SDS TECHNICAL INFORMATION

SDS 900 SERIES
REAL-TIME FORTRAN II

SDS 90 10 48B

January 1966

SCIENTIFIC DATA SYSTEMS

Price: $1.25

SDS 900 SERIES
REAL-TIME FORTRAN II

SDS 90 10 48B January 1966

SCIENTIFIC DATA SYSTEMS/1649 Seventeenth Street/Santa Monica, California/UP 1-0960

Printed in U.S.A.

REVISIONS

This publication, SDS 90 10 48B, dated January 1966, supersedes the SDS 920/930 Real-Time FORTRAN II Technical Manual, SDS 90 10 48A. All revisions and corrections that appeared in an addendum to that manual, dated November 1965, have been incorporated in this latest edition, and they have been indicated by a vertical line in the margin of the page.

The inclusion of these changes modifies the manual so that it now applies to both 920/930 FORTRAN II and to 910/925 FORTRAN II; hence the change in the title of the manual to "SDS 900 Series Real-Time FORTRAN II."

SDS PROGRAM LIBRARY
PROGRAM DESCRIPTION

Catalog No.: 920/930 = 212017B
910/925 = 112028

IDENTIFICATION:	SDS 900 Series Real-Time FOkTRAN II
AUTHOR:	R. Derin/C. Martin, SDS; Beckman Instruments, Inc.
ACCEPTED:	17 September 1965
COMPUTER CONFIGURATION:	Any 900 Series Computer with at least 8000 words of memory.
PURPOSE:	To provide a Real-Time FORTRAN II System for the 900 Series Computers
PROGRAMMED OPERATORS:	FORTRAN Run-Time System
SUBROUTINES REQUIRED:	None
STORAGE:	See individual program descriptions (i.e., Compiler, Loader, and Run-Time Monitor)
TIMING:	Not applicable
SOURCE LANGUAGE:	SYMBOL, META-SYMBOL
LOADING PROCEDURE:	See SDS 900046 FORTRAN II Operations Manual
USE:	See SDS 900003 FORTRAN II Reference Manual

CONTENTS

1. INTRODUCTION

This manual describes the SDS/Beckman Real-Time FORTRAN II System for SDS 920/930 Computers. The "real-time" capability in the system was implemented by Beckman Instruments, Inc.

The manual has been designed for use by experienced systems programmers, and it contains a detailed description of the implementation of 920/930 RTF II.

The primary design criteria for 920/930 RTF II was to produce a real-time system with full interrupt and recursive capabilities. To implement this required: 1) modifications and additions to the standard SDS FORTRAN II system at the compiler, loader, run-time monitor, and library levels; and 2) creation of special purpose programmed operators (POPs). 920/930 RTF II is a FORTRAN based system operating under the run-time monitor; however, SYMBOL capability has been provided for, and most of the system routines can be called either by FORTRAN or by SYMBOL code.

2. COMPILER

SECTION 1. FORTRAN INTERRUPT SUBPROGRAMS

A. FINT (or CONNECT)

Form: CALL subroutine (arg 1, arg 2, . . . , arg m, FINT(n)) or CONNECT subroutine (arg 1, arg 2, . . . , arg m), n

This function causes "subroutine" to be called when interrupt n occurs. The arguments arg 1, arg 2, . . . , arg m are set up for subroutine at the time of the interrupt. No special coding is required for subroutine, but it must be prepared to accept m arguments.

subroutine = name of any normally coded FORTRAN subroutine
arg i = any integer or floating point constant or variable; should not be an expression
n = integer constant, variable, or expression between 0 and 31 but not greater than the number established by ASSIGN.

Examples:
```
CALL SUB1 (FINT (2 * N+3))
CALL SUB2 (1.0, X, J, 3, FINT (0))
CONNECT SUB3 (A, B), 2
```

B. SINT

Form: CALL subroutine (SINT(n))

This function causes "subroutine" to be called when interrupt n occurs. The coding for subroutine should be in SYMBOL language because all of the interrupt housekeeping must be done in subroutine.

subroutine = name of a SYMBOL coded subroutine
n = integer constant, variable, or expression between 0 and 31 but not greater than the number established by ASSIGN.

Example: CALL SUB3 (SINT (N+1))

C. RELEASE

Form: CALL RELEASE (n)

This subprogram releases any subroutine that is set up to be called by interrupt n.

n = integer constant, variable, or expression between 0 and 31, but not greater than the number established by ASSIGN.

Example: CALL RELEASE (14)

D. ASSIGN

Form: CALL ASSIGN (n)

This subprogram assigns n as the number of systems interrupts that will be recognized.

n = integer constant, variable, or expression between 0 and 32. Until changed, the number of allowable interrupts is assumed to be 16.

Example: CALL ASSIGN (12)

E. ARM

Form: CALL ARM (n)

This subprogram arms the interrupt specified by n.

n = integer constant, variable, or expression between 0 and 31, but not greater than the number established by ASSIGN.

Example: CALL ARM (5)

F. DISARM

Form: CALL DISARM (n)

This subprogram disarms the interrupt specified by n.

n = integer constant, variable, or expression between 0 and 31, but not greater than the number established by ASSIGN.

Example: CALL DISARM (14)

G. CONDITION

Form: CALL CONDITION

This subprogram disarms and clears all system interrupts. Input/output interrupts are not disturbed.

Example: CALL CONDITION

H. CONNECT

Form: CONNECT RTN(A, B, C, . . .), n

This subprogram attaches the subroutine "RTN" to interrupt line n along with arguments A, B, C.

n = integer constant, variable, or expression between 0 and 31, but not greater than the number established by ASSIGN.

Example: CONNECT CHECK (L), 10

SECTION 2. MISCELLANEOUS FORTRAN CHANGES AND ADDITIONS

A. PAUSE

Form: PAUSE n

This statement causes the typewriter to type PAUSE n; then the program waits for the operator to switch Breakpoint 4.

n = unsigned integer or blank.

B. EXIT

Form: CALL EXIT

This subprogram is used to terminate the running of a program. It causes the typewriter to type *EXIT* and the computer to halt. When the halt is cleared, the monitor is reinitialized for another run.

C. STOP

Form: STOP

This statement causes the typewriter to type *STOP*. The computer then halts. When the halt is cleared, the computer is reinitialized for another run.

SECTION 3. MODIFICATIONS TO THE COMPILER

A. The compiler was modified so that it would generate real-time programs. In addition, some recursive features have been included:

1. BRS POP

   Wherever there is a subprogram link, the compiler now generates a BRS or BRS* instruction instead of the BRM or BRM* instruction.

2. Internal Functions

   Internal functions start immediately with the first word of code instead of beginning with an HLT cell. The BRR of the internal function is now replaced with a RETURN POP.

3. SUBROUTINE and FUNCTION Subprograms

   Instead of an HLT cell, the subprograms now generate a SAVTMP POP as the first instruction. Changes have been made so that a SUBROUTINE may call itself and a FUNCTION may use itself. All BRR instructions previously generated are now replaced with BRU RELTMP instructions.

4. Subprogram Linkage

   All of the system subprogram link references have been increased by 32 cells to allow room for interrupt cells.

5. PAUSE

   The pause statement now generates code that causes a type out of the PAUSE message and then goes into a Breakpoint wait.

SECTION 4. FORTRAN LANGUAGE EXTENSIONS

A. Boolean Statements

1. A Boolean statement is indicated by a B in column 1 of the first line of the statement.
2. There are two types of Boolean statements, the assignment statement and the function definition statement.
3. All operators are considered Boolean operators except for the operators in subscript expressions and function argument expressions.
4. All variables, constants, and functions must be integer mode except for those in subscript expressions and function argument expressions.

B. Boolean Operators

1. The following set of symbols constitutes the Boolean operators:

   - \- complement
   - * and
   - / exclusive or
   - + or

2. In the absence of clarifying parentheses the - operations are performed first, then the * and / operations are performed as encountered, and finally the + operations are performed.

3. The - operator is a unary operator that operates only on the term following it. Therefore, it may directly follow another operator.

C. Boolean Constants

1. All Boolean constants must be in octal integer format.
2. They will be right justified if they are less than eight digits.

D. Symbolic Code Statements

1. A symbolic code statement is indicated by an S in column 1 of the first line of the statement.
2. Statement numbers may be used with these statements.
3. The operation code is the first field encountered in the statement.
4. The variable field follows the operation code and is separated from it by at least one space.
5. The tag field follows the variable field and can only be , 2.
6. The comments field is the last field within the statement and must begin with a virgule (/).

E. Symbolic Operation Code

1. The operation code may be indicated by either a three letter mnemonic or by a three digit octal number between 100 and 177.
2. All EOM and SKS type instructions must be entered as such and not with their specific mnemonic codes.
3. No pseudo-operations or data defining operations are allowed.

F. Symbolic Variable Field

1. The variable field may contain spaces.
2. The variable field can consist of the following:

   a. FORTRAN scalar variables
   b. Unsubscripted FORTRAN array variables
   c. FORTRAN array variables with numeric subscripts
   d. FORTRAN dummy variables
   e. Local FORTRAN function names
   f. Previously used subprogram names
   g. Statement number, indicated by a number followed by an S
   h. Decimal address, indicated by a decimal number with no preceding zero

i. Octal address, indicated by an octal number preceded by zero

j. Current address, indicated by a $

k. FORTRAN literal, indicated by an = followed by any legal FORTRAN constant

l. Nothing, in which case the variable is considered to be zero

3. The variable field may contain a simple expression of the form V ± N, where V is one of the above variables and N is a decimal integer.

4. Indirect addressing, is indicated by an asterisk at the beginning of the variable field.

G. Logical Operations

1. The following operations constitute the set of logical operators and have processing precedence, as shown, from top to bottom:

.NOT. This unary operator produces logical inversion of the term following it.

.AND. This binary operator produces "true" if the terms it connects are "true", otherwise "false".

.OR. This binary operator produces "true" if either connected term is "true", otherwise "false".

.EOR. This binary operator produces "true" if the connected terms represent opposed states and "false" if they represent identical states.

2. Only the sign bit of each 24-bit word will be considered by logical operations. All positive numbers will be "false" (first bit = zero) while all negative numbers will be "true" (first bit = one). Logical data may be any type. Data generated by the logical operations will be all zeros in bits other than the sign bit.

3. The following logical constants are acceptable in expressions:

.TRUE.
.FALSE.

Example: A = .TRUE.

4. Mixed expressions are allowed.

SECTION 5. COMPILER OPERATIONS (Breakpoint Settings)

Breakpoint switch settings:

BP1	RESET (UP)	Punch object program
	SET (DOWN)	Suppress punching
BP2	RESET (UP)	Type source statements
	SET (DOWN)	Suppress typing
BP3	RESET (UP)	Source from paper tape
	SET (DOWN)	Source from cards
BP4	RESET (UP)	Source listing to printer
	SET (DOWN)	Source listing to typewriter

3. LOADER

MODIFICATIONS TO THE LOADER

A. Origin of the Loader

The origin of the loader was moved up so that it could accommodate the new subprogram links. These links are 32 cells higher in core.

B. EOFIX and LISTFX

The loader now sets up two special cells used by the run-time system. EOFIX is the bottom of erasable store and the start of the argument list. LISTFX is the top of erasable storage and the start of the work list. These cells are used by the run-time system to initialize the list markers.

C. Main Program Load Address

The main program load address is now around cell 3200_8.

D. Input Buffer

The maximum buffer size has been increased to 200_{10} words.

E. Program Map

The map has been incorporated with the loader. The map is printed on the console typewriter prior to loading the Run-Time Monitor, and is selected with Breakpoint 2 set. Optionally, setting Breakpoint 1 and 2 prints the map on the line printer.

F. SYMBOL

The loader will now accept SYMBOL coded statements in-line with FORTRAN.

4. RUN-TIME MONITOR

SECTION 1. MEMORY LAYOUT DURING RUN TIME

Core storage is divided into four sections during run time. Starting in lower memory and proceeding to the end of storage, the four sections are organized as follows:

A. Run-Time Monitor

The monitor is composed of constants, flags, subroutine links, and special routines, all used by the operating program at run time. It is broken down as follows:

1. Temporary storage cells
2. List marker cells
3. Programmed operator links
4. Library subroutine links
5. Constant pool
6. Format scan temporary storage cells
7. System initialize routines
8. Programmed operator routines
9. List processing routines
10. Typewriter output routine

B. Run-Time Program

The run-time program consists only of those routines required to perform functions defined by the user. It is broken down as follows:

1. FORTRAN compiled main program
2. FORTRAN compiled subroutines, if required
3. FORTRAN compiled functions, if required
4. SYMBOL coded subroutines, if required
5. FORTRAN library routines

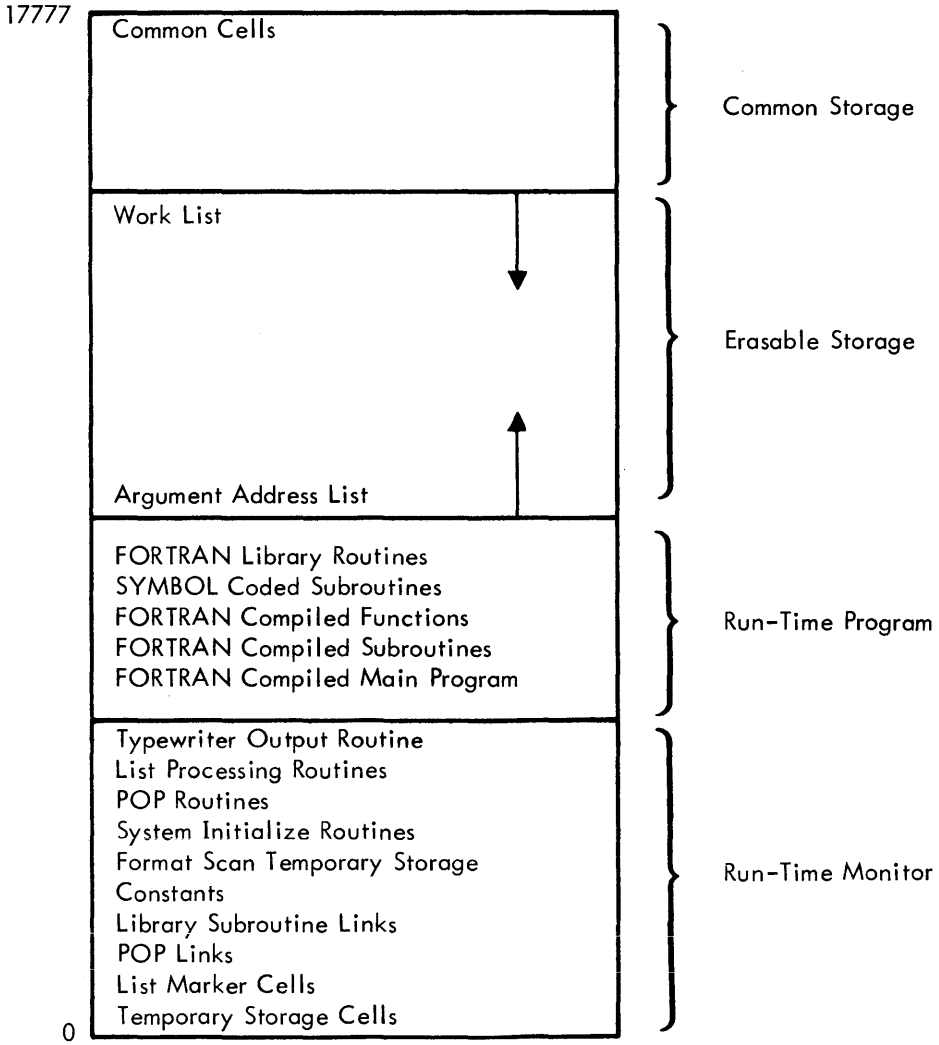
C. Erasable Storage

Erasable storage is a block of memory extending from the end of the run-time programs up to the bottom of common storage. It contains two lists defined as follows:

1. The argument address list, which occupies the bottom of erasable storage and builds toward the top
2. The work list, which occupies the top of erasable storage and builds toward the bottom

D. Common Storage

Common storage is the block of memory allocated by COMMON statements in the FORTRAN program. It is located at the very top of memory. If there were no COMMON statements, then there will be no common storage and erasable storage will extend to the top of memory.



SECTION 2. USE OF THE WORK LIST

The work list is the heart of the real-time monitor. It is used to save recursive variables, subroutine returns, I/O buffers, and items that must be saved during interrupts.

A. Location of the List

The work list is located at the top of erasable storage and builds downward toward the top of the argument list. It is not allowed to extend below the argument list since this would be a memory overflow condition.

B. List Marker Cell

There is a cell in memory that always points to the bottom of the work list, i.e., it contains the address of the last word of the list. This pointer cell is named LIST.

C. TMP Storage Cells

There is a block of 15 temporary storage cells in the run-time monitor labeled TMP1 through TMP15. Before these cells may be used for storage, their contents must be saved on the bottom of the work list. After these cells have been used, they must be restored with their original contents. TMP cells are saved with the ASN n instruction and restored with the RLS n instruction, where n is the number of cells. The use of TMP cells in conjunction with ASN and RLS makes possible recursive entry into a subroutine that requires temporary storage.

D. ZIP Storage Cells

There is a block of five temporary storage cells in the run-time monitor labeled ZIP1 through ZIP5. These cells may be used immediately without first saving them. They may be thought of as an auxiliary set of registers. They are saved in the list only when an interrupt occurs.

E. FORTRAN Storage Cells

When a FORTRAN compiled subroutine is entered recursively, its temporary storage cells and recursive variables are saved in the work list. They are restored when leaving the subroutine.

F. FORTRAN I/O Buffer Space

All the FORTRAN I/O routines use the work list for buffer space. They add to the list the number of buffer cells needed. After using the buffer, the list is reduced to its original size.

G. Interrupt Usage of the List

Interrupts set up by the FINT function cause the work list to be used for storing all registers and interrupt-vulnerable cells.

I. Coding used in Connection with the Work List

1. Code to Save an Item in the List

```
SKR  LIST      reduce marker address, no skip
LDA  LIST      pick up the list marker
SKG  EADR1     test to see if it exceeds the limit
BRU  overflow  overflow if it does
LDA  item      pick up the item
STA* LIST      save it in the list
```

2. Code to Restore an Item from the List

```
LDA* LIST        pick up the item
STA  item        restore it
MIN  LIST        increase marker address
```

3. Programmed Operator to Put an Item in the List

```
PUT  item
     or
PUT* item link   this performs the same operation as in 1
                 without changing the A register
     or
PUT  item, 2
```

4. Programmed Operator to get an Item from the List

```
GET  item
     or
GET* item link   this performs the same operation as in 2
                 without changing the A register
     or
GET  item, 2
```

5. Programmed Operator to Assign the First n Cells of TMP Storage from the List

```
ASN  n           cells TMP1 through TMPn are copied into the list
```

6. Programmed Operator to Release the First n Cells of TMP Storage from the List

```
RLS  n           cells TMP1 through TMPn are restored from the list
```

7. Programmed Operator to Branch to a Subroutine and Save the Return in the List

```
BRS  subroutine
     or
BRS* subroutine link   used in place of a BRM, branches to subroutine
                       instead of subroutine + 1
     or
BRS  subroutine, 2
```

8. Code to Return from a Subroutine

```
BRU  RETURN      used in place of BRR subroutine
```

SECTION 3. USE OF THE ARGUMENT LIST

A. The argument list provides the link between a subroutine and its caller. It is used by the calling routine to tell the subroutine where to find its arguments.

1. Location of the List

The list is located in erasable storage. It starts at the bottom of erasable and builds upward toward the work list. It is not allowed to extend above the work list, as this would be a memory overflow condition.

2. List Marker Cells

There are three main marker cells associated with the list. The cell that marks the bottom of the list is named E0ADR. The top of the list is marked by a cell named EADR1. The current cell being processed by the subroutine is marked by a cell named EADR2. In addition, there are two cells containing the same marker address as E0ADR. They are E0IND, which contains an indirect bit, and E0TAG, which contains an index bit.

3. Placing Arguments in the List

When a calling routine has arguments to transmit a subroutine, it places the address of the arguments in the list. The first argument's address is in the bottom cell of the list and the last argument's address is in the top cell. In addition, if the argument is floating point, bit 5 of the corresponding list cell will be on.

4. Obtaining Arguments from the List

Arguments for machine language programs may be obtained by using the E0TAG or E0IND cells. They allow the subroutine to indirectly address cells in the list and thus obtain arguments. The way that FORTRAN compiled subroutines obtain arguments is to use the start dummy (STRTDM), the compiler-generated dummy setup procedure, and end dummy (ENDDMY) routines. In addition to obtaining the arguments, these routines also check the mode and number of arguments. See Chapter 5, Section 2, "System Routines."

SECTION 4. PROGRAMMED OPERATORS (POPs)

The FORTRAN system incorporates a set of special purpose POPs designed particularly for FORTRAN programs. Some of these have HELP/SYMBOL functional counterparts; however, the POP entry locations may not coincide since the two programmed operator sets are not identical. (FORTRAN, for example, does not require a single precision, fixed point, square root POP.) Thus, the standard SYMBOL assembler should not be used to assemble FORTRAN subroutines unless they contain no POPs. OPDs should be used to define these POPs.

Real-Time FORTRAN II contains the run-time POPs listed in the SDS 920/930 FORTRAN II Operations Manual plus the following additions:

102 SKR - Reduces M by 1, skips if negative

103 SKE - Skips if A equals M

106 FEO - Floating Exclusive OR

124 MUL - Multiplies A by M

127 DIV - Divides A by M

130 SKB - Skips if M and B do not compare ones

133 ADM - Adds A and M

134 FOR - Floating OR

137 FAN - Floating AND

143 XMA - Exchanges memory and A

146 RCH - Register changes

152 SKD - Differences exponents and skips

160 ASN - Assign Temporary Storage. Puts TMP cells in work list.

161 RLS - Release Temporary Storage. Restores TMP cells from the work list.

162 BRS - Branch and Save Return. Used to enter a recursive subroutine.

163 RTN - Return. Used to leave a recursive subroutine.

164 SAV - Saves FORTRAN temporary storage on recursive entries to FORTRAN compiled subroutines.

165 PUT - Puts the operand in the work list.

166 GET - Gets the operand from the work list.

Note: Care should be taken in using POPs that access floating point quantities in memory. These POPs double the index register before accessing the floating point variable since floating point quantities require two words of storage. The original value of the index register is restored prior to exit.

POP numbers above 166 may be used by the programmer to write his own Run-Time POPs. They must be written in machine language, and they will be loaded and relocated by the LOADER. This does not apply to cell 177_8, which is the list marker address.

SECTION 5. INTERRUPT PROCESSING

A. Interrupt Cells

There are 18 or 34 interrupt cells used by the system. Each must reference some routine by way of a BRM instruction. The routines referenced, however, may be changed during the running of the program. The assignment of the interrupt cells is as follows:

Cell (in octal)	Assignment
31	W buffer I1 interrupt
33	W buffer I2 interrupt
200–217	Patchable system interrupts
220–237	Patchable system interrupts (optional with hardware configuration)

B. Priority of Interrupts

The routines assigned to the lowest numbered interrupt cells have the highest priority. Therefore, if a routine assigned to interrupt cell 200 is being executed and an interrupt signal for cell 33 occurs, the 33 interrupt will be serviced immediately. If, however, a 201 interrupt occurs, it must wait until the 200 interrupt routine is finished before it can be serviced.

C. Enabling of Interrupts

In order for any interrupt to be serviced, the enable instruction (EOM 20002) must be executed; otherwise, interrupt signals will be ignored. All interrupts are disabled with EOM 20004.

D. Assignment of W Buffer Interrupts

Each I/O routine assigns its own W buffer interrupt routines. When an I/O routine is entered, one of its first tasks is to place appropriate BRM instructions in cells 31 and 33. Since it is possible that cells 31 and 33 might be changed by another I/O routine when an interface interrupt occurs, these cells are saved before and restored after the interrupt is serviced.

E. Assignment of Patchable System Interrupts

During the system initialization phase of the run-time monitor, the patchable system interrupt cells (200–231) are plugged with BRM instructions which reference "do-nothing" routines. All that a do-nothing routine does is clear its interrupt and return. Then during the running of the program, assignments of subroutines may be made to these interrupt cells. These assignments may be made by using either the FINT, CONNECT, or the SINT functions, or simply by storing a BRM instruction in the interrupt cell.

F. Releasing Interrupt Assignments

The assignment of a subroutine to an interrupt cell may be released in two ways. One way is to assign another subroutine to the interrupt cell. This automatically releases any prior assignment. The other way is to use the RELEASE subroutine. This subroutine plugs a reference to a do-nothing routine into the interrupt cell.

SECTION 6. RUN-TIME ERROR CONDITIONS

A. When an error condition is detected during the running of a program, two things happen. First, the appropriate error message is output through the typewriter. Second, the program waits for Breakpoint 4 to be changed from its current setting. If the operator decides the program may continue, he should change the Breakpoint switch; otherwise, he should terminate program execution.

B. Error Message	Meaning
ERR AGTO	Assigned GO TO variable never assigned. Result: Unpredictable.
ERR ARGM	An argument of the wrong mode was given to a FORTRAN compiled subprogram. Result: Unpredictable.

Error Message	Meaning
ERR ARGN	The wrong number of arguments was given to a FORTRAN compiled subprogram. Result: If too many, extra ones are ignored. If too few, whatever arguments remain in the argument list will be used.
ERR ARM	An illegal number of interrupt lines was given to ARM or DISARM. Result: No action is taken.
ERR ASGN	An illegal number of interrupt lines was given to ASSIGN. Result: No action is taken.
ERR CGTO	The computed GO TO value is outside the allowable range. Result: Go the the first statement number in the list.
ERR EFIA	An E, F, I, or A is missing from the FORMAT. Result: Processing proceeds without output of the variables.
ERR EXP	The argument given to EXP was greater than 176. Result: The answer is set to the maximum floating point value.
ERR FCHR	An illegal character was found in the FORMAT. Result: The scan for the next specification begins, i.e., the character is treated as if it were a comma.
ERR ICHR	There is an illegal input character. Result: The scan begins for the next field, i.e., treats the character as if it were a comma.
ERR IFSL	The IF SENSE LIGHT value was not 1-24. Result: The test is assumed off.
ERR IFSS	The IF SENSE SWITCH value was not 1-4. Result: The test is assumed off.
ERR INT	The argument give to the FINT function was invalid. Result: No interrupt assignment is made.
ERR INUM	The characteristic of an input number exceeds ± 99. Result: The number is set to zero.
ERR LOG	The argument given to ALOG was either negative or zero. Result: The answer is set to zero.
ERR N**F	A negative number was raised to a nonintegral floating point power. Result: The absolute value of the number is used.
ERR OEXP	An output exponent is greater than 99. Result: The exponent is cleared to zero and the resulting number is output.
ERR RELSE	An illegal interrupt line number was given to RELEASE. Result: No action taken.
ERR PRTY	An input parity error has occurred. Result: The input record is read again. This means that card or tape must be repositioned or the typewriter message must be keyed in again.
ERR SNLT	The SENSE LIGHT value was not 0-24. Result: No action is taken.
ERR SNT1	The SINT function was used improperly. Result: An attempt is made to make the proper interrupt assignment.
ERR SNT2	The argument given to the SINT function was invalid. Result: No interrupt assignment is made.
ERR SQRT	The argument given to SQRT was negative. Result: The square root of the absolute value is taken.
ERR RLSE	The argument given to RELEASE was invalid. Result: No action is taken.
ERR 0**N	Zero was raised to a nonpositive power. Result: (0**0) will be 1 or 1.0, and (0**negative) will be the maximum possible integer or floating number.

SECTION 7. RECURSIVE FEATURES OF REAL-TIME FORTRAN

A. A byproduct of Real-Time FORTRAN II is the ability of subprograms to be recursive. This ability is needed for real-time programming because subroutine "X" could be interrupted by a routine which also calls subroutine X. This implies that there must be a way to keep track of the recursive calls so that their respective returns will be made to the proper place. Furthermore, temporary variables used must be saved and restored for each recursive entry into a subroutine; otherwise, the latest recursive entry into a subroutine would destroy the temporary variables being used by an earlier entry.

1. Recursive Variables

The Recursive variables are those that are temporary in nature. These would be the variables used for indexing through arrays and counting through DO loops, those saved as intermediate results, and so on. In this system recursive variables have arbitrarily been chosen as those scalar variables that do not appear in COMMON or EQUIVALENCE statements. If a dummy variable (an argument used by a subroutine or function) is to be used recursively, a recursive variable should be set equal to it and used in its place.

2. Examples of Recursive Programming

Two simple subprograms are shown below to illustrate recursive programming. The first is a subroutine that calls itself and the second is a function that uses itself.

a. Compute the characteristic of a number

```
      SUBROUTINE COMPUTE (NDUMMY)
      COMMON CHAR
      NUMBER = NDUMMY
      IF (NUMBER - 10) 1, 2, 2
1     CHAR = 0
      RETURN
2     CALL COMPUTE (NUMBER/10)
      CHAR = CHAR + 1
      RETURN
      END
```

b. Compute the factorial of a number

```
      FUNCTION IFACTORIAL (NDUMMY)
      N = DUMMY
      IF (N) 1, 1, 2
1     IFACTORIAL = 1
      RETURN
2     IFACTORIAL = N*IFACTORIAL (N-1)
      RETURN
      END
```

SECTION 8. MODIFICATIONS TO EXISTING RUN-TIME ROUTINES

A. The FORMAT scan routines and the POP routines have been modified to operate in a real-time environment.

1. Subroutine Linkages

All subroutines have been modified so that they no longer begin with HLT cells. The BRM instructions have been changed to BRS POPs and the BRR instructions changed to BRU RETURN instructions.

2. Floating Point POPs

The temporary storage cells used by the floating point POP subroutines are all ZIP storage. This, then, does not slow down the subroutines as would TMP storage because of the assigning and releasing required.

3. FORMAT Scan Routines

The temporary storage required by the FORMAT scan routines is much more than the 15 TMP cells can handle. Therefore, when the FORMAT scan is entered recursively, all of its temporary storage is saved in the work list.

SECTION 9. LIST PROCESSING ROUTINES

A. PUTPOP (programmed operator routine)

1. Function

The purpose of PUTPOP is to take the POP operand and put it on the bottom of the work list.

2. Input

The POP operand is the only input.

3. Method

The method used is first to reduce the list marker address and test to see if the list exceeds its limit. If it does, the overflow routine is executed; otherwise, the POP operand is placed on the bottom of the list.

4. Subroutines

Name	Function
OVFLO	prints the overflow message and halts

5. POP Definition

PUT OPD 16500000

6. Timing

25 cycles or .200 ms.

B. GETPOP (programmed operator routine)

1. Function

The purpose of GETPOP is to get the contents of the cell at the bottom of the work list and store it in the POP operand cell.

2. Input

The POP operand is the only input.

3. Method

The method used is first to pick up the bottom list cell, store it in the POP operand cell, and then increase the list marker address.

4. POP Definition

GET OPD 16600000

5. Timing

20 cycles or .160 ms.

C. ASNPOP (programmed operator routine)

1. Function

The purpose of ASNPOP is to take the number of TMP cells indicated by the POP address and put them on the bottom of the work list.

2. Input

The input is a number in the POP address.

3. Method

Space is created in the work list to accommodate the required number of TMP cells. If the list exceeds its limit, the overflow routine is exeucted; otherwise, the TMP cells are copied into the space just provided at the bottom of the list.

4. Subroutines

Name	Function
OVFLO	prints the overflow message and halts.

5. POP Definition

ASN OPD 16000000

6. Timing

35+7n cycles or .280+ .056n ms., where n is the number of TMP cells to be saved.

D. RLSPOP (programmed operator routine)

1. Function

The purpose of RLSPOP is to take the number of cells indicated by the POP address from the bottom of the work list and put them in TMP storage.

2. Input

The input is a number in the POP address.

3. Method

The required number of cells are copied from the bottom of the list into TMP storage, and the list marker address is incremented to reflect this reduction of the list.

4. POP Definition

RLS OPD 16100000

5. Timing

37+7n cycles or .296+ .056 ms., where n is the number of TMP cells to be restored.

E. BRSPOP (Programmed operator routine)

1. Function

The purpose of BRSPOP is to branch to the POP operand and save the location of the POP in the work list.

2. Input

The input is the POP operand.

3. Method

A branch instruction is made using the effective address of the POP. The location of the POP is placed on the bottom of the work list. Then the branch instruction is executed. If the list exceeds its limit, the overflow routine is executed instead.

4. Subroutines

Name	Function
OVFLO	prints the overflow message and halts

5. POP Definition

BRS OPD 16200000

6. Timing

32 cycles or .256 ms.

F. RETURN (programmed operator routine)

1. Function

The RETURN POP is used in conjunction with the BRS POP. It gets the return address that the BRS put on the bottom of the list. It then returns to that address and restores the overflow indicator to what it was when the BRS was executed. Because the address of the RETURN POP is not used, a simple BRU RETURN may be substituted for the POP.

2. Method

The method used is to get the return address from the bottom of the work list, reset the overflow indicator, and then perform a BRR with the return address as the operand.

3. POP Definition

RETURN OPD 16300000
or OPD 00100163

4. Timing

18 cycles or .144 ms.

G. SAVTMP (programmed operator routine)

1. Function

The purpose of SAVTMP is to save temporary storage and recursive variables when a routine is entered recursively. It is used exclusively by FORTRAN compiled subroutines.

2. Method

Because the SAVTMP POP is inserted as the first instruction of a FORTRAN compiled subroutine, it serves as a location marker for finding the temporary storage and recursive variables. In addition, it also serves as an entry flag for the subroutine. When the subroutine is entered, the POP and its location are saved. If the sign bit of the POP is negative, meaning that this is a recursive entry, the temporary storage and recursive variables are saved in the list. If the sign bit is positive, meaning that this is not a recursive entry, it is changed to negative.

3. Subroutines

Name	Function
ANSPOP	Store TMP cells in the work list

4. Timing

See RELTMP (below).

H. RELTMP

1. Function

The RELTMP routine is used in conjunction with SAVTMP POP. Its purpose is to restore the temporary storage and recursive variables saved by SAVTMP. It is used also to initiate the return from a FORTRAN compiled subroutine.

2. Method

The entry flag of the subroutine is restored and tested to see if this was a recursive entry. If it was not, a return is made to the calling program; if it was, the temporary storage and recursive variables are restored before returning to the calling program.

3. Subroutines

Name	Function
RLSPOP	releases TMP cells from the work list

4. Timing

63 cycles or .504 ms., if it was not a recursive entry; $81+17n$ cycles or $.648+.136n$ ms., where n is the number of cells to be restored, if it was a recursive entry.

SECTION 10. MONITOR SUPPORT ROUTINES

A. SYSINI

1. Function

The purpose of the system initialize routine, SYSINI, is to prepare the monitor for the start of the main program.

2. Method

The method used is to reset all the monitor switches, plug all the interrupt cells with do-nothing routines, clear any active interrupts, reset the work list and argument list marker cells, reset the subroutine recursive entry switches, disable the system interrupts, type the starting message, halt, and branch to the main program.

B. ERROR

1. Function

The purpose of ERROR is to indicate an execution error by typing the comment ERR plus a code word. After typing the error, this routine branches to the Breakpoint Wait routine.

2. Input

The input is a four-character code word located just below the BRS ERROR instruction.

3. Method

The error message is placed at the bottom of the work list, so as not to be vulnerable to recursive entries into ERROR. After typing out the message and releasing it from the list, the BPWAIT routine is entered.

4. Subroutines

Name	Function
ASNPOP	assigns TMP cells to the work list
RLSPOP	releases TMP cells from the work list
PUTPOP	puts a cell on the bottom of the list
TYPO	outputs a message through the typewriter
BPWAIT	waits for Breakpoint 4 to be switched

C. BPWAIT

1. Function

The purpose of the Breakpoint Wait routine, BPWAIT, is to provide a wait in the program that does not halt the computer.

2. Method

Breakpoint 4 is tested until the operator switches it; then a return is made back to the calling program.

3. Subroutines

Name	Function
OVFLO	prints the overflow message and halts
ASNPOP	assigns TMP cells to the work list

4. POP Definition

SAVTMP OPD 16400000

5. Timing

70 cycles or .560 ms., if it is not a recursive entry; 103+ 17 n cycles or .824+ .136 n ms., where n is the number of cells to be saved, if it is a recursive entry.

D. OVFLO

1. Function

The purpose of the overflow routine, OVFLO, is to indicate that there is no more space left in erasable storage. The message OVERFLOW AT xxxxx is typed, where xxxxx is the point in the program where overflow occurred.

2. Input

The overflow point is put in location zero.

3. Method

The method used is to disable interrupts and disconnect any I/O. The overflow point is converted to an octal number and then typed out in the overflow message. After the message is typed, a halt occurs. When the halt is cleared, the monitor is reinitialized.

E. SAVE

1. Function

The purpose of the SAVE routine is to save all the registers and interrupt-vulnerable cells when an interrupt occurs. SAVE then executes the interrupt subroutine, after which it restores the registers and vulnerable cells.

2. Input

Because this routine is not entered by a BRS, the return address is found in the index register. The index register is also used to find the location of the interrupt subroutine.

3. Method

The A, B, and X registers, the five ZIP cells, the floating overflow indicator, the two buffer interrupts cells, and the marker for the argument list bottom are all saved on the bottom of the work list. The contents of the argument list are saved by setting the marker for the bottom of the argument list equal to the marker for the top of the argument list. After the interrupt routine has been executed, the above process is reversed.

4. Timing

92 cycles or .736 ms are used to save everything
89 cycles or .712 ms are used to restore everything

SECTION 11. TYPO-TYPEWRITER OUTPUT ROUTINE

A. Function

The purpose of TYPO is to output data to the typewriter.

B. Input

A register = word count of record
X register = location of output record

C. Returns

Entry address + 1, only

D. Method

The routine operates under I1 and I2 interrupts, and data is output in 4-character-per-word mode.

E. Timing

66.7 ms per character (15 char/sec)

5. LIBRARY

SECTION 1. MODIFICATIONS TO EXISTING LIBRARY ROUTINES

A. System Subprogram Links

In order to accommodate the interrupt cells, the subprogram links were moved up 32 cells, and all references to these subprogram links were moved up accordingly.

B. Subroutine Entries

All subroutine entry points were modified so that they no longer begin with HLT cells; instead, they begin with the first program instructions.

C. Subroutine Exits

Instead of the BRR instruction, the subroutines now return with a BRU RETURN instruction.

D. Subroutine Calls

When calling another subroutine, the BRS POP is now used instead of the BRM instruction.

E. Temporary Storage

All references to temporary storage have been changed to reference either TMP or ZIP cells. If TMP cells are used, the appropriate ASN and RLS POPs are used to assign and release them.

SECTION 2. SYSTEM ROUTINES

The system routines listed below cannot be called by name. They are given octal numbers ranging from 241 to 304 and are used in much the same manner as programmed operators; that is, the linkage to them is stored in locations 241 to 304, and they are entered by a BRS POP. Only those routines called for implicitly in the program will actually be loaded.

The description of each routine in the following list includes:

1. Octal number
2. Name
3. Operation performed
4. Memory Storage used
5. Other system routines required, if any

241 STRTDM - Start of dummies. Used by FORTRAN subprograms in obtaining arguments from the calling program.
Memory: 4 words

242 ENDDMY - End of dummies. Used in conjunction with above in obtaining arguments.
Memory: 11 words

243 STOP - Stop. Types *STOP* and halts.
Memory: 10 words

244 IFSNSW - If Sense Switch. Performs the IF SENSE SWITCH test.
Memory: 19 words

245 IFSNLT - If Sense Light. Performs the IF SENSE LIGHT test.
Memory: 21 words

246 COMPGO - Computed Go To. Performs the computed GO TO.
Memory: 14 words

247 ACCEPT - Accept. Reads information from the console typewriter.
Memory: 10 words
Requires: 275 (INITFS)

250 ACCTAP - Accept Tape. Reads from paper tape.
Memory: 10 words
Requires: 275 (INITFS)

251 PRINT - Print. Same as TYPE.

253 PNCHTP - Punch Tape. Punches paper tape.
Memory: 10 words
Requires: 275 (INITFS)

254 TYPE - Type. Types on the console typewriter.
Memory: 9 words
Requires: 275 (INITFS)

255 SQRT - Square Root. Takes the square root of an argument. This system routine may also be called by name.
Memory: 79 words

256 READ - Read. Reads BCD cards.
Memory: 10 words
Requires: 275 (INITFS)

263 ENDIOL - End input/output list. Used by all input/output lists.
Memory: 32 words

264 IFOVL - If overflow. Tests status of floating-point overflow indicator and branches accordingly.
Memory: 6 words

267 SENSLT - Sense Light. Sets sense light.
Memory: 19 words

270 POWER - Power. Raises an argument to a floating-point or integer power.
Memory: 100 words
Requires: ELOGF and EXPF

271 FIX - Fix. Converts floating-point number to integer.
Memory: 3 words

272 FLOAT - Float. Converts integer to floating-point number.
Memory: 3 words

273 IOLUSA - Input/output list subscripted array. Used during input and output of arrays when listed without subscripts (e.g., TYPE 3, A).
Memory: 30 words

274 PAUSE - Pause. Types PAUSE and an integer.
Memory: 39 words

275 INITFS - Initialize format scan. Used in conjunction with the FORMAT scan routines in the run-time system.
Memory: 55 words
Requires: 276 (BINBCD)

301 READTP - Read Tape. Read from magnetic tape in binary mode.
Memory: 132 words

302 WRITAP - Write Tape. Write on magnetic tape in binary mode.
Memory: 115 words

303 ENFILE - Write End of File. Write an end-of-file mark on magnetic tape.
Memory: 61 words

304 REWIND - Rewind Tape. Rewind magnetic tape to beginning-of-tape mark.
Memory: 6 words

SECTION 3. INTERRUPT ROUTINES

A. FINT (or CONNECT) – FORTRAN Interrupt

1. Function

The purpose of the FORTRAN interrupt routine, FINT, is to assign a FORTRAN compiled subroutine or a SYMBOL coded subroutine to an interrupt line. (FINT may also be called with a CONNECT statement.) The linkage is set up so that when an interrupt occurs the arguments at that moment are set up and the subroutine is entered.

2. Input

The interrupt line number is the first argument of the argument list. The subroutine location is the effective address of the first BRS instruction following the BRS FINT instruction.

3. Method

First, a search is made for the BRS that contains the subroutine location. Next, a linkage is set up so that when the desired interrupt occurs the BRM in the interrupt cell will branch to a small routine assigned to that cell. This routine will enter the SAVE subroutine and supply it with the beginning location of the desired argument setup instructions. After the interrupt subroutine has been executed and the SAVE subroutine has restored everything, a return is made to the small routine. It in turn clears the interrupt.

4. Error Messages

Code	Meaning
INT	The interrupt line number is not valid or was given as a floating-point number

5. Subroutines

Name	Function
ERROR	Outputs the error message
SAVE	Saves the interrupt-vulnerable cells and enters the interrupt subroutine

B. SINT - SYMBOL Interrupt

1. Function

The purpose of the SYMBOL interrupt routine, SINT, is to assign a SYMBOL coded subroutine to an interrupt line. This subroutine must do its own interrupt housekeeping and must clear the interrupt when it returns.

2. Input

The interrupt line number is the first argument of the argument list. The subroutine location is the effective address of the first BRS instruction following the BRS SINT instruction.

3. Method

A search is made for the BRS that contains the subroutine location. Next a linkage is set up so that when an interrupt occurs on the desired interrupt line there will be a direct BRM to the interrupt routine.

4. Error Messages

Code	Meaning
SNT1	SINT was used improperly
SNT2	The interrupt line was an invalid argument

5. Subroutines

Name	Function
ERROR	Outputs the error message

C. RELEASE - Release Interrupt

1. Function

The purpose of release is to release the subroutine assigned to a particular interrupt line.

2. Input

The interrupt line number is in the first argument of the argument list.

3. Method

The desired interrupt cell is filled with a BRM to its associated do-nothing routine.

4. Error Messages

Code	Meaning
RELSE	The interrupt line number argument was invalid

5. Subroutines

Name	Function
ERROR	Outputs the error message

D. ARM (or DISARM) - Arm or Disarm Systems Interrupts

1. Function

The purpose of these routines is to initiate the ARM interrupt and DISARM interrupt hardware option.

2. Method

The interrupts are armed or disarmed by using an EOM and a POT word that is built from the argument. These routines do not disable the interrupts. Interrupt cell 217_8 is always armed but not disarmed. All "hanging" systems interrupts are cleared.

E. CONDITION - Disarm Systems Interrupts

1. Function

This routine will disarm all systems interrupts without argument transfer.

2. Method

The systems interrupts are disarmed by using the DISARM routine with the run-time constant ASIGN to define the number of interrupts.

F. ASSIGN - Assign Number of Interrupts

1. Function

This routine will change the number of defined systems interrupts by the argument.

2. Method

ASIGN in the run-time monitor is substituted with the valid argument.

SECTION 4. INPUT/OUTPUT ROUTINES

A. TYPI - Typewriter Input

1. Function

The purpose of TYPI is to input data through the typewriter, and to detect buffer errors.

2. Input

A register = location of input buffer area.

3. Output

X register = character count of record.
LOCATION contains input data.

4. Returns

Entry address + 1, buffer error during input.
Entry address + 2, normal.

5. Method

The routine operates under I1 interrupt control, and data enters the computer in the single character mode. Leading carriage returns (C/R) are ignored, and +/- followed by a C/R voids an entry. Data followed by a C/R terminates input. Although data enters in single character mode, it is packed four characters per word in the buffer area.

6. Timing

Type-in speed of user.

B. CARD - Card Input

1. Function

The purpose of CARD is to read up to 80 columns of data from a card, and to detect input errors.

2. Input

A register = location of input buffer area.

3. Output

X register = character count of record (0 if blank card).
LOCATION contains input data.

4. Returns

Entry address + 1, buffer error during input.
Entry address + 2, normal.

5. Method

The routine operates under I1 and I2 interrupt control, and data enters the computer in the single character mode. Card blanks (60) are converted to code (12) blanks during input, and the character count is determined by the column containing the last non-blank character. Data is packed four characters per word in the buffer area.

6. Timing

240 ms per card (250 cards/minute).

C PPTO - Paper Tape Output

1. Function

The purpose of PPTO is to punch data on paper tape, with or without leader, in 4-character-per-word mode.

2. Input

A register = word count of record.
B register = 0 denotes leader.
≠ 0 denotes no leader.
X register = location of output record.

3. Returns

Entry + 1, only.

4. Method

The routine operates under I1 and I2 interrupt control, and outputs data in 4-character-per-word mode, with or without leader.

5. Timing

16.7 ms per character (60 characters/second).

D. FPTI - Paper Tape Input

1. Function

The purpose of FPTI is to input data from paper tape in the single character mode, and to detect buffer errors.

2. Input

A register = location of input buffer area.

3. Output

X register = character count of record.
LOCATION contains input data.

4. Returns

Entry address + 1, buffer error during input.
Entry address + 2, normal.

5. Method

The routine operates under I1 and I2 interrupt control. Data enters the computer in single character mode. Delete codes (77) are ignored, and input is terminated by a C/R (52), or gap. Data is packed four characters per word in the buffer area.

6. Timing

3.33 ms per character (300 characters/second).

F. READTP - Read Magnetic Tape Binary

1. Function

Reads one logical record from magnetic tape in the binary mode.

2. Input

Tape unit in the A register.

3. Subroutines

SETUP	temporary storage setup.
RTAPE	read magnetic tape.
CALCKS	calculate checksum.

4. Error Messages

LRR (unit) short logical record; record truncated.
CKS (unit) checksum error; record processed.

5. Memory Allocation

94 cells.

G. WRITAP - Write Magnetic Tape Binary

1. Function

Writes one logical record on magnetic tape in the binary mode.

2. Input

Tape unit in the A register.

3. Subroutines

SETUP	temporary storage setup.
CALCKS	calculates checksum.
WTAPE	write magnetic tape.

4. Memory Allocation

81 cells.

H. SETUP - Initialize Temporary Storage

1. Function

Initializes temporary storage for binary input/output.

2. Memory Allocation

26 cells.

I. CALCKS - Calculate Checksum

1. Function

Calculates checksum for physical binary record.

2. Result

Folded checksum is in the A register.

3. Memory Allocation

20 cells.

J. READIT - Read Magnetic Tape BCD

1. Function

Reads magnetic tape in the BCD (even parity) mode.

2. Input

Tape unit in the A register.

3. Subroutines

INITFS	initializes FORTRAN format scan.
RTAPE	reads magnetic tape.

4. Memory Allocation

14 cells

K. WRITOT - Write Magnetic Tape BCD

1. Function

Writes magnetic tape in the BCD (even parity) mode.

2. Input

Tape unit in the A register.

3. Subroutines

INITFS	initializes FORTRAN format scan.
WTAPE	writes magnetic tape.

4. Memory Allocation

11 cells.

L. RTAPE - Read Magnetic Tape Binary or BCD

1. Function

Reads magnetic tape in either binary or BCD mode. (Odd or even parity, respectively.)

2. Input

Tape unit in A register.
Word count in X register.
Location in B register SIGN bit on - binary.
SIGN bit off - BCD.

3. Error Messages

EOF (unit) end of file detected, another record read.
ETR (unit) end of tape detected; continues.
RDT (unit) read error uncleared after 10 tries; bad record processed.

4. Subroutines

SETIOT sets up I/O list, etc.
ERRMSG controls tape positioning following error conditions; outputs error messages.
RESET resets temporary storage.

5. Memory Allocation

67 cells.

M. WTAPE - Write Magnetic Tape Binary or BCD

1. Function

Writes a record on magnetic tape in either binary or BCD mode. (Odd or even parity, respectively.)

2. Input

Tape unit in the A register.
Word count in the X register.
Location in the B register: SIGN bit on - binary.
SIGN bit off - BCD.

3. Error Messages

FPT (unit) file protected; hangs until cleared.
ETW (unit) end of tape detected; continues.
WRT (unit) write error; tape erased after five tries and record is rewritten.

4. Subroutines

SETIOT sets up I/O list, etc.
RESET resets temporary storage.
ERRMSG controls tape positioning following error conditions; outputs error messages.
ERASE erases tape.

5. Memory Allocation

80 cells.

N. SCAN - Move Magnetic Tape

1. Function

Moves magnetic tape one physical record in either direction.

2. Input

Tape unit in the A register.
Direction in the B register: negative, forward; positive, backward.

3. Subroutines

SETIOT	sets up I/O list.
RESET	resets temporary storage.

4. Memory Allocation

37 cells.

O. BKSPAC - Backspace Magnetic Tape

1. Function

Backspaces magnetic tape one logical record.

2. Input

Tape unit in the A register.

3. Subroutines

SETIOT	sets up I/O list.
ERRMSG	controls tape errors; outputs error messages.
RESET	resets temporary storage.
SCAN	moves magnetic tape one record.

4. Memory Allocation

93 cells.

P. ERASE - Erase Magnetic Tape

1. Function

Erases magnetic tape 4.5 inches.

2. Input

Tape unit in the A register.

3. Subroutines

SETIOT	sets up I/O list.
RESET	resets temporary storage.

4. Memory Allocation

38 cells.

Q. ENFILE - Write End of File on Magnetic Tape

1. Function

Writes an end of file mark on magnetic tape.

2. Input

Tape unit in the A register.

5. Library

3. Error Messages

FPT (unit) file protected; computer hangs up until cleared.
WEF (unit) backspaces and rewrites EOF until error is cleared.

4. Subroutines

SETIOT sets up I/O list, etc.
RESET resets temporary storage.
ERRMSG controls error conditions; outputs error messages.

5. Memory Allocation

53 cells.

R. REWIND - Rewind Magnetic Tape

1. Function

Rewinds magnetic tape.

2. Input

Tape unit in the A register.

3. Subroutines

SETIOT sets up I/O list, etc.
RESET resets temporary storage.

4. Memory Allocation

9 cells.

S. SETIOT - Set Up I/O Instructions

1. Function

Sets up I/O instructions for all magnetic tape operations in temporary storage. Checks validity of the tape unit number and the density of the magnetic tape unit.

2. Error Messages

TPNO (tape unit in A) unacceptable tape number; uses 3 low-order bits as tape number.
BPI (tape unit) tape density error; awaits change of density.

3. Memory Allocation

63 cells.

T. RESET - Reset I/O Instructions

1. Function

Resets temporary storage saved by SETIOT.

2. Memory Allocation

10 cells.

U. ERRMSG - Magnetic Tape Error Message

1. Function

Controls error conditions found in magnetic tape routines; outputs error messages.

2. Input

Error code in the index register.

3. Subroutines

SCAN moves tape one record.

4. Memory Allocation

47 cells.

V. PRINT - Print on Line Printer

1. Function

Prints a line on the high speed printer.

2. Subroutines

INITFS initializes FORTRAN format scan.
PRTR outputs information to the printer.

3. Memory Allocation

9 cells.

W. PRTR - Output to Line Printer

1. Function

Outputs information to the high speed printer.

2. Method

Limits output to 33 words. Changes trailing carriage return to a blank.

3. Memory Allocation

87 cells.

SECTION 5. MISCELLANEOUS ROUTINE

PAUSE - Pause in Execution

1. Function

The purpose of PAUSE is to cause the message PAUSE x to be typed out and to wait for Breakpoint 4 to be switched before continuing. This routine is called by the PAUSE x statement in FORTRAN where x is some integer or a blank.

2. Input

The input is a positive integer in binary form located in the word following the BRS PAUSE instruction.

3. Output

The message PAUSE x is output where x is some integer or is blank.

4. Method

The integer is converted for output with leading zeros eliminated. The entire pause message is then placed on the bottom of the work list and is output through the typewriter. Finally the message is released from the list and the Breakpoint Wait routine is entered..

5. Subroutines

Name	Function
TYPO	outputs a message through the typewriter.
BPWAIT	waits for Breakpoint 4 to be switched.