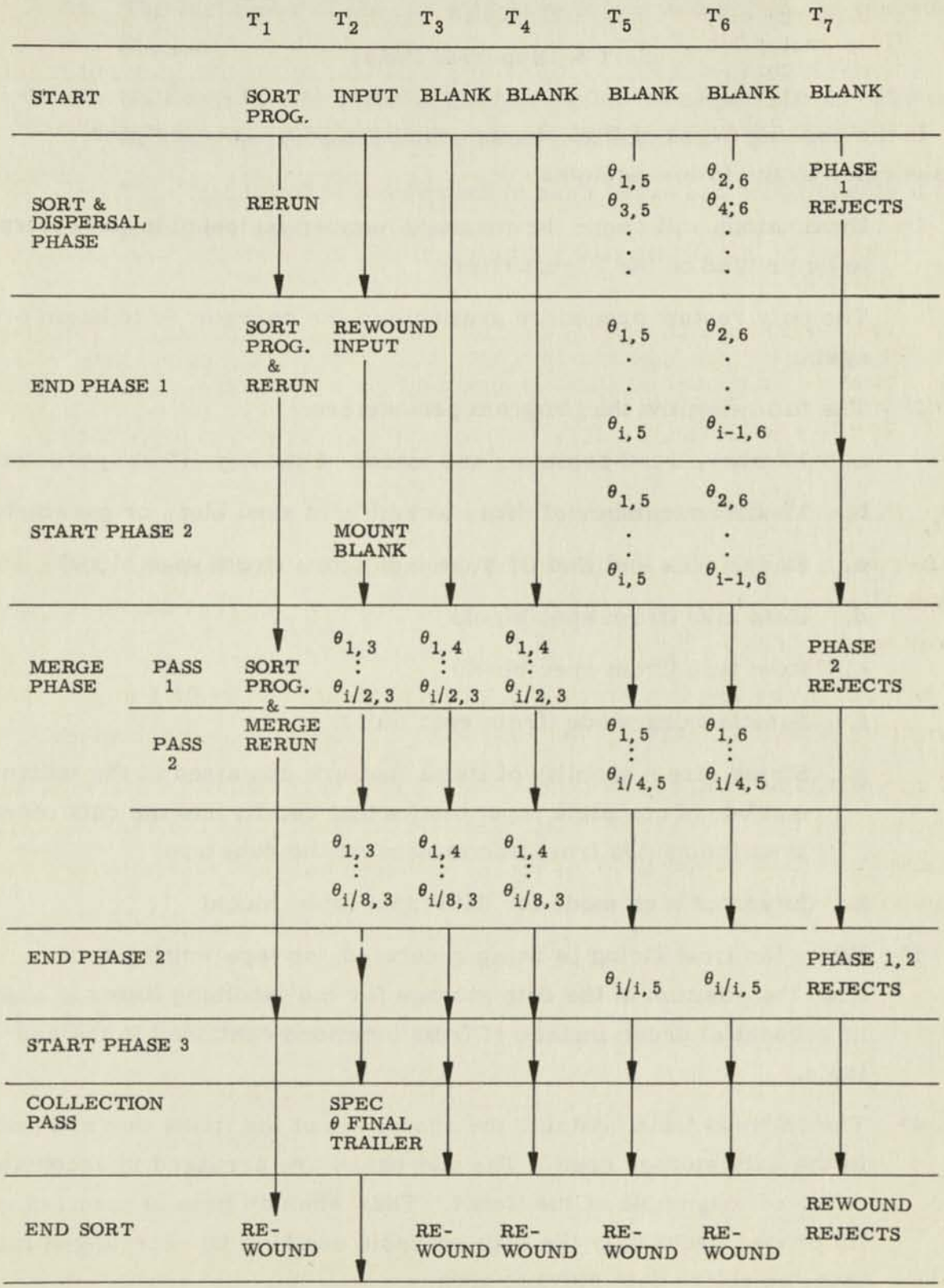
$0_{i,j}$ = The i^{th} string of the data. It is on tape j.

After each merge pass the number of strings is at least halved. At "END SORT" the output may be either on Tape 3 or Tape 5. It is shown for illustrative purposes as T_3.

For a rerun the two input reels are always mounted on reels 5 and 6, irrespective of which units they were on when the merge was interrupted.

If only six tape units are available, tape unit 3 (T_3) cannot be dialed in during the sort phase. At receipt of the message "ENDSRT" on the Flexowriter, a blank tape must be mounted instead of the input tape and this unit must be dialed as tape unit 3 (T_3).

TAPE LAYOUT (BASIC)



SORT1 - FLOW CHARTS

Sort & Dispersal Phase

In the ensuing pages of flow charts, numbers with superscripted asterisks refer to the following notes:

1* Error stops will cause the message number indicated in the rectangle to be printed on the Flexowriter.

The only restart procedure available to the operator is to begin over again.

2* The following are the program parameters:

a. Number, word position, and masks of the keys (from parameters)

b. Maximum number of data blocks (from spec block or parameters)

c. End Of File and End Of Tape indicators (from spec block)

d. Item size (from spec block)

e. Item type (from spec block)

f. Sumchecking mode (from spec block)

g. String size = quantity of items that are contained in the maximum number of complete input blocks that can fit into the data storage area (computed from information on the data tape)

h. Interrupt sign mode for data input (spec block)

3* While the first string is being generated, no tape writing occurs. Also the position in the data storage for the incoming items is assigned in sequential order instead of from locations contained in the address table.

4* The address table contains the addresses of the items that are contained in the data storage area. The addresses are arranged in ascending order of magnitude of the items. Thus when an item is inserted in its proper order only the address table needs to be rearranged rather than the entire data storage area.

5* The last block of the file may have fewer items than the preceding blocks.

6* This test is a function of the number of items per block. The output blocking is the same as the input blocking.

7* This test is made on cumulative hash totals and block counts if they are present in the trailer block.

Insert Routine

The following symbols appear in the flow chart of the Insert Routine:

Argument:	The item currently being processed.
Entry:	The item, whose address is already in the Address Table, with which we are currently matching the Argument.
U:	Upper bound of the Address Table (this is fixed).
L:	The current lower bound of the Address Table (this is decreased by 1 after each item's address is inserted).
TOP:	The position in the Address Table that contains the address of an item that is equal to or greater than the Argument.
BOT:	The position in the Address Table that contains the address of an item that is equal to or less than the Argument.
I2:	Index register 2.
M:	Current median value of table.

8.2 MOBIDIC MERGE ROUTINE

Symbolic Label: MERG1

PURPOSE:

To merge two (2) files of items formed in the Hi-Reliability mode and composed of internally sequenced tape blocks. Each file must be contained on not more than one (1) rull reel (3600 ft.) of magnetic tape.

USAGE:

A. Preparation of Cards or Tapes

An input parameter tape must be used to give the necessary control information to the MERGE program.

The parameter tape requesting a merge shall be punched with the following labels and parameters. Each parameter will be followed immediately, and terminated by, a carriage return.

Parameter Number	Question or Statement	Number of Characters Required	Allowable Choices
1	This is a Merge	5	mergl
2	No. of input reels	3	2rl
3	Today's date	9	Standard date/time group
4*	Name of file	18	Exactly as it appears in File spec block
5*	Number of file	4	Exactly as it appears in File spec block plus a period
6	Format of file	3	rel-high reliability
7**	Error checking desired?	6	fullck, ignore, ncxxx
8	Number of allowable errors	3	xxe
9	Security classification	6	Same as file spec block

Parameter Number	Question or Statement	Number of Characters Required	Allowable Choices
10	Password	6	Password
11	Installation Code	6	Same as file spec block
12	Output File name	18	As you wish them to appear in the output file spec block
13	Output File number	4	As you wish them to appear in the output file spec block plus a period
14	Number of Keys	6	xxkeys where xx = 00-08
15	For compatibility with sort generator format	2	00
16T	Key and word in item	10	xxkydzzzz
17T	mask	12	12 octal digits
18	stop code	1	stop code

* - These labels are repeated for each input file

** - There are three sum checking modes. They are: full check (fullck) which sum checks the input blocks on their block hash total and reforms the block hash on the output data; ignore check (ignore) which designates that no sum-checking on the block hash is to be performed; and new check (ncxxxx) which checks the block hash, and the sumcheck word for the file and recomputes the sum check on the output tape using word xxx of each item. Results are carried in the trailer block.

T - These parameters are repeated for each key.

Specific Example

Suppose we have a main file and a weekly transaction file. The main file is fully sequenced and the transaction file is sequenced in daily lots. It is desired to make merge run, producing a new master file.

merg1	We are to start by merging
2rl	We have two input reels
26062305e	The days date in standard date/time group
Main File	The file name of the first reel
001..	The file number of the first reel
Transaction file	The file name of the second reel
001.	The file number of the second reel
rel	The input has a high reliability format
nc0241	We desire to sumcheck word 241 in the output file
03e	We will allow 3 errors before stopping sort
secret	The security classification is secret
washdc	The installation code is Washington D. C.
Merged main file	The output file name
001.	The output file number
04keys	The total number of keys is four
00	Generator compatibility format characters
01kywd0240	The first key is contained in word 240
773000000000	Bits 36-31, 29, 28 are to be used
02kywd0111	The second key is contained in word 111
00000000077	Bits 6-1 are to be used
03kywd0002	The third key is contained in word 2
7700000000077	Bits 36-31, 6-1 are to be used
04kywd0100	The last key is contained in word 100
760000000007	Bits 36-32, 3-1 are to be used
(stop code)	

B. Input Format

The input is composed of a file of items, formed in the Hi-Reliability mode, each of whose blocks is internally sequenced.

C. Output Format

The output is composed of a tape file containing a fully sequenced and labeled standard tape in the Hi-Reliability mode. (See writeup entitled SPECIFICATIONS FOR STANDARD FIELDATA TAPES - revised 31 July 1959.)

Error Halts

In case of error, a message of the form ERRNN is printed on the Flexowriter where NN is the error number.

01 Unreadable Parameter Tape

02 Unreadable Tape Specification Block

03 Unreadable File Specification Block

04 Format Error in Parameter Tape

05 Unmatching Password

06 Unmatching Security Code

07 Unmatching File Name

08 Unmatching File Number

09 Number of Errors is greater than the number of Allowable Errors

10 Machine Error or Possible Bug in Program

11 Partial Item In Block (i. e., Item split across blocks)

12 When the number of allowable errors = 0 and the item or block count in the trailer is unequal to the respective computed counts.

13 Unmatching Trailer Checksum and the Number of Allowable Errors = 0

Error Halts (Cont.)

14 Number of unsorted blocks > the Number of allowable errors

15 Over maximum number of data blocks

Sence Flip-Flops Used

1-16 They must be set in the "neutral" position on the console.

Number of Storage Locations

This program is 1500 orders long; however, most of the remaining core is used for buffering.

Mergl

Approximate Timing Formulas

N_B = number of blocks

N_i = number of items

W_i = words/item

W_B = words/block

N^* = number of blocks optimumly packed

T = number of time in microseconds

K = number of intermediate merge phases

S = number of items/string

First Merge Pass

$$T = 5.1 \left\{ 144[125 + N_i W_i] + 13.2 \times 10^3 [N_B + 2] \right\} + 10^6$$

Intermediate Merge Passes

$$T = 2.2 \left\{ 144[125 + N_i W_i] + 13.2 + 10^3 [N^* + 2] \right\}$$

$$K = \text{Log}_2 [\frac{N_i}{S}] - 2$$

Collection Pass

$$T = N_i \left\{ 240 \times W_i + 800 \right\} + N_B \left\{ 26WB + 6.6 \times 10^3 + 128 \right\} + 34.2 \times 10^3$$

RESTRICTIONS:

1. Each input file must be contained on one tape reel and must be in the Hi-Reliability format.
2. Fixed size items only.
3. No item may exceed 510 words.
4. Words containing control information must be positive.
5. Total control field size less than or equal to 8 words.
6. No item may be "split" across tape blocks.
7. All keys must be in the same position in both files of items.
8. Data items must be sequenced, at least between blocks.
9. 8192 words of core must be available.
10. At least six (6) tape units and two (2) converters must be available.

METHOD:

MERG1 performs the following steps in its operation:

(a) Reads itself in memory and transfers control to the starting line.

(b*) Reads in the parameter tape, checks labels, checks for any inconsistency. Finishes the initialization. Generates coding necessary to pick up control fields. If the labels do not agree or if there is any inconsistency an appropriate error printout is made, otherwise control passes to step (d).

(c*) Reads in the specification blocks of each input file, picks up parameters from these specification blocks and starts initializing itself.

(d*) Continues the read-in of the input tapes, sequence checks the incoming items, if items arrive out of sequence within a block MERG1 labels and rejects the block. Continues by preselecting the next input and checking for any input errors on the previous read. If no error is indicated it passes to step (e). In case of an error, an automatic re-read sequence is initiated. If after three reads the information is still in error, it will be labeled and written on a reject tape. In any case control passes to step (e).

(e*) Merges and writes out the merged input data, checking for a "break" in the sequencing of blocks from either file. (This is known as the "single step down" condition.) If none, returns to step (d); otherwise copies the sequence from the other input file testing for the "break" in its sequencing. (This is known as the "double step down" condition.) As soon as this occurs, MERG1 terminates the string produced on the output, swaps output tapes and tests for exhaustion of the input reels. If they are exhausted, control passes to step (f). Otherwise control returns to step (d).

(f) When the input tapes have been exhausted, input and output tapes are swapped, rerun information is written and control returns to step (d). Finally there will exist on each output tape only one long string; at this time control is passed to step (g).

(g) This is the final tape pass known as the collection pass. In this pass the final merge is performed, the specification block(s) are updated and written, hash totals are checked, word rearrangement is performed, and other bookkeeping tasks completed. There is an option at this time for the operator to print the trailer and specification blocks of the final tape(s).

* These macro steps are performed with either partial or full overlapping.

FLOW CHARTS

GENERALIZED MERGE ROUTINE

NOTATION

BUFFERS:

R_{ij} Input buffers (4) — i = 5 or 6; j = 1 or 2

i = input index

5 = input reel 1 on tape drive 5

6 = input reel 2 on tape drive 6

j = buffer index

there is an active and an alternating buffer for each input tape

W_k = Output buffers (2) — k = 1 or 2

there is an active and a passive output buffer

INDICATORS:

FP first pass

DO output of SORT 1

RD non-output of SORT 1

CP collection pass

ME machine error test

SYMBOLS:

EOF end of file of tape

EOS end of string of input

Par. parameter

RC Read Check Subroutine

ERROR STOPS:

Error numbers are printed on the flexowriter. The only restart procedure available to the operator is to begin all over again.

8.3 FORMAT FOR TAPE SPECIFICATION BLOCKS - FORMAT FOR TAPE AND FILE TRAILER BLOCKS

The discussion below concerns itself with procedures that must be followed by programming personnel when writing out data on to tape if it is anticipated that such data will eventually require sorting or merging through use of the SORT I or MERGE I programs. Design of these programs is such that they will recognize the formats given in the tables presented in subsequent paragraphs and no others. Failure to label a given table in accordance with the formats given will result in erroneous operation when the SORT I and MERGE I programs are used.

8.3.1 Format for Tape Specification Block

Octal and Fieldata configurations for the Tape Specification Block are given in Table 8-1 below. Remarks that appear in the Fieldata and Remarks column and have an associated brace or arrow are further amplified in the explanatory discussion immediately following the table.

TABLE 8-1. OCTAL AND FIELDATA CONFIGURATIONS FOR TAPE SPECIFICATION BLOCK

Word Number	Octal	Fieldata and Remarks
1	040303013106	↙→→↑ T A
2	251205302512	P E Δ S P E
3	101613161006	C I F I C A
4	311624230507	T I O N Δ B
5	212410200404	L O C K ↙ ↙
6	110631120531	D A T E Δ T
7	162212051427	I M E Δ G R
8	243225055302	O U P Δ : ↓
9	050000353535	Δ x x x Value of Date-Time Group
10	353535353535	x x x x x x in Decimal (9 characters)
11	040130121032	↙↑ S E C U
12	271631360510	R I T Y Δ C
13	210630300005	L A S S Δ
14	005302050000	: ↓ Δ
15	322310210630	x x x x x x Value of Security Class (6 characters)
16	040131062512	↙↑ T A P E
17	052506303034	Δ P A S S W
18	242711055305	O R D Δ : Δ
19	020000000000	↓
20	353535353535	x x x x x x Value of Tape Password (6 characters)
21	040116233031	↙↑ I N S T
22	062121063116	A L L A T I
23	242305102411	O N Δ C O D
24	120553050200	E Δ : Δ ↓
25	670605301010	x x x x x x Value of Installation Code (6 characters)
26	040131062512	↙↑ T A P E
27	052306221205	Δ N A M E Δ
28	530502000000	: Δ ↓
29	353535353535	x x x x x x Value of
30	353535353535	x x x x x x Tape Name
31	353535353535	x x x x x x (18 characters)
32	040131062512	↙↑ T A P E
33	052332220712	Δ N U M B E
34	270553050200	R Δ : Δ ↓
35	000000353535	x x x← Value of Tape Number (3 characters)
36	040127121221	↙↑ R E E L

TABLE 8-1. OCTAL AND FIELDATA CONFIGURATIONS FOR TAPE SPECIFICATION BLOCK (Cont.)

Word Number	Octal	Fieldata and Remarks
37	052332220712	Δ N U M B E
38	270553050200	R Δ : Δ ↓
39	000000353535	x x x ← Value of Current Reel Number (3 characters)
40	010524130502	↑ Δ O F Δ ↓
41	000000353535	x x x Value of Final Reel Number (3 characters)
42	040131062512	/ ↑ T A P E
43	053012233116	Δ S E N T I
44	231221300553	N E L S Δ :
45	050200000000	Δ ↓
46	077307730773	M ; B ; C ;
47	040107212410	/ ↑ B L O C
48	200521060712	K Δ L A B E
49	210553050237	L Δ : Δ ↓ x Value of Tape Sentinel (1 character)
50	040107212410	/ ↑ B L O C
51	200510243223	K Δ C O U N
52	310553020500	T Δ = ↓ Δ
53	000035353535	x x x x ← Value (Decimal) of Total Block Count
54	040123322207	/ ↑ N U M B
55	122705241305	E R Δ O F Δ
56	131621123005	F I L E S Δ
57	530502606061	: Δ ↓ x x x Value (Decimal) of the Number of Files on this Tape (3 characters)
58	040113162112	/ ↑ F I L E
59	052306221205	Δ N A M E Δ
60	530502000000	: Δ ↓
61	353535353535	x x x x x x } Value of 1st File Name
62	353535353535	x x x x x x } on this Tape
63	353535353535	x x x x x x } (18 characters)
64	040113162112	/ ↑ F I L E
65	052332220712	Δ N U M B E
66	270553050200	R Δ : Δ ↓
67	000000353535	x x x { Value of File Number of Above File Name (3 characters)
68	040107212410	/ ↑ B L O C
69	200521241006	K Δ L O C A
70	311624230553	T I O N Δ :
71	050260606062	Δ ↓ x x x x Decimal Value Given Location of 1st File
72	050131240502	Δ ↑ T O Δ ↓
73	000035353535	x x x x } Decimal Value Giving Ending Location of 1st File

In reference to the Sort 1 and Merge 1 programs, the following manipulations of the Tape Specification Block are performed for the output Tape Specification Block:

1. Date Time Group is set as specified by Parameter Tape.
2. The security code is also set from the Parameter Tape.
3. The password is also set from the Parameter Tape.
4. The installation code is set from the Parameter Tape.
5. The tape name is unaltered and not used.
6. The tape number is unaltered and not used.
7. The Reel Number is set to 001 of 001.
8. The Tape sentinels are preserved.
9. The block label character is preserved.
10. The Block count is computed and set by the sort scanner from estimated file size and item size obtained from the file spec.
11. Number of files is set to 001.
12. The file name is set from the parameter tape. It is also used for matching purposes on input.
13. The file number is set from the parameter tape. It is also used for matching purposes on input.
14. The ending Block location is computed from the file size and item size. On input the block location is used by the Sort to locate the file specification block.

8.3.2 Format for File Specification Block

Octal and Fieldata configurations for the File Specification Block are given in Table 8-2. Remarks that appear in the Fieldata and Remarks column and have an associated brace or arrow are further amplified in the explanatory discussion immediately following the table.

TABLE 8-2. OCTAL AND FIELDATA CONFIGURATIONS FOR FILE SPECIFICATION BLOCK

Word Number	Octal	Fieldata and Remarks
1	040303011316	/→ →↑ F I
2	211205302512	L E △ S P E
3	101613161006	C I F I C A
4	311624230507	T I O N △ B
5	212410200404	L O C K //
6	110631120531	D A T E △ T
7	162212051427	I M E △ G R
8	243225055302	O U P △ : ↓
9	050000353535	△ x x x } Value of Date Time Group in Decimal (9 Characters)
10	353535353535	x x x x x x }
11	040130121032	//S E C U
12	271631360510	R I T Y △ C
13	210630300553	L A S S △ :
14	050200000000	△ ↓
15	322310210630	x x x x x x { Value of File Security Class (6Characters)
16	040113162112	//F I L E
17	052506303034	△ P A S S W
18	242711055305	O L D △ : △
19	020000000000	↓
20	353535353535	x x x x x x { Value of File Password (6 Characters)
21	040116233031	//I N S T
22	062121063116	A L L A T I
23	242305102411	O N △ C O D
24	120553050200	E △ : △ /
25	670605301010	x x x x x x { Value of Installation Code (6 Characters)
26	040113162112	/↑ F I L E
27	052306221205	△ N A M E △
28	530502000000	: △ ↓
29	353535353535	x x x x x x } Value of File Name (18 Characters)
30	353535353535	x x x x x x }
31	353535353535	x x x x x x }

TABLE 8-2. OCTAL AND FIELDATA CONFIGURATIONS FOR FILE SPECIFICATION BLOCK (Cont.)

Word Number	Octal	Fieldata and Remarks
32	040113162112	/↑ F I L E
33	052332220712	Δ N U M B E
34	270553050200	R Δ : Δ ↓
35	000000353535	x x x — Value of File Number (3 Characters)
36	040127121221	/↑ R E E L
37	052332220712	Δ N U M B E
38	270553050200	R Δ : Δ ↓
39	000000353535	x x x — Value of Current Reel Number (3 Characters)
40	010524130502	↑ Δ O F Δ ↓
41	000000353535	x x x — Value of Last Reel Number (3 Characters)
42	040113162112	/↑ F I L E
43	053012233116	Δ S E N T I
44	231221300553	N E L S Δ :
45	050200000000	Δ ↓
46	077307730773	M ; C ; C ;
47	040107212410	/↑ B L O C
48	200521060712	K Δ L A B E
49	210553050236	L Δ : Δ ↓ x — Value of End of File Sentinel This File (1 Char.)
50	040131362512	/↑ T Y P E
51	053127061621	Δ T R A I L
52	122705530502	E R Δ : Δ ↓
53	000013162112	x x x x — Either File or Tape if Multireel (4 Characters)
54	040116311222	//I T E M
55	053032221015	Δ S U M C H
56	121020053424	E C K Δ W O
57	271105530502	R D Δ : Δ ↓
58	000060606060	x x x x — Value of Location of Sum Check Word
59	040116311222	//I T E M
60	053136251205	Δ T Y P E Δ

TABLE 8-2. OCTAL AND FIELDATA CONFIGURATIONS FOR FILE SPECIFICATION BLOCK (Cont.)

Word Number	Octal	Fieldata and Remarks
61	530502000000	: Δ ↓
62	003535353535	x x x x x } Either Fixed or Varys (5 Characters)
63	040116311222	↙↑ I T E M
64	053016371205	Δ S I Z E Δ
65	530502000000	: Δ ↓
66	000035353535	x x x x } Value (Decimal) of Item Size
67	050131240502	Δ ↑ T O Δ ↓
68	000035353535	x x x x } Value (Decimal) of Maximum Item Size
69	040111063106	↙↑ D A T A
70	052224111205	Δ M O D E Δ
71	530502233023	: Δ ↓ x x x ← Either NSN or ISN
72	040131243106	↙↑ T O T A
73	210513162112	L Δ F I L E
74	053016371205	Δ S I Z E Δ
75	530502000000	: Δ ↓
76	003535353535	x x x x x } Value (Decimal) of Item Count of File
77	040125062731	↙↑ P A R T
78	160621051316	I A L Δ F I
79	211205301637	L E Δ S I Z
80	120553050200	E Δ : Δ ↓
81	006060606060	x x x x x } Decimal Value of Item Count - This file this tape
82	040113162112	↙↑ F I L E
83	051324272206	Δ F O R N A
84	310553050200	T Δ : Δ ↓
85	001516271221	x x x x x } Either Hire L or Relax
86	040122063505	↙↑ M A X Δ
87	110631060507	D A T A Δ B
88	212410200530	L O C K Δ S
89	163712055305	I Z E Δ : Δ
90	023535353535	↓ x x x x x } Value (Decimal) of Maximum Block Size

In reference to the file specification block, the Sort manipulates the fields as follows:

1. Date time group-set from the parameter tape.
2. Security Class, file password, installation code, file name, and file number are used for matching purposes with the parameter tape.
3. Reel number is preserved, but not used.
4. File sentinels are preserved, and used, at present, to show a block label sentinel.
5. Block label is preserved, and used by the Sort to detect an EOF condition.
6. Type trailer is preserved, but assumed to be FILE and not used.
7. Item sumcheck is preserved, and used, if not zero, by the Sort for the value of the sumchecking location.
8. Item type is checked by Sort for FIXED. If FIXED is not present, an error halt results.
9. Item size is preserved and used by Sort for initialization.
10. Data mode is preserved and used by Sort for all In-Out orders.
11. Total and partial file size are assumed equal and used to compute various factors for Sort initialization.
12. File format is checked for HIREL. An error halt will result if any other configuration is present.
13. Max data block size is computed on basis of item size and reblocking will occur if possible for optimum tape efficiency.

8.3.3 Format for File Trailer Block

Octal and Fieldata configurations for the File Trailer Block are given in Table 8-3. Remarks that appear in the Fieldata and Remarks column and have an associated brace or arrow are further explained in the explanatory discussion following the table.

TABLE 8-3. OCTAL AND FIELDATA CONFIGURATIONS FOR FILE TRAILER BLOCK

Word Number	Octal	Fieldata and Remarks
1	xx0440010000	x 044_8 001_8 0000_8 Block Label 1st Character EOF Sentinel
2	043136251205	/type Δ
3	530513162112	: Δxxxx File or Tape
4	042306221205	/name Δ
5	530502000000	: Δ↓
6	353535353535	xxxxxx } Value of
7	353535353535	xxxxxx } File Name
8	353535353535	xxxxxx } (18 characters)
9	042332220712	/numbe
10	270553050000	r Δ : Δ
11	000000353535	xxx ← Value of File Number (3 characters)
12	041106311205	/date Δ
13	311622120514	time Δg
14	272432250553	roup Δ :
15	050000358535	Δ xxx } Value (decimal) of Date-Time
16	353535353535	xxxxxx } Group (9 characters)
17	040721241020	/block
18	051024322331	Δcount
19	055305000000	Δ : Δ
20	000035353535	xxxx ← Value (DEC) of File Block Count including this block (4 characters)
21	041631122205	/item Δ
22	102432233105	count
23	530500000000	: Δ
24	353535353535	xxxxxx ← Value (DEC) of File Item Count (6 characters)
25	040721241020	/block
26	053032071024	Δsubco
27	322331055305	unt Δ : Δ
28	000035353535	xxxx ← Decimal Value of File Block Count this Reel including this block (4 characters)
29	041631122205	/item Δ
30	303207102432	subcou
31	233105530500	nt Δ : Δ
32	353535353535	xxxxxx ← DEC Value of File Item Count this Reel (6 characters)
33	041316211205	/file Δ
34	303222101512	sumche
35	102005530500	ck Δ : Δ
36	353535353535	xxxxxx ← Value (Binary) of the File Hash Sum

With respect to the file trailer block, the Sort makes use of its fields as follows:

1. Block label - uses sentinel to determine end of data.

2. The file name is filled in on output as is the file number. These will be the same as those appearing in the file specification block.

3. Date time group is filled in on output as in the file name, etc.

4. The block count and item count are used by the first pass of the sort to check against its own computed counts. The Sort's computed counts are filled in on output.

5. The block and item subcounts are not used, and are set equal to the block and item counts.

6. If there was an item sumcheck word, the file sumcheck word will contain its value and will be tested against the sort's computed counts.

8.3.4 Format for Tape Trailer Block

Octal and Fieldata configurations for the Tape Trailer Block are given in Table 8-4.

With respect to the tape trailer block, the sort program does not read it or write it at present.

TABLE 8-4. OCTAL AND FIELDATA CONFIGURATIONS FOR TAPE TRAILER BLOCK

Word Number	Octal	Fieldata and Remarks
1	xx0510010000	x 051_8 001_8 0000_8 Block Label 1st Character EOT Sentinel
2	043136251205	/type Δ
3	530531062512	: Δ tape
4	042306221205	/name Δ
5	530502000000	: Δ ↓
6	353535353535	x x x x x x } Tape Name goes here
7	353535353535	x x x x x x }
8	353535353535	x x x x x x }
9	042332220712	/numbe
10	270553050000	r Δ : Δ
11	000000353535	x x x ← Tape Number
12	041106311205	/date Δ
13	311622120514	time Δ g
14	272432250553	roup Δ :
15	050000353535	Δ x x x } Date - Time Group
16	353535353535	x x x x x x }
17	041024222521	/compl
18	123112051316	ete Δ fi
19	211230055305	les Δ : Δ
20	000000006061	0 1
21	042506273116	/parti
22	062105131621	al Δ fil
23	123005530500	es Δ : Δ
24	000000000061	1
25	043124310621	/total
26	051631122230	Δ items
27	055305000000	Δ : Δ
28	353535353535	x x x x x x ← Item Count
29	042106303105	/last Δ
30	131621120530	file Δ s
31	252116310553	plit Δ :
32	050000000023	Δ n
33	040721241020	/block
34	300521063031	s Δ last
35	051316211205	Δ file Δ
36	530535353535	: Δ x x x x ← Total Block Count including this block
37	041631122230	/items
38	052106303105	Δ last Δ
39	131621120553	file Δ :
40	050000000000	Δ
41	353535353535	x x x x x x ← Item Count

8.4 GENERAL UTILITY PACKAGE

The utility package comprised of the following subroutines:

1. Core Dump - LPR or Flex. or PTP8
2. Tape Dump - LPR or Flex. or PTP8
3. Transfer Trace - LPR or Flex. or PTP8
4. Snapshot Routine - LPR or Flex. or PTP8
5. Data Generator - MT2(41)
6. Utility Read Routine
7. MAP Output Loader - Cards or paper
8. Save and Restore Routine

All of the above can be entered either from the console or directly (via a TRL) from a running program. At the end of execution, all registers, indices, and flip-flops are restored to the status they were in upon entry to the subroutine. Thus, it is possible to have periodic core dumps, tape dumps, and snapshots during the normal execution of a program.

The subroutine system is written so that it may be used in its entirety or in parts which simplifies the process of adding to or modifying the original routine.

The package is contained in a symbolic deck and has as its origin the symbolic location ORG. By punching an EQU card for the symbol ORG, the deck can be assembled in any block of 638 (counting 24 word card buffer) locations desired by the programmer. The same EQU card should be included in the programmer's deck so that references within his program to the utility package will be assembled correctly. A symbolic paper tape is also supplied.

A separate description of each subroutine follows. Any mentioning of printed output refers to the line printer for the card version and the flexowriter for the paper tape version. The symbol "out" is used in the device field of all printing output orders, thus allowing output printing to be specified as any MOBIDIC output device.

8.4.1 Core Dump:

Purpose: To dump the contents of the registers, indices, and flip-flops as well as specified core locations on to the output device.

Entry:

L	MOV	DCW, ORG + 189
L+1	trl	ORG + 114
L+2	RETURN	

DCW is a location the contents of which have the following meaning:

Sign bit:	+ dump in octal
	- dump in alphanumeric
Gamma Beta Field:	number of locations to be dumped
Address Field:	first location to be dumped

Upon dumping the required locations, registers are restored to their original status and control is transferred to L+2.

Entry from console:

Set the WSR keys as indicated for location DCW. Set ASR keys to ORG + 4. Press "start at ASR". Upon completion of routine, machine will HALT at location ORG + 18.

Format of output is as follows:

Line 1 DUMP from XXXXXX to XXXXXX

Line 2 Address of first word 1st word 2nd word 3rd 4th

Line 3 Address of fifth word etc. 5th word 6th word etc.

If all four locations to be printed on a line contain zeros, the line is not printed. Any gaps in the address sequence caused by such omissions are indicated by the sentence "zeros are omitted". However, negative zeros are printed.

Other Subroutines Used:

Save and Restore Routines

Snapshot Routine

8.4.2 Tape Dump:

Purpose: To dump a specified number of blocks of tape onto the output device.

Entry:

L	CLA	READ
L+1	LOD	COUNT,,ORG
L+2	TRL	ORG+210
L+3	RETURN	

Read:

This location contains the actual read instruction that the subroutine will use to read in one block (or a certain number of words) from the tape unit specified in the read instruction into the core location of the address of that instruction. If READ is negative, the tape is read ISN; if positive, it is read non ISN.

The tape must be positioned before entering the routine. The tape is not repositioned upon completion of the dumping.

Example: If value of READ is 70 40140 10000 it will cause 1 block to be read from tape 40_8 into location 10000_8 and dumped on the output device. This procedure is repeated by the number of times indicated by "COUNT".

Count:

The number in "COUNT" will determine how many times the "READ" instruction is executed (usually number of blocks).

Entry from Console:

Set the WSR keys as indicated for location READ. Set the Q register as indicated for location COUNT.

Set ASR keys to ORG+7. Press "start at ASR". Upon completion of routine, machine will halt at location ORG+18.

If the machine indicates a read error on reading the tape, the word "ERRT" is printed and the dump proceeds normally. Tape is read with NHC set so that only DVA's stop the dump.

Format of Output:

TAPE Δmm Δbbbb Δblocks

Blk 01 kkk words

1st word 2nd word 3rd 4th

5th 6th etc.

Blk 02 kkk words etc.

Zeros are omitted as in core dump.

kkk = number of words in block.

mm = device address in octal.

bbbb = number of blocks (decimal) to be dumped.

Other Subroutines Used:

Core Dump

8.4.3 Transfer Trace Routine:

Purpose: To execute the snapshot program after each transfer instruction is executed by the program.

Entry: There are two possible entries to the routine as follows:

1) TRL ORG+10

This will produce a snapshot only if the conditions of the instruction are met, e.g., TRP ZETA.

If the Accumulator is plus, a snapshot will occur.

2) TRL ORG+12

This will produce a snapshot after the execution of each transfer instruction.

Tracing will then commence at the instruction following the TRL instruction.

In order to leave the tracing mode, the program should execute a TRL ORG+398.

It should be noted that location zero is used by the trace program, and also the TRA flip-flop.

Entry from Console:

Put the address of the location (where tracing is to commence) into the PCS. Set the ASR to ORG+10 or ORG+12 depending on the type of trace desired. Then press "Start at ASR".

The trace must be halted by pressing the "HLT" button unless a halt is to be executed by the program being traced.

Format of Output:

See Snapshot routine.

Other Subroutines Used:

Save and Restore Routine.

8.4.4 Snapshot Printout:

Purpose: To cause the contents of the registers, indices and flip-flops to be printed out on the output device.

Entry: L TRL ORG + 36

L + 1 RETURN

When the snapshot has been printed, the registers, indices and flip-flops are restored to their original status and control is returned to L + 1.

Format of Output: PCS AR QR IR1 IR2 IR3 IR4 SENSE FLIP-FLOPS

If a flip-flop is reset, a "0" is printed.
If a flip-flop is set, a "1" is printed.

Entry from Console:
Set ASR to ORG + 2. Press "start at ASR". At end of execution machine will halt at ORG + 18.

Other Subroutines Used: Save and Restore Routine.

8.4.5 Data Generator

Purpose: To generate binary blocks of data on magnetic tape (41_8) in the standard block format. The structure of the data is controlled by the parameters. There are two entries. The first entry (NOSPEC) will rewind the tape first while the second entry (YESPEC) will start to write data from its present position. In each case, a file trailer is generated as the last block.

Entry 1: TRL ORG + 403

Entry 2: TRL ORG + 411

An octal paper tape with the following parameters must be ready in the tape reader:

1. ROKα, PTP8, 10_{10} α = Location to read parameters
2. Number of words per block
3. Starting value for data generator
4. Stepping constant
5. Number of words per logical record

6. Number of Blocks to generate
7. ISN indicator
8. File name
9. File name
10. File name
11. File name
12. Word of all sevens

A HIREL File trailer is written at the end of the data. The trailer is written in the same mode as the data. Block counts, file name, and item counts are filled into the trailer.

After generation of the tape, control is returned to the instruction following the TRL order. The tape is not rewound nor are the registers restored.

Entry from Console:

Entry 1: Set ASR switches to ORG + 14. Press start at ASR.
Entry 2: Set ASR switches to ORG + 411. Press start at ASR.

The computer will halt at ORG + 18 after tape generation.

Sample: If a Parameter tape were punched as follows:

0 720122005050	Control
0 000000000764	500_{10} words per Block
0 000000000777	Starting Constant
10000000000001	Stepping Constant
0 000000000024	Number of words per item (20_{10})
0 000000000024	Number of Blocks per file (20_{10})
0 000000000000	Data is non 13N
0 300622252112	Sample
0 051106310605	ΔDataΔ
0 141223122706	GenerΔ
0 312427056061	TORΔ01
0 777777777777	SENTINEL

The data would be generated as follows:

Twenty (decimal) Blocks of data + one 36 (decimal) word file trailer. Each item would be 20 (decimal) words long and the first would be 20 words of 776 (octal), the second 20 words of 775 octal, etc. There would be 25 (decimal) items to each tape block. Each tape block of data (not trailer) would be 501 (decimal) words long where the first word is the standard block label. If entered at ORG +403 the tape (41_8) will be rewound, tape erased (5 blocks), and written. If entered at ORG + 411 writing of the tape will occur first.

Other Subroutines Used: Utility Read Routine.

8.4.6 Utility Read Routine

Purpose: To read batches of specified octal words from paper tape into specified locations in core memory.

Entry: TRL ORG + 561

On return to main program

Will not restore ACC, QRG, or PCS

Format of Paper Tape

First word on the tape (13 octal characters) must be ROK XYZ, PTR8, LMN. Where XYZ is the starting location for read in of data and LMN is the number of words to read in. Next LMN words are the data to be read in. The next word may be either another ROK for more data or a word of seven's which will terminate the routine. If the first word is octal, but not a ROK or sentinel an error halt will occur.

Example:

0720012001000	Read 1 word into 1000_8
0431000107320	
0720022000333	Read 2 words into 333_8
0000000000000	
1042023426161	
0777777777777	Terminate Read

Other Subroutines Used: None

8.4.7 MAP Output Loader:

Purpose: To read the absolute binary paper tape or cards of a MAP assembly into core.

1. Paper Tape

   Entry: TRL ORG + 21

   Entry from Console: Set ASR to ORG and press "Start at ASR"

2. Cards

   Entry: TRL ORG + 589

Special Features:

1. Checks hash sum and indicates error, if any.
2. A punch in the 12R sign bit will cause the hash sum to be ignored.
3. A punch in the 12L high order magnitude bit for a Transfer Card will cause a halt before entering the program.

   This latter punch may be automatically recorded on the Binary Transfer Card by punching "End X, 1" on the symbolic end card. After this halt, depressing the Start at PC push-button will effect normal transfer to routine.

Other Subroutines Used: None

8.4.8 Save and Restore Routine

Save:

Purpose: To save the contents of the registers and indices.

Entry:

L	MOV	IRET, ORG + 113
L + 1	TRU	ORG + 90
L + 2	RETURN	

where IRET contains a TRU L + 2

When the registers and indices have been saved, control is transferred to location ORG + 113.

Restore:

Purpose: To restore the contents of the registers and indices to their conditions when saved.

Entry: TRU ORG + 99

Exit is made with a TRS instruction after the PCS is restored to its SAVED condition.

Other Subroutines Used: Snapshot Routine

SECTION IX

MALFUNCTION AND CONTROL ROUTINES

9.1 REGISTER AND CORE SAVER

Symbolic Label: RACS1

PURPOSE:

1. To write on magnetic tape a self-restoring copy of core memory and selected registers under either programmer or operator control.

The following information is saved by RACS1:

a. All of core memory except those locations required to "bootstrap" the save routine into memory.

b. The following registers:

1. The accumulator
2. The Q-Register
3. The B-Register
4. Each index register
5. The "PCS" register
6. Real-Time output register
7. Real-Time address register

c. The status of each general purpose sense flip-flop.

d. The status of the following special flip-flops:

1) OVA	4) IOA2	7) ROP1
2) ROR38	5) RPE	
3) IOA1	6) ROBB	

2. To restore a previously "saved" copy of core memory and selected registers under either programmer or operator control. To print on the Flexowriter the status of any flip-flop which had been manually set or reset (only in the case of operator control).

Coding Information

1. Calling Sequence - Save Routine

A. Under Programmer Control

The programmer may save the contents of the converters (RACS1 cannot). He must execute a routine similar to the one below:

```
MOV   BRG, COM        Save BRG
MOV   IR2, COM + 1    Save IR2
LOD   *+ 2, 0, IR2
SNS   *+ 1, 0, TPE
WAN   5, MT2, 511     Write 5 blocks of tape erase.
TRX   *- 2, 1, 0
SNS   *+ 1, 0, ISN
WAN   0, MT2, NEC     To create the correct amount of space on
                      MT2, for the restore routine.
SKP   MT1, 2          To skip over the "bootstrap" blocks on RACS
                      tape.
SNS   *+ 1, 0, ISN
WAN   0, MT2, NEC1    To save enough of core to make room for
                      RACS.
SEN   *, 0, MT2       Wait for write to finish.
RAN   0, MT1, 257     Read in RACS.
SEN   *, 0, MT1       Wait for RACS to finish reading to memory.
LOD   COM + 1, 0,     Restore IR2
        IR2
LOD   COM, 0, BRG     Restore BRG
TRL   PRSAU           Transfer to "programmer save" entrance.
```

NEC = Number of core locations (215_8 or 141_{10}) used by the restore program.

NEC1 = Number of core locations (460_8 or 304_{10}) used by save and restore routine.

B. Under Operator Control

The operator should save the contents of the converters and the contents of the program counter. In addition, he should record the status of the ISN flip-flop and put the switch in the reset position.

The operator may then read the first block of MT1 into locations 8182_{10} (17766_8) thru 8191_{10} (17777_8) and transfer control to location 8182_{10} (17766_8). This will result in the execution of the routine shown above but with the following exceptions:

1. The "SKP" instruction will not be executed.
2. The last instruction will be a transfer to "OPSAV", the operator save routine.

The execution of these orders will "bootstrap" the main RACS1 program into core.

2. Calling Sequence - Restore Routine

A. Under Programmer Control

The programmer need only set the ISN flip-flop, read the first block from MT2 into location zero, and transfer control to the "Programmer Restore" entrance.

RACS1 will not print out any console conditions when memory is restored via the "Programmer Restore" entrance.

B. Under Operator Control

The operator is only required to set the ISN flip-flop, read the first block from MT2 into location zero, and transfer control to the "Operator Restore" entrance.

Information about the status of the sense flip-flops which were manually locked will be printed on the Flexowriter to enable the operator to restore them to their original settings.

Other Subroutines Required

None

Error Halts

None

Sense Flip-Flops Used

None

Number of Storage Locations

305

"Common" Locations Used

31

Approximate Time

1. Programmer Control - takes approximately 5 seconds for a seven memory machine. (770 ms + 695 ms/memory unit of 4096 words).

2. Operator Control - basically the same as the above plus the time required for operator manipulation and the time required to print the condition of any manually locked flip-flops.

RESTRICTIONS:

1. Ten consecutive memory locations must be sacrificed to provide space for the "bootstrap" program. These locations, 8182_{10} - 8191_{10} were arbitrarily selected but may be changed by any installation with only minor changes to the RACS1 program.

2. The RACS1 program will usually be contained on magnetic tape 1 (MT1) and it will normally use MT2 as its "save" tape and to obtain restoring data.

3. The positions in which the programmer has his "bootstrap" program must be at least NEC1 (465_8 or 309_{10}) locations after location zero.

4. Since the RACS1 program has to overwrite itself with core memory before executing a return to the original program, the programmer should take care that his return address is in an area not used by the RACS1 program. He should make sure that all of core memory is restored before he continues execution of the original program. This may be done easily be sensing MT2.

METHOD:

To save core and register information the following information is written on MT2 in ISN mode:

1. Block No. 1 Restore Routine and register flip-flop conditions.
   Block No. 2 Partial memory dump (equivalent in length to size of the RACS program).
   Block No. 3 Remainder of memory dump.

2. To restore core and register information.
   The restore routine first restores the condition of the registers and flip-flops. When this task is completed it overwrites itself with original core memory as is contained in blocks No. 2 and No. 3 on MT2. When restoration is completed the program halts if under operator control, or returns to the orginal program, if under programmer control. (see Appendix A, page 9-6.)

REMARKS:

At present RACS1 occupies core locations $0\text{-}465_8$.

"PRSAV"	(programmer save)	$= 244_8$
"OPSAV"	(operator save)	$= 251_8$
"PRRST"	(programmer restore)	$= 0_8$
"OPRST"	(operator restore)	$= 4_8$

APPENDIX A

FORMAT OF TAPES USED BY RACS

1. MT1 (NSIN)

Operator	
Bootstrap routine	Blocks 1 + 2
RACS1	Block 3

2. MT2 (ISN)

"Restore" program with register and F. F. Conditions (NEC words)	Block 1
Core used by RACS1 (NEC1 words)	Block 2
Remaining cores	Block 3

9.2 TAPE ERROR ROUTINE

Symbolic Label: TERM1

PURPOSE:

To provide a means of testing and guaranteeing (as far as possible) that information just written on magnetic tape was not recorded on a "bad" spot.

To provide a log on the 8-hole paper tape punch, consisting of a written record of the "bad" blocks thereby giving a measure of the "goodness" of the tape.

USAGE:

A. Preparation of Cards or Tapes

An input magnetic tape with information written thereon, and a blank tape. (See Method)

B. Input Format

See Section D, Coding Information

C. Output Format

The input magnetic tape with all "bad" areas of tape erased and all recorded information "moved" to good areas on the input tape. All possible incorrect data from the input magnetic tape recorded on the originally blank tape. (See Method)

D. Coding Information

Calling Sequence

```
A)      TRL     TERM, 1, TYPE
A + 1)  HLT     BUFST, NSI, BLKLB
A + 2)  MOV     BUFSZ, TOL
A + 3)  RAN     MRJT, MTEST, NUMBK
```

A + 4) (Tape Label)

A + 5) (Label 1) only the bits 36-31 are used for labelling

A + 6) <——— Returns control here

Entry and Exit Conditions

Entry Conditions	Exit Conditions
ACC: BLKNUM	ACC: (Transient Quantity)

Definition of Terms Used Above

BLKNUM - is an integer using bits 1-12 telling the number of the first block. This number is used for printing and identification.

BLKLB - is a 1 or 0 respectively indicating whether a block label is available as the first word of each block on MTEST. (For definition of block label see SPECIFICATIONS FOR STANDARD FIELDATA TAPES revised 31 July 1959)

TYPE - is either 1 or 0 respectively showing whether BUFST contains all the output information in sequence or BUFST is for use as eraseable storage.

BUFST - is the starting location of the buffer.

NSI - is either 1 or 0 respectively showing whether the information to be tested was written in the ISN mode or not.

BUFSZ - is the size of the buffer starting at BUFST, (e.g. it must be at least 1 block in length for corrective action to take place).

TOL - is the maximum number of "bad" blocks TERM will accept per entry.

MRJT - is the tape address for any rejected information.

MTEST - is the tape address of the tape to be tested.

NUMBK - is the number of consecutive blocks to be tested on MTEST per entry. (e.g., the count of the number of blocks back from the read head)

(Tape name) - is an alphanumeric word giving the tape name or abbreviation for the log. It will be printed as the log header.

(Label 1) - is an alphanumeric word used to mark the block label of any blocks which may contain erroneous information. Will be ignored if BLKLB is 0. Tests 36-31 of label 1 will be masked into bits 36-31 of the first word of each possibility erroneous data block.

Other Subroutines Required

TERM1 will, in certain instances, automatically re-enter itself. For example, when the reject tape is used, or whenever errors are detected which TERM1 can correct.

Error Halts

1. When TOL is exceeded.
2. When BUFSZ is too small.
3. When the tape address or the block count is unmeaningful.
4. When TERM1's block counts become unsychronized.

Sense Flip-Flops

TERM1 uses five (5) sense flip-flops. SFF1-SFF4 are used internally by the program and must initially be in neutral. In the case of the number of errors exceeding the TOLerance, TERM1 will halt to allow the operator to set the reset SFF5. If SFF5 is set, TERM will pick up the WSR in which a tape address and a block count must have been placed by the operator prior to hitting "start at PC". The octal block count must occupy bits 1-15 and the octal tape address must occupy bits 16-21. If SFF5 is reset, TERM will rewind MTST, restore registers and return to the main program.

Common Locations Used

None

Approximate Time

(Depends on the number of blocks being checked and the number of errors encountered)

E. Accuracy Information

If information is memory contained - 100% reliability, but if not memory contained the particular block or blocks in question have a low reliability rate.

RESTRICTIONS:

1. If more than one block of information is to be checked, the blocks must all be in the same sign mode (i.e., either all ISN or all non-ISN) and must all be of the same size (have the same number of words).
2. No block may exceed 511 words in length.
3. A single block may be of any size from 1 to 511 words.
4. The Tolerance (TOL) must not be greater than 4095.
5. Tapes on which "label 1" may be written should have a block label as their first word, else data may be overlain by the label.
6. TERM1 does not take end of tape into consideration.

METHOD & DESCRIPTION OF TERM1 ACTION:

1. On the initial entrance, MTEST is read reverse into non-existent core (60000_8) and as each block is read, the IOA is checked. This process continues until NUMBIK blocks have passed the tape head. If no alarm has been detected during the reverse pass, the tape is repositioned, and an entry on the punch is made counting the number of blocks checked. If an IO alarm is present on entry to TERM1, this condition will also be printed (since IOA cannot be set). The main program is then re-entered.
2. If any I-O alarms were noted, and TOL was exceeded, an option is allowed to either copy all of MTEST or to rewind and halt.

3. If any I-O alarms were noted and TOL was not exceeded, either of two sequences take place:

   a. If all the information scanned by TERM1 is present in BUFSZ in sequence, all "bad" blocks are erased, all information from BUFSZ is rewritten (using old good blocks). The remaining information is written in a new area (area of tape just "ahead" of the initial starting point). This area is then automatically checked by new entry to TERM1. No use is made of label 1 or MTRJT in this case.

   b. If any information scanned by TERM1 is not available in core, BUFSZ will be considered to be eraseable storage.

      1) If BUFZ is large enough to contain NUMBLK blocks of information, all blocks are read into BUFSZ, label 1 is placed in the first word of each block. All bad blocks are copied onto MTRJT and all bad areas of MTEST erased. The information (BUFSZ) is then rewritten into the good areas of MTEST and the remaining information written on a new area of MTEST. TERM1 is automatically re-entered to test this new area on both MTEST and MTRJT.

      2) If BUFSZ is not large enough to contain all the tape information, but is large enough to contain one (1) block, a slower correction cycle is initiated. All tape blocks from the initial bad block are moved up past the initial read ahead position, and all the area from the first "bad" block to the initial read head position is erased. Those "bad" blocks are also labelled, if specified, and written on MTRAJT, but are not tested.

      3) If BUFSZ is not large enough to contain one block, then a message is punched out, no erasure or "move up" is done, but the tape is repositioned and control is returned to the main program.

This routine will print (punch) the following status report(s). If the printing is to be suppressed, TRU *+ 1's may be inserted to bypass printing. Those fields shown below with ( ) are figurative, while those fields without ( ) are literals.

Log Entries (Punch)

1. (Tape Label) Block (xxxx)?	This is the name of the tape. We have read block twice, once with error, once without; considered good.
2. (Tape Label) (xxxx) Good	We have found no bad blocks, xxxx were tested and are good.
3. IOAXX originally	IOAXX was on when TERM was entered. We have turned it off.
4. (IOE) exceeded tol (TOL)	(IOE) errors encountered which were more than (TOL).
5. Copying on (TA), (xxxx)	We are copying on (TA) xxxx blks from the beginning of MTEST.
6. (xxxx) Blks copied RWD (MTEST)	We have finished copying xxxx blocks from MTEST to TA and are rewinding MTEST.
7. BUFSZ too small	BUFSZ is not big enough to hold NUMBLK Blocks.
8. Block counts off	The internal counts are out of synchronization. We must halt. (Flex also)
9. Multiple errors cannot fix	We cannot ascertain trouble with tape. Seems to be more than parity. We will halt. (Flex also)
10. BUFSZ less than 2 Blks. xxxx Blks bad	The buffer will not hold two blocks. We switch to an extremely slow mode.
11. BUFSZ less than (ADC) We cannot do any corrections xxxx Blks bad	The Buffer size is smaller than (ADC) value. We can do no corrective action. We will reposition and exit from TERM.
12. Block xxxx written on (MRJT)	The numbered block has been written on MRJT.

13.	xxxx Blocks on (MRJT) from MTEST	Total count of blocks written on MRJT.
14.	Machine Errors, cannot continue	Cannot ascertain trouble, will halt. (Flex also)
15.	Erased xxxx Blks, xxxx were good am retesting.	Have finished copy and corrected. Will now test those areas not previously written on.

SECTION X

MOBIDIC TAPE SELECTION AND UPDATING ROUTINE

10.1 MOBIDIC TAPE SELECTION AND UPDATING ROUTINE

Symbolic Label: TSUP1

PURPOSE:

The Tape Selection and Updating Routine (TSUP1) is a service routine designed for the following general uses:

1. To update Magnetic Tapes.
2. To check tapes mounted on a tape drive.
3. To select the appropriate blocks of information specified by the programmer.

In order to accomplish these objectives in the most general sense, TSUP1 is provided with twelve control functions some of which have subcontrol functions. The following set of general control functions show the extent of TSUP1's vocabulary. For the specific formats and capabilities of each word in TSUP1's vocabulary, see the section entitled CONTROL WORD SPECIFICATIONS. By a judicious use of the specific control words, TSUP1 may be made to selectively rewrite a tape.

Equipment requirements:

1. Three magnetic tape units.
2. Two converter units.
3. Two memory units.

GENERAL CONTROL FUNCTIONS

The control functions of TSUP1 encompass the following five categories:

1. The check reading of tapes.
2. The rewriting of specified blocks or words on a tape.
3. The positioning of a tape.

4. The selective comparison of tapes.

5. The copying of one tape to another.

In category 5 above there are four sub-categories as follows:

a. Straight copy of a specified amount of information.

b. Copying with selective replacement of information.

c. Copying with selective insertion of information.

d. Copying with selective deletion of information.

Some of these categories apply only to magnetic tapes, some apply to both magnetic and paper tape. See section entitled CONTROL WORD SPECIFICATIONS for further discussion.

USAGE

In order for TSUP1 to perform a control function, information must be supplied to it. If only a standard amount of information is required, this information is grouped and called the "control word". Some of TSUP1's control functions require more than a standard amount of information. In this case, the additional information is grouped into a "subcontrol word". A set of control and subcontrol words form a control function.

TSUP1 requires for its operation a sequential string of control words. Each control word will be fully processed before the next control word is executed. Any erroneous control words or improper usage of the control or subcontrol words will cause TSUP1 to make an error indication, skip the erroneous control word, and execute the next control word in sequence. The list of control words is entered to TSUP1 from paper tape. The control words may, if desired, be "keyed in" manually.

All input expressions; control words, subcontrols, words to be inserted, replaced etc., must be typed on a separate line so that a carriage return separates them. All must begin at the extreme left of the paper, so that their left margin is straight. No indentation is permitted. Octal words need not include high order zeros.

The series of subcontrols belonging to a given control word must be terminated by the character e on a line by itself and to the extreme left. The entire set of control functions must be terminated by the three-character word end on a line by themselves and to the extreme left. If the control words are to be read in one batch, a stop code must follow the end characters. If they are to be read in one at a time, a stop code must follow the last character of the information belonging to each subcontrol.

GENERAL FORMAT OF INPUT CONTROL AND SUBCONTROL WORDS

In order to systematize the control and subcontrol words and to make their use convenient, a general format was designed for both the control words and the subcontrol words. These formats are fixed in size. The size of the control word is fixed at 23 characters including spaces, while the size of the subcontrol word is fixed at 14 characters including spaces. A control word must be separated from the next control word or subcontrol word by a carriage return. In all the following descriptions of the control and subcontrol words, small letters have been used to indicate a literal expression and capital letters indicate that a symbol has been used.

The general format for a control word is:

KKK Z II L TTT OO L XXX

Where:

KKK - is a three letter mnemonic abbreviation for the control function.

Z - is a single character denoting the sign status of the specified tapes. It is either y for ISN or n for non ISN.

II - is two alphanumeric digits denoting the octal address of the input device.

L - is a single alphabetic character restricted in choice to either r for rewind of the device immediately preceding it, p for reposition without check read of the device, c for reposition with check read and i for ignore positioning. When II is the address of a paper tape device, this character is ignored.

TTT - has varying uses depending on the control function. See section CONTROL WORD SPECIFICATIONS for specific meaning.

OO - is two alphanumeric digits denoting the octal address of the second or output device.

XXX - is 3 alphanumeric digits expressing in decimal, the total number of blocks or words over which the control word is to exercise its function.

In any case where the above defined character sets do not apply, the format of the control word will contain an alphanumeric zero.

The general format of a subcontrol word is:

DDD EEE FFF GG

Where:

DDD - is three decimal digits denoting the number of consecutive words or blocks over which the operations are to be performed.

EEE - is three decimal digits denoting the block at which operations will be started.

FFF - varies depending on whether control word is the "word" type or "block" type.

GG - is the octal address of a tape on which the information to be inserted or replaced has been written. It may be the same device the subcontrol words are coming from.

CODING INFORMATION

CONTROL WORD SPECIFICATIONS

In the following, 0 will be used to indicate an alphanumeric zero and O will signify the letter "o". Small letters will be used to indicate literals and capitals for symbolic expressions. It is convenient to first list those control functions which have no subcontrol.

CONTROL FUNCTIONS WITHOUT SUBCONTROL

A. Straight selective copy.

cpy Z II L TTT OO L XXX

This function copies information from device II onto device OO.

Notes:

1. II may be either a magnetic or paper tape device.

2. OO must be magnetic tape.

3. If II is a magnetic tape address and TTT is zero, XXX blocks are copied from II to OO.

4. If II is a magnetic tape address and TTT is not zero, XXX words beginning at word TTT will be copied on to OO.

5. If II is a paper tape device, TTT is not used and should be zero.

6. If II is a paper tape device, and XXX is zero, paper tape will be copied up to a "stop code".

7. If II is a paper tape device, and XXX is not zero, XXX words will be copied from II to OO.

8. The other control characters perform as explained under INPUT FORMAT.

B. Comparison of two magnetic tapes.

cpr Z II L TT0 OO L XXX

This function compares the information on tape II and OO. In each case of discrepancy a message consisting of the two disagreeing words and the tape locations, i.e., block and word, is written on device TT.

Notes:

1. Both II and OO must be magnetic tape addresses.

2. A word by word comparison is made over XXX blocks.

3. Whenever a block of II is unequal in size to the corresponding block of OO, the extra words are written on device TT.

4. TT may only be the paper tape punch or the flexowriter.

5. The other control characters perform as explained under INPUT FORMAT.

C. Positioning of magnetic tapes.

pos 0 II L TTT OO L XXX

The function allows the positioning of tapes II and OO either forward or backward TTT and XXX blocks respectively. Their positioning is independent.

Notes:

1. Both II and OO must be magnetic tape addresses.

2. The L characters are used to denote respective direction of motion of II and OO. Therefore either b for backward or f for forward are the only permissible characters for the L fields.

3. Magnetic Tape II will be positioned TTT blocks forward or backward depending on its L field.

4. Magnetic tape OO will be positioned XXX blocks forward or backward depending on its L field.

D. Rewinding of Magnetic tapes.

rwd 0 II 0 TT 0 OO 0 XX 0

This function rewinds up to three tapes.

Notes:

1. II, TT, OO are magnetic tape addresses.

2. When less than three tapes are to be rewound, the other fields must be zero filled.

E. Check reading.

crd Z II L TTT OO L XXX

This function reads and checks for parity error on tapes II and OO. It provides for a re-read sequence when parity error is detected.

Notes:

1. Tapes II and OO will be check read. They may be either paper or magnetic tape. If either is paper tape, no re-read can be done.

2. Both tapes must have been written in the same sign node, either ISN or NISN. If they have not, an I/O alarm may occur.

3. TTT is a count associated with II. If II is magnetic tape, TTT is a count of the number of blocks to be checked on II. If II is a paper tape, then TTT is a count of the number of words to be checked.

4. If either II or OO is a paper tape device, a zero should be used in its L field.

5. If parity error occurs, and automatic re-read is initiated. This is performed three times. If the error persists, the block number and the tape address are written on the flexowriter and the routine skips the bad block and continues.

F. Updating the block label.

ubl Z II L TTT OO L 000

This function updates the block labels of TTT Blocks on magnetic tape II. It provides an option for updating the block labels on the input tape (II) or producing a new tape (OO) with updated block labels.

Notes:

1. II is the tape whose block labels are to be checked.
2. TTT blocks on II will be checked.
3. If OO and II are the same, any updating will be done on II.
4. Both the word count and the block hash sum are checked by this control function.
5. If the word count is incorrect, the corrected word count will be inserted in the block label.
6. If the block hash sum is incorrect, an automatic re-read is initiated. If the error persists, the new hash sum is inserted.
7. Tape II must have a block label on each block to be tested. This block label must follow the standard block label conventions as depicted in the document SPECIFICATIONS FOR STANDARD FIELDATA TAPES revised 31 July 1959.

CONTROL FUNCTIONS WITH SUBCONTROL

The following control functions will require at least one subcontrol. A maximum of twelve subcontrols are allowed per control word. In general, the control word specifies tapes to be operated on and each subcontrol specifies a sub-operation on those tapes. Each set of subcontrols is terminated by the character e on a line by itself.

Within this area there are two general types. The first type operates on blocks while the second operates on words.

BLOCK CONTROL FUNCTION

A. Block insertion.

isb Z II L TTT OO L 000
DD EEE FFF GG

This function copies EEE blocks from II, then skips FFF-1 blocks on GG, and copies DDD blocks from GG. The copies blocks are put on tape OO. This is, in effect, the combination of several simpler control functions.

Notes:

1. Three tapes are involved in this operation, i.e., II OO, GG. II and OO must be magnetic while GG may be either magnetic or paper.
2. The input tape II and the inserting tape GG together form the inserted output tape OO.
3. TTT is the total number of blocks to be retrieved from tape II.
4. DDD is the number of blocks which are to be inserted after block EEE.
5. If a set of subcontrols are used, the numeric values of EEE for each subcontrol must be ascending, or an error printout will occur. That is, if a subcontrol refers to an insertion after block K, the next subcontrol may not refer to a block less than or equal to K.

B. Block Deletion.

dlb Z II L TTT OO 000

DDD EEE 000 00

This function copies TTT blocks from tape II, deleting blocks specified by each subcontrol. GG is not used.

Notes:

1. II and OO must be magnetic tapes.
2. DDD consecutive blocks are deleted starting with block EEE.
3. TTT is the total number of blocks on which deletions are to be performed. That is, OO will have less than TTT blocks on completion of the control function.

C. Block Replacement.

rpb Z II L TTT OO 000

DDD EEE FFF GG

This function copies TTT blocks from II into OO replacing specified blocks by blocks from GG.

Notes:

1. II, OO and GG must be magnetic tapes and the sign mode of each must be the same.

2. Each subcontrol specifies one set of replacements.

3. Up to 12 subcontrols may be used.

4. EEE blocks are copied from tape II, then DDD blocks starting at block FFF are copied from GG while DDD blocks are skipped over on tape II. This process is performed for each subcontrol.

5. When all subcontrols have been performed, additional blocks are copied from II until tape GG holds TTT blocks.

6. Replacement begins at block EEE rather than the block after EEE as insertion does.

WORD CONTROL FUNCTIONS

In all cases of the word control functions, the GG field of each subcontrol is zero filled, e.g. GG<-00. If a number of subcontrol words are given EEE must be ascending.

A. Insertion by word

isw z II L TTT OO L 000

DDD EEE FFF 00

WORD 1

WORD 2

WORD 3

e

This function copies TTT blocks from tape II into tape OO inserting information as specified by the subcontrol words.

Notes:

1. II and OO must be magnetic tape addresses.
2. TTT specifies the total number of blocks from II to be copied.
3. DDD words will be inserted after word FFF of block EEE of the input tape II.
4. DDD must equal the number of WORD 1, WORD 2's, etc which appear directly after the subcontrol.
5. As many words as desired up to 510 may be inserted consecutively.
6. If a block, by virtue of the number of insertions, becomes greater than 511 words, an error is indicated.
7. The words to be inserted must be octal numbers. They may never have more than 12 digits and a sign. If no sign is specified, positive will be assumed.

B. Deletion by word

dlw Z II L TTT OO L 000

DDD EEE FFF 00

This function copies TTT blocks from tape II, deleting the words specified by the subcontrol words, onto tape OO.

Notes:

1. II and OO must be magnetic tapes.
2. TTT specifies the number of blocks on II to be copied.
3. DDD specifies the number of consecutive words to be deleted.
4. EEE specifies the block in which the deletion is to occur.
5. FFF is the word immediately after which deletions are to occur.
6. Up to 511 consecutive words may be deleted with one subcontrol.

C. Replacement by word

rpw Z II L TTT OO L 000

DDD EEE FFF 00

WORD 1

WORD 2

e

The function copies TTT blocks from magnetic tape II, replacing words as specified by the subcontrol, onto tape OO.

Notes:

1. II and OO must be magnetic tape addresses.
2. TTT specifies the total number of blocks from II to be copied.
3. The replacing word (s) e.g., WORD 1, WORD 2 etc., must be expressed as octal numbers with or without sign.

4. The replacement starts at word FFF of block EEE and continues for DDD words.

5. Up to 12 subcontrol words may be written for each control word.

6. Up to 511 words may be replaced.

ERROR HALTS

There are nine (9) specific error halts in TSUP1. More can be added if desired. When an error is encountered TSUP1 will print on the flexowriter a Fieldata e, a number from 1 to 9 and the control word causing the difficulty. The computer will be halted.

If the "start at PC" switch is depressed, the control word causing the error will be skipped and the next control word processed.

A list of the error printouts and their cause is below:

Error printout	Cause
e1	The KKK field is illegal.
e2	There are more than 12 subcontrol words attached to the control word printed out.
e3	There is a non-octal character in the subcontrol word.
e4	The subcontrol words for an insertion control word are causing the output block to be larger than 511 words.
e5	The EEE portions of a set of subcontrol words are incorrectly ordered.
e6	The type of device is inconsistent, e.g. paper tape specified instead of magnetic tape.
e7	The first L field of the position control function is neither b nor f.

e8	The second L field of the position control field is neither b nor f.
e9	An illegal character exists in the control word printed out.

Sense Flip-Flops

Sense flip-flops 1 and 2 are used by TSUP1 for control purposes. The setting of all other flip-flops is immaterial.

Sense flip-flop 2 is normally reset and is used only in case of error. It allows the erroneous control word to be corrected.

Sense flip-flop 1 indicates whether the control words are to be read all at once or one at a time. If it is set one control function at a time will be read, however a "stop code" must be placed on the control tape immediately after the control function. If it is reset, all control words are read in to core and then executed.

The use of Sense flip-flop 2 can be seen through an example. Suppose we had 100 control functions to be performed and after the 57th. had been performed an error was detected in the 58th. control word. This would cause an error print-out and a halt. Sense flip-flop 2 should then be set, the corrected control word punched and loaded in the reader, and the start at PC switch depressed. This causes the corrected control word to be read in and executed. Upon completion of the function, the computer will print out "Reset SFF2". After this is done the computer will continue execution of the rest of the control words.

Number of Storage Locations

2500 words.

REMARKS

All block counts, etc., in TSUP's vocabulary are relative to the present tape position. That is, block number 1 is the block to be read next.

For example, if a tape had been read forwards any number of blocks, and it was now desired to delete the 10th word in the next block, the EEE field would 001.

SYLVANIA ELECTRONIC SYSTEMS
Government Systems Management
for GENERAL TELEPHONE & ELECTRONICS

63 SECOND AVENUE, WALTHAM, MASS.