DATA GENERAL
CORPORATION
Southboro,
Massachusetts 01772
(617) 485-9100

PROGRAM

FORTRAN IV RUN TIME LIBRARY
USER'S MANUAL

ABSTRACT

Data General's FORTRAN IV Run Time Library is an implementation of the ANSI FORTRAN standard X3.9-1966, with many extensions. The primary extension is that all run time routines are reentrant. Routines in the library permit integer, single and double precision real, and single and double precision complex arithmetic and transcendental functions. String and byte handling, array manipulation, and I/O conversion routines are also provided.

Also provided in the Run Time Library is a set of real time routines, which permit the writing of multitask real time (RT) FORTRAN IV programs.

Techniques are given for interfacing the reentrant, relocatable library code with assembly language programs.

093-000068-03

Original Release	-	December 1971
First Revision	-	March 1972
Second Revision	-	July 1972
Third Revision	-	February 1973

This revision of the FORTRAN IV Run Time Library User's Manual, 093-000068-03, supersedes 093-000068-02 and constitutes a major revision. The primary change is the addition of many real time routines. However, other changes of importance have been made and are contained in the list of changes following the Index.

TABLE OF CONTENTS

SINGLE PRECISION FLOATING POINT ROUTINES (Continued)

MIXED MODE ROUTINES (Continued)

APPENDIX E - REAL TIME FORTRAN (Continued)

FORTRAN RUN TIME LIBRARY

USER'S MANUAL

INTRODUCTION

The purpose of this manual is to provide FORTRAN and assembly language programmers with information about the Data General FORTRAN IV Run Time Libraries. Users should familiarize themselves with DGC publications 093-000053 "FORTRAN IV USER'S MANUAL," 093-000017 "ASSEMBLER," 093-000040 "EXTENDED ASSEMBLER," and 093-000039 "RELOCATABLE LOADER."

RUN TIME LIBRARY STRUCTURE

All subroutines in the DGC FORTRAN Run Time Library (hereafter called "the library") possess certain common features. The primary identifying feature is their reentrant nature. They are also relocatable; most entry points to these routines are in page zero memory. This reentrant, relocatable nature makes the routines suitable for use in Time-Sharing environments, as well as in assembly language programs and in FORTRAN object programs.

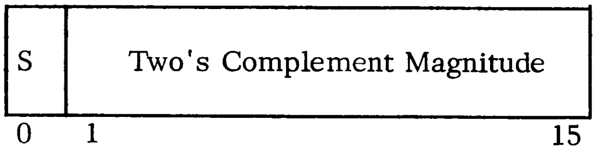
Primarily because of the difference in the assembly of .SYSTM calls, different revisions of the Run Time Library must be used depending upon the operating system supporting the main program. Programs supported by either the Stand-Alone Operating System (SOS) or the Disk Operating System (DOS), must use the DOS Run Time Library package. Programs supported by either the Real Time Disk Operating System (RDOS) or the non-disk Real Time Operating System (RTOS) must use the RDOS Run Time Library package.

Descriptions of routines in the main section of this manual, Chapter 2 "Integer Routines" through Chapter 13 "Array Handling Routines," are common to all operating system types except where noted. Multitask real-time routines, required by programs run with a real time operating system, and all routines peculiar to the RDOS FORTRAN IV library, are detailed in Appendix E.

Fixed Point Numbers

Fixed point numbers (integers) are represented by 16-bit words. Bit 0 contains the sign (0 if positive, 1 if negative). Bits 1 through 15 express the magnitude of the number in two's complement notation. All fixed point numbers are regarded as integers by library routines. The range of values that may be expressed by fixed point numbers is $-(2^{15}-1)$ to $+2^{15}-1$, or $-32,767_{10}$ to $32,767_{10}$ (the fixed point number 100000 is an illegal signed number since attempting to obtain its two's complement returns the same number). Zero must be expressed by an all zero word.

Fixed Point Numbers (Continued)

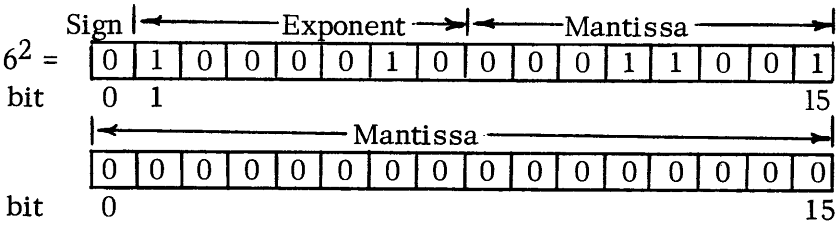


Real Numbers

Real numbers may be either single or double precision, and each precision may be in packed or unpacked form. Numbers on the number stack are always unpacked. Numbers elsewhere are usually in packed form.

Single precision floating point numbers (SPFL) in packed format occupy two sequential sixteen bit words. Packed SPFL numbers are expressed as a sign, a binary fractional mantissa 24_{10} bits long and an exponent to the base 16 to which is added an offset of 100_8 (excess 64 notation). Decimal exponents of values from 16^{-64} through 16^{+63} are represented as 0 through 177_8. Negative numbers are formed by setting the sign bit of the positive representation of the number to a 1. Thus $+25.0_{10}$ becomes 041031, 000000 and -25.0_{10} becomes 141031,000000 .

$+25_{10}=31_8=.062_8*16^2$ =

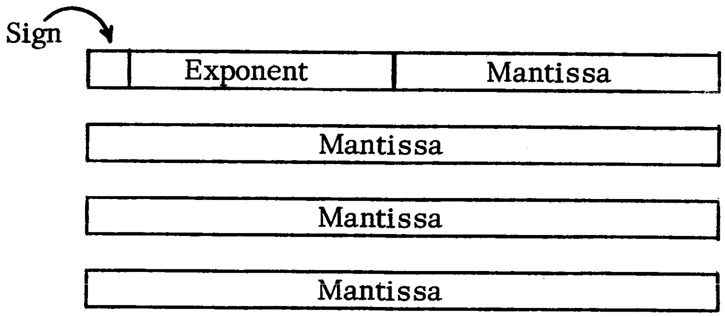


Packed Single Precision Real Format

Zero is represented as two sequential all zero words, but any SPFL number input to a routine with an all zero mantissa is considered zero. An SPFL number is considered normalized if at least one binary 1 is found in the first four positions of the mantissa (bits positions 8 through 11 inclusive). All SPFL numbers input to the library routines must be normalized. The range of values of an SPFL number are $2.4 * 10^{-78}$ to $7.2 * 10^{75}$ with significance in excess of 6 decimal digits.

Double precision floating point real numbers (DPFL) in packed form occupy four sequential 16 bit words. The first word allocates bit positions for the sign, exponent, and most significant portion of the mantissa as does a packed SPFL number. The remaining words express the rest of the mantissa. Rules for normalization and expressing negative numbers and zero are the same as for SPFL numbers. The range of alues is identical to the range for SPFL numbers, $2.4*10^{-78}$ to $7.2*10^{75}$, with significance in excess of 16 decimal digits.

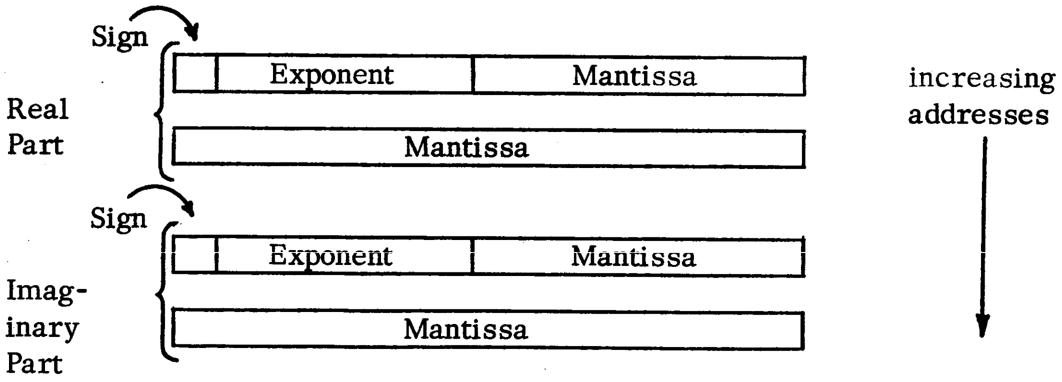
Real Numbers (Continued)



Double Precision Real Representation

Complex Numbers

Single precision complex numbers in packed format are composed of two sequential packed SPFL numbers, the first expressing the real portion of the number and the second expressing the imaginary portion of the complex quantity. Four sequential memory locations are required, and the bit definitions, range and significance which apply to SPFL numbers apply also to the real and imaginary portions of single precision complex numbers.



Single Precision Complex Representation

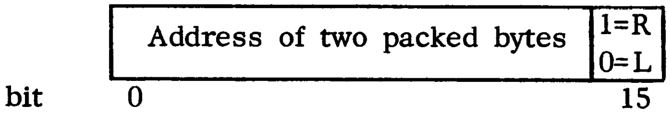
Double precision complex numbers in packed format consist of two sequential DPFL numbers. The first expresses the real portion of the number and the second expresses the imaginary portion. Eight sequential locations are required, and the bit definitions, range and significance figures which apply to DPFL numbers apply also to the real and imaginary portions of double precision complex numbers.

Byte Manipulation

Bytes and byte pointers used in the run time library are identical to bytes and pointers discussed in "How To Use the Nova Computers," with one exception: bit 15 is set to zero if the left byte is pointed to , and bit 15 is set to a 1 of the right byte is pointed to. Thus, to insure that packing occurs left to right, the

Byte Manipulation (Continued)

pseudo-op ".TXTM 1" should be included in any source program which generates ASCII text messages.



Byte Pointer

STACK STRUCTURE AND LINKAGE

The following discussion details the structure of the various run time stacks and the means used to access variables on these stacks in a single task FORTRAN environment. Single task FORTRAN includes both DGC stand-alone FORTRAN, DOS-supported FORTRAN, and RDOS single-task FORTRAN. For a description of stack structure and the partitioning of the run time stack area in multitask real time FORTRAN, see Appendix E.

SP Stack

The SP stack is a block of sequential locations with a page zero pointer, SP, used for general purpose temporary storage by subroutines which have no run time stacks. The SP stack is used primarily by the single and double precision routines. The following example shows how it might be used to save and then restore AC1:

```
STA   1,@SP
ISZ   SP
      .
      .
      .
DSZ   SP
LDA   1,@SP
```

SP stack overflow is a fatal error undetected by the library routines.

Number Stack

The Number Stack is a block of locations reserved for the storage of numeric values either as input or output for the arithmetic routines, or for temporary computational storage by these routines. The default size of this stack is 630 octal locations, although this size may be redefined by the user at assembly time by means of the following statements:

```
.ENT        .FLSZ
.FLSZ = n
```

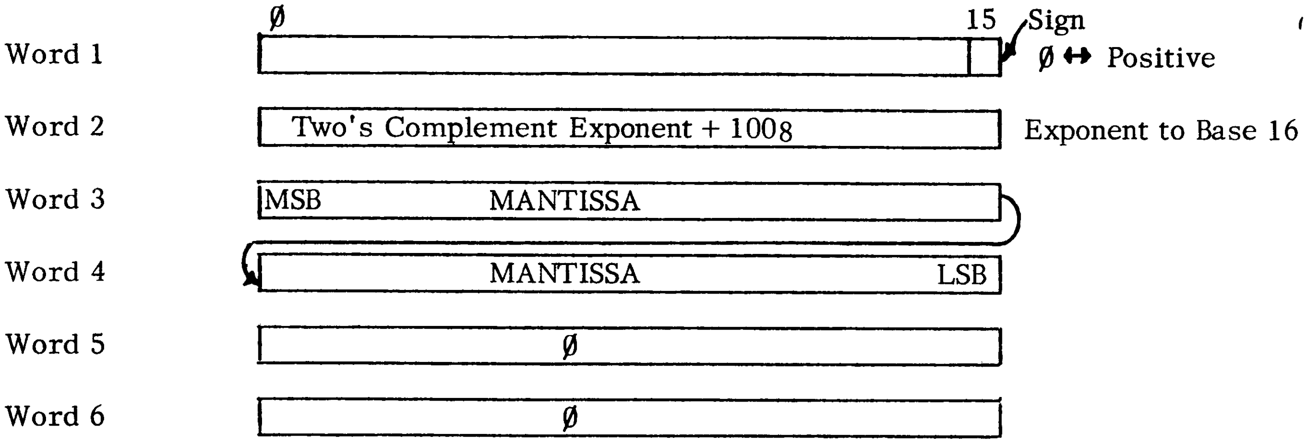
where n represents an absolute integer expression that can be evaluated at assembly time.

The maximum length of the number stack will then be equal to 2 * n + 30_8. (These two statements might be included in the FORTRAN source program, with an A in column one, so that they will be passed on to the assembler directly; see the FORTRAN IV User's Manual, Chapter 1.)

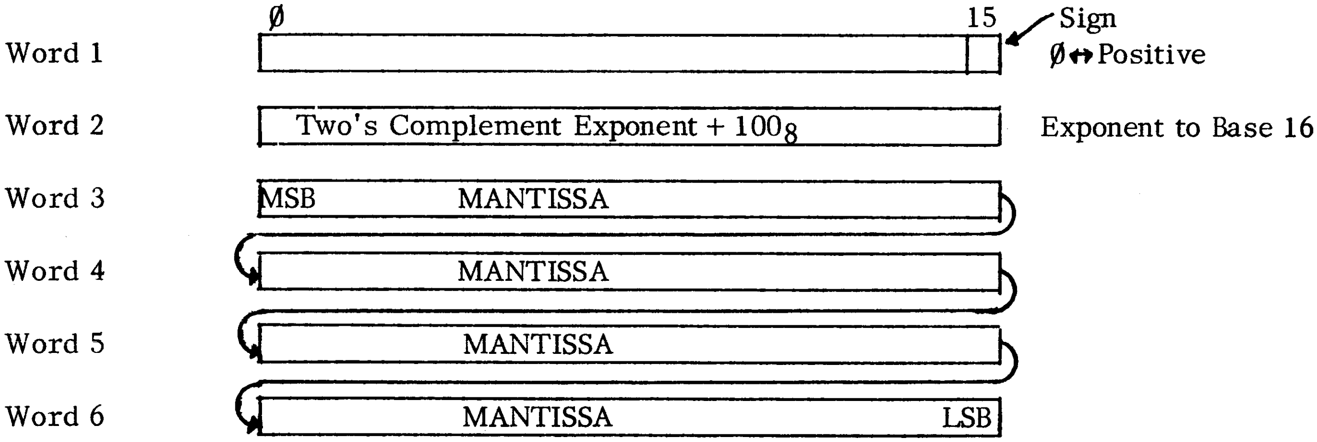
The entire storage of the number stack is seldom used. Instead the number stack expands dynamically as numbers are loaded onto it and contracts as they are removed. It never exceeds its maximum allotted area however. This stack is built in the direction of increasing addresses. The end of the number stack is pointed to by .NDSP; NSP (also called FLSP) points to the beginning of the most recently loaded number on the stack, the current top of the stack.

All numbers on the number stack are stored in sequential six word frames (or multiples of six word frames); this format is called unpacked format, and is shown in the following illustrations. Any attempt to load a number onto an already filled number stack will cause overflow error message FENSO to be issued. No such check or message is issued for number stack underflow, an attempt to load a number below the first frame on the number stack.

Number Stack (continued)



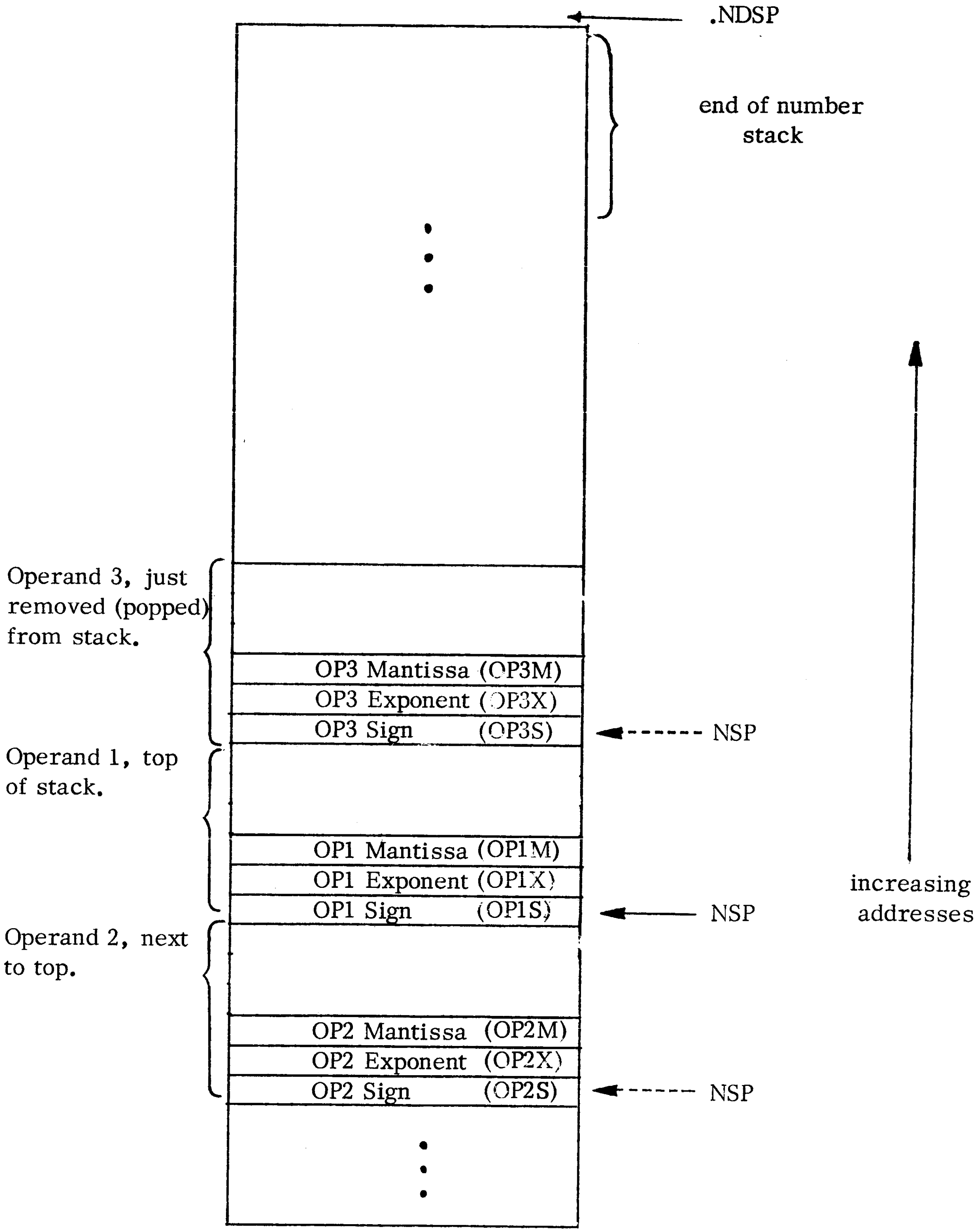
Unpacked SPFL Number Map



Unpacked DPFL Number Map

An integer to be loaded onto the number stack is first converted to an unpacked SPFL number. Single precision complex numbers are composed of two sequential unpacked SPFL numbers; the first (topmost) SPFL number represents the imaginary portion, and the next SPFL number represents the real portion of the complex number. Similarly, unpacked double precision complex numbers are composed of two sequential unpacked DPFL numbers.

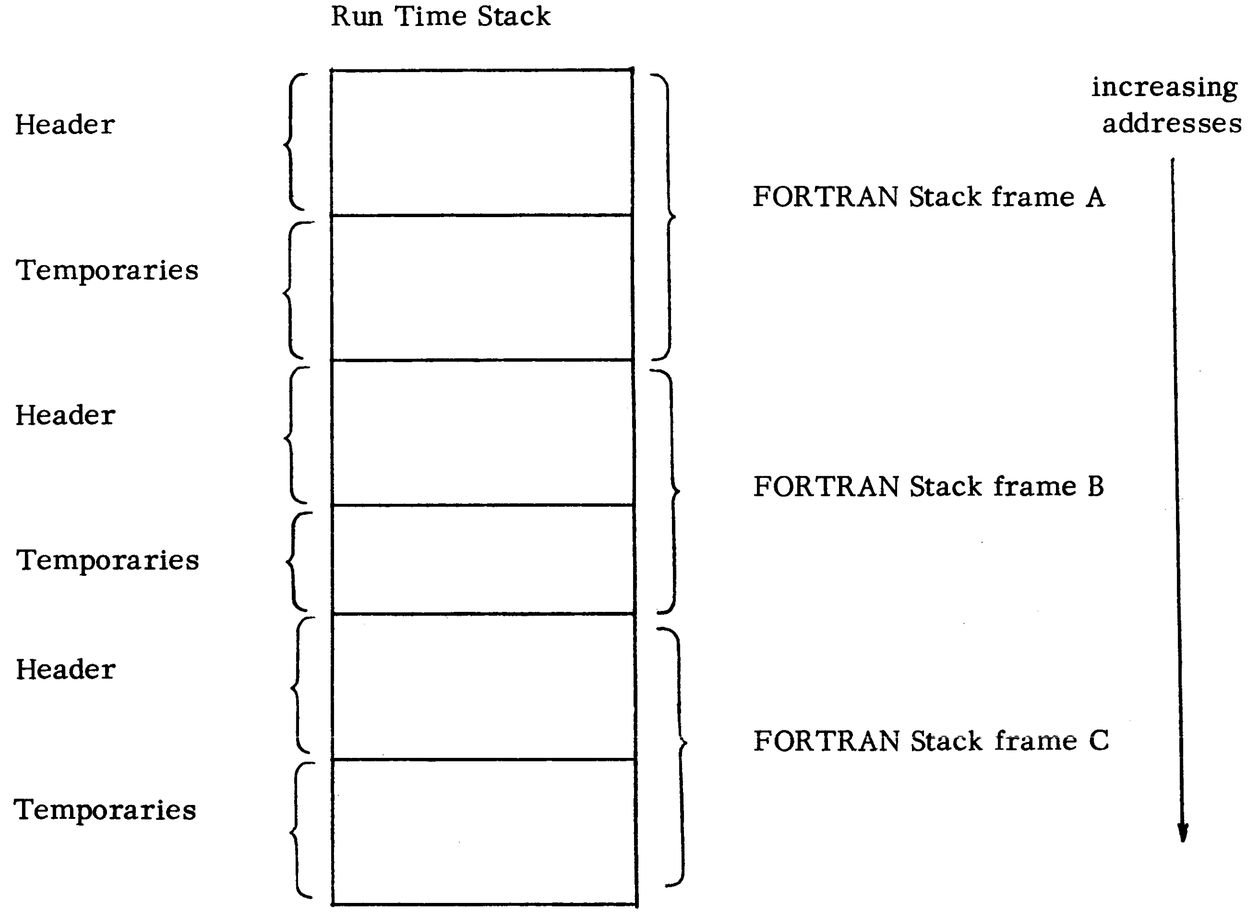
In the following illustration of the number stack broken arrows denote noncurrent values of NSP.



Number Stack Map

FORTRAN Linkage Stack

FORTRAN Linkage Stack frames are variable length blocks of sequential locations allocated for use by the main program and each run time subroutine requiring temporary storage. Each FORTRAN frame is composed of an initial 11 octal word header, and most routines require a varying length series of temporary locations following each header.



Each header location is used to store a specific type of information pertaining to the subroutine which owns it, and each header location is at a fixed displacement from a pointer called the current FSP. PARF, the FORTRAN parameter tape, defines FSP to be stored in cell 16_8, and also defines the fixed displacements and mnemonic assignments of each location in the stack header. The following illustration names these displacements and shows what information they contain.

FORTRAN Linkage Stack (Continued)

address	contents
FSP = cell 16	; 003000 = current contents of FSP

200_8 NREL locations		cell 2577 is the last temporary location of the previous stack frame.		
	Stack Header	cell 2600	FLGT	;Length of variable portion of this stack, 1 in this example.
		cell 2601	FOSP	;Old FSP
		cell 2602	FPLP	;Unused
		cell 2603	FEAD	;Entry address to the last routine called by this routine.
		cell 2604	FCRY	;State of carry at the time this routine issues a subroutine call. Bit 15 = Ø ↔ carry was Ø.
		cell 2605	FACØ	;Contents of ACØ when this routine issues a subroutine call.
		cell 2606	FAC1	;Contents of AC1 when this routine issues a subroutine call.
		cell 2607	FAC2	;Contents of AC2 when this routine issues a subroutine call.
		cell 2610	FRTN	;Address of next sequential instruction when the routine issues a subroutine call.
	Temporary	cell 2611	FTSTR or TMP	;Temporary storage available for use by this routine.
		(cell 2612	will be the FLGT of the FORTRAN stack frame belonging to the subroutine called by this routine.)	
		cell 3000	FSP	;Contents might be anything.

FORTRAN STACK

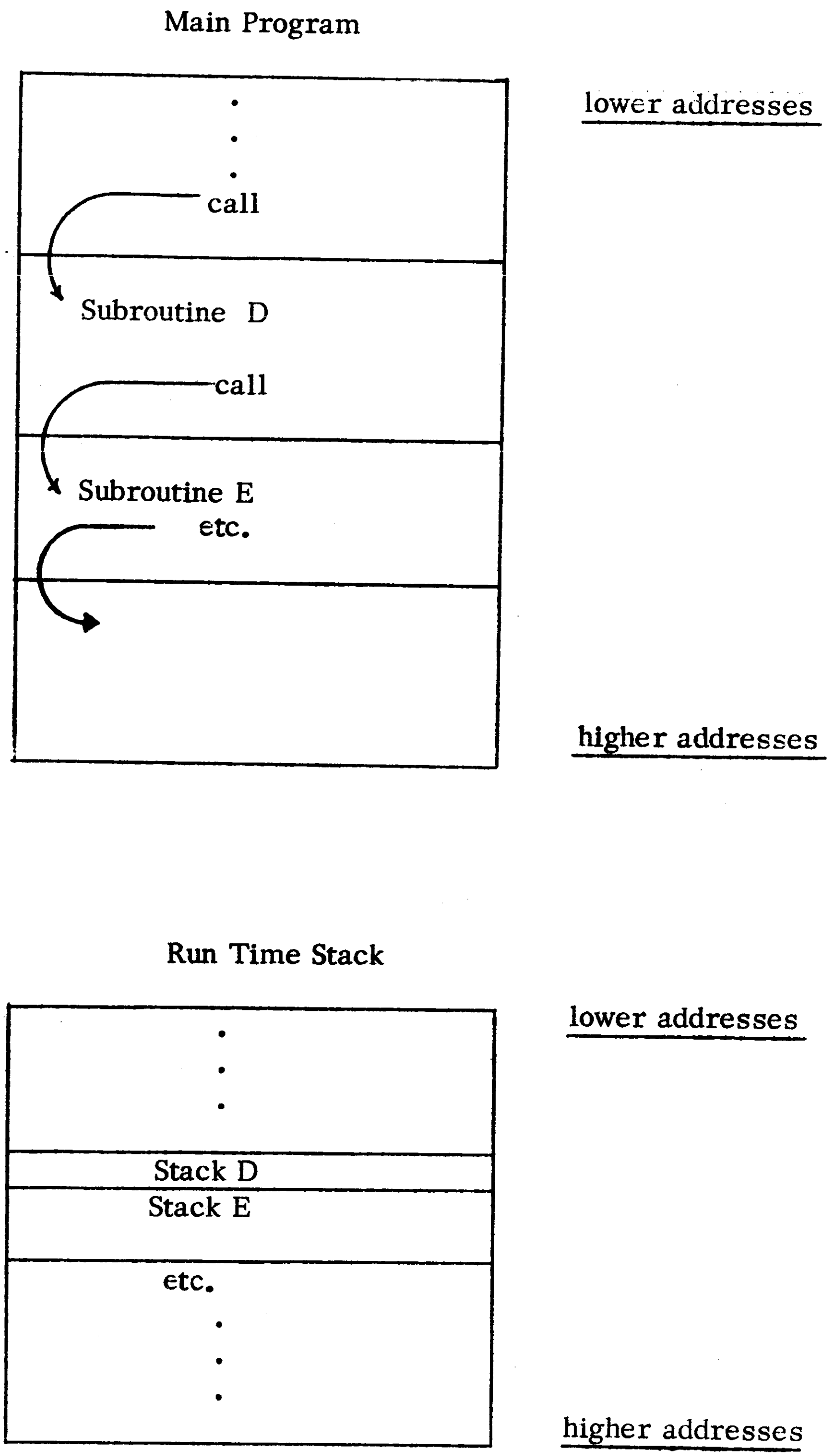
There is a page zero pointer, called QSP, which may reside anywhere from 20 through 377_8, that points to FAC2. This is the location where AC2 will be stored should the current routine call out, and it tracks FSP by an offset of -171_8. This pointer is used for immediate temporary storage by the FORTRAN linkage subroutines. For example STA 2,@QSP frees AC2 while STA 2,@FSP is not acceptable. QSP is defined as an external displacement in .I, the run time initializer.

Following FRTN is the series of temporary locations used for general purpose storage. The first of these is called FTSTR or TMP. The calling routine's accumulators, carry and return addresses are always recoverable from its stack header at locations FCRY through FRTN. FPLP is not currently used by the library routines.

In reality, the stack mnemonics are negative displacements which are added to the indexable center (FSP) of the current stack to obtain the effective address locations used for header and temporary storage. For simplicity's sake, we refer to FLGT ... FTSTR as though they were the effective addresses themselves. Similarly, we refer to the current FSP and QSP, by which we really mean the current contents of cells FSP and QSP. These mnemonics are defined in the PARF parameter tape, a portion of which is listed below.

FSP	=	USP (USP is predefined in the assembled to be 16_8.)
FRTN	=	-170
FAC2	=	-171
FAC1	=	-172
FAC0	=	-173
FCRY	=	-174
FEAD	=	-175
FPLP	=	-176
FOSP	=	-177
FLGT	=	-200
FTSTR	=	-167
TMP	=	FTSTR
FZD	=	-200

The area occupied by the FORTRAN stack frames, called the Run Time Stack, expands and contracts dynamically with the execution of the main program, expanding when more nests of subroutines are called or as subroutines are called which demand temporary storage. As the Run Time Stack expands, any FORTRAN stacks created earlier for subroutines already executed are overwritten by the new stacks. Stack overflow is said to occur if more storage area than the memory available at run time is demanded; AFSE is a page zero word used to determine the end or uppermost memory location available for the entire Run Time stack. It is declared as an entry by the library.



Stack Creation for Nested Subroutines

In every case, upon subroutine entry AC2 will then be set to contain a pointer to the calling program's stack frame (the old FSP) and AC3 will contain a pointer to the called routine's stack frame (new FSP) if one has been allocated, or -1 if no frame has been allocated. Upon return to the caller, carry and all registers except AC3 will be restored to their original values. AC3 will contain the caller's FSP.

Inter-Subroutine Linkage, FLINK

The library contains a set of subroutines called the FLINK module which enables the calling of other library routines and performs all stack frame creation/deletion and maintenance functions required. FLINK forms the nucleus of the run time subroutine's communications facility.

Library routines, including FLINK, have two types of entry points: page zero (.ZREL) or normally relocatable (.NREL) locations outside page zero. Those with .ZREL entries must be specified in an .EXTD statement, while those with .NREL entries require .EXTN statements. The following lists the mnemonic entries of the FLINK subroutines:

.EXTN	.EXTD
FCALL (JSR @.FCALL)	.FCALL
FRCAL	
FSAV (JMP @.FSAV)	.FSAV
FRET (JSR @.FRET)	.FRET
FQRET	

In reality, FRCAL and FQRET have page zero entries too, but these have not been entered with a .ENT statement and are not available for programming use.

The following table highlights the purpose of each FLINK subroutine.

FCALL (or JSR @.FCALL)	Used to call a library routine by its .NREL entry point. Also performs FSAV functions.
FRCAL	Used to call a library routine with its .NREL entry contained in AC2. Also performs FSAV functions.
FSAV (or JMP @.FSAV)	Used to maintain the caller's header, allocate a frame for the called subroutine, and update FSP.
FRET (or JSR @.FRET)	Used to return control to the caller, restore the caller's registers and carry, and update FSP.
FQRET	Provide a quick return to a caller when the called subroutine has no stack frame; restore the caller's registers and carry.

FSAV and an integer stack length word must immediately precede any subroutine which has a page zero entry point. The method of calling such a routine is JSR @.ADR where .ADR represents the page zero address containing the entry point (less two):

```
                  .ZREL
     .SBR:        SBR-2

                  .NREL
(page zero call)  JSR @.SBR
                  .
                  .
                  .
                  FSAV
                  n
     SBR:         True beginning of the subroutine
                  .
                  .
                  .
                  FRET (or FQRET)
```

The Stack Length word, SLW, labeled n in the illustration, may take on positive integral values Ø or -1. If the SLW is equal to -1, no stack header nor any temporary storage locations will be allocated for the called subroutine. In addition, no further calls can be made from the called routine. Subroutines which have a -1 SLW use the FLINK subroutine FQRET for exit and return to the next sequential address following the original subroutine call unless the user modifies FRTN. Subroutines with a -1 SLW typically provide quicker call and return to the caller, since no creation or maintenance of a stack for the called subroutine is required.

If the SLW is either zero or a positive integer, a new stack frame is created for the called subroutine, and the subroutine FRET must be used to provide a return of program control to the caller. If the SLW is Ø, a "bare bones" stack consisting of only a stack header is created; this would provide for the storage and restoral of the values in accumulators ACØ through AC3 and the state of carry should this subroutine make a call to another routine.

If the SLW is a positive integer, then a stack is created with both a header and the specified number of temporary storage locations.

Whenever one subroutine with a stack allocated for itself calls another subroutine with a stack, the contents of ACØ through AC2, carry, and the return address of the call are stored on the caller's stack, AC3 is set to the FSP value of the stack belonging to the new, called subroutine and AC2 is set to the FSP of the caller's stack. Should the called subroutine have no stack allocated for itself, AC2 is set to the caller's FSP but AC3 is left free for general purpose use.

If a subroutine in the library has no page zero entry, FCALL (also part of FLINK) may be used to perform the subroutine call, and the form of the call is:

```
FCALL
SBR
```

where SBR represents the .NREL entry point to the routine. Subroutines called by FCALL need not be preceded by FSAV since FCALL performs the functions of FSAV, although such subroutines must be preceded by a stack length word. Subroutines which have normal entry points in page zero can also be called by means of FCALL to the .NREL entry point. (Note that this type of call requires 2 words as opposed to 1 word.)

The FLINK module contains one other subroutine which permits the calling of a subroutine by its .NREL entry point: FRCAL. Subroutines called by means of FRCAL must have their entry points preceded by appropriate SLWs and as with FCALL, no FSAV is needed preceding the SLW. FRCAL is not followed by the name of the subroutine to be called; instead, AC2 is set to the address of the subroutine to be called, and then the instruction FRCAL is issued. FRCAL accomplishes the same functions as FCALL.

FORTRAN Addressing

The placing of current FSP values in AC3, and next-to-most-recent values of FSP in AC2 by FLINK permits an addressing scheme called FORTRAN Addressing, which is used by the library and the FORTRAN Compiler.

FORTRAN addressing extends the NOVA family addressing scheme in two ways:

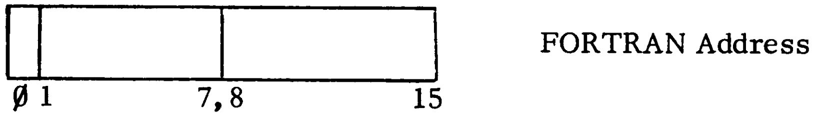
1. Variables on the stack are referenced relative to that stack's FSP.
2. Full word addressing for all absolute addresses is effected by subroutines .LDØ and .STØ.

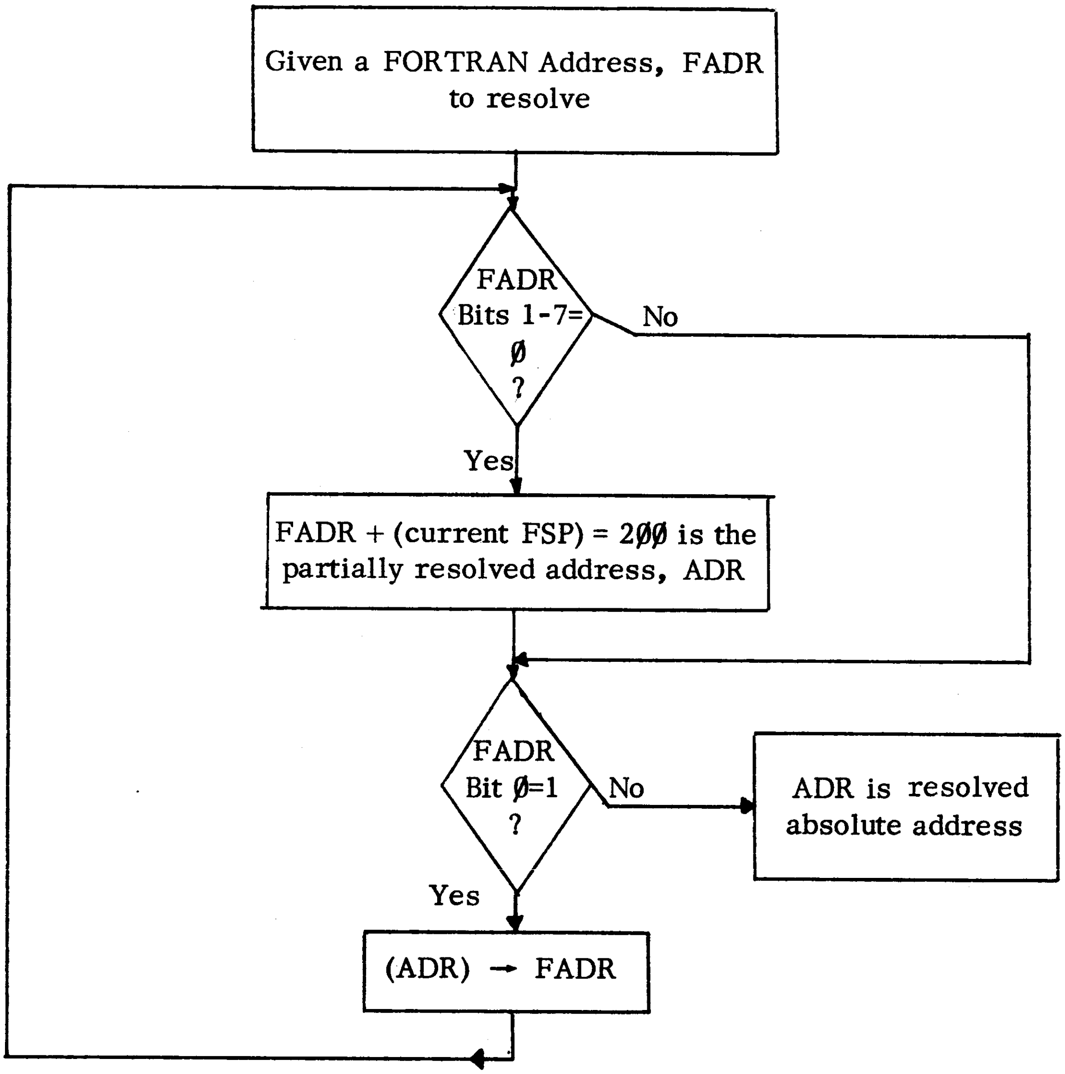
Since NOVA family computers can address 256_{10} words in an indexed instruction, using a bias of -200 through +177 each address on the stack can be referenced by using the centerpoint, FSP, and an offset stack displacement. Indirect stack displacements are also generated for dummy arguments of a function or a subroutine. Stack addresses are encoded as being between Ø and 377_8 inclusive, or as between 100000_8 and 100377_8 (the address of a variable, not the variable itself). FORTRAN addresses, when referring to locations on a frame, are equal to the displacement relative to FSP minus FZD (=-200). Thus the FORTRAN address of FLGT (=-200, see PARF) is equal to Ø, since FLGT - FZD = -200 - (-200) = Ø. Using similar reasoning, all direct FORTRAN stack addresses are positive, with a range of Ø through 377 inclusive.

FORTRAN STACK FRAME

FORTRAN address ∅	FLGT
FORTRAN address 1	FOSP
FORTRAN address 2	FPLP
FORTRAN address 3	FEAD
FORTRAN address 4	FCRY
	•
	•
	•
FORTRAN address 200	FSP

FORTRAN addresses greater than 377 are treated as absolute .NREL addresses. The chart on the following page illustrates the decisions made by library routines in interpreting FORTRAN addresses.





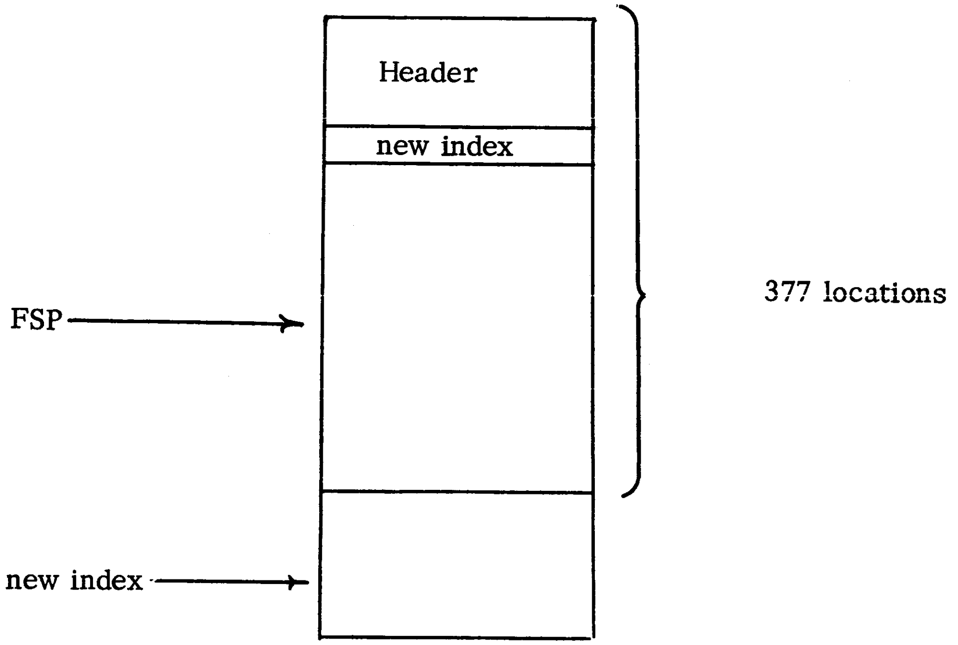
Given a FORTRAN Address, FADR to resolve
FADR Bits 1-7= ø ?
No
Yes
FADR + (current FSP) = 2øø is the partially resolved address, ADR
FADR Bit ø=1 ?
No
ADR is resolved absolute address
Yes
(ADR) → FADR

Since the most recent FSP is always placed in AC3 by the linkage routines (FLINK), any of 377 locations on a frame can be address in such instructions as:

```
LDA             ø, -167, 3
    which is equivalent to
LDA             ø, TMP, 3
```

As indicated earlier, stack frames may have lengths exceeding 377 locations. If frames exceed 377 locations, variations on the FORTRAN addressing scheme must be employed, possibly by placing pointers to new index values in the frame so that all locations may be accessed:

Jumbo Frame



FORTRAN array handling presents another means of accessing locations on a stack (see Appendix D).

The following illustrations give examples of FORTRAN addressing applications.

To adjust a caller's FRTN (without using further linkage routines which will be discussed), the following method might be employed:

```
              FCALL
              NAME
              Parameter
              Next Sequential Instruction

              Stack Length Word
NAME:         LDA     Ø, @FRTN, 2             ;Parameter→ ACØ
              ISZ     FRTN, 2                 ;Return can now be made to
              .                               ;the NSI
              .
              .
              FRET
```

One of the duties of FSAV is to preserve a caller's registers upon issuance of a further call. In order to do this, a register must be freed. The following example shows how FSAV's use of QSP accomplishes this end.

```
            .ZREL
TEMP:       .BLK      1
            .NREL
FSAV:       STA   2, TEMP
            LDA   2, FSP                     Without Using QSP
            STA   Ø, FACØ, 2
            STA   1, FAC1, 2
            STA   3, FRTN, 2
            LDA   Ø, TMP
            STA   Ø, FAC2, 2


FSAV:       STA   2, @QSP
            LDA   2, FSP
            STA   Ø, FACØ, 2                 Using QSP
            STA   1, FAC1, 2
            STA   3, FRTN, 2
```

QSP may also be used for temporary storage by a routine provided it is not being so used when a call out is made to a subroutine by means of FLINK.

```
            JSR   NAME
            Next Sequential Instruction



NAME:       STA   3, @QSP
            .
            .
            .
            LDA   3, FSP
            JMP   @FAC2, 3
```

In spite of the fact that FLINK restores the original values of a caller's registers, it is possible to pass results to a caller in one of the free registers. The following example illustrates one possible method.

```
            FCALL
            NAME
            .
            .
            .
            SLW
NAME:       .
            .
            .
            LDA     3,FSP
            LDA     2, FOSP, 3
            STA     1, FACØ, 2              ;The result is returned in
            FRET                            ;the caller's ACØ.
```

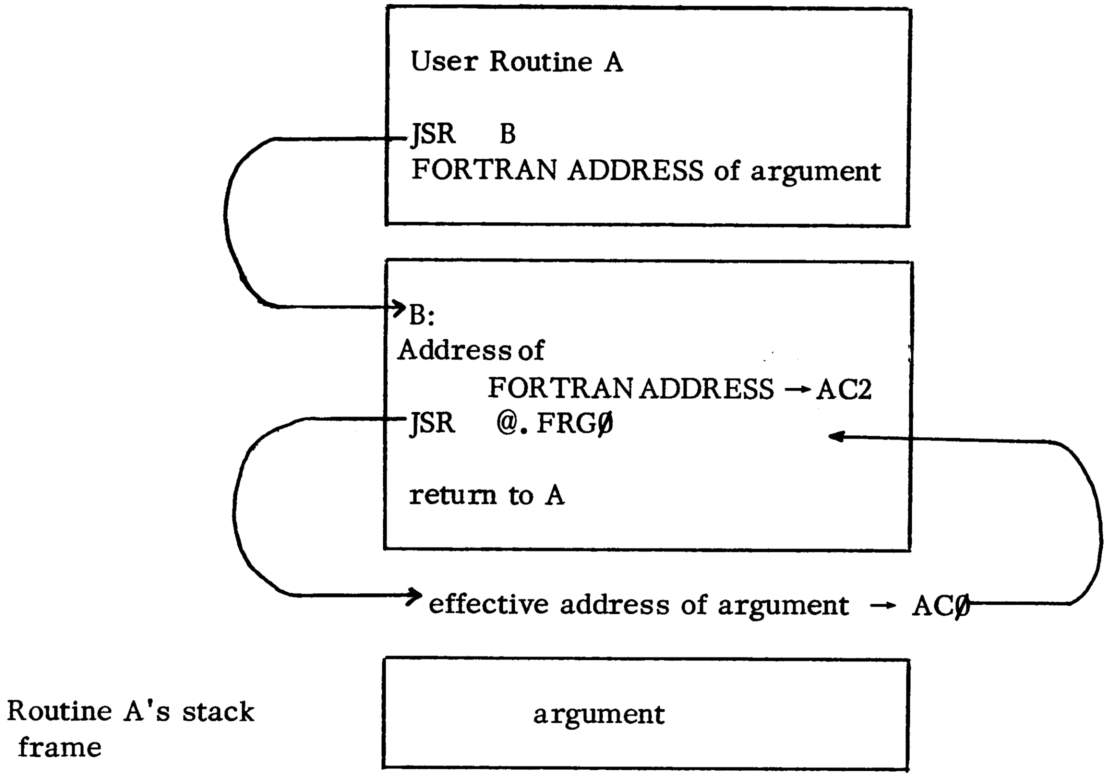
Similarly, conditional return can be provided by altering the caller's FRTN:

```
            FCALL
            NAME
            Return if condition 1 satisfied
            Return if condition 2 satisfied
            .
            .
            .
            SLW
NAME:       .
            .
            LDA     2,FOSP,3
            SUB     Ø, 1, SZR               ;Condition 2 satisfied?
            ISZ     FRTN, 2                 ;Yes
            FRET
```

Library Conversion of Fortran Addresses to Absolute Addresses

Several library routines are available for transforming FORTRAN addresses into absolute addresses: FRGØ/FRG1, MAD/MADO, FRGLD, CPYARG/CPYLS, and FARG. In addition to performing effective address calculation, FRGLD loads the contents of this address in ACØ. CPYARG/CPYLS and FARG transfer effective addresses to the caller's stack. FRGØ computes the effective address of a stack frame displacement with respect to the current FSP, while FRG1 performs this calculation with respect to the next most current FSP.

FRGØ operation



Subroutine B must not specify a stack frame.

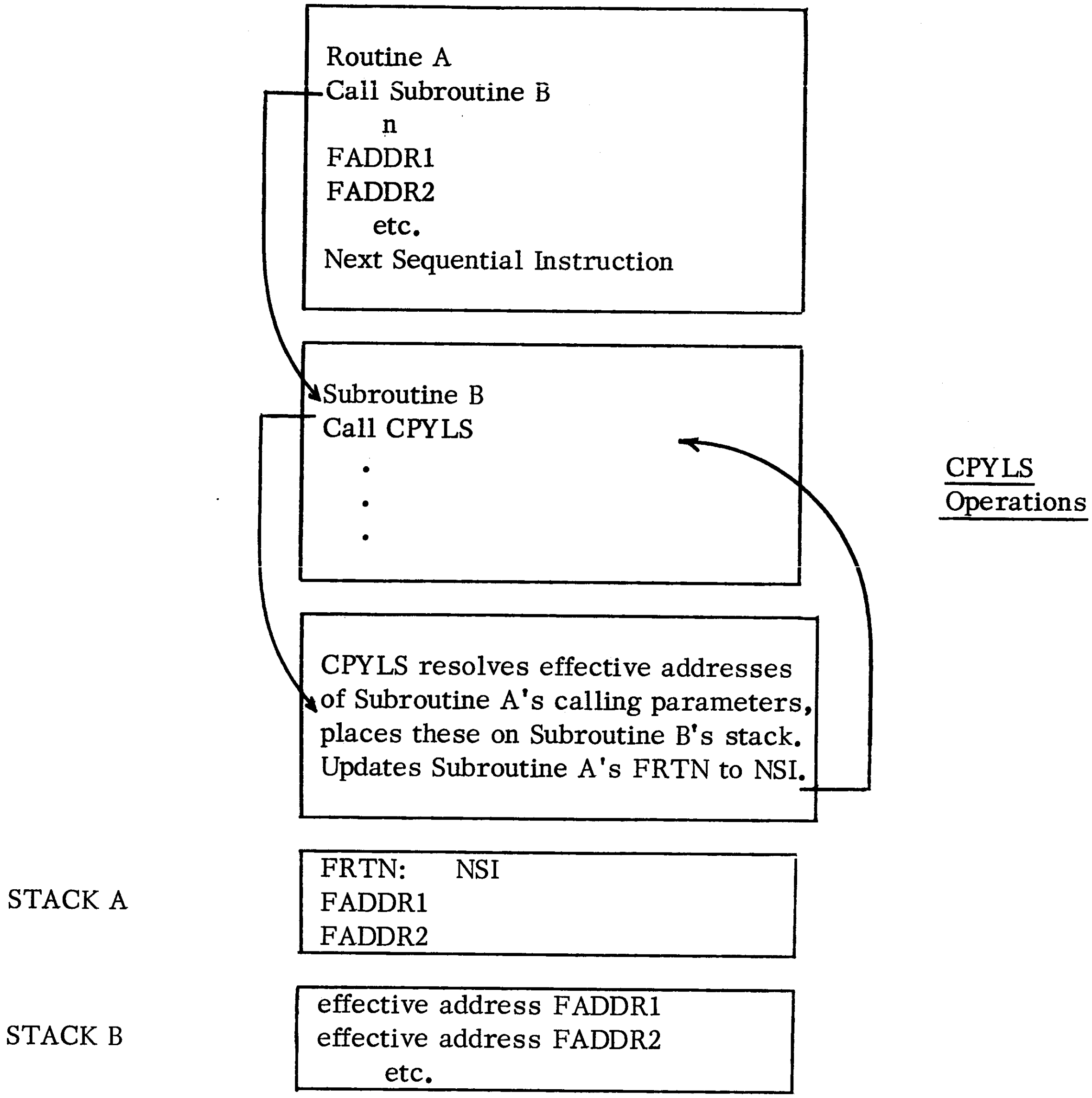
FRGLD computes the effective address of an argument stored at a FORTRAN ADDRESS, and then loads the contents of that address; ACØ receives the argument. If the address is a stack frame displacement, it must be a displacement on the next -most-recently created frame. FRGLD calls FRG1 to resolve the effective address of the argument.

The MAD/MADO module also computes effective addresses from FORTRAN ADDRESSES. If the address exceeds 377_8, then it is resolved as either an absolute NREL address or as an indirect address needing further resolution as shown on page 1-15. If the address is from 0 to 377_8 inclusive, the address is a stack frame displacement. The question "Which stack frame?" is answered by the entry which was selected to this module. If MAD entry, then the caller's stack frame is meant and the current FSP is used as a base for the address calculation. The resulting effective address is returned in AC2. The MAD/ MADO module itself has no stack frame, and does not restore the accumulators or carry to their entry values when return is made to the caller.

Passing Arguments from the Caller

There are two subroutines available in the library for resolving FORTRAN A DDRESSES passed by a caller and storing them on the stack frame of called subroutine B: FARG and CPYARG/CPYLS. FARG is used to pass argument addresses to the stack frame of the called subroutine without restoring caller B's accumulators and state of carry upon return to B.

CPYARG/CPYLS performs the same function, but restoring the original contents of accumulators and state of carry upon return to caller B. The only difference between CPYARG and CPYLS lies in the calling sequences which each accepts.



Routine A
Call Subroutine B
n
FADDR1
FADDR2
etc.
Next Sequential Instruction

Subroutine B
Call CPYLS
•
•
•

CPYLS resolves effective addresses of Subroutine A's calling parameters, places these on Subroutine B's stack. Updates Subroutine A's FRTN to NSI.

CPYLS Operations

STACK A

FRTN: NSI
FADDR1
FADDR2

STACK B

effective address FADDR1
effective address FADDR2
etc.

Returning Results to the Caller

The order of a calling sequence generated by the FORTRAN statement CALL SUB2 ($P_1 \ldots P_n$) is as follows:

```
JSR @.FCALL
SUB2
n
FORTRAN ADDRESS of Parameter 1
.
.
.
FORTRAN ADDRESS of Parameter n
Next Sequential Instruction
```

The called subroutine, SUB2, must fetch the FORTRAN ADDRESSES of each of the parameters, perform its function on the parameters, and return the result it has obtained back to the caller at the FORTRAN ADDRESS of the result (which may be one of the parameters). This it can do by first calling CPYARG (or CPYLS), using the effective addresses it has received, and then by returning the result to the caller's stack.

One way of returning this result is to load it into an accumulator and then store it:

```
STA   @Ø, TMP, 3
```

where AC3 contains the current (i.e, SUB2's) FSP. Assuming parameter 1 is the result address, TMP would contain the effective address of SUB1's Result, since the list of addresses of SUB1's parameters were transferred in order onto SUB2's stack. The effective address of the Result was transferred to SUB2's FORTRAN address TMP, the effective address of the second parameter was transferred to SUB2's TMP + 1, and so on by CPYARG. Often in assembly language programs it will be helpful to assign mnemonics to the displacements of the temporary storage locations following TMP, especially in cases where many of these storage locations are being used.

Stack Allocation at Run Time

Before the main FORTRAN program may be run, there must be an initial allocation of the primordial stacks, pointers to unlabeled common must be set up and the FORTRAN Channel Assignment Table must be set up to define the relationships between actual device drivers and logical FORTRAN channels. This whole complex task of initialization is performed by single task .I at the beginning of run time. (Multitask .I, found in library FMT.LB, is discussed in Appendix E.)

Stack Allocation at Run Time (Continued)

Single task .I is a subroutine from the first library file, consisting of 134 octal .NREL locations. .I also allocates a stack for itself, 60 octal locations plus header, where the Channel Assignment Table is placed. .I is called by the operating system (either the Stand-Alone or Disk Operating Systems) at the beginning of Run Time. At the end of the successful running of the FORTRAN program, return will be made to .I which transfers control unconditionally to the STOP routine. STOP outputs the message "STOP 999" to the system output device, and returns control to the operating system.

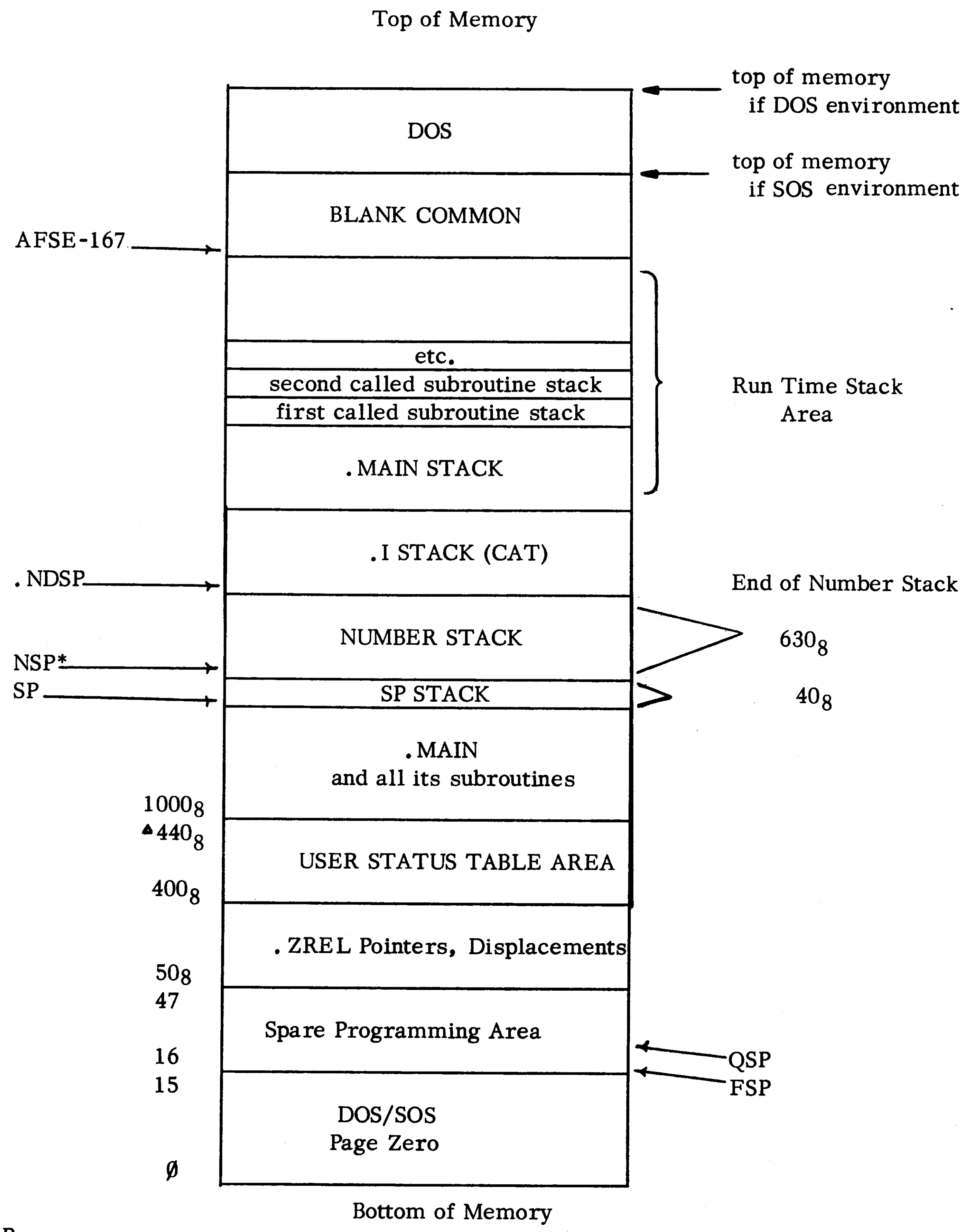
A system call is issued at the start, .SYSI, which initializes system I/O (this call is a no-op to DOS and RDOS). Then 40 octal locations are allocated for the SP stack immediately following the last loaded run time subroutine; a pointer to the beginning of the SP stack is also created.

Immediately following the last location in the SP stack, the number stack is defined and is allocated if floating point arithmetic is used in .MAIN or any of its subprograms. This stack will be 630 octal words long, or 30 octal plus twice whatever value a user has specified in a .FLSZ statement. The default value sets aside enough storage for 68 single precision complex numbers, or 17 double precision complex numbers.

After allocating the number stack (or after allocating the SP stack if no number stack is called for), a pointer to the beginning of the run time stack is defined, and .I's stack is allocated here. .MAIN's stack frame will be created, as soon as transfer is made to .MAIN .

Before this happens, a check is made to see whether or not there is enough room for blank common allocation, and blank common is allocated at the high end of memory if there is room enough for both it and the stacks which have been allocated. If not, a memory overflow message will be output and the system will wait for operator intervention. Assuming there is enough room, .NMAX is updated to acknowledge the stack allocations by means of a system call, .MEMI. The Channel Assignment Table is initialized and deposited in .I's temporary stack area, and program control is given to .MAIN.

The fact that no memory overflow is detected by .I in no way implies that there will be enough core space for all stack allocations which will be necessary at the peak requests of run time. Instead, a stack overflow check is made by the FLINK module each time a stack allocation request is issued, and a stack overflow error message is issued if insufficient space is detected. The following illustration depicts a typical map of memory during the execution of .MAIN. A simplified version of .I is given in Appendix B which illustrates the elements of .I as they might be used in an assembly language program. AFSE,referenced in both versions of .I, is a pointer used to determine whether stack overflow has occurred.



*NSP moves
towards NDSP
as numbers are
loaded on the stack.

▲start of .MAIN
in SOS environment.

MEMORY MAP OF .MAIN AT RUN TIME

USING THE RUN TIME LIBRARY

Structure of Subroutine Descriptions

Each subroutine description is subdivided into the following categories: Title, Purpose, Calling Sequence(s), Subroutine Size (and Timing), Supporting Routines (and Displacements), and Notes to User.

The "Title" is a name selected to describe the subroutine (or subroutines) being discussed. Usually these correspond to loader-recognized titles. A list of loader-recognized titles is given in the Summary Table in Appendix A.

The "Purpose" section is followed by "Calling Sequence" which illustrates the subroutine entry point, input parameters, and output result. Requisite inputs and output results are enclosed in parentheses. In most cases where alternate entry points or alternate entry means (like FCALL and JSR @ entries) are possible, the JSR@ entry will usually be given in the calling sequence, with the FCALL entry point listed in the "Notes to User."

The section titled "Subroutine Size" gives the octal number of locations required in both page zero and in the remainder of core memory for this subroutine (or set of subroutines if this subroutine is part of a load module). The figures do not include the storage requirements of any auxiliary subroutines required and called by this routine for support. Subroutines with the same Loader Title in the Summary Table share common coding in a load module. Thus, if either one or all of the subroutines in a module are loaded, the core storage requirements for these subroutines are the same and are equal to the size given for any of the subroutines in the module.

Selected subroutines have been measured to determine their typical execution times, and these are given in "Subroutine Size and Timing." The following comparisons of typical execution times of single precision real arithmetic functions run on the SUPERNOVA are given to illustrate the advantage obtained by using the hardware fixed point multiply/divide option.

Single Precision Real Subroutine	Typical Execution Time Using Software Multiply/Divide	Typical Execution Times With Hardware Multiply/Divide
SIN	3.3 ms	2.0 ms
COS	3.0 ms	1.9 ms
TAN	4.2 ms	2.5 ms
ATAN	3.6 ms	2.2 ms
EXP	4.9 ms	2.9 ms
LN	3.7 ms	2.4 ms
TANH	6.3 ms	4.2 ms
SQRT	2.9 ms	1.7 ms
ALOG10	4.1 ms	2.6 ms
ATAN2	5.3 ms	3.4 ms

SAMPLE SUPERNOVA EXECUTION TIMES

The following chart compares the typical execution times of the NOVA, NOVA 1200, and NOVA 800. The NOVA 1200 and NOVA 800 figures are given both with and without the use of the integer hardware multiply/divide option.

Double Precision Real Subroutine	NOVA with software multiply, divide	NOVA 1200 with software multiply, divide//with hardware multiply, divide	NOVA 800 with software multiply, divide//with hardware multiply, divide
FAD2*#	1.3 ms	.55 ms//.55 ms	.35 ms//.35 ms
FSB*#	1.4 ms	.55 ms//.55 ms	.35 ms//.35 ms
FML2*	5.7 ms	2.2 ms//1.2 ms	1.2 ms//.7 ms
FDV2*	11.4 ms	4.3 ms//2.0 ms	2.3 ms//1.4 ms
DPWER	180 ms	69 ms//37 ms	39 ms//25 ms
RIPWR	3 ms	2.5 ms//1.5 ms	1.5 ms//1.0 ms

* includes time required for two floating load operations and one floating store operation.

the integer multiply/divide routine is not used by this routine.

Thus to estimate the execution times for the NOVA 800 or NOVA 1200 with or without the integer hardware multiply/divide option, a series of conversion constants can be derived. The following series of conversion constants can be used to estimate the execution times of FORTRAN run time routines for which the execution time on the NOVA is already given. "With" means "with the hardware integer multiply/divide hardware," and "without" means "without the hardware."

NOVA 800 without:	NOVA 800 with	as	1	:	.66
NOVA 1200 without:	NOVA 1200 with	as	1	:	.55
NOVA without:	NOVA 1200 without:	as	1	:	.40
NOVA without:	NOVA 800 without	as	1	:	.25

External subroutines, pointers, and flags found elsewhere in the library which support each routine are indicated by the category "Supporting Routines." External normals will be listed to the left of the semicolon, external displacements to the right.

Note that displacements defined on the PARF and PARU tapes are not listed in this section. Consequently, it is good programming practice to always assemble these tapes with any user written subroutines.

"Notes to User" specifies whether error messages can be generated, and whether the contents of accumulators and the state of carry are restored to their entry values upon exit from this routine. The statement that accumulators and carry are restored should be understood to be qualified because AC3 is always set to contain the current FSP and AC2 the next most current FSP upon subroutine entry. Upon return AC2 is the caller's original AC2. Moreover, error messages may be issued by subroutines supporting a main routine which, itself, is incapable of issuing such messages.

Finally, subroutine descriptions are arranged alphabetically by title within each of twelve categories. The list and order of appearance of these categories is as follows:

Integer
Single Precision Real
Double Precision Real
Single Precision Complex
Double Precision Complex
Mixed Mode
String/Byte Manipulation
Pointers/Displacements
Stack Linkage, Initialization
Input/Output
Miscellaneous Fortran Support
Array Handlers

Interfacing Assembly Language Routines to FORTRAN Programs

If it is desired to write a function or subroutine in assembly language which will be called by a FORTRAN program, or which call FORTRAN programs or subprograms, several points must be borne in mind:

1. First 5 letters in name must be unique and distinct from library defined entries.
2. Include the statement .ENT name.
3. Select a unique title (.TITL title).

The code generated by the FORTRAN statement CALL NAME (x, y, z) is as follows:

```
        .EXTN NAME
        JSR      @.FCAL
        NAME
        n                          ;Where n represents the number of arguments
        FORTRAN ADDRESS of x
        FORTRAN ADDRESS of y
        FORTRAN ADDRESS of z
```

All externals which are to be resolved in the displacement field of an instruction at load time are specified by .EXTD. Examples of these are page zero entries and page zero flags. All other externals (FCALL entries, primarily) are specified by .EXTN.

The lower case n in the above calling sequence represents an integer equal to the number of parameters in the calling sequence. The .FCAL routine saves accumulators, carry and the current FSP, and allocates a stack frame for the called subprogram. The statement .EXTN NAME need appear only once in a program.

The converse of the calling sequence generated by the FORTRAN CALL statement is the receiving sequence. This is the means by which the calling parameters are fetched by the called subroutine. The form of the receiving sequence generated by the use of the FORTRAN statement SUBROUTINE is as follows:

```
         FS
  NAME:  JSR  @.CPYL
         .
         .
         .
```

FS in the above illustration is the number of temporary locations required by the subroutine NAME in the FORTRAN stack. FS must be large enough to provide for the maximum number of arguments expected by the routine. .CPYL converts the n argument address to effective addresses and places these addresses in locations TMP, TMP + 1, ... TMP+n-1 on its FORTRAN stack. Even if no arguments are passed, .CPYL is still called to correct the contents of FRTN so that program control will return to the next sequential FORTRAN statement.

Lastly, the assembly language code generated by the FORTRAN RETURN statement is JSR @.FRET, called FRET earlier. As mentioned earlier, FRET restores accumulators, carry, the contents of FSP, and places the current FSP in AC3.

Appendix C lists two library routines and a sample program which calls routines from the library to illustrate the linkage principles discussed above.

To call a FORTRAN subroutine or function in an assembly level program, care must be exercised to assure that arguments passed to the subprogram agree in number, order and type with the arguments required by the subprogram. Given the following FORTRAN subroutine statements,

SUBROUTINE name $(arg_1, arg_2, \ldots, arg_n)$

the assembly language code required to call this subroutine would be:

```
.EXTN  name
FCALL
name
N
FORTRAN ADDRESS 1
FORTRAN ADDRESS 2
.
.                    }      Argument Addresses
.
FORTRAN ADDRESS n
```

In like fashion, given the following FORTRAN function,

FUNCTION name $(arg_1, arg_2, \ldots arg_n)$

The assembly language code required to call this function would be:

```
.EXTN name
FCALL
name
N + 1
FORTRAN ADDRESS of result
FORTRAN ADDRESS 1
FORTRAN ADDRESS 2
.
.                    }           Argument Addresses
.
FORTRAN ADDRESS n
```

If the argument list is empty in either a subroutine or function definition, N=Ø must be specified explicitly.

Finally, if any text strings are to be passed to FORTRAN routines, the first must be preceded by a statement to force the storing of text in left to right order: .TXTM 1.

INTEGER ROUTINES

BASC

Purpose: To convert an unsigned fixed point number to an ASCII string of six octal digits converted to ASCII.

Calling Sequence: (ACØ contains the Byte Pointer to the returned string. AC1 contains the number to be converted.)

```
FCALL
.BASC
```

(Leading zeroes are not suppressed; string is terminated with a null byte, ACØ contains updated pointer to null byte.)

Supporting Routines: FSAV, FRET; .STBT .

Subroutine Size: 35 octal locations of normally relocatable memory are required.

Notes to User: The input fixed point number is of the following form, with N representing an octal digit:

$N_6N_5N_4N_3N_2N_1$

The output ASCII string is in the following form, where A_n corresponds to N_n:

A_6A_5
A_4A_3
A_2A_1
Ø Ø

No error messages are output.
Contents of accumulators, carry are restored.

.BASC must be specified with an .EXTN statement.

BDASC

Purpose: To convert an unsigned fixed point number to a string of ASCII decimal characters.

Calling Sequence: (ACØ contains the output string pointer. AC1 contains the number which is to be converted.)

```
FCALL
.BDASC
```

(Leading zeroes are suppressed; string is terminated with a null byte. ACØ contains the updated pointer to the null byte.)

Supporting Routines: FSAV, FRET; .STBT .

Subroutine Size: 62 octal locations of normally relocatable memory are required.

Notes to User: No error messages are generated.

Contents of accumulators, carry are restored.

.BDASC must be specified with an . EXTN statement.

IABS

Purpose: To compute the absolute value of an integer argument.

Calling Sequence:

```
JSR    @IA.S
FORTRAN ADDRESS of result        ;A NON-NEGATIVE INTEGER.
FORTRAN ADDRESS of argument      ;ANY INTEGER
```

(The location containing the result will be expressed as a FORTRAN ADDRESS immediately following the call).

Supporting Routines: FRET, FSAV; CPYARG .

Subroutine Size: One page zero location and 11 octal locations of normally relocatable memory.

Notes to User: Original states of accumulators, carry restored upon exit.

No error messages are generated.

IA.S must be referenced by an .EXTD statement.

This routine has an FCALL entry point, .IABS .
.IABS must be referenced by an .EXTN statement.

IDIM

Purpose: To compute the positive difference of two integers I and J.

Calling Sequence:

JSR @ID.M
FORTRAN ADDRESS of result
FORTRAN ADDRESS of I
FORTRAN ADDRESS of J

(If I-J ≤ Ø, the result is Ø; otherwise, the result is the difference I-J).

Supporting Routines: FSAV, FRET; .FARG .

Subroutine Size: One page zero location and 13 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are restored.

No error messages are generated.

ID.M must be referenced by an .EXTD statement.

This routine has an FCALL entry point, .IDIM .
.IDIM must be referenced by an .EXTN statement.

IPWER

Purpose: To raise an integer to an integer power, with an integer result.

Calling Sequence: (The integral base is in AC1, and the integral power is in ACØ)

```
JSR    @.IPWR
```

(The result is deposited in AC1).

Supporting Routines: MPY; SP, .RTES .

Subroutine Size: One page zero location and 53 octal locations of normally relocatable memory.

Notes to User: Original states of accumulators and carry are lost.

If overflow occurs, or if a zero base was input to the routine, an error message is issued.

. IPWR must be referenced by an . EXTD statement.

ISIGN

Purpose: To transfer the sign of one integer to another integer.

Calling Sequence:

JSR @ IS.GN
FORTRAN ADDRESS of result
FORTRAN ADDRESS of integer receiving the sign
FORTRAN ADDRESS of integer whose sign is being transferred.

Supporting Routines: FRET, FSAV; .FARG .

Subroutine Size: One page zero location and 14 octal locations of normally relocatable memory.

Notes to User: Original states of accumulators, carry are restored.

No error messages are generated.

IS.GN must be referenced by an .EXTD statement.

This routine has an FCALL entry point, .ISIGN .
.ISIGN must be referenced by an .EXTN statement.

MAXØ, MINØ

Purpose:

To select the smallest (MINØ) or the largest (MAXØ) member from a set of integers, expressing the selection as an integer.

Calling Sequence:

JSR @MA.Ø (or @MI.Ø)
N (an integer constant specifying the number of members +1 in the set of integers.)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of I_0
FORTRAN ADDRESS of I_1
.
.
.
FORTRAN ADDRESS of I_{N-1}

(The integer result is stored at the FORTRAN ADDRESS immediately following N.)

Supporting Routines:

FRET, FSAV; .FARG .

Subroutine Size:

Two page zero locations and 44 octal locations of normally relocatable memory.

Notes to User:

Accumulators, carry are restored upon exit. No error messages are generated.

FCALL entry points are MINØ and MAXØ.

MAXØ and MINØ must be referenced by an .EXTN statement. MA.Ø and MI.Ø must be referenced by an .EXTD statement.

MOD

Purpose: To fetch the remainder of an integer quotient when integer I_1 is divided by integer I_2.

Calling Sequence:

```
JSR    @MO.
FORTRAN ADDRESS of  result
FORTRAN ADDRESS of  integer I1
FORTRAN ADDRESS of integer I2
```

(The location of the result is expressed as a FORTRAN ADDRESS immediately following the call.)

Supporting Routines: FSAV, FRET; .FARG, .SDVD .

Subroutine Size: One page zero location and 11 octal words of normally relocatable memory.

Notes to User: In the case of an illegal division, an error return will be made by .SDVD, and a zero result will be returned.

Original contents of accumulators, carry will be restored upon exit.

MO. must be referenced by an .EXTD statement. This routine has the FCALL entry point .MOD .

MPY, DVD, MPYØ

Purpose:

To enable the use of the unsigned hardware multiply/divide option on a NOVA family computer and restore FSP upon exit.

Calling Sequences:

(AC1 and AC2 contain the multiplier and multiplicand upon input to the routine; contents of ACØ will be added to the product.)

MPY

(The product of AC1 and AC2 is computed, and the entry contents of ACØ is added to the product. This sum is returned with the more significant half in ACØ, the less significant half in AC1. AC3 contains the caller's FSP upon exit.)

(AC1 and AC2 contain the multiplier and multiplicand upon entry.)

MPYØ

(The product of AC1 and AC2 is returned with the less significant half in AC1, and the more significant half in ACØ. AC3 contains the caller's FSP upon exit.)

(The high and low parts of the dividend are in ACØ and AC1, the divisor is in AC2.)

DVD

(The remainder is in ACØ, the quotient is in AC1, AC2 is unchanged, and carry is clearea; AC3 is set to FSP. Upon overflow, carry is set, FSP is placed in AC3, and return is made with the accumulators unchanged.)

Supporting Routine:

.SVØ .

Subroutine Size:

Three page zero locations, 31 octal locations of normally relocatable memory for NMPYD ; 3 page zero locations and 15 locations of normally relocatable memory for HMPYD (see Notes to User).

MPY, DVD, MPYØ (Continued)

Notes to User:

Tape NMPYD (099-000011) must be loaded at link load time for the NOVA. For the NOVA 1200, 800 and SUPERNOVAs, load tape HMPYD (099-000009).

MPY, MPYØ, and DVD must each be specified in a .EXTN statement.

MPY, MPYØ, DVD

Purpose:

To perform unsigned integer multiplication and division on NOVA family machines lacking the hardware multiply/divide.

Calling Sequences:

Same as for machines with the hardware multiply/divide option. (See page 2-11)

Supporting Routine:

. SVØ .

Subroutine Size and Timing:

Three page zero locations and 33 octal locations of normally relocatable memory.

Typical execution times for MPYØ are:
74 μs on the Supernova and 349 μs on the Nova.

Typical execution times for MPY are:
73 μs on the Supernova and 343 μs on the Nova.

Typical execution times for DVD are:
96 μs on the Supernova and 491 μs on the Nova.

Notes to User:

Tape MULT, (099-000008), must be loaded for software multiply/divide on all NOVA family machines.

MPY, MPYØ, DVD must each be specified in an . EXTN statement.

SDVD

Purpose: To perform a division of two signed integers.

Calling Sequence: (ACØ contains the signed divisor, AC1 contains the signed dividend.)

```
JSR    @.SDVD
```

(ACØ contains the signed remainder, AC1 the signed quotient.)

Supporting Routines : DVD; .RETS, SP.

Subroutine Size: One page zero location and 46 octal locations of normally relocatable memory.

Notes to User: Division by zero or input value 2^{15} will cause an error message to be issued, with a zero quotient and remainder.

Original states of accumulators and carry will be preserved except as noted.

.SDVD must be referenced by an .EXTD statement.

SMPY

Purpose: To perform a multiplication of two signed integers.

Calling Sequence: (ACØ contains the signed multiplicand, AC1 contains the signed multiplier)

```
JSR    @.SMPY
```

(AC1 contains the signed result; the result is Ø if overflow occurs.)

Supporting Routines: MPY; .RTES, SP .

Subroutine Size: One page zero location and 24 octal locations of normally relocatable memory.

Notes To User: Original contents of accumulators and carry are preserved except as noted. An error message is output if overflow occurs.

.SMPY must be referenced by an .EXTD statement.

SINGLE PRECISION FLOATING POINT ROUTINES

ABS

Purpose: To compute the absolute value of any real number.

Calling Sequence: (The number whose absolute value is to be calculated is on top of the number stack.)

```
JSR    @.ABS
```

(The absolute value of the original number is on top of the number stack.)

Supporting Routine: FSAV, FQRET ; NSP .

Subroutine Size: One page zero location and 6 locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit. No error messages are generated.

ABS., XAS., and DABS., are each equivalent to JSR @.ABS .

.ABS must be referenced by an .EXTD statement. ABS., XAS., and DABS. must each be referenced by an .EXTN statement.

This routine has an FCALL entry point, ABS . ABS must be referenced by an .EXTN statement.

AINT

Purpose: To truncate a single precision real number.

Calling Sequence: JSR @AI. T
FORTRAN ADDRESS of number to be truncated.

(The truncated real is placed on the number stack.)

Supporting Routines: none; . FRGØ, FFLD1, NSP, SP .

Subroutine Size: One page zero location and 60 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are not restored upon exit from this routine. No error messages are generated.

JSR @XA. T is equivalent to JSR @AI. T .

XA. T and AI. T must each be referenced by an . EXTD statement.

ALOG

Purpose: To compute the single precision real natural logarithm of a single precision real positive argument x.

Calling Sequence: (Input argument x is placed on the top of the number stack)

ALOG.

(Output result replaces x on the top of the number stack.)

Supporting Routines: FSAV, FRET;.RTER, .FARG, FSB1, FAD1, FML1, FLIP1, FDV1, FFLD1, FPLY1, FCLE1, FXFL1, NSP, FRLD1 .

Subroutine Size: Two page zero locations and 205 octal locations of normally relocatable memory are required.

Notes to User: The single precision real base 10 logarithm function has an alternate entry point in this routine (see AL. GØ).

ALOG. must be referenced by an .EXTN statement.

In the case of a zero argument, an error message is given and the largest possible real number is returned as a result.

In the case of a negative argument, an error message is given and the logarithm of the absolute value of the argument is computed.

Accumulators and carry are restored upon exit from this routine.

This routine has an FCALL entry point, ALG. ALG must be referenced by an .EXTN statement.

AL.GØ

Purpose:

To compute the single precision real base 10 logarithm of a single precision real argument x.

Calling Sequence:

JSR @AL.GØ
FORTRAN ADDRESS of x
(Output result is placed on the top of the number stack.)

Supporting Routines :

FSAV, FRET; .RTER, .FARG, FSB1, FAD1, FML1, FXFL1, FDV1, FPLY1, FFLD1, FCLE1, FLIP1, FRLD1, NSP .

Subroutine Size and Timing:

Two page zero locations and 205 octal locations of normally relocatable memory are required.

Typical execution times are 17 ms for the NOVA with software multiply/divide and 2.6 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

The single precision real natural logarithm function, ALOG., has an alternate entry point in this routine.

AL.GØ must be referenced by an .EXTD statement.

Accumulators and carry are restored upon exit from this routine. This routine has the FCALL entry point .ALG10.

AMAX1, AMIN1

Purpose: To select the smallest (AMIN1) or the largest (AMAX1) member from a set of single precision real numbers, expressing the selection as a single precision real number.

Calling Sequences:

```
JSR    @AM.X1       (or AM.N1)
N  (an integer constant specifying the number of members
    in the set.)
FORTRAN ADDRESS of R0
FORTRAN ADDRESS of R1
         .
         .
         .
FORTRAN ADDRESS of RN-1

(The result is placed on the number stack.)
```

```
FCALL
AMAX1      (or AMIN1)
N+1     (where N is an integer constant specifying the number
           of members in the set.)
FORTRAN ADDRESS of RESULT
FORTRAN ADDRESS of R0
FORTRAN ADDRESS of R1
         .
         .
         .
FORTRAN ADDRESS of RN-1

(The result is expressed as a single precision real stored
at the FORTRAN ADDRESS of the result given in the calling
sequence.)
```

Supporting Routines: FSAV, FRET; FFLD1, FFST1, FCLT1, .FARG .

Subroutine Size: Two page zero locations and 74 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit from the routine. No error messages are generated. AM.X1 and AM.N1 must be referenced by an .EXTD statement. AMAX1 and AMIN1 must be referenced by an .EXTN statement.

AMOD

Purpose: To fetch the remainder in the quotient of two single precision real arguments.

Calling Sequence:

```
JSR    @ AM.D
FORTRAN ADDRESS of dividend
FORTRAN ADDRESS of divisor
```

(Result is placed on the top of the number stack.)

Supporting Routines: none; FFLD1, FDV1, FML1, .FRGØ, NSP .

Subroutine Size: One page zero location and 100 octal locations of normally relocatable memory.

Notes to User: If the quotient causes overflow or underflow, an error message is output by FDV1 and no meaningful result is obtained.

Contents of accumulators, carry are lost.

AM.D must be referenced by an .EXTD statement.

ATAN, ATAN2

Purpose:

To compute the real arctangent of either a real argument x or the quotient of two real arguments, y/x.

Calling Sequences:

One real argument x

(Input argument x on top of the number stack.)

```
          ATAN.
```

(Output argument replaces x on the number stack.)

Two real arguments y and x

```
JSR    @AT.N2
FORTRAN ADDRESS of y
FORTRAN ADDRESS of x
```

(Output argument is placed on top of the stack.)

Supporting Routines:

FSAV, FRET; FAD1, FML1, FDV1, FPLY1, FSB1, FLIP1, FCLT1, FNEG1, FFLD1, FRLD1, NSP .

Subroutine Size and Timing:

Two page zero locations and 222 octal locations of normally relocatable memory are required.

Typical execution times are 13 ms for the NOVA with software multiply/divide and 2.2 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

Although the routine will accept input arguments of any size, results computed by the routine will fall within the following ranges:

$-\pi/2 < \mathrm{ATN}(x) < \pi/2.$

$-\pi \leq \mathrm{ATN2}(x, y) \leq \pi$.

ATAN, ATAN2 (Continued)

Notes to User:

Overflow is possible as the divisor x approaches zero. In the case of overflow + or - $\pi/2$ is returned.

ATAN. must be referenced by an .EXTN statement. AT.N2 must be referenced by an .EXTD statement.

ATAN. has an FCALL entry point, ATN . ATN must be referenced by an .EXTN statement.

COS

Purpose:

To compute the real cosine of an argument x expressed as a single precision real number.

Calling Sequence:

(Input argument x is placed on the top of the number stack.)

```
COS.
```

(Output result replaces x on the number stack.)

Supporting Routines:

FSAV, FRET; FPLY1, FDV1, FML1, FSB1, FNEG1, FBRK1, FRLD1, NSP .

Subroutine Size and Timing:

Two page zero locations and 145 octal locations of normally relocatable memory are required.

Typical execution times are 13 ms for the NOVA with software multiply/divide and 1.9 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

The single precision real sine function has an alternate entry point in this routine.

In the case of large arguments of the form $2n\pi + \theta$, $-\pi \leq \theta \leq \pi$, when n becomes very large, significant digits in the result will be lost.

COS. must be referenced by an .EXTN statement.

This routine has an FCALL entry point, CS. CS must be referenced by an .EXTN statement.

DIM

Purpose: To compute the positive difference of two single precision real numbers, R and S.

Calling Sequence:

```
JSR    @DI.
FORTRAN ADDRESS of R
FORTRAN ADDRESS of S
```

(If R-S ≤ Ø the result is zero; otherwise, the result is the difference R-S. The result is placed on the number stack.)

JSR @XD. is equivalent to JSR @DI.

Supporting Routines: none;.FRGØ, FFLD1, FCLT1, FSB1 , FRLD1, .NSP .

Subroutine Size: One page zero location and 32 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are lost.

DI. and XD. must be referenced by an .EXTD statement.

EXP

Purpose:

To compute the real value of e^x with x any single precision floating point argument.

Calling Sequence:

(Input argument x on top of number stack.)

```
EXP.
```

(Output result replaces x on top of number stack .)

Supporting Routines :

FSAV, FRET; .RTER, FPLY1, FSGN1, FSB1, FDV1, FML1, FLIP1, FBRK1, FRLD1, NSP .

Subroutine Size and Timing:

One page zero location and 160 octal locations of normally relocatable memory are required.

Typical execution times are 15 ms for the NOVA with software multiply/divide and 2.9 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

If x is the input argument, the routine performs the following calculation:

$$e^x = x * \log_e 2 = 2^{(I+F)}$$

where I and F are the integral and fractional portions of the power whose base is 2. The argument x of e^x must be selected so that $I < 175_8$.

If either underflow or overflow occurs, an error message is typed on the TTY printer, and zero or the greatest possible real value replaces x on the stack.

In the case of very large I values, where $I \geq n*2^{16}$, an error message is output by FLFX1 (which is called by FBRK1).

EXP. must be referenced by an .EXTN statement.

This routine has an FCALL entry point, EXPO . EXPO must be referenced by an .EXTN statement.

EXPC

Purpose: To calculate the value $e^x - 1$ with x a single precision real number.

Calling Sequence: (Input argument x on top of number stack.)

```
          JSR    @EXPC
```

(Result replaces input on number stack.)

Supporting Routines: none; FML1, FLIP1, FPLY1, FSB1, FDV1,.NR1, FRLD1, SP, NSP .

Subroutine Size: One page zero location and 100 octal locations of normally relocatable memory are required.

Notes to User: Original accumulator contents and state of carry upon entry to routine are lost.

x must be selected such that $0 \leq Z < 1/2$ where $Z = x * \log_{10} e$.

No error message is given when x is selected to yield a value of Z outside the acceptable range.

EXPC must be referenced by an . EXTD statement.

FAD1, FSB1

Purpose: To add (subtract) two single precision real numbers.

Calling Sequences:

FAD1

(The sum of the top, OP1, and next-to-top, OP2, numbers on the number stack is computed; OP1 is popped and the sum replaces OP2.)

FSB1

(The top number on the stack, OP1, is subtracted from the next-to-top number, OP2; OP1 is popped, and the value OP2-OP1 replaces OP2.)

Supporting Routines: MPY, DVD; SP, FLSP, .NDSP, .RTES .

Subroutine Size and Timing: 17 octal page zero locations and 754 octal locations of normally relocatable memory are required.

Typical execution time on the NOVA with software multiply/divide is 1.0 ms if inputs to FAD1 have like signs or the input subtrahend to FSB1 is negative. Otherwise, the typical execution time is 1.1 ms. On the SUPERNOVA with hardware fixed point multiply/divide, typical execution times are 250 µs and 275 µs with the same qualifications given above on the inputs to these subroutines. Each of these times includes the time necessary to perform 1 floating store and 2 floating load operations.

Notes to User: Original states of accumulators, carry are lost. FFLD1, FFST1, FXFL1, FLFX1, FSGN1, FML1, FDV1, FNEG1, FCLE1, FCLT1, FCGE1, FCGT1, and FCEQ1 also have entry points in the single precision floating point module.

An error message is generated upon overflow or underflow of the result. Results are normalized before being placed on the number stack. FAD1 and FSB1 must be referenced by an .EXTN statement.

FCLT1, FCLE1, FCEQ1, FCGE1, FCGT1

Purpose:

To compare the size and sign of two single precision real numbers, and set the carry bit to a one if the specified condition is true. Conditions which may be examined are as follows:

OP2 < OP1 -- FCLT1
OP2 ≤ OP1 -- FCLE1
OP2 = OP1 -- FCEQ1
OP2 ≥ OP1 -- FCGE1
OP2 > OP1 -- FCGT1

where OP1 is the top number on the number stack (i.e., the most recently loaded number) and OP2 is the next-to-top number on the stack (the next most recently loaded).

Calling Sequence:

(The two numbers to be compared are loaded on the number stack.)

FCLT1 (or FCLE1, FCEQ1, etc.)

(Carry is set to a one if the comparison yields an affirmative result, otherwise carry is set to a zero. Both compared numbers are popped from the stack.)

Supporting Routines:

MPY, DVD; .RTES, .NDSP, SP, FLSP .

Subroutine Size:

17 octal page zero locations and 754 octal locations of normally relocatable memory are required.

Notes to User:

Original states of accumulators, carry are lost.

FFLD1, FFST1, FXFL1, FLFX1, FSGN1, FAD1, FSB1, FML1, FDV1, FNEG1 also have entry points in the single precision floating point module.

No error messages are generated.

FCLT1 (FCLE1, etc.) must be referenced by an .EXTN statement

FFLD1, FFST1

Purpose:

To unpack and load a single precision real number onto the number stack (FFLD1). To pack and store a single precision real number from the number stack into a FORTRAN ADDRESS (FFST1).

Calling Sequences:

```
FFLD1
FORTRAN ADDRESS of packed number
```

(The number is unpacked and loaded on the number stack.)

```
FFST1
FORTRAN ADDRESS of destination
```

(The number stack is popped, and the popped number is packed and stored at the specified FORTRAN ADDRESS, with rounding.)

Supporting Routines:

MPY, DVD; .RTES, .NDSP, SP, FLSP .

Subroutine Size and Timing:

17 octal page zero locations and 754 octal locations of normally relocatable memory are required.

Typical execution times are 250μs for FFLD1 and FFST1 on the NOVA, and 50μs for FFLD1 and FFST1 on the SUPERNOVA.

Notes to User:

Original states of accumulators and carry are lost. FXFL1, FCLT1, FLFX1, FSGN1, FAD1, FSB1, FML1, FDV1, FNEG1, FCLE1, FCGE1, FCGT1, and FCEQ1 all have entry points in the single precision floating point module. JSR @DB. E is equivalent to FFLD1.

No error message is given if an attempt is made to store more numbers than exist on the number stack. A stack overflow message is generated whenever an attempt is made to load onto an already filled number stack. The most significant bit of the fourth byte of the word to be stored is checked. If set, the third byte is incremented before the floating store is accomplished.

FFLD1, FFST1 (Continued)

An error message is generated whenever a truncation of significant exponent digits occurs as the result of packing an unpacked number.

FFLD1 and FFST1 must be referenced by an .EXTN statement. DB.E must be referenced by an .EXTD statement.

FML1, FDV1

Purpose:

To multiply (divide) two single precision real numbers.

Calling Sequences:

```
FML1
```

(The product of the top, OP1, and next-to-top, OP2, numbers on the number stack is computed; OP1 is popped, and the product replaces OP2.)

```
FDV1
```

(The quotient of the next-to-top, OP2, and top, OP1, numbers on the number stack is computed; OP1 is popped, and OP2/OP1 replaces OP2.)

Supporting Routines:

MPY, DVD; .RTES, .NDSP, SP, FLSP .

Subroutine Size and Timing:

17 octal page zero locations and 754 octal locations of normally relocatable memory are required.

Typical execution times are 2.1 ms for FML1 and 2.5 ms for FDV1 on the NOVA with software multiply/divide. Typical execution times are 320 µs for FML1 and 340 µs for FDV1 on the SUPERNOVA with hardware multiply/divide. Each of these times includes the time necessary to perform 1 floating store and 2 floating load operations.

Notes to User:

Original states of accumulators, carry are lost.

FFLD1, FFST1, FXFL1, FLFX1, FSGN1, FAD1, FSB1, FCGE1, FNEG1, FCLE1, FCLT1, FCGT1, and FCEQ1 also have entry points in the single precision floating point module.

An error message is generated upon underflow or overflow. Results are normalized before being placed on the number stack. FML1 and FDV1 must be referenced by an .EXTN statement.

FNEG1

Purpose: To change the sign of any real number at the top of the number stack.

Calling Sequence: FNEG1

(The sign of the number on the top of the number stack is changed.)

Supporting Routines : MPY, DVD; .RTES, SP, FLSP , .NDSP .

Subroutine Size: 17 octal page zero locations and 754 locations of normally relocatable memory are required.

Notes to User: The contents of ACØ,AC1 and the original state of carry are preserved.

FFLD1, FFST1, FXFL1, FLFX1, FML1, FDV1, FSGN1, FAD1, FSB1, FCLE1, FCLT1, FCGE1, FCGT1 and FCEQ1 also have entry points in the single precision floating point module.

No error messages are generated.

FNEG1 must be referenced by an .EXTN statement.

FPWER

Purpose: To raise a non-negative single precision real base to a single precision real power.

Calling Sequence: (The real power is loaded onto the number stack, and the real base is placed just below the power on the stack.)

FPWR1

(The real power is removed from the number stack, and the result replaces the base at the top of the stack.)

Supporting Routines: none; FLIP1, .RTES, FRLD1, FML1, ALOG., EXP, NSP, SP

Subroutine Size and Timing: One page zero location and 53 octal locations of normally relocatable memory.

Typical execution times are 31 ms for the NOVA with software multiply/divide and 4.9 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost.

This routine generates an error message upon receipt of a negative base argument, and returns the negative base as a result; error messages generated upon underflow or overflow are given by the supporting routines.

FPWR1 must be referenced by an .EXTN statement.

FSGN1

Purpose: To examine the sign of a single precision real number.

Calling Sequence: (The number which is to be examined is at the top of the number stack)

FSGN1

(ACØ is returned with -1, Ø, or 1 corresponding to a negative, zero, or positive state of the examined number. The examined number is popped from the stack.)

Supporting Routines: MPY, DVD; .RTES, .NDSP, SP, FLSP .

Subroutine Size: 17 octal page zero locations and 754 octal locations of normally relocatable memory are required.

Notes to User: Original states of accumulators, carry are lost.

FFLD1, FFST1, FXFL1, FLFX1, FAD1, FSB1, FML1, FDV1, FNEG1, FCLT1, FCLE1, FCEQ1, FCGE1, and FCGT1 also have entry points in the single precision floating point module.

No error messages are generated.

FSGN1 must be referenced by an .EXTN statement.

PLY1

Purpose: To compute a polynomial function P(x) where x is a single precision real argument.

Calling Sequence: (Input argument x on top of the number stack).
(ACØ contains the starting address of LIST+1. See "Notes to User".)

FPLY1

(Output result replaces x on top of the number stack.)

Supporting Routines : none; FRST1, FML1, FAD1, FRLD1, NSP, SP .

Subroutine Size: One page zero location and (34 octal + 4 * order of polynomial) locations of normally relocatable memory are required for this routine and its accompanying order-and-coefficients list.

Notes to User: P(x) is of the form $P(x) = C_0 + C_1X^1 + C_2X^2 + \ldots C_nX^n$

where $C_0 \ldots C_n$ are real coefficients and all powers of X are positive integers.

The structure of the order-and-coefficients list is as follows:

LIST: Single precision fixed point value expressing the order of the polynomial.
LIST+1: Real coefficient C_n in unpacked format.
.
.
.
LIST+5: Real coefficient C_{n-1} in unpacked format.
.
.
.
LIST + 4(n-1) + 1: Real coefficient C_0 in unpacked format.
.
.
.

FPLY1 must be referenced by an .EXTN statement,

RATN1

Purpose: To calculate the arctangent of a quotient of two single precision real arguments loaded onto the number stack.

Calling Sequence: (The argument denominator, OP1, is at the top of the number stack. The argument numerator, OP2, is at the frame following OP1 on the number stack.)

RATN1

(Argument OP1 is removed from the number stack and the arctangent of OP2/OP1 replaces the input argument OP2 on the number stack.

Supporting Routines: none; ATAN., FDV1, FRLD1, FSB1, SP, NSP.

Subroutine Size: One page zero location and 35 octal locations of normally relocatable memory.

Notes to User: The original contents of accumulators and carry are lost upon exit.

RATN1 must be referenced by an .EXTN statement.

SIGN

Purpose: To transfer the sign of one single precision real number to another single precision real number.

Calling Sequences:

```
JSR    @SI.N
FORTRAN ADDRESS of R1
FORTRAN ADDRESS of R2
```

(The sign of R2 is transferred to R1 which is then stored on the number stack.)

```
FCALL
SIGN
Integer 3
FORTRAN ADDRESS of Result
FORTRAN ADDRESS of R1
FORTRAN ADDRESS of R2
```

(The sign of R2 is transferred to R1 which is then stored at the FORTRAN ADDRESS of the result.)

Supporting Routines : FRET, FSAV; FFLD1, FFST1, .FARG, NSP.

Subroutine Size: One page zero location and 33 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit. No error messages are generated.

SI.N must be referenced by an .EXTD statement.
SIGN must be referenced by an .EXTN statement.

SIN

Purpose:

To compute the real sine of an argument x expressed as a single precision real number.

Calling Sequence:

(Input argument x is placed on the top of the number stack)

SIN.

(Output result replaces x on the number stack)

Supporting Routines :

FSAV, FRET; FNEG1, FML1, FSB1, FBRK1, FPLY1, FDV1, FRLD1, NSP.

Subroutine Size and Timing:

Two page zero locations and 156 octal locations of normally relocatable memory are required.

Typical execution times are 16 ms for the NOVA and 2.0 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

The single precision real cosine function has an alternate entry point in this routine.

In the case of large arguments of the form $2n\pi + \theta$, $-\pi \leq \theta \leq \pi$, when n becomes a very large integer, significant digits in the result will be lost.

SIN. must be referenced by an .EXTN statement.

SINH

Purpose: To compute the hyperbolic sine of a single precision real number.

Calling Sequence:

```
(Input argument on number stack.)

            JSR    @.SHIN

(Result is left on the number stack.)
```

```
            JSR    @SI.H
            FORTRAN ADDRESS of argument

(Result is left on the number stack.)
```

Supporting Routines: FRET, FSAV; NSP, EXP, EXPC, FDV1, FRLD1, FLIP1, FCLT1, FML1, FSB1, FFLD1, .FARG .

Subroutine Size: Two page zero locations and 66 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit. No error messages are generated.

.SHIN and SI.H must be referenced by an .EXTD statement.

.SHIN and SI.H have FCALL entry points, .SHIN and SNH. .SHIN and SNH must be referenced by an .EXTN statement.

SQRT

Purpose: To compute the single precision real square root of any non-negative single precision real argument x.

Calling Sequence: (Input argument x on top of number stack.)

```
SQRT.
```

(Output result replaces x on top of number stack.)

Supporting Routines : FSAV, FRET; .RTER, FAD1, FDV1, FLIP1, FRLD1, NSP .

Subroutine Size and Timing: One page zero location and 142 octal locations of normally relocatable memory are required.

Typical execution times are 13 ms for the NOVA with software mutliply/divide and 1.7 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: If the input argument is negative, an error message is output and the square root of the absolute value of the argument is computed.

SQRT. must be referenced by an .EXTN statement.

Original contents of accumulators and carry are restored upon exit from this routine.

This routine has the FCALL entry point SQR .

TAN

Purpose: To compute the single precision real tangent of x, any single precision real argument.

Calling Sequence: (Input argument x on top of number stack.)

TAN.

(Output result replaces x on top of number stack.)

Supporting Routines : FSAV, FRET; FNEG1, FML1, FSB1, FBRK1, FLIP1, FPLY1, FDV1, FRLD1, NSP.

Subroutine Size and Timing: One page zero location and 116 octal locations of normally relocatable memory are required.

Typical execution times are 19 ms for the NOVA with software multiply/divide and 2.4 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: TAN. must be referenced by an .EXTN statement. Original contents of accumulators and carry are restored upon exit from this routine.

This routine has an FCALL entry point, TN. TN must be referenced by an .EXTN statement.

TANH

Purpose:	To calculate the hyperbolic tangent of a single precision real number.
Calling Sequence:	JSR @TA.H FORTRAN ADDRESS of the argument (The result is loaded onto the top of the number stack.)
Supporting Routines:	FSAV, FRET; .FARG, FRLD1, FRST, FSB1, FAD1, FML1, FDV1, FFLD1, FCLE1, FLIP1, EXP., FNEG1, NSP.
Subroutine Size:	One page zero location and 126 octal locations of normally relocatable memory.
Notes to User:	Original contents of accumulators and carry are restored upon exit. Error messages, if issued, will originate from the supporting routines. TA.H must be referenced by an .EXTD statement. This routine has an FCALL entry point, TNH. TNH must be referenced by an .EXTN statement.

DOUBLE PRECISION FLOATING POINT ROUTINES

DATAN, DATAN2

Purpose:

To calculate the arctangent of a double precision real number (DATA.) or the arctangent of a quotient of two double precision real numbers (DA. N2).

Calling Sequences:

(The single argument whose arctangent is to be calculated is loaded onto the number stack.)

```
DATA.
```

(The result replaces the input argument on the number stack.)

```
JSR    @DA. N2
FORTRAN ADDRESS of argument dividend
FORTRAN ADDRESS of argument divisor
```

(The arctangent of the quotient of the input argument is loaded onto the number stack.)

Supporting Routines:

FSAV, FRET; FDV2, FSB2, FFLD2, FPLY2, FCLT2, FRLD2, FLIP2, .FARG, NSP , FML2, FAD2 .

Subroutine Size and Timing:

Two page zero locations and 301 octal locations of normally relocatable memory.

Typical execution times for DA. N2 are as follows:
120 ms for the NOVA with software multiply/divide, and
14 ms for the SUPERNOVA with hardware fixed point multiply/divide. Typical execution times required for DATA. are as follows: 74 ms for the NOVA with software multiply/divide and 15 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

The sign of the result is the same as the sign of the single input argument or argument quotient.

Original contents of accumulators and carry are restored.

DATAN, DATAN2 (Continued)

JSR @XA.N2 and JSR @DA.A2 are each equivalent to JSR @DA.N2. Likewise, XAAN. is equivalent to DATA.

DATA. and XAAN. must be referenced by an .EXTN statement. DA.A2, XA.N2 and DA.N2 must be referenced by an .EXTD statement.

DCOS , DSIN

Purpose: To calculate the sine (DSIN.) or cosine (DCOS.) of a double precision real number.

Calling Sequence: (The input argument is loaded onto the number stack.)

DCOS. (or DSIN.)

(The result replaces the input argument on the number stack.)

Supporting Routines: FSAV, FRET; FPLY2, FBRK2, FML2, FDV2, FLIP2, FSB2, FRLD2, NSP .

Subroutine Size and Timing: Two page zero locations and 161 octal locations of normally relocatable memory.

Typical execution times for DCOS. are as follows: 86 ms for the NOVA with software multiply/divide and 12 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Typical execution times for DSIN. are 90 ms for the NOVA with software multiply/divide and 11 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User : Original contents of accumulators and carry are restored upon exit.

XCS. is equivalent to DCOS. and XSN. is equivalent to DSIN.

DCOS., DSIN., XCS., and XSN. must be referenced by an .EXTN statement.

DEXP

Purpose:

To calculate the value e^x with x any double precision real number.

Calling Sequence:

(The input argument is loaded onto the number stack.)

XEP.

(The result replaces the input argument on the number stack.)

Supporting Routines:

FSAV, FRET; .RTER, FSGN2, FRLD2, FSB2, FML2, FDV2, FLIP2, FPLY2, FRST2, FAD2, NSP, FBRK2 .

Subroutine Size and Timing:

One page zero location and 232 octal locations of normally relocatable memory. Typical execution times are 76 ms for the NOVA with software multiply/divide and 11 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

Original contents of accumulators and carry are restored upon exit.

An error message is issued upon overflow or underflow and either the largest possible value or zero is returned as a result.

DEXP. is equivalent to XEP.

DEXP. and XEP. must be referenced by an .EXTN statement.

This routine has an FCALL entry point, DEXP . DEXP must be referenced by an .EXTN statement.

DEXPC

Purpose: To calculate the value e^x-1 with x a double precision real number.

Calling Sequence: (The input argument is loaded onto the number stack.)

```
JSR    @DEXPC
```

(The result replaces the input argument on the number stack.)

Supporting Routines : none; FRLD2, FML2, FLIP2, FPLY2, FSB2, FDV2, .NR1, NSP, SP .

Subroutine Size: One page zero location and 137 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost.

Any error messages will be generated by the supporting routines .

The range of values for input arguments to this routine is restricted such that $\emptyset \leq \log_{10}e * x < 1/2$.

DEXPC must be referenced by an .EXTD statement.

DLOG, DLOG10

Purpose:

To calculate either the natural logarithm or the logarithm to the base 10 of a double precision real number.

Calling Sequences:

(The argument whose natural logarithm is to be calculated is loaded onto the number stack.)

```
        DLOG.
```

(The result is loaded onto the top of the number stack.)

```
JSR   @DL.GØ (or @XA.GØ)
```

FORTRAN ADDRESS of argument whose base 10 logarithm is to be calculated.

(The result is loaded onto the number stack.)

Supporting Routines:

FSAV, FRET; .RTER, .FARG, FFLD2, FML2, FCLT2, FLIP2, FSB2, FDV2, FAD2, FRLD2, FPLY2, FXFL2, NSP

Subroutine Size and Timing:

Two page zero locations and 275 octal locations of normally relocatable memory. Typical execution times for the natural logarithm function are as follows: 99 ms for the NOVA with software multiply/divide, and 13 ms for the SUPERNOVA with hardware fixed point multiply/divide. Typical execution times for the base 10 logarithm function are 103 ms for the NOVA with software multiply/divide and 14 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

If the input argument is negative an error message is issued, the argument is forced positive and the logarithm is then calculated.

Upon receipt of a zero input argument the largest possible negative number will be returned.

DLOG, DLOG10 (Continued)

DLOG. and XAOG. are equivalent; DL.GØ and XA.GØ are equivalent.

Original contents of accumulators and carry are restored upon exit.

DLOG. and XAOG. must be referenced by an .EXTN statement.
DL.GØ and XA.GØ must be referenced by an .EXTD statement.

DMAX1, DMIN1

Purpose: To select the smallest (DMIN1) or largest (DMAX1) member from a set of double precision real numbers, expressing the selection as a double precision real number.

Calling Sequences:

```
JSR   @DM.X1     (or DM.N1)
N  (an integer constant specifying the number of members
    in the set.)
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
        .
        .
        .
FORTRAN ADDRESS of DRN

(The largest or smallest member of the set is placed on
the number stack.)
```

```
FCALL
DMAX1     (or DMIN1)
N+1     (N is an integer constant specifying the number of
          of members in the set.)
FORTRAN ADDRESS of RESULT
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
        .
        .
        .
FORTRAN ADDRESS of DRN

(The largest or smallest member of the set is placed at the
FORTRAN ADDRESS of the result.)
```

Supporting Routines: FSAV, FRET; .FARG, FFST2, FFLD2, FCLT2 .

Subroutine Size: Two page zero locations and 72 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit from the routine.

DMAX1, DMIN1 (Continued)

No error messages are generated.

JSR @XA.X1 is equivalent to JSR @DM.X1 , and
JSR XA.N1 is equivalent to JSR @DM.N1 .

DMAX1 and DMIN1 must be referenced by an .EXTN statement. DM.X1, DM.N1, XA.X1, and XA.N1 must be referenced by an .EXTD statement.

DMOD

Purpose: To fetch the modulus of two double precision real numbers (i.e., the remainder of their quotient.)

Calling Sequence:

```
JSR    @DM.D
FORTRAN ADDRESS of DR1           ;DIVIDEND
FORTRAN ADDRESS of DR2           ;DIVISOR
```

(Result is placed on the top of the number stack.)

JSR @XA.D is equivalent to JSR @DM.D .

Supporting Routines : none; FFLD2, FDV2, FML2, .FRGØ, NSP, SP .

Subroutine Size: One page zero location and 127 octal locations of normally relocatable memory.

Notes to User: If the quotient DR1/DR2 causes overflow or underflow, an error message will be output by FDV2 and no meaningful result will be returned.

DM.D and XA.D must be referenced by an .EXTD statement.

DPWER

Purpose: To raise a non-negative double precision real number to a double precision real power.

Calling Sequence: (The real power is loaded onto the number stack, and the real base is placed just below the power on the stack.)

FPWR2

(The real power is removed from the stack, and the result replaces the base at the top of the stack.)

Supporting Routines : none; FLIP2, .RTES, .FFLD2, FML2, FRLD2, DLOG., DEXP., NSP, SP .

Subroutine Size: and Timing: One page zero location and 55 octal locations of normally relocatable memory.

Typical execution times are 180 ms for the NOVA with software fixed point multiply/divide and 24 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost.

This routine generates an error message and returns the base as the result upon receipt of a negative base argument; error messages generated upon underflow or overflow are given by the supporting routines.

FPWR2 must be referenced by an .EXTN statement.

DSIGN

Purpose:

To transfer the sign of one double precision real number to another double precision real number.

Calling Sequences:

```
JSR    @DS.GN
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
```

(The sign of DR2 is transferred to DR1 which is then stored on the number stack.)

```
FCALL
DSIGN
Integer 3
FORTRAN ADDRESS of Result
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
```

(The sign of DR2 is transferred to DR1 which is then stored at the FORTRAN ADDRESS of the result.)

Supporting Routines :

FSAV, FRET; FFLD2, FFST2, .FARG, NSP .

Subroutine Size:

One page zero location and 33 octal locations of normally relocatable memory.

Notes to User:

Accumulators, carry are restored upon exit.

No error messages are generated.

JSR @XS.N is equivalent to JSR @DS.GN .

DSIGN must be referenced by an .EXTN statement. DS.GN and XS.N must be referenced by an .EXTD statement.

DSINH

Purpose: To calculate the hyperbolic sine of a double precision real number.

Calling Sequence:

```
(The argument is placed on the number stack.)

          JSR    @.DSHIN

(The result replaces the argument on the number stack.)
```

```
          JSR    @DS.NH
          FORTRAN ADDRESS of argument

(The result is placed on the number stack.)
```

Supporting Routines: FRET, FSAV; NSP, DEXP, DEXPC, FDV2, FRLD2, FLIP2, FCLT2, FML2, FSB2, .FARG, FFLD2 .

Subroutine Size: Two page zero locations and 72 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit. No error messages are generated. XS.H is equivalent to .DSHIN .

DS.NH, .DSHI and XS; H must be referenced by an .EXTD statement.

.DSHIN and DS.NH have FCALL entry points, DSINH and DSNH . DSINH and DSNH must be referenced by an .EXTN statement.

DSQRT

Purpose: To calculate the square root of a double precision real number.

Calling Sequence: (The input argument is loaded onto the number stack.)

DSQR.

(The result replaces the input argument on the number stack.)

Supporting Routines: FSAV, FRET; FRLD2, FML2, FAD2, FLIP2, FDV2, FPLY2, .RTER, NSP.

Subroutine Size and Timing: One page zero location and 127 octal locations of normally relocatable memory.

Typical execution times are 82 ms for the NOVA with software multiply/divide and 8.1 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are restored upon exit.

An error message is output by this routine upon receipt of a negative argument. In this case, the argument is forced positive and the square root of the positive quantity is calculated.

XSRT. is equivalent to DSQR .

DSQR. and XSRT. must be referenced by an .EXTN statement.

This routine has an FCALL entry point, DSQR . DSQR must be referenced by an .EXTN statement.

DTAN

Purpose: To calculate the tangent of a double precision real number.

Calling Sequence: (The input argument is loaded onto the number stack.)

DTAN.

(The result replaces the input argument on the number stack.)

Supporting Routines: FSAV, FRET; FML2, FDV2, FRLD2, FSB2, FBRK2, FPLY2, FLIP2, NSP .

Subroutine Size: and Timing: One page zero location and 165 octal locations of normally relocatable memory.

Typical execution times are 84 ms for the NOVA with software multiply/divide and 9.3 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are restored upon exit.

XTN. is equivalent to DTAN.

DTAN. and XTN. must be referenced by an .EXTN statement.

This routine has an FCALL entry point, DTN .
DTN must be referenced by an .EXTN statement.

DTANH

Purpose:

To calculate the hyperbolic tangent of a double precision real number.

Calling Sequence:

JSR @DT.NH
FORTRAN ADDRESS of argument

(The result is loaded onto the number stack.)

Supporting Routines:

FSAV, FRET; FAD2, FML2, FDV2, FFLD2, DEXP., FSB2, FLIP2, FCLT2, FRST2, FRLD2, DEXPC, .FARG, NSP .

Subroutine Size and Timing:

One page zero location and 136 octal locations of normally relocatable memory.

Typical execution times are 185 ms for the NOVA with software multiply/divide, and 21.5 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User :

JSR @XT.H is equivalent to JSR @DT.NH .

Original contents of accumulators and carry are restored upon exit from this subroutine. If any error messages are generated they will be generated by the supporting routines.

DT.NH and XT.H must be referenced by an .EXTD statement.

This routine has an FCALL entry point, DTNH . DTNH must be referenced by an .EXTN statement.

FAD2, FSB2

Purpose:

To add (subtract) two double precision real numbers.

Calling Sequences:

FAD2

(The sum of the top, OP1, and next-to-top, OP2, numbers on the number stack is computed; OP1 is popped and the sum replaces OP2.

FSB2

(The top number on the stack, OP1, is subtracted from the next-to-top number, OP2; OP1 is popped, and the value OP2-OP1 replaces OP2.)

Supporting Routines:

MPY, DVD; SP, FLSP, .NDSP, .SVØ, .RTES .

Subroutine Size and Timing:

17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Typical execution time on the NOVA with software multiply/divide is 1.3 ms if inputs to FAD2 have like signs or the input subtrahend to FSB2 is negative. Otherwise, the typical execution time is 1.4 ms. On the SUPERNOVA with hardware fixed point multiply/divide, typical execution times are 300 µs and 400 µs with the same qualifications given above on the inputs to these subroutines. Each of these times includes the time necessary to perform 1 floating store and 2 floating load operations.

Notes to User:

Original states of accumulators, carry are lost. FFLD2, FFST2, FXFL2, FLFX2, FSGN2, FML2, FDV2, FNEG2, FCLT2, FCLE2, FCEQ2, FCGE2, and FCGT2 also have entry points in the double precision floating point module.

An error message is generated upon underflow or overflow of result. Results are normalized before being placed on the number stack. FAD2 and FSB2 must be referenced by an .EXTN statement.

FCLT2, FCLE2, FCEQ2, FCGE2, FCGT2

Purpose:

To compare the size and sign of two double precision real numbers, and set the carry bit to a one if the specified condition is true. Conditions which may be examined are as follows:

OP2 < OP1 -- FCLT2
OP2 ≤ OP1 -- FCLE2
OP2 = OP1 -- FCEQ2
OP2 ≥ OP1 -- FCGE2
OP2 > OP1 -- FCGT2

where OP1 is the top number on the number stack (i.e., the most recently loaded number) and OP2 is the next-to-top number on the stack.

Calling Sequence:

(The two numbers to be compared are loaded on the number stack.)

FCLT2 (or FCLE2, etc.)

(Carry is set to a one if the comparison yields an affirmative result, otherwise carry is set to a zero. Both compared numbers are popped from the stack.)

Supporting Routines:

MPY, DVD; SP, FLSP, .NDSP, .SVØ, .RTES .

Subroutine Size:

17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Notes to User:

Original states of accumulators, carry are lost.

FFLD2, FFST2, FXFL2, FLFX2, FSGN2, FAD2, FSB2, FML2, FDV2, and FNEG2 have entry points in the double precision floating point module.

No error messages are generated.

FCLT2 (FCLE2, etc) must be referenced by an .EXTN statement.

FFLD2, FFST2

Purpose:

To unpack and load a double precision real number onto the number stack (FFLD2).

To pack and store a double precision real number from the number stack into a FORTRAN ADDRESS (FFST2).

Calling Sequences:

FFLD2
FORTRAN ADDRESS of packed number

(The number is unpacked and loaded onto the number stack.)

FFST2
FORTRAN ADDRESS of destination

(The number stack is popped, and the popped number is packed and stored at the specified FORTRAN ADDRESS, with rounding.)

Supporting Routines:

MPY, DVD; SP, FLSP, .NDSP, .SVØ, .RTES .

Subroutine Size and Timing:

17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Typical execution times are 500 µs for FFLD2 or FFST2 on the NOVA, and 100 µs for FFLD2 or FFST on the SUPERNOVA.

Notes to User:

Original states of accumulators and carry are lost.

FXFL2, FLFX2, FSGN2, FAD2, FSB2, FML2, FDV2, FNEG2, FCLT2, FCLE2, FCEQ2, FCGE2, and FCGT2 also have entry points in the double precision floating point module. JSR @XD. E is equivalent to FFLD2.

No error message is given if an attempt is made to store more numbers than exist on the number stack. A stack overflow message is generated whenever an attempt is made to load onto a filled number stack. The most significant bit of the eighth byte of the word to be stored is checked. If set, the 7th byte is incremented before the floating store is accomplished.

FFLD2, FFST2 (Continued)

An error message is generated whenever a truncation of significant exponent digits occurs as the result of packing an unpacked number.

FFLD2 and FFST2 must be referenced by an .EXTN statement. XD.E must be referenced by an .EXTD statement.

FML2, FDV2

Purpose: To multiply (divide) two double precision real numbers.

Calling Sequences:

```
FML2

(The product of the top, OP1, and next-to-top, OP2,
numbers on the number stack is computed; OP1 is
popped, and the product replaces OP2 on the stack.)
```

```
FDV2

(The quotient of the next-to-top, OP2, and top, OP1,
numbers on the number stack is computed; OP1 is
popped, and OP2/OP1 replaces OP2.
```

Supporting Routines: MPY, DVD; SP, FLSP, .NDSP, .SVØ, .RTES .

Subroutine Size and Timing: 17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Typical execution times are 5.7 ms for FML2 and 11.4 ms for FDV2 on the NOVA with software multiply/divide. Typical execution times on the SUPERNOVA with hardware fixed point multiply/divide are 700 µs for FML2 and 1.28 ms for FDV2. Each of these times includes the time necessary to perform 1 floating store and 2 floating load operations.

Notes to User: Original states of accumulators, carry are lost.

FFLD2, FFST2, FXFL2, FSGN2, FAD2, FSB2, FNEG2, FLFX1, FCLT2, FCLE2, FCEQ2, FCGE2, and FCGT2 also have entry points in the double precision floating point module.

An error message is generated upon underflow or overflow of result.

Results are normalized before being placed on the number stack.

FML2 and FDV2 must be referenced by an .EXTN statement.

FNEG2

Purpose: To change the sign of any real number at the top of the number stack.

Calling Sequence: FNEG2

(The sign of the number on top of the number stack is changed)

Supporting Routines: MPY, DVD; SP, FLSP, .NDSP, .SVØ, .RTES .

Subroutine Size: 17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Notes to User: The contents of ACØ, AC1, and the original state of carry are preserved.

FFLD2, FFST2, FXFL2, FLFX2, FSGN2, FAD2, FSB2, FML2, FDV2, FCGE2, FCLT2, FCLE2, FCEQ2, and FCGT2 also have entry points in the double precision floating point module.

No error messages are generated.

FNEG2 must be referenced by an .EXTN statement.

FPLY2

Purpose: To compute a polynomial function P(x) where x is a double precision real number.

Calling Sequence: (The input argument x is at the top of the number stack. ACØ contains the starting address of LIST + 1. See Notes to User.)

FPLY2

(The output result replaces the input argument on the number stack.)

Supporting Routines: none; FRLD2, FML2, FAD2, FRST2, NSP, SP .

Subroutine Size: One page zero location and (34 octal + 6 * order of polynomial) locations of normally relocatable memory are required for this routine and its accompanying order-and-coefficients list.

Notes to User: P(x) is of the form $P(x) = C_0 + C_1x^1 + C_2x^2 \ldots + C_nx^n$

where C_0 through C_n are double precision real coefficients and all powers of x are positive integers.

The structure of the order-and-coefficients list is as follows:

LIST: Single precision fixed point number expressing the order of the polynomial
⋮
LIST+1: Double precision Real coefficient C_n in unpacked form
⋮
LIST+7: Double precision Real Coefficient C_{n-1} in unpacked form
⋮
LIST+6(m-n+1)*+1
Double precision Real coefficient C_0 in unpacked form.

Original states of accumulators and carry are lost. FPLY2 must be referenced by an .EXTN statement.

* where m is the order of the polynomial and x is the number of the term.

FSGN2

Purpose: To examine the sign of a double precision real number.

Calling Sequence: (The number which is to be examined is at the top of the number stack)

FSGN2

(ACØ is returned with -1, 0, or 1 corresponding to a negative, zero, or positive state of the examined number. The examined number is popped from the stack.)

Supporting Routines: MPY, DVD; SP, FLSP, .NDSP, .SVØ, .RTES .

Subroutine Size: 17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Notes to User: Original states of accumulators, carry are lost.

FFLD2, FFST2, FXFL2, FAD2, FSB2, FML2, FDV2, FNEG2, FCLT2, FCLE2, FCEQ2, FCGE2, and FCGT2 also have entry points in the double precision floating point module.

No error messages are generated.

FSGN2 must be referenced by an .EXTN statement.

RATN2

Purpose: To calculate the arctangent of a quotient of two double precision real arguments loaded onto the number stack.

Calling Sequence: (The argument denominator, OP1, is at the top of the number stack. The argument numerator, OP2, is at the frame following OP1 on the number stack.)

RATN2

(Argument OP1 is removed from the number stack, and the arctangent of OP1/OP2 replaces the input argument OP2 on the number stack.)

Supporting Routines: none; DATA., FDV2, FRLD2, FSB2, NSP, SP .

Subroutine Size: One page zero location and 37 octal locations of normally relocatable memory.

Notes to User: The original contents of accumulators and carry are lost upon exit from this routine.

RATN2 must be referenced by an .EXTN statement.

SINGLE PRECISION COMPLEX ROUTINES

CABS

Purpose: To obtain the absolute value of a single precision complex number.

Calling Sequence:

```
JSR    @CA.S
FORTRAN ADDRESS of argument
```

(The absolute value of the argument is loaded onto the number stack.)

Supporting Routines: none; .FRGØ, FFLD1, .RCABS, SP .

Subroutine Size: One page zero location and 17 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost upon exit from this routine. Stack overflow messages may be issued by the supporting routines.

The result obtained by this routine is a real number, thus occupying only one 6-word frame on the number stack.

CA.S must be referenced by an .EXTD statement.

CAD1, CSB1

Purpose: To add (CAD1) the topmost two single precision complex numbers on the number stack or to subtract (CSB1) the top single precision complex number on the number stack from the next-to-top single precision complex number on the stack.

Calling Sequence: (The two arguments are loaded onto the number stack.)

CAD1 (CSB1)

(The top argument is removed from the stack, and the sum or difference replaces the second argument.)

Supporting Routines: none; FAD1, FRST1, FRLD1, .NR2, SP, NSP .

Subroutine Size: Two page zero locations and 22 octal locations of normally relocatable memory.

Notes to User: Original accumulator contents and carry are not restored upon exit from this routine. Error messages are generated by supporting routines upon overflow or underflow.

CAD1 and CSB1 must be referenced by an .EXTN statement.

CCEQ1

Purpose: To compare two single precision complex numbers for identity.

Calling Sequence: (The two complex numbers to be examined are the topmost numbers on the number stack.)

CCEQ1

(Carry is set to a one if they are equal, otherwise, it is set to a zero. The two complex numbers are removed from the number stack.)

Supporting Routines: none; FCEQ1, .NR2, .NR1, .NR3, FRST1, FRLD1, NSP, SP.

Subroutine Size: One page zero location and 21 octal locations of normally relocatable memory.

Notes to User: The original contents of accumulators, carry are lost.

CCEQ1 must be referenced by an .EXTN statement.

CCOS

Purpose: To compute the cosine of a single precision complex number.

Calling Sequence: (A complex argument is loaded on the top of the number stack.)

CCOS.

(The cosine of the argument is expressed as a single precision complex number and replaces the input argument on the number stack.)

Supporting Routines: none; COS., SIN., .SHIN, EXP., .NR2, FAD1, FML1, FRST1, FRLD1, FLIP1 , SP, NSP .

Subroutine Size and Timing: One page zero location and 43 octal locations of normally relocatable memory are occupied by this routine.

Typical execution times are 111 ms for the NOVA with software multiply/divide and 17 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost.

Error messages will be generated by EXP. or FML1 upon overflow or underflow.

CCOS. must be referenced by an .EXTN statement.

CDV1

Purpose: To divide one single precision complex number by another.

Calling Sequence: (The argument divisor is placed on the top of the number stack, and the dividend is immediately below the divisor.)

CDV1

(The divisor is removed from the number stack and the quotient replaces the dividend on the number stack.)

Supporting Routines: none; FRLD1, FCLE1, FDV1, .NR2, CML1, FLIP1, FML1, .NR3, FAD1, FRST1, .NR1, SP, NSP .

Subroutine Size: One page zero location and 75 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are not restored upon exit from the routine. Error messages are generated by supporting routines upon overflow or underflow.

CDV1 must be referenced by an .EXTN statement.

CEXP

Purpose: To compute the value e^c with c any single precision complex number.

Calling Sequence: (The complex argument is loaded onto the number stack.)

CEXP.

(The complex result replaces the argument on the number stack.)

Supporting Routines : none; EXP., COS., SIN., .NR2, FLIP1, FRLD1, FRST1, FML1, SP, NSP.

Subroutine Size and Timing: One page zero location and 24 octal locations of normally relocatable memory are required by this routine.

Typical execution times are 47 ms for the NOVA with software multiply/divide and 7.8 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost.

Error messages are generated upon underflow or overflow.

CEXP. must be referenced by an .EXTN statement.

CFST1

Purpose: To pack and store a single precision complex number located on the number stack.

Calling Sequence: (The argument is at the top of the number stack.)

CFST1
FORTRAN ADDRESS to receive the argument

(The top two six word frames are removed from the stack.)

Supporting Routines: none; .FRGØ, FFST1, SP.

Subroutine Size: One page zero location and 17 octal locations of normally relocatable memory.

Notes to User: Original accumulator contents and state of carry are both lost upon exit from this routine. No error messages are generated.

The argument on the number stack occupies two sequential six-word frames, with the top frame containing the imaginary portion of the argument. After the argument has been packed and stored at the indicated FORTRAN ADDRESS it occupies only four sequential locations, with the first pair of words containing the real portion of the argument.

CFST1 must be referenced by an .EXTN statement.

CLIP1, CLIP2

Purpose: To swap positions of the two topmost complex numbers on the number stack (whether single or double precision or both.)

Calling Sequence: (Two complex numbers are on the top of the number stack.)

CLIP1 (or CLIP2)

(The positions of the two complex numbers are interchanged.)

Supporting Routines: none; .NR1, .NR2, .NR3, .FLIP, SP, NSP .

Subroutine Size: One page zero location and 15 octal locations of normally relocatable memory.

Notes to User: CLIP1 and CLIP2 are equivalent.

Original contents of accumulators and carry are lost.

CLIP1 and CLIP2 must be referenced by an .EXTN statement.

CLOAD

Purpose: To unpack and load a single precision complex number onto the number stack.

Calling Sequence: CFLD1
FORTRAN ADDRESS of the packed real and imaginary portions of the complex number

(The complex number is unpacked and loaded onto the number stack.)

Supporting Routines: none; .FRGØ, FFLD1, SP.

Subroutine Size: One page zero location and 16 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost. Upon number stack overflow an error message will be issued by FFLD1.

The FORTRAN ADDRESS following the call points to four sequential stack locations containing first the real portion (in single precision packed format) and then the imaginary portion (also in single precision packed format) of the argument. The argument is then unpacked and loaded onto the number stack in two sequential six word frames. The top frame contains the imaginary portion and the next-to-top frame contains the real portion of the argument.

CFLD1 must be referenced by an .EXTN statement.

CLOG

Purpose: To compute the natural logarithm of a single precision complex number.

Calling Sequence: (The single precision argument is loaded onto the number stack.)

CLOG.

(The result replaces the input argument on the number stack.)

Supporting Routines: none;.NR2, RATN1, FRLD1, FRST1, ALOG., CLIP1, RCABS, SP, NSP.

Subroutine Size and Timing: One page zero location and 21 octal locations of normally relocatable memory are required by this routine.

Typical execution times are 60 ms for the NOVA with software multiply/divide and 8.3 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost upon exit from this routine. Error messages are generated upon underflow or overflow by the supporting routines.

CLOG. must be referenced by an .EXTN statement.

CML1

Purpose: To multiply two single precision complex numbers by one another.

Calling Sequence: (The two arguments are loaded onto the number stack.)

CML1

(The topmost argument is removed, and the product replaces the second argument on the number stack.)

Supporting Routines: none; FML1, FRLD1,FAD1,FRST1,FSB1,.NR1,.NR2,.NR3,SP,NSP.

Subroutine Size: One page zero location and 47 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost upon exit from this routine. Error messages generated upon underflow or overflow are issued by supporting routines.

CML1 must be referenced by an .EXTN statement.

CNEG1, CNEG2

Purpose: To negate the real and imaginary parts of any complex number.

Calling Sequence: (The complex number to be negated is at the top of the number stack.)

CNEG1 (or CNEG2)

(The negated complex number replaces the input argument on the number stack.)

Supporting Routine: NSP .

Subroutine Size: One page zero location and 6 locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost.
No error messages are generated.

CNEG1 and CNEG2 must be referenced by an .EXTN statement.

CONJG

Purpose: To produce the conjugate of any complex number.

Calling Sequence: (The complex number whose conjugate is to be obtained is loaded onto the number stack.)

CONJ.

(The sign of the imaginary portion of the input argument is complemented, replacing the original value.)

Supporting Routine: NSP

Subroutine Size: One page zero location and five locations of normally relocatable memory.

Notes to User: The original contents of carry and accumulators AC3 and AC2 are lost; no error messages are generated.

AC3 contains FSP upon exit from this routine.

This routine accepts both single and double precision complex numbers as input arguments.

XCNJ. and DCON. are each equivalent to CONJ.

CONJ., XCNJ. and DCON. must be referenced by an .EXTN statement.

CPWR1

Purpose: To raise a single precision complex number to a single precision complex power.

Calling Sequence: (The complex power is on the top of the stack, the complex base is immediately below it.)

CPWR1

(The power and base are removed from the stack; the complex result is loaded on the stack.)

Supporting Routines: none; CLOG., CEXP., CML1, .NR3, .NR2, FRLD1, FRST1, SP .

Subroutine Size: One page zero location and 20 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are lost. Error messages can arise from the supporting routines.

CPWR1 must be referenced by an .EXTN statement.

CSIN

Purpose: To compute the sine of a single precision complex number.

Calling Sequence: (The single precision complex argument is input on the top of the number stack.)

CSIN.

(The result replaces the input argument on the number stack.)

Supporting Routines: none; COS., SIN., .SHIN, EXP., .NR2, FAD1, FML1,FRST1, FRLD1, FLIP1, SP, NSP .

Subroutine Size and Timing: One page zero location and 42 octal locations of normally relocatable memory are required by this routine.

Typical execution times are 100 ms on the NOVA with software multiply/divide and 15 ms on the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Accumulators, carry are lost. Any error messages generated will be issued by the supporting routines.

CSIN. must be referenced by an .EXTN statement.

CSQRT

Purpose: To compute the square root of a single precision complex number.

Calling Sequence: (The complex argument is placed at the top of the number stack.)

CSQR.

(The result replaces the input argument on the number stack.)

Supporting Routines: none; .NR2, FRLD1, RATN1, FLIP1, CLIP1, FML1, SQRT., RCABS, SIN., COS.,. FRST1, SP, NSP ·

Subroutine Size and Timing: One page zero location and 47 octal locations of normally relocatable memory are required by this routine.

Typical execution times are 89 ms on the NOVA with software multiply/divide and 12 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost.

CSQR. must be referenced by an .EXTN statement.

RCABS

Purpose:

To obtain the absolute value of a single precision complex number located on the number stack.

Calling Sequence:

(The complex argument is loaded onto the number stack.)

JSR @RCABS

(The complex argument is removed from the number stack, and the absolute value of the argument is loaded there.)

Supporting Routines :

none; .NR1, .NR2, SQRT,, FLIP1, FML1, FDV1, FAD1, FRLD1, FCLE1, SP, NSP .

Subroutine Size:

One page zero location and 42 octal locations of normally relocatable memory.

Notes to User:

Original contents of accumulators and carry are lost. Error messages are generated by supporting routines.

RCABS must be referenced by an .EXTD statement.

REAL, AIMAG

Purpose: To fetch either the real or the imaginary parts of a single precision complex number.

Calling Sequences:

```
JSR    @RE.L
FORTRAN ADDRESS of complex number
```

(The real portion of the compl ex number is loaded onto the number stack.)

```
JSR    @AI.AG
FORTRAN ADDRESS of complex number
```

(The imaginary portion of the complex number is loaded onto the number stack.)

Supporting Routines: none; .FRGØ, FFLD1, SP

Subroutine Size: Two page zero locations and 22 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are lost upon exit. No error messages are generated.

RE.L and AI.AG must be referenced by an .EXTD statement.

DOUBLE PRECISION COMPLEX ROUTINES

CAD2, CSB2

Purpose: To add (CAD2) the topmost two double precision complex numbers on the number stack or to subtract (CSB2) the top double precision complex number on the stack from the next-to-top double precision number on the stack.

Calling Sequence: (The two arguments are loaded onto the number stack.)

CAD2 (or CSB2)

(The top argument is removed from the stack, and the sum or difference replaces the second argument.)

Supporting Routines: none; FAD2, FRST2, FRLD2, .NR2, SP, NSP.

Subroutine Size: Two page zero locations and 22 octal locations of normally relocatable memory.

Notes to User: Original accumulator contents and carry are not restored upon exit from the routine. Error messages are generated by supporting routines upon overflow or underflow.

CAD2 and CSB2 must be referenced by an .EXTN statement.

CCEQ2

Purpose: To compare two double precision complex numbers for identity.

Calling Sequence: (The two complex numbers to be examined are the topmost numbers on the number stack.)

CCEQ2

(Carry is set to a one if they are equal, otherwise, it is set to zero. The two complex numbers are removed from the number stack.)

Supporting Routines: none; FCEQ2, .NR2,.NR1,.NR3, FRST2, FRLD2, SP, NSP .

Subroutine Size: One page zero location and 21 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are lost.

CCEQ2 must be referenced by an .EXTN statement.

CDV2

Purpose: To divide one double precision complex number by another.

Calling Sequence: (The argument divisor is placed on the top of the number stack, and the dividend is immediately below the divisor.)

CDV2

(The divisor is removed from the number stack and the quotient replaces the dividend on the number stack.)

Supporting Routines : none; FRLD2, FCLE2, FDV2, CML2, FLIP2, FML2, .NR3, .NR2, .NR1, FAD2, FRST2, SP, NSP.

Subroutine Size: One page zero location and 101 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are not restored upon exit from this routine. Error messages are generated by supporting routines upon overflow or underflow.

CDV2 must be referenced by an .EXTN statement.

CFST2

Purpose: To pack and store a double precision complex number located on the number stack.

Calling Sequence: (The argument is at the top of the number stack.)

CFST2
FORTRAN ADDRESS to receive the argument

(The top two six-word frames are removed from the stack.)

Supporting Routines: none; .FRGØ, FFST2, SP .

Subroutine Size: One page zero location and 20 octal locations of normally relocatable memory.

Notes to User: Original accumulator contents and state of carry are both lost upon exit from this routine. No error messages are generated.

The argument on the number stack occupies two sequential six-word frames, with the top frame containing the imaginary portion of the argument. After the argument has been packed and stored at the indicated FORTRAN ADDRESS it occupies eight sequential locations, with the first group of four words containing the real portion of the argument.

CFST2 must be referenced by an .EXTN statement.

CML2

Purpose: To multiply two double precision complex numbers by one another.

Calling Sequence: (The two arguments are loaded onto the number stack.)

CML2

(The topmost argument is removed, and the product replaces the second argument on the number stack.)

Supporting Routines : none; DCLO., DCEX., .NR2, .NR3, FRLD2, FRST2, SP, FML2, FAD2, FSB2 .

Subroutine Size: One page zero location and 47 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost upon exit from this routine. Error messages are generated upon overflow or underflow by supporting routines.

CML2 must be referenced by an .EXTN statement.

Double Precision Complex

CPWR2

Purpose: To raise a double precision complex number to a double precision complex power.

Calling Sequence: (The complex power is on the top of the stack, the complex base is immediately below it.)

CPWR2

(The power and base are removed from the stack; the complex result is loaded on the stack.)

Supporting Routines: none; DCLO., DCEX., CML2, .NR2, .NR3, FRLD2, SP FRST2 .

Subroutine Size: One page zero location and 20 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are lost upon exit from this routine. Error messages can arise from the supporting routines.

CPWR2 must be referenced by an .EXTN statement.

DCABS

Purpose: To obtain the absolute value of a double precision complex number.

Calling Sequence:

```
JSR    @DC.BS
FORTRAN ADDRESS of argument
```

(The absolute value of the argument is loaded onto the number stack.)

Supporting Routines: none; .FRGØ, FFLD2, RDCABS, SP .

Subroutine Size: One page zero location and 22 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost upon exit from this routine. Stack overflow messages may be issued by the supporting routines.

The result obtained by this routine is a real number, thus occupying only one 6-word frame on the number stack.

JSR   @XC.S is equivalent to JSR   @DC.BS.

XC.S and DC.BS must be referenced by an .EXTD statement.

DCCOS

Purpose: To compute the cosine of a double precision complex number.

Calling Sequence: (The double precision complex argument is placed on the top of the number stack.)

DCCO.

(The result replaces the argument on the number stack.)

Supporting Routines : none; DCOS., DSIN., .DSHIN, DEXP., .NR2, FAD2, FML2, FRST2, FLIP2, SP, NSP , FRLD2 .

Subroutine Size and Timing: One page zero location and 45 octal locations of normally relocatable memory are occupied by this routine.

Typical execution times are 580 ms for the NOVA with software multiply/divide and 89 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Accumulators and carry are lost upon exit from this routine. Any error messages generated will be issued by the supporting routines.

XCOS. is equivalent to DCCO.

XCOS. and DCCO. must be referenced by an .EXTN statement.

DCEXP

Purpose: To compute the value e^{c} with c any double precision complex number.

Calling Sequence: (The complex argument is loaded onto the number stack.)

DCEX.

(The complex result replaces the argument on the number stack.)

Supporting Routines : none; DEXP., DCOS., DSIN., .NR2, FLIP2, FRLD2, FRST2, FML2, SP, NSP.

Subroutine Size and Timing: One page zero location and 24 octal locations of normally relocatable memory are required by this routine.

Typical execution times are 295 ms for the NOVA with software multiply/divide and 36.5 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original contents of accumulators and carry are lost upon exit from this routine. Error messages are generated upon underflow or overflow.

XCXP. is equivalent to DCEX.

DCEX. and XCXP. must be referenced by an .EXTN statement.

DCLOD

Purpose: To unpack and load a double precision complex number onto the number stack.

Calling Sequence:

CFLD2
FORTRAN ADDRESS of the packed real and imaginary portions of the complex number.

(The complex number is unpacked and loaded onto the number stack.)

Supporting Routines: none; .FRGØ, FFLD2, SP.

Subroutine Size: One page zero location and 21 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost. Upon number stack overflow an error message will be issued by FFLD2.

The FORTRAN ADDRESS following the call points to eight sequential stack locations containing first the real portion (in double precision packed format) and then the imaginary portion (also in double precision packed format) of the argument. The argument is then unpacked and loaded onto the number stack in two sequential six word frames. The top frame contains the imaginary portion and the next-to-top frame contains the real portion of the argument.

CFLD2 must be referenced by an .EXTN statement.

DCSIN

Purpose: To compute the sine of an angle expressed as a double precision complex number.

Calling Sequence: (The double precision complex argument is input on the top of the number stack.)

DCSI.

(The result replaces the argument on the number stack.)

Supporting Routines : none; DCOS., .DSHIN, DS.NH, DEXP., .NR2, FAD2, FML2, FRST2, FRLD2, FLIP2, SP, NSP.

Subroutine Size and Timing: One page zero location and 44 octal locations of normally relocatable memory are occupied by this routine.

Typical execution times are 585 ms for the NOVA with software multiply/divide and 90 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Accumulators and carry are lost. Any error messages generated will be issued by the supporting routines.

XCIN. is equivalent to DCSI.

XCIN. and DCSI. must be referenced by an .EXTN statement.

DCSQR

Purpose: To compute the square root of a double precision complex number.

Calling Sequence: (The complex argument is placed at the top of the number stack.)

DCSQ.

(The result replaces the input argument on the number stack.)

Supporting Routines: none; .NR2, FRLD2, RATN2, FLIP2, CLIP2, FML2, DSQR., DSIN., DCOS., FRST2, SP, NSP, RDCABS .

Subroutine Size and Timing: One page zero location and 51 octal locations of normally relocatable memory are occupied by this routine.

Typical execution times are 655 ms for the NOVA with software multiply/divide and 70.5 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User: Original states of accumulators and carr y are lost.

XCQR. is equivalent to DCSQ.

DCSQ. and XCQR. must be referenced by an .EXTN statement.

DDCLO

Purpose:

To compute the natural logarithm of a double precision complex number.

Calling Sequence:

(The double precision argument is loaded on the number stack.)

XCOG.

(The result replaces the input argument on the number stack.)

Supporting Routines:

none; .NR2, RATN2, FRLD2, FRST2, DLOG., CLIP2, RDCABS, SP, NSP.

Subroutine Size and Timing:

One page zero location and 21 octal locations of normally relocatable memory are required.

Typical execution times are 430 ms for the NOVA with software multiply/divide and 45.5 ms for the SUPERNOVA with hardware fixed point multiply/divide.

Notes to User:

Original contents of accumulators and carry are lost upon exit from this routine. Error messages are generated upon underflow or overflow by the supporting routines.

DCLO. is equivalent to XCOG.

DCLO. and XCOG. must be referenced by an .EXTN statement.

DREAL , DAIMG

Purpose: To fetch the real or complex parts of a double precision complex number.

Calling Sequences:

```
JSR    @DR.AL
FORTRAN ADDRESS of complex number

(The real portion of the complex number is loaded on the
number stack.)
```

```
JSR    @DA.MG
FORTRAN ADDRESS of complex number

(The imaginary portion of the complex number is loaded
on the number stack.)
```

Supporting Routines: none; .FRGØ, FFLD2, SP

Subroutine Size: Two page zero locations and 24 octal locations of normally relocatable memory.

Notes to User: JSR @XR.L is equivalent to JSR @DR.AL, and
JSR @XA.AG is equivalent to JSR @DA.MG .

Original contents of accumulators, carry are lost; no error messages are generated.

DR.AL, DA.MG, XR.L, and XA.AG must be referenced by an .EXTD statement.

RDCABS

Purpose: To obtain the absolute value of a double precision complex number located on the number stack.

Calling Sequence: (The complex argument is loaded onto the number stack.)

```
JSR    @RDCABS
```

(The complex argument is removed from the number stack, and the absolute value of the argument is loaded there.)

Supporting Routines : none; .NR1, .NR2, DSQR.,FLIP2, FML2, FDV2, FAD2, FRLD2, FCLE2, SP, NSP .

Subroutine Size: One page zero location and 44 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost. Error messages are generated by supporting routines.

The result obtained by this routine is a real number, thus occupying only one 6-word frame on the number stack.

RDCABS must be referenced by an .EXTD statement.

MIXED MODE ROUTINES

AMAXØ, AMINØ

Purpose: To select the smallest (AMINØ) or the largest (AMAXØ) member from a set of integers, expressing the selection as a single precision real value.

Calling Sequences:

JSR @AM.NØ (or @AM.XØ)
N (an integer constant specifying the number of members in the set)
FORTRAN ADDRESS of I_0
FORTRAN ADDRESS of I_1
.
.
.
FORTRAN ADDRESS of I_{N-1}

(The result is expressed as a single precision real on the top of the number stack.)

FCALL
AMAXØ (or AMINØ)
N+1 (where N is an integer constant specifying the number of members in the set.)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of I_0
FORTRAN ADDRESS of I_1
.
.
.
FORTRAN ADDRESS of I_{N-1}

(The result is expressed as a single precision real number stored at the FORTRAN ADDRESS of the result given in the calling sequence.)

Supporting Routines: FSAV, FRET; .FARG, FXFL1, FFST1 .

Subroutine Size: Two page zero locations and 76 octal locations of normally relocatable memory.

AMAXØ, AMINØ (cont'd)

Notes to User:

Accumulators, carry are restored upon exit from the routine. No error messages are generated.

AM. NØ and AM. XØ must be referenced by an .EXTD statement. AMAXØ, AMINØ must be referenced by an .EXTN statement. AM.NØ and AM. XØ have FCALL entry points AMNØ and AMXØ respectively.

BREAK

Purpose: To separate a single precision real number x into its integral and fractional components.

Calling Sequence: (Input argument x on top of the number stack.)

```
FBRK1
```

(Output fractional result replaces x on the number stack; integral result is placed in ACØ.)

Supporting Routines : none; FXFL1, FLFX1, FSB1, FRLD1, NSP, SP .

Subroutine Size: One page zero location and 15 octal locations of normally relocatable memory are required.

Notes to User: Original contents of all accumulators and the state of carry upon exit from this routine are lost.

Upon exit from this routine ACØ is loaded with the integral portion of the argument, expressed as a single precision fixed point number. The fractional portion of the argument is expressed as a single precision real value.

FLFX1 will generate an error message whenever the integral portion of the argument exceeds the range $\pm(2^{15}-1)$.

FBRK1 must be referenced by an .EXTN statement.

CMPLX

Purpose: To construct a single precision complex number from two single precision real numbers.

Calling Sequences:

```
JSR    @CM. LX
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion

(A complex number is formed and loaded on the number stack.)
```

```
FCALL
CMPLX
Integer 3
FORTRAN ADDRESS of result
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion

(A complex number is formed and is then stored at the FORTRAN
ADDRESS of the result.)
```

Supporting Routines : FSAV, FRET;   FFST1, .FARG, FFLD1

Subroutine Size: One page zero location and 40 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are restored upon exit. No error messages are generated.

CM. LX must be referenced by an .EXTD statement.
CMPLX must be referenced by an .EXTN statement.

CRCX1

Purpose: To convert a packed single precision real number R to a single precision complex number of the form R + Øi.

Calling Sequence: CRCX1
FORTRAN ADDRESS of single precision real argument R

(The real argument R becomes expanded to a complex number of the form R + Øi, which is loaded on the number stack.)

Supporting Routines: none; .FRGØ, FFLD1, FRLD1, SP.

Subroutine Size: One page zero location and 22 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost; any error messages issued will be generated by the supporting routines.

CRCX1 must be referenced by an .EXTN statement.

CRCX2

Purpose: To convert a packed double precision real number D to a double precision complex number of the form D + Øi .

Calling Sequence:

CRCX2
FORTRAN ADDRESS of double precision real argument D

(The real argument D becomes expanded to a complex number of the form D + Øi , which is loaded onto the number stack.)

Supporting Routines: none; .FRGØ, FFLD2, FRLD2, SP .

Subroutine Size: One page zero location and 24 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost; an error message will be generated by FFLD2 or FRLD2 upon number stack overflow.

CRCX2 must be referenced by an .EXTN statement.

CXFL1

Purpose: To convert an integer I to a single precision complex number of the form I + Øi.

Calling Sequence: CXFL1
FORTRAN ADDRESS of the integer argument I

(The integer argument I becomes expanded to a complex number of the form I + Øi which is loaded onto the number stack.)

Supporting Routines : none; FXFL1, FRLD1, .FRGØ, SP .

Subroutine Size: One page zero location and 21 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost; an error message will be issued by a supporting routine upon stack overflow.

CXFL1 must be referenced by an .EXTN statement.

CXFL2

Purpose: To convert an integer I to a double precision complex number of the form I + Øi.

Calling Sequence:

```
CXFL2
FORTRAN ADDRESS of the integer argument I
```

(The integer argument I becomes expanded to a double precision complex number I + Øi which is loaded onto the number stack.)

Supporting Routines: none; FXFL2, FRLD2, .FRGØ, SP .

Subroutine Size: One page zero location and 23 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost; an error message will be issued by a supporting routine upon stack overflow.

CXFL2 must be referenced by an .EXTN statement.

DBREAK

Purpose: To separate a double precision real number into its integral and fractional components.

Calling Sequence: (The input argument is loaded onto the number stack.)

FBRK2

(The integral portion is expressed as a single precision fixed point value which is loaded into ACØ. The fractional component replaces the input argument on the number stack.)

Supporting Routines: none; FXFL2, FLFX2, FRLD2, FSB2, NSP, SP .

Subroutine Size: One page zero location and 15 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost.

Error messages generated will be issued by the supporting routines.

FBRK2 must be referenced by an .EXTN statement.

DCMPLX

Purpose: To construct a double precision complex number from two double precision real numbers.

Calling Sequences:

```
JSR    @DC.PX
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion
```

(A complex number is formed and loaded on the number stack.)

```
FCALL
DCMPL
Integer 3
FORTRAN ADDRESS of result
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion
```

(A complex number is formed and is then stored at the FORTRAN ADDRESS of the result.)

Supporting Routines : FSAV, FRET; .FARG, FFLD2, FFST2, SP .

Subroutine Size: One page zero location and 43 octal locations of normally relocatable memory.

Notes to User: JSR @XC.LX is equivalent to JSR @DC.PX.

Original contents of accumulators and carry are restored upon exit. No error messages are generated.

DCMPL must be referenced by an .EXTN statement.
XC.LX and DC.PX must be referenced by an .EXTD statement.

DIPWR

Purpose: To raise a double precision real base to an integer power.

Calling Sequence:

FIPR2
FORTRAN ADDRESS of the integer power
FORTRAN ADDRESS of the real base

(The real result is loaded onto the number stack.)

Supporting Routines: FSAV, FRET; .FARG, FLIP2, FDV2, FML2, FRST2, FRLD2, FFLD2, NSP.

Subroutine Size and Timing: One page zero location and 52 octal locations of normally relocatable memory.

Typical execution times on the NOVA with software multiply/divide are 5 ms where the integer power, I, equals Ø, or 7ms + 5 ms * (I-1) where I ≥ 1. Where I ≤ -1, the execution time equals 17.5 ms +(-I-1)* 5.5 ms.

Typical execution times on the SUPERNOVA with hardware multiply/divide are 425 μs where I = Ø, and 1 ms + (I-1) *.6 ms where I ≥ 1. Execution times where I ≤ -1 are correspondingly larger.

Each of the above execution times includes the time required for one floating store operation.

Notes to User: Original contents of accumulators and carry are restored upon exit from this routine; error messages upon overflow or underflow will be issued by supporting routines.

FIPR2 must be referenced by an .EXTN statement.

This routine has an FCALL entry point, DIPWR . DIPWR must be referenced by an .EXTN statement.

FLIP

Purpose: To interchange number stack positions of two single or double precision real numbers.

Calling Sequences:

(ACØ and AC1 point to two six-word frames -- usually on the number stack, but they could be anywhere -- which are to be swapped.)

```
        JSR    @.FLIP
```

(The contents of the two frames are now exchanged.)

(The two topmost frames on the number stack contain variables which will be interchanged.)

```
        FLIP1, FLIP2
```

(The two topmost variables on the number stack are swapped.)

FLIP1 and FLIP2 are equivalent.

Supporting Routines: SP, NSP .

Subroutine Size: Two page zero locations and 26 octal locations of normally relocatable memory are required.

Notes to User: Original accumulator contents and state of carry are lost.

Six word frames on the number stack may contain either single or double precision real variables.

.FLIP must be referenced by an .EXTD statement.
FLIP1 and FLIP2 must be referenced by an .EXTN statement.

FRLD1, FRLD2

Purpose: To load any unpacked real number onto the number stack.

Calling Sequences:

(ACØ contains the address of the sign word of a single precision real number which is to be loaded onto the number stack.)

```
FRLD1
```

(The single precision real number is loaded onto the top of the number stack.)

(ACØ contains the address of the sign word of a double precision real number which is to be loaded onto the number stack.)

```
FRLD2
```

(The double precision real number is loaded onto the top of the number stack.)

Supporting Routines: none; .RTER, NSP, SP, .NDSP .

Subroutine Size: Two page zero locations and 32 octal locations of normally relocatable memory.

Notes to User: Original contents of accumulators, carry are lost.

A fatal error message is generated upon stack overflow.

An unpacked single precision real number in normally relocatable memory occupies four sequential memory locations. Nonetheless, this four word block is expanded to 6 words (by padding the two least significant mantissa words with zeroes) so that all frame lengths on the number stack will be of equal size.

FRLD1 and FRLD2 must be referenced by an .EXTN statement.

Mixed
Mode

FRST1, FRST2

Purpose:

To store any real number located on the number stack at a specified address, in unpacked form.

Calling Sequences:

(Address to receive sign word of single precision real number is contained in ACØ.)

FRST1

(The single precision number is stored, unpacked, at the four sequential addresses specified, and the number is popped from the number stack.)

(Address to receive sign word of double precision real number is contained in ACØ)

FRST2

(The double precision number is stored, unpacked, at the six sequential addresses specified, and the number is popped from the number stack.)

Supporting Routines:

SP, NSP .

Subroutine Size:

Two page zero locations and 25 octal locations of normally relocatable memory are required.

Notes to User:

Original states of accumulators, carry are lost.

No error messages are generated.

No check is made by this routine to ascertain whether or not there really is a number on the number stack.

FRST1 and FRST2 must be referenced by an .EXTN statement.

FXFL1, FLFX1

Purpose:

To convert a fixed point number to an unpacked single precision real, and load it on the number stack (FXFL1).

To pop a single precision real number from the number stack, convert it to fixed point format, and store it at a specified FORTRAN ADDRESS (FLFX1).

Calling Sequences:

```
FXFL1
FORTRAN ADDRESS of fixed point number I
```

(I is converted to a single precision floating point number which is loaded on the number stack.)

```
FLFX1
FORTRAN ADDRESS to receive I
```

(The top member of the number stack is converted to a fixed point number I, the stack is popped, and I is stored at the FORTRAN ADDRESS following the call.)

Supporting Routines :

MPY, DVD; .RTES, SP, FLSP .

Subroutine Size:

17 octal page zero locations and 754 octal locations of normally relocatable memory are required.

Notes to User:

Original states of accumulators and carry are lost.

FFLD1, FFST1, FML1, FDV1, FSGN1, FAD1, FSB1, FNEG1, FCLE1, FCLT1, FCGE1, FCGT1, and FCEQ1 also have entry points in the single precision floating point module.

An error message is generated if FXFL1 attempts to load an already filled number stack.

FXFL1, FLFX1 (Continued)

An error message is issued if the input argument to FLFX1 falls outside the range $[-2^{15}+1, +2^{15}-1]$; a signed maximum integer is returned as a result. If the input argument for FLFX1 is in the range $< -1 , +1 >$, zero is returned as a result.

No error message occurs if FLFX1 is called with an empty number stack.

FXFL1 and FLFX1 must be referenced by an .EXTN statement.

JSR @FL.AT is equivalent to FLFX1 and must be referenced by an .EXTD statement.

FXFL2, FLFX2

Purpose:

To convert a fixed point number to an unpacked double precision real, and load it on the number stack (FXFL2).

To pop a double precision real number from the number stack, convert it to fixed point format, and store it at a specified FORTRAN ADDRESS (FLFX2).

Calling Sequences:

FXFL2
FORTRAN ADDRESS of fixed point number I

(I is converted to a double precision floating point number which is loaded on the number stack.)

FLFX2
FORTRAN ADDRESS to receive I

(The top number of the number stack is converted to a fixed point number I, the stack is popped, and I is stored at the FORTRAN ADDRESS following the call.)

Supporting Routines: MPY, DVD; .RTES, SP, FLSP, .NDSP, .SVØ .

Subroutine Size:

17 octal page zero locations and 1233 octal locations of normally relocatable memory are required.

Notes to User:

Original states of accumulators and carry are lost.

FFLD2, FFST2, FSGN2, FAD2, FSB2, FML2, FDV2, FNEG2, FCLT2, FCLE2, FCEQ2, FCGE2 and FCGT2 also have entry points in the double precision floating point module.

An error message is generated if FXFL2 attempts to load an already filled number stack.

FXFL2, FLFX2 (Continued)

Notes to User:

An error message is issued if the input argument of FLFX2 falls outside the range [$-2^{15}-1$, $+2^{15}-1$] ; a signed maximum integer is returned as a result. If the input argument for FLFX2 is in the range < -1, +1 > , zero is returned as a result.

No error message occurs if FLFX2 is called with an empty number stack.

JSR @DF.OT is equivalent to FXFL2.

FXFL2 and FLFX2 must be referenced by an .EXTN statement.

DF.OT must be referenced by an .EXTD statement.

IDINT

Purpose: To truncate a double precision real number and express the result as a fixed point number.

Calling Sequence:

```
JSR     @ID. NT
FORTRAN ADDRESS of location where result is to be stored
FORTRAN ADDRESS of real number DR to be truncated
```

(DR is truncated, converted to a fixed point number, and is stored at the FORTRAN ADDRESS following the call.)

Supporting Routines: FSAV, FRET; FLFX2, FFLD2, .FARG .

Subroutine Size: One page zero location and 11 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit. Error messages will be generated if the truncated real number exceeds $+2^{15}-1$ or is less than $-(2^{15}-1)$.

JSR @XI. is equivalent to JSR @ID. NT .

ID. NT and XI. must be referenced by an .EXTD statement.

This routine has an FCALL entry point, .IDIN .
.IDIN must be referenced by an .EXTN statement.

IFIX

Purpose: To truncate a single precision real number and express it as a fixed point number.

Calling Sequence:

```
JSR    @IF.X
FORTRAN ADDRESS of integer result.
FORTRAN ADDRESS of real value to be truncated
```

Supporting Routines: FSAV, FRET; .FARG, FFLD1, FLFX1, NSP

Subroutine Size: One page zero location and 21 octal locations of normally relocatable memory.

Notes to User: JSR @XI.X is equivalent to JSR @IF.X .

Accumulators, carry are restored upon exit.

IF.X and XI.X must be referenced by an .EXTD statement.

This routine has an FCALL entry point, .IFIX . .IFIX must be referenced by an .EXTN statement.

INT

Purpose: To truncate a single precision real and express the result as the nearest integer.

Calling Sequence:

```
JSR    @IN.
FORTRAN ADDRESS of location where result is to be stored
FORTRAN ADDRESS of real number R to be truncated
```

(R is truncated, converted to a fixed point number and is stored at the FORTRAN ADDRESS following the call.)

Supporting Routines : FSAV, FRET; .FARG, FFLD1, FLFX1 .

Subroutine Size: One page zero location and 11 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit.

If the truncated real is greater than 2^{15}-1 or less than -(2^{15}-1)FLFX1 will generate an error message.

Result = Sign of argument * largest integer ≤ |argument|.

IN. must be referenced by an .EXTD statement.

This routine has an FCALL entry point, .INT .
.INT must be referenced by an .EXTN statement.

MAX1, MIN1

Purpose: To select the smallest (MIN1) or the greatest (MAX1) member from a set of single precision real numbers, expressing the selection as a fixed point number.

Calling Sequence:

JSR @MA.1 (or @MI.1)
N+1 (where N is a fixed point number equal to the number of members in the set being examined.)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of R_1

.
.
.

FORTRAN ADDRESS of R_n

(The result is a fixed point number stored at the FORTRAN ADDRESS of the result given in the calling sequence.)

Supporting Routines: FSAV, FRET; .FARG, FLFX1, FFLD1, FCLT1 .

Subroutine Size: Two page zero locations and 46 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit. An error message is generated if the truncated real number exceeds 2^{15}-1 or if it is less than -(2^{15}-1).

JSR @XA.1 is equivalent to JSR @MA.1, and JSR @XI.1 is equivalent to JSR @MI.1 .

FCALL entry points are MAX1 and MIN1.

MA.1, MI.1, XA.1, and XI.1 must be referenced by an .EXTD statement. MAX1 and MIN1 must be referenced by an .EXTN statement.

.NR1

Purpose: To obtain a pointer to the first frame below the top frame of the number stack.

Calling Sequence: JSR @.NR1

(Pointer is returned in ACØ).

Supporting Routine: NSP .

Subroutine Size: One page zero location and five locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost. No error messages are generated.

AC3 loses FSP upon exit.

A frame is understood to be a block of six consecutive locations on the number stack.

.NR1 must be referenced by an .EXTD statement.

.NR2

Purpose: To obtain a pointer to the second frame below the top frame of the number stack.

Calling Sequence:

```
JSR     @.NR2
(Pointer is returned in ACØ.)
```

Supporting Routine: NSP.

Subroutine Size: One page zero location and five locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost. No error messages are generated.

AC3 loses FSP upon exit.

A frame is understood to be a block of six consecutive locations on the number stack.

.NR2 must be referenced by an .EXTD statement.

.NR3

Purpose: To obtain a pointer to the third frame below the top frame of the number stack.

Calling Sequence: JSR @.NR3

(Pointer is returned in ACØ.)

Supporting Routine: NSP.

Subroutine Size: One page zero location and five locations of normally relocatable memory.

Notes to User: Original contents of accumulators and carry are lost. No error messages are generated.

AC3 loses FSP upon exit.

A frame is understood to be a block of six consecutive locations on the number stack.

.NR3 must be referenced by an .EXTD statement.

RIPWR

Purpose:

To raise a single precision real base to an integer power.

Calling Sequence:

FIPR1
FORTRAN ADDRESS of the integer power
FORTRAN ADDRESS of the real base

(The real result is loaded onto the number stack.)

Supporting Routines :

FSAV, FRET; FLIP1, FDV1, FML1, FRST1, FRLD1, .FARG, FFLD1, NSP .

Subroutine Size and Timing:

One page zero location and 50 octal locations of normally relocatable memory.

Typical execution times on the NOVA with software multiply/divide are 1.45 ms where I = Ø, and 3 ms + (I-1) *1.7 ms where I ⩾ 1. Where I ⩽ -1, NOVA execution times are 5.3 ms + (-I-1) * 1.6 ms .

Typical execution times on the SUPERNOVA with hardware multiply/divide are 360 µs where I = Ø, and 550 µs + (I-1) * 180 µs where I ⩾ 1. Execution times where I < -1 are correspondingly larger.

Each of the above execution times includes the time required for one floating store operation.

Notes to User:

Original contents of accumulators and carry are restored upon exit from this routine.

Error messages will be issued by supporting routines whenever appropriate.

FIPR1 must be referenced by an .EXTN statement.

This routine has an FCALL entry point, RIPWR . RIPWR must be referenced by an .EXTN statement.

STRING/BYTE MANIPULATION ROUTINES

COMP

Purpose: To compare two character strings for identity.

Calling Sequence: (String byte pointers in ACØ and AC1).

```
JSR    @.COMP
```

(Return is to the next sequential address if the strings match, and to one after the next sequential address if they do not match.)

Supporting Routines: FSAV, FRET; .LDBT .

Subroutine Size: One page zero location and 34 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are restored upon exit; no error messages are generated.

Each string must be terminated with a null byte.

The FCALL entry is COMP .

.COMP must be referenced by an .EXTD statement.
COMP must be referenced by an .EXTN statement.

LDBT, STBT

Purpose: To load or store a byte by means of a byte pointer.

Calling Sequence:

```
(ACØ contains byte pointer)

            JSR   @.LDBT

(AC1 contains the byte, right justified)
```

```
(AC1 contains word whose right byte is to be stored.
ACØ contains byte pointer.)

            JSR   @.STBT
```

Supporting Routine: .SVØ .

Subroutine Size: Two page zero locations and 30 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are lost except ACØ; AC3 contains FSP upon exit. No error messages are generated.

Byte pointer is left unchanged upon exit.

.LDBT and .STBT must be referenced by an .EXTD statement.

LOAD, STORE

Purpose: To permit the loading or storing of any accumulator except AC3 from or into any absolute address.

Calling Sequences:

```
JSR    @.LDØ   (.LD1, .LD2)
Any absolute address

(ACØ -- or AC1, AC2 -- is loaded with the contents of the
absolute address.)
```

```
JSR   @.STØ  (.ST1, .ST2)
Any absolute address

(The contents of ACØ -- or AC1, AC2 -- is stored at the
absolute address.)
```

Supporting Routines: None.

Subroutine Size: Six page zero locations and 17 octal locations of normally relocatable memory.

Notes to User: This routine uses QSP for temporary storage, so the existence of at least one Fortran stack frame is required for the operation of this routine.

The value of FSP contained in AC3 prior to the call is restored in AC3 upon exit from the routine.

No error messages are generated upon attempting to reference a non-existent location.

The six above-named entries must be referenced by an .EXTD statement.

MOVE

Purpose: To move all (MOVE) or part of (CMOVE) a byte string.

Calling Sequences:

(ACØ contains the byte pointer to the beginning of the source string. AC1 contains the byte pointer to the beginning of the destination string. The source byte string is terminated by an all zero byte.)

```
        FCALL
        MOVE
```

(AC1 points to the null byte in the destination string.)

(ACØ contains the byte pointer to the beginning of the source string. AC1 contains the byte pointer to the beginning of the destination string. AC2 contains the number of bytes which are to be moved.)

```
        FCALL
        CMOVE
```

(AC1 points to the last byte moved to the destination string.)

Supporting Routines: FSAV, FRET; .LDBT, .STBT .

Subroutine Size: 44 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit from this routine.

No error messages are generated. No check is made by CMOVE to determine if the value in AC2 exceeds the number of bytes in the source string. The original source string remains unaltered in both move operations. Both MOVE and CMOVE must referenced by an .EXTN statement.

MOVEF

Purpose: To move a block of words.

Calling Sequence:

JSR @.MOVE
Word Count
FORTRAN ADDRESS of word block
FORTRAN ADDRESS of word block destination

Supporting Routines: .MAD, QSP, SP

Subroutine Size: One page zero location and 23 octal locations of normally relocatable memory.

Notes to User: Original states of accumulators and carry are lost.

Upon completion of this routine, the word block is found both at its original location and at the destination location.

MVBC

Purpose: To move a byte string.

Calling Sequence: (Upon entry to this routine, accumulators contain the following parameters:

ACØ, the byte pointer to the present byte string;
AC1, the byte pointer to the destination of the string;
AC2, the number of bytes in the string.)

```
JSR   @.MVBC
```

Supporting Routines: FSAV, FRET, FQRET; .LDBT, .STBT .

Subroutine Size: Two page zero locations and 37 octal locations of normally relocatable memory are required for this routine.

Notes to User: Accumulators and carry are restored upon exit from this routine.

Bytes are packed left to right:

BYTE 1		BYTE 2	
Bit 0	Bit 7	Bit 8	Bit 15

Byte pointer structure is as follows:

Memory Address		Byte Selector	
Bit 0	Bit 14	Bit 15	(Ø = Left)
			(1 = Right)

Upon exit from the routine, the byte string is found both at the specified destination and at its original location.

.MVBT has an alternate entry point in this routine. .MVBC must be referenced by an .EXTD statement.

This routine has a FCALL entry point MVBC.

MVBT

Purpose: To move a byte string.

Calling Sequence: (Upon entry to this routine, accumulators contain the following parameters:

ACØ, byte pointer to the present byte string;
AC1, byte pointer to the destination of the string;
AC2, terminal character in the byte string.)

```
JSR    @.MVBT
```

Supporting Routines: FSAV, FRET, FQRET; .LDBT, .STBT .

Subroutine Size: Two page zero locations and 37 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit from this routine.

Upon completion of this routine, the byte string is found both at the specified destination and at its original location.

.MVBC has an alternate entry point in this routine.

.MVBT must be referenced by an .EXTD statement.

This routine has an FCALL entry point, MVBT . MVBT must be referenced by an .EXTN statement.

MVF

Purpose: To move blocks of whole words within core memory.

Calling Sequence: (Beginning address of the word block to be moved is in ACØ; the destination address is in AC1, the number of words in the block is specified as a positive integer in AC2.)

```
JSR    @.MVF
```

Supporting Routines: FSAV, FQRET ; none .

Subroutine Size: One page zero location and 16 octal locations of normally relocatable memory are required.

Notes to User: Accumulators and carry are restored upon exit.

No error messages are generated.

The original word block remains unaltered.

.MVF must be referenced by an .EXTD statement.

This routine has an FCALL entry point, MVF . MVF must be referenced by an .EXTN statement.

MVZ

Purpose: To clear blocks of memory words.

Calling Sequence: (Beginning address of block in AC1, number of words in the block to be zeroed is in ACØ.)

```
JSR    @.MVZ
```

Supporting Routines: FSAV, FQRET ; none .

Subroutine Size: One page zero location and 13 octal locations of normally relocatable memory are required for this routine.

Notes to User: Accumulators and carry are restored upon exit from this routine.

No error messages are generated.

.MVZ must be referenced by an .EXTD statement.

This routine has an FCALL entry point, MVZ . MVZ must be referenced by an .EXTN statement.

POINTERS/DISPLACEMENTS

Dummy Module

Purpose: The dummy module, found only in the RDOS FORTRAN library, defines dummy values for three FORTRAN run time flags so that they will never be listed at load time as being undefined. The three flags are: QTCK, FLSP, and .FLSZ .

Program:

```
          .TITLE DUMMY
          .ENT   QTCK,FLSP,.FLSZ

QTCK  = -1
FLSP  = -1
.FLSZ = -1

          .END
```

.FLSP

Purpose:

To enable .I to determine whether or not real or complex arithmetic is used, so that it may decide whether or not to allocate core space for the number stack.

Program:

```
        .NREL
.FLSP:  FLSP
        .END
```

Notes to User:

.FLSP must be specified by an .EXTN statement.

.FLSP will always be loaded along with the run time initialization program, .I . If real or complex arithmetic is used by the main program, the FPTRS module will have been loaded and resolved, assigning a location to the number stack pointer which is equivalent to FLSP.

.FLSP contains the default value 000377 at load time unless the FPTRS module has been loaded, in which case it will contain the resolved value for FLSP, which is some other ZREL address. .I will examine .FLSP to see whether it contains 377 or other ZREL address, and either allocate space for the number stack or not depending upon the result of this test.

Stack Pointers for Real and Complex Arithmetic (FPTRS module)

Purpose:

To define a page zero pointer, NSP (or FLSP), to the current top of the number stack. This position will also be used by .I at initialization time to determine whether or not arithmetic routines have been used and thus whether the number stack should be allocated.

Program:

```
          .ZREL
FLSP:     Ø
NSP=      FLSP
          .END
```

Notes to User:

NSP and FLSP are synonymous labels for the page zero location containing a pointer to the current top of the number stack. This module will be loaded only if real arithmetic routines are called for by .MAIN.

FLSP is the label of a ZREL location other than 377_8. This label is tested by .I (see .FLSP, "Notes to User") which then either allocates a number stack or not, depending on the result of this test.

FLSP and NSP must both be referenced by an .EXTD statement.

Run Time Page Zero Locations (FPZERO module)

Purpose: These page zero locations are reserved for use by run time routines.

Definitions:

SP - A pointer to the Return Address Stack, which is a stack located after the .I stack, and whose size is determined by .I . Utilized by routines which do not use any of the FSAV family for storage of return addresses for exiting subroutines, and for miscellaneous storage.

.NDSP - Pointer to one greater than the topmost possible location in the number stack.

SUCOM - Start of unlabeled common.

.OVFL - A flag used to indicate whether or not overflow (or underflow) has occurred, and therefore whether error messages should be issued. If all zero, no overflow has occurred; if set to a one, overflow has occurred.

AFSE - Indication of the end (top most memory location) of available run time stack area.

.IOCAT - Pointer to the I/O Channel Assignment Table's starting address.

.SOSW - Flag indicating whether or not the Stand-Alone Operating System has been loaded. If non-zero, SOS was loaded.

.SVØ - Return save for zero level routines like MPY.

QSP - Pointer to FAC2.

Note to User: Each of the above-named locations must be referenced by an .EXTD statement. Under RDOS , TVR is defined to be the starting address of the series of page zero locations.

LINKAGE AND INITIALIZATION ROUTINES

CPYARG, CPYLS

Purpose: To transfer effective addresses of a caller's argument list to its called subroutines's stack.

Calling Sequences:

```
            FCALL
            SUBR
            N               ;N=NUMBER OF ARGUMENTS IN LIST
            FADDR           ;FORTRAN ADDRESSES
            FADDR
              .
              .
SUBR:         .
              .
              .
```

(ACØ contains the number of arguments to be passed.)

```
            JSR    @.CPYA   ;ADDRESS OF CALLER'S
                            ;ARGUMENTS ARE NOW
                            ;ON SUBR STACK.
```

```
            FCALL
            SUBR
            N               ;N=NUMBER OF ARGUMENTS IN LIST
            FADDR
            FADDR
              .
              .
SUBR:         .
              .
              .
            JSR    @.CPYL   ;ADDRESSES OF CALLER'S
                            ;ARGUMENTS ARE NOW ON
                            ;SUBR STACK.
```

Supporting Routines: FSAV, FRET; .MADO .

Subroutine Size: Two page zero locations and 42 octal locations of normally relocatable memory.

CPYARG, CPYLS (Continued)

Notes to User: This routine is more generalized than FARG ; accumulators and carry are preserved upon exit.

CPYLS updates the caller's return address (stored in FRTN) to the next sequential instruction following the caller.

.CPYL and .CPYA must be referenced by an .EXTD statement.

CPYARG has an FCALL entry point, CPYAR . CPYAR must be referenced by an .EXTN statement.

FARG

Purpose: To fetch a called subroutine's argument addresses, when these are stored as FORTRAN ADDRESSES immediately following the caller.

Calling Sequence: (ACØ contains the number of argument addresses to be fetched.)

```
          JSR @.FARG
```

(Caller's argument addresses are stored on current stack. Caller's FRTN is updated.)

Supporting Routine: SP.

Subroutine Size: One page zero location and 34 octal locations of normally relocatable memory are required.

Notes to User: Caller's ACØ, AC1 contents are lost. .FARG must be referenced by an .EXTD statement.
The following example illustrates the use of .FARG:

```
                .ZREL
AL.GØ:          .ALG1Ø-2
                .NREL
.MAIN:          .
                .
                .
.CAL1:          JSR       @AL.GØ   ;THIS IS THE CALLING
                                   ;ROUTINE
                FADDR  of ARGUMENT
                .
                .
                .
                FSAV
                3
.ALG1Ø:         SUBZL     Ø,Ø      ;PUT 1 IN ACØ, SINCE
                                   ;THERE IS ONLY ONE
                                   ;ARGUMENT FOLLOWING
                                   ;THE MAIN CALLER.
.CAL2 :         JSR       @. FARG  ;ARGUMENT ADDRESS IS
                .                  ;STORED ON ALG1Ø'S STACK.
                .
```

FARGØ

Purpose: To calculate the effective address of an argument on the current stack frame (.FRGØ) or the next most current stack frame (.FRG1) given its FORTRAN ADDRESS pointed to by AC2.

Calling Sequences: (FORTRAN ADDRESS is pointed to by AC2.)

```
JSR    @.FRGØ (or .FRG1)
```

(The address is returned in ACØ.)

Supporting Routine: SP.

Subroutine Size: Two page zero locations and 24 octal locations of normally relocatable memory are required.

Notes to User: Original states of accumulators, carry are lost.

This routine avoids the need for reserving stack storage, and is also useful when an argument list is variable in length and contains single word arguments.

.FRGØ and .FRG1 must be referenced by an .EXTD statement.

FQRET

Purpose: To provide return from a called subroutine which neither requires temporary storage nor calls other subroutines.

Calling Sequence:

```
         FSAV
         -1                      ;NO TEMP STORAGE
SUBR:    LDA     Ø, MNE, Ø
          .
          .                      ;NO FURTHER
          .                      ;SUBROUTINE CALLS
         FQRET
```

Supporting Routines: .I; AFSE, .RTEØ .

Subroutine Size: Five page zero locations and 140 octal locations of normally relocatable memory are required.

Notes to User: All subroutines which neither call others nor require temporary storage (i.e., all subroutines lacking stack frames) must use FQRET for return to the caller.

FQRET must be specified in an .EXTN statement.

Caller's accumulators, original state of carry are restored upon exit from the called subroutine.

FCALL, FRCAL, FSAV, and FRET have alternate entry points in this routine.

FRCAL

Purpose:

To call a subroutine whose address is containeo in AC2, and create a stack for this subroutine if needed.

Calling Sequence:

```
                .ZREL
.SUBR:          SUBR
                .
                .
                .
                .NREL
                LDA          2, .SUBR
                FRCAL
                .
                .
                .
                Ø            ;ZERO STACK LENGTH WORD
SUBR:           .
                .
                .
```

Supporting Routines:

.I; AFSE, .RTEØ .

Subroutine Size:

Five page zero locations and 140 octal locations of normally relocatable memory ate required.

Notes to User:

FRCAL creates a new stack for the called routine (if needed) and allocates temporary storage on the new stack if this is required. The stack length word immediately preceding the called routine determines whether or not a stack will be created and whether temporaries on the stack will be allocated. The following summarizes the possible stack length words:

SLW = -1 No stack, no temporaries will be created for the called routine.

SLW = Ø A stack will be created to permit deeper subroutine calls; no temporary storage is allocated on this stack.

SLW = +1 A stack will be created with I temporary storage locations allocated.

Upon entry to SUBR, ACØ AC1 and carry will be the same as the calling program's; AC2 will contain the calling program's FSP,and AC3 will contain the called program's FSP.

FRCAL (Continued)

A fatal error message is generated if insufficient core storage is available for the creation of the called routine's stack.

FCALL, FSAV, FRET, and FQRET have alternate entry points in this routine.

Caller's accumulators (except AC3) and original state of carry will be restored by FRET or FQRET upon return to the next sequential instruction following the call, and AC3 will contain the caller's FSP.

FRCAL must be specified in an .EXTN statement.

FRET

Purpose: To restore a caller's accumulators and state of Carry upon exit from the called subroutine, and return to the next instruction following the caller.

Calling Sequence:

```
        .ZREL
.SUBR:  SUBR-2
        .NREL
        JSR   @.SUBR
NEXT:   MOV   1,1
         .
         .
         .
        FSAV
        5
SUBR:   LDA   Ø, Ø, 2
         .
         .
         .
        FRET          ;RESTORE CALLER'S ACCUMULATOR'S
                      ;RETURN TO NEXT
```

Supporting Routines: .I; AFSE, .RTEØ .

Subroutine Size: Five page zero locations and 140 octal locations of normally relocatable memory are required.

Notes to User: FRET is equivalent to JSR @. FRET . FRET must be referenced by an . EXTN statement, . FRET by an . EXTD statement.

FRET also restores caller FSP, loads it into AC3 before return.

FRCAL, FSAV, FCALL, FQRET have alternate entry points in this routine.

FRGLD

Purpose: To fetch the contents of the FORTRAN ADDRESS pointed to by AC2.

Calling Sequence: (FORTRAN ADDRESS is pointed to by AC2.)

```
JSR    @.FRGLD
```

(Result is returned in ACØ.)

Supporting Routines: none; .FRG1, SP .

Subroutine Size: One page zero location and 10 octal locations of normally relocatable memory are required.

Notes to User: Original states of accumulators and carry are lost.

If the FORTRAN ADDRESS is a stack frame displacement, it is resolved with respect to the next-most-current stack frame, the caller's caller's frame.

.FRGLD must be referenced by an .EXTD statement.

FSAV

Purpose:

To save a caller's accumulators and state of carry upon a subroutine page zero call, create a new stack frame with temporary storage allocated (if needed), and check for stack overflow.

Calling Sequence:

```
FSAV
 I
```

(I is a stack length word; see below and Notes to User).

```
       .ZREL
.SUBR:  SUBR-2
       .NREL
           .
           .
           .
       JSR    @.SUBR
NSI:   MOV    2,3            ;NEXT SEQUENTIAL INSTRUCTION
           .                 ;FOLLOWING RETURN FROM SUBR
           .
           .
       FSAV                  ;SAVE ACCUMULATORS, CARRY
SLW:   5                     ;TYPICAL STACK LENGTH WORD
SUBR:  LDA    Ø, Ø, 2        ;FIRST TRUE CALLED INSTRUCTION
```

Supporting Routines:

.I; AFSE, .RTEØ .

Subroutine Size:

Five page zero locations and 140 octal locations of normally relocatable memory are required.

Notes to User:

The stack length word (SLW) following FSAV can be equal to either -1, Ø, or any positive integer I. The following summarizes the meanings of these stack length words:

SLW= -1 No stack, no temporaries will be created for the called routine; no further calls are made from the called routine.

SLW= Ø A stack without temporary storage allocated is created for the called routine. The called routine calls some other routine.

FSAV (Continued)

SLW= +I	A stack will be created with I temporary locations allocated for use by the called routine; the called routine may call other routines.

Upon entry to SUBR, ACØ, AC1 and carry will be the same as the calling programs's; AC2 will contain the calling program's FSP and AC3 will contain the called program's FSP.

A fatal error message is generated if insufficent core storage is available for the creation of the called subroutine's stack.

FCALL, FRCAL, FRET and FQRET have alternate entry points in this routine.

Caller's accumulators (except AC3) and original state of carry will be restored upon return to the next sequential instruction following the subroutine call by FRET or FQRET. AC3 will contain the caller's FSP.

JSR @. FSAV is equivalent to FSAV. FSAV must be referenced by an . EXTN statement, . FSAV by an . EXTD statement.

.I

Purpose: To allocate number and SP stacks, and blank and unlabeled common for FORTRAN compiled program, initialize the Run Time Stack, and to construct pointers to them in a SOS, DOS or single task RDOS environment. (For a description of the multitask real time initializer, also labeled .I, see Appendix E.)

Calling Sequence: Program control is not transferred to .I in the manner that all other library routines receive control. Instead of being called, .I simply receives program control when the loaded program is started. This is due to the fact that the .END statement in this routine has the argument .I, whereas each other library routine is terminated by a simple .END statement.

Upon completion of the initialization procedure, .I issues an FCALL to the assembly language routine having the entry .MAIN. At the completion of .MAIN, it transfers control to the CLI by calling STOP under DOS. Under RDOS, .I calls STOP which then transfers control back to .I after outputting STOP 999 ↲ on the console. The system performs an effective halt, JMP., under SOS. For more information concerning the use of .I by an assembly language routine, see Appendix B.

Supporting Routines: CATIN, FCALL, .FLSP, .FLSZ, .MAIN; .STOP, .IOCAT, SP, NDSP, .WRCH, AFSE, SUCOM .

Subroutine Size: 144 octal locations of normally relocatable memory under DOS, 153 locations under RDOS. A 60 octal word temporary run time stack is also reserved for .I, and is used by the operating system.

Notes to User: The following describes the functions performed by .I in the sequence that they occur.

A system call, .SYSI, is issued to initialize system I/O under SOS (this is a no-op to DOS), and then a system reset (.RESET) is issued. Forty octal locations are then allocated for the SP stack immediately following the last loaded run time subroutine. A -1 is placed in the first location of the SP stack, and a pointer to the next location in the stack is

.I (Continued)

Notes to User: (con'd)

created. The SP stack is nothing more than a series of temporary locations for use by subroutines which have no stack set aside for their use.

Next, the number stack pointer is defined and number stack storage is allocated if floating point arithmetic is used in .MAIN, the FORTRAN program which is about to be run. This storage will be 630 octal words long or 30 octal plus twice whatever a user has specified in a .FLSZ statement. The default value creates enough room for 68 single precision real numbers (34 double precision real or single precision complex numbers, or 17 double precision complex numbers). After the allocation of the number stack (or after the allocation of the SP stack if no number stack is called for), a pointer to the beginning of the run time stack is defined, and .I's stack with 60 octal temporary storage locations is allocated; the Channel Assignment Table will be placed in these locations.

Next, a check is made to see whether or not there is room enough for blank common allocation, and blank common is allocated at the high end of memory. .NMAX is now updated with the system call .MEMI; the Channel Assignment Table is initialized and placed in the .I stack with an FCALL to CATIN.

After this, the main program, .MAIN, is called and upon its completion return is made to .I which transfers control to the CLI by calling STOP under DOS. Under RDOS, .I calls STOP which then transfers control back to .I after outputting STOP 999 ↵ on the console. The system performs an effective halt, JMP., under SOS.

Three additional entries exist in the RDOS single task .I which return control to either the CLI or to the debugger: FERTN, FERT1, and FERTØ. FERTN transfers control to the CLI via the call .SYSTM, .RTN . FERTØ transfers control to the CLI via the call .SYSTM, .ERTN . FERT1 transfers control to the debugger.

MAD, MADO

Purpose: To resolve an effective address from a given FORTRAN ADDRESS.

Calling Sequences:

(Input FORTRAN ADDRESS in AC2; current (i.e., caller's) FSP is base used in calculation).

```
JSR    @.MAD
```

(AC2 contains effective address upon exit; AC3 does not contain caller's FSP on exit.)

(Input FORTRAN ADDRESS in AC2; base FSP in AC1.)

```
JSR      @.MADO
```

(AC2 contains effective address upon exit; AC3 does not contain caller's FSP on exit.)

Supporting Routines: None.

Subroutine Size: Two page zero locations and 25 octal locations of normally relocatable memory.

Notes to User: Accumulators, carry are not restored upon exit. No error messages are generated.

.MAD and .MADO must be referenced by .EXTD statements.

INPUT/OUTPUT ROUTINES

CATIN, IMIO

Purpose:

To initialize the I/O Channel Assignment Table. This table lists the default assignments of the logical FORTRAN channels, and is used to maintain information about new assignments made by calls to FOPEN/FCLOS.

Calling Sequence:

(ACØ contains the starting address of the I/O Channel Assignment Table.)

```
FCALL
CATIN
```

(The three word entries for each of the 16 FORTRAN logical channels are set to the following states:

WORD 1, Closed ASCII file
WORD 2, -1 or word address of default file name
WORD 3, Random Record Length Ø.)

Supporting Routines:

FQRET, .SOS; .IOCAT, .SOSW .

Subroutine Size:

110 octal locations of normally relocatable memory for DOS; 105 locations for RDOS.

Notes to User:

Original contents of accumulators and carry are restored upon exit. CATIN and IMIO must be referenced by an .EXTD statement.

The Stand-alone Operating System will be force loaded.

There are 16 entries in the I/O Channel Assignment Table, one for each FORTRAN logical channel. The following table lists the FORTRAN channels and their default assignments where applicable:

Logical Channel Number	Default Assignment
0	none
1	none
2	none

CATIN, IMIO (Continued)

Logical Channel Number	Default Assignment
3	none
4	none
5	none
6	Plotter ($PLT)
7	none
8	TTY punch ($TTP)
9	Card Reader ($CDR)
1Ø	TTY Printer ($TTO)
11	TTY Keyboard ($TTI)
12	Line Printer ($LPT)
13	High Speed Paper Tape Reader ($PTR)
14	High Speed Paper Tape Punch ($PTP)
15	TTY Reader ($TTR)

A table labeled IMIO (and given as an entry along with CATIN) is located in the CATIN module. IMIO consists of a block of 16 words with a structure identical to the above table. Table entries which have default assignments contain the absolute CATIN module address of a byte string consisting of the appropriate four letter device name ($PLT, $TTR, etc.).

The I/O Channel Assignment Table is built in .I's stack at initialization time. This table consists of a block of 16 sequential three word entries, one entry for each FORTRAN logical channel with default assignments given in IMIO. The structure of each three word entry is as follows:

		bit 0	bit 1		bits 10 thru 15
TYPICAL	Word 0	OPEN switch	BINARY/ASCII switch	▨	DOS I/O Channel No.
I/O CATALOGUE	Word 1	FILE NAME POINTER			
ENTRY	Word 2	RECORD LENGTH OF RANDOM RECORDS			

The OPEN switch is set to a zero only if the referenced channel has been opened. The BINARY/ASCII switch is set to a zero only if ASCII mode has been selected. The DOS I/O CHANNEL field contains the DOS I/O channel number for this FORTRAN logical channel. (See the DOS or RDOS User's Manual, Chapter 4, "Command Word Format.") This field has meaning only if the logical channel is open.

The FILE NAME POINTER may be one of two things. If the file is closed, the pointer is simply the word address of a four letter file name or -1. If the file is open, the pointer is a byte pointer to some file name text string.

The RECORD LENGTH OF RANDOM RECORDS is Ø if the file has not been opened as a random file. Otherwise it is the integer record length in bytes of random records in the file.

Default values for each three word entry are given in the Calling Sequence description.

CHSAV, CHRST

Purpose: To permit the rereading or rewriting of FORTRAN records on disk. The method is to first save the status of a FORTRAN channel (CHSAV), issue any number of reads or writes, and then restore the original status of the channel (CHRST). Records processed between the status save and status restore operations may then be reread or rewritten.

Calling Sequences:

(An integer array has been created with a two word block allocated for storage of the channel status information.)

```
FCALL
CHSAV
2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of first word in the two word block
```

(CHSAV has been called previously.)

```
FCALL
CHRST
2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of the first word in the two word block
  containing previously saved channel status data.
```

Supporting Routines: FRET; .CPYL, .RTER, .IOCAT .

Subroutine Size: 63 octal locations of normally relocatable memory for DOS; 62 locations for RDOS.

Notes to User: Accumulators and carry are restored upon exit. Both routines will issue a non-fatal error message if the specified channel has not been opened. CHRST will also issue a non-fatal error message if an attempt is made to restore channel status information which was not previously saved. The status of more than one channel may be saved in the same array. For example, an array declared as I(2,100) can be used to save up to 100 blocks of channel status information.

COUT

Purpose: To input or output a character on a teletype.

Calling Sequences:

```
(ACØ contains the character to be output, right justified)

JSR    @.COUT

(The character is output to a TTY printer/punch)
```

```
JSR    @.CIN
(ACØ contains a character input from a TTY reader/keyboard).
```

Supporting Routines: FSAV, FQRET ; none.

Subroutine Size: Two page zero locations and 23 octal locations of normally relocatable memory.

Notes to User: If the character output was a carriage return, a line feed will also be output.

This routine can only be used with either the stand alone or disk operating systems.

No error messages are generated by this routine; accumulators, state of carry will be restored.

Characters input via . CIN will also be echoed on the TTY printer/punch. .COUT and . CIN must be referenced by . EXTD statements.

. COUT has an FCALL entry, COUT. COUT must be referenced by an . EXTN statement. . CIN has an FCALL entry point, CIN . CIN must be referenced by an . EXTN statement.

DELETE

Purpose: To delete a disk file.

Calling Sequence:

```
FCALL
DELET
Integer 1
FORTRAN ADDRESS of file name
```

Supporting Routines: FCALL, FCLOS, FRET; .COMP, .CPYL, .IOCAT, .RTER

Subroutine Size: Forty-three locations of normally relocatable memory are required under DOS, 100 under RDOS.

Notes to User: The file name is an ASCII byte string. This routine makes a system call,

```
.DELET
```

Before issuing the .DELET command, a check is made to determine whether or not the file has been closed. If the file is open on one channel, it will be closed and error message FEOPN will be issued. If the file is open on more than one channel, the file is closed on all these channels.

If there is no disk file directory entry corresponding to the file name byte string, the routine simply returns control to the caller; no error message is issued.

Original contents of accumulators and carry are restored.

Good practice dictates the use of DELET in program initialization to preclude the attempted writing of an already existing file.

DELET must be referenced by an .EXTN statement. DFILW and RLSE have alternate entry points in this routine.

FCLOS

Purpose: To free a FORTRAN logical channel and close the file associated with that channel.

Calling Sequence:

FCALL
FCLOS
Integer 1
FORTRAN ADDRESS of logical channel number

(A call can now be made to FOPEN requesting the free channel.)

Supporting Routines: FSAV, FRET, IMIO; .IOCAT, .RTER, .CPYL, .SOSW .

Subroutine Size: 41 octal NREL locations under DOS, 57 locations under RDOS.

Notes to User: The logical channel number is an integer constant with a value between Ø and 15_{10}.

Original accumulator's contents, carry are restored upon exit from this routine.

FCLOS must be specified in an .EXTN statement.

To close a channel under RDOS CLOSE may also be used (see Appendix E).

FFILE

Purpose:

To position a sequential file which has been assigned a FORTRAN Channel Number.

Calling Sequence:

```
JSR   @.FFIL
File Positioning Code
FORTRAN ADDRESS of FORTRAN Channel Number
```

(File Positioning Codes are: 1, position the file at its initial record; 2, close the file associated with this channel.)

Supporting Routines :

FSAV, FRET, FCLOS, FSEEK, FCALL; .CPYARG, .RTER, .IOCAT, .FCALL .

Subroutine Size:

One page zero location and 76 octal locations of normally relocatable memory.

Notes to User:

Accumulators and carry are restored upon exit from this routine.

This routine must be supported by the disk operating system.

I/O error conditions and unopened files will cause error messages to be generated.

.FFIL must be referenced by an .EXTD statement.

This routine has an FCALL entry point, FFIL . FFIL must be referenced by an .EXTN statement.

FOPEN

Purpose: To open a FORTRAN channel.

Calling Sequence:

JSR @.FOPEN
Integer number of arguments to follow - - 2 through 4 possible.
FORTRAN ADDRESS of logical channel number.
FORTRAN ADDRESS of file name
optional FORTRAN ADDRESS of binary specifier
optional FORTRAN ADDRESS of random record byte length

(The specified channel is now assigned to the named file.)

Supporting Routines: FSAV, FRET; .RTER, .CPYL, .SOSW, .IOCAT.

Subroutine Size: One page zero location and 141 octal locations of normally relocatable memory under DOS; one page zero location and 162 NREL locations under RDOS.

Notes to User: Logical channel numbers are represented by integer constants with values from Ø through 15_{10}.

The file name is an ASCII byte string terminated by a null byte. Likewise, the binary specifier is a single word ASCII byte string consisting of an ASCII B, left justified, followed by a null byte. If a binary specifier is given, the named file is opened with all particular device characteristics inhibited, e.g., such functions as a rubout character following a tab character output by a paper tape punch.

The random record length parameter, given only when random devices are selected, is an integer specifying the random record length in bytes. If the file does not exist, a file is created and then opened. This file is organized sequentially under DOS, randomly under RDOS.

This routine must be supported by either a disk or stand-alone operating system. Accumulators and carry are restored upon exit from this routine. The FCALL entry to this routine is FOPEN. .FOPEN must be referenced by an .EXTD statement; FOPEN must be referenced by an .EXTN statement.
Random access is permitted only under a disk supported operating system.

This routine has an FCALL entry point, FOPEN. FOPEN must be referenced by an .EXTN statement.

FREAD, FWRIT

Purpose:

To perform formatted or free form FORTRAN input (. FREAD) or output(. FWRIT) of ASCII data, or to perform FORTRAN input (. BRD) or output (. BWR) of binary data.

Calling Sequence:

(Binary data is to be read.)

```
JSR    @.BRD
FORTRAN ADDRESS of the logical channel number
Ø
ELEMENT DESCRIPTOR SEQUENCE(s) (see Notes to User)
5
```

(Binary data is to be written.)

```
JSR    @.BWR
FORTRAN ADDRESS of the logical channel number
Ø
ELEMENT DESCRIPTOR SEQUENCE(s) (see Notes to User)
5
```

(ASCII data is to be read or written in free format.)

```
JSR    @.FREAD   (or @.FWRIT)
FORTRAN ADDRESS of the logical channel number
Ø
ELEMENT DESCRIPTOR SEQUENCE(s) (see Notes to User)
5
```

FREAD, FWRIT (Continued)

```
(Formatted ASCII data is to be read or written.)

JSR @.FREAD (or @.FWRIT)
FORTRAN ADDRESS of the logical channel number
FORTRAN ADDRESS of the beginning of the format
      statement text string.
ELEMENT DESCRIPTOR SEQUENCE(s) (see Notes to User)
5
```

Supporting Routines:

FSAV, FRET,MPY,DVD; .WRTS, .REDS, .ALLOC, .THREAD .FRG1, .FRGLD, .READL, .WRITL, .RDFCH, .RTER, .RDFLD, .STBT, .LDBT, .MVBC, .ARYSZ, .FSBR, .WRCH, SP, .SVØ. (FERTØ is also used under RDOS.)

Subroutine Size:

Four page zero locations and 3665 octal locations of normally relocatable memory. This module also has the unusually large run time stack frame size of 256 octal locations including header.

Notes to User:

Contents of accumulators and carry are restored upon exit from this routine.

.FREAD, .FWRI, .BRD, and .BWR must be referenced by an .EXTD statement.

If the contents of the first word in the format text string (see formatted I/O) are ØØ24Ø1, then the first four bytes in this string are ignored. This permits FREAD to be used by the FORTRAN compiler, which always precedes the format text string with JMP @.+1. JMP @.+1 assembles to ØØ24Ø1.

The ELEMENT DESCRIPTOR SEQUENCES describe in detail the nature of each data type in the list of elements to be input or output. Each SEQUENCE is in reality a set of eight possible calling sequences. One sequence is selected to describe each data element in the input/output list.

FREAD, FWRIT (Continued)

Thus the FORTRAN statement:

WRITE (1∅) 'REAL RESULT IS', X

generates a call to .FWRIT with two ELEMENT DESCRIPTOR SEQUENCES. One is sequence 6 for the outputting of the text string 'REAL RESULT IS';the other is sequence ∅ to output the real variable X.

The first word of each ELEMENT DESCRIPTOR SEQUENCE is an integer tag, labeling the type of sequence which is to follow. The following list summarizes the integers and their corresponding sequences.

Integer	Data Element Type
∅	Variable
1	Array Element
2	Array
3	Left Parenthesis
4	End of loop Right Parenthesis
6	String
7	End of file address
8	Error return address

Integer 5 is used as a flag to terminate the entire calling sequence.

Following are the detailed ELEMENT DESCRIPTOR SEQUENCE parameters for each data element type, with accompanying example FORTRAN statements which generate them. Combining the appropriate ELEMENT DESCRIPTOR SEQUENCE with one of the Calling Sequences given above yields a complete FORTRAN input/output calling sequence.

Variable Data Element Sequence

∅
Integer variable type (see below)
FORTRAN ADDRESS of variable

FREAD, FWRIT (Continued)

FORTRAN Statement

```
READ (11,1) TEST
```

FORTRAN Object Code

```
JSR    @.FREA
.C1
L2.
Ø
2
V.+Ø                    ;TEST
5
```

The integer variable type is an integer, 1 through 5, which specifies the type of variable in the I/O list. The following variable types correspond to integers 1 through 5 in the following fashion:

1	Integer, logical, alphabetic/hollerith
2	SPFL
3	DPFL
4	SPCX
5	DPCX

Thus the code generated by

```
READ (11,1) TEST
```

yields the following complete call to FREAD:

```
                                    JSR          @.FREA
FORTRAN ADDRESS of      }  --->     .C1
Logical Channel Number  }
FORTRAN ADDRESS of      }
Format Statement Text   }  --->     L2.
String                  }
Variable ELEMENT        }        {  Ø
DESCRIPTOR              }  --->  {  2
SEQUENCE                }        {  V.+Ø
Terminator                          5
```

The second ELEMENT DESCRIPTOR SEQUENCE describes an Array Element in the I/O list.

FREAD, FWRIT (Continued)

Array Element Sequence

```
1
N  (see below)
FORTRAN ADDRESS of Three Word Specifier
Ø
FORTRAN ADDRESS of Subscript 1
•
•
•
FORTRAN ADDRESS OF Subscript N-1
```

FORTRAN Statements

```
DIMENSION  NAME (25)
•
•
•
READ (11,1ØØ) NAME (1)
```

FORTRAN Object Code

```
•
•
•
JSR    @.FREA
.C3
L2.
1
3
V.+Ø                        ;NAME
Ø
.C2
5
```

N in the array element sequence is an integer equal to the number of parameters following N excluding the list terminator flag, 5.

The Three Word Specifier is described in Appendix D, "Array Structure and Handling."

The ELEMENT DESCRIPTOR SEQUENCE for entire arrays in FORTRAN I/O lists follows.

FREAD, FWRIT (Continued)

Array Descriptor Sequence

2
FORTRAN ADDRESS of Three Word Specifier

FORTRAN Statements

```
DIMENSION A(1Ø)
.
.
.
READ BINARY (13) A
```

FORTRAN Object Code

```
JSR    @.BRD
.C3
Ø
2
V.+Ø                  ;A
5
```

There are two possible ELEMENT DESCRIPTOR SEQUENCES for left parenthesis data elements, depending upon whether the parenthesis is significant or not. Left parentheses are significant only in implied DO-loops or nests of implied DO-loops.

Insignificant Left Parenthesis Sequence

3
FORTRAN ADDRESS of useless-right-parenthesis flag

This flag is an integer 4, to be described in the INSIGNIFICANT-RIGHT-PARENTHESIS Sequence, sequence 4.

FORTRAN Statement

```
READ (11,1) (TEST)
```

FREAD, FWRIT (Continued)

FORTRAN Object Code

```
JSR    @.FREA
.C1
L2.
3                   Insignificant left parenthesis sequence
L3.
Ø
2
V.+Ø
4                   ;TEST
5
```

Significant Left Parenthesis Sequence

3
FORTRAN ADDRESS of useful-right-parenthesis Flag

This flag is an address not equal to integer 4, and will be described in the SIGNIFICANT-RIGHT-PARENTHESIS sequence, sequence 4.

FORTRAN Statements

```
DIMENSION  TEST 1 (1Ø)
:
READ (11,1) ( TEST1(I), I = 1,7)
```

FORTRAN Object Code

```
JSR    @.FREA
.C3
L2.
3                   Significant left parenthesis sequence
L3.
1
3
V.+Ø                ;TEST1
Ø
V.+3                ;I
4
V.+3
.C2
.C4
.C2
.-4
5
```

FREAD, FWRIT (Continued)

Corresponding to the two left parenthesis sequences there are two right parenthesis ELEMENT DESCRIPTOR SEQUENCES.

Insignificant Right Parenthesis Sequence

```
4
```

FORTRAN Statement

```
READ (11,1) (TEST)
```

FORTRAN Object Code

```
JSR   @.FREA
.C1
L2.
3
L3.
ø
2
V.+ø
4            ← insignificant right parenthesis sequence
5
```

Significant Right Parenthesis Sequence

```
4
FORTRAN ADDRESS OF Indexing Variable
FORTRAN ADDRESS of Start Value
FORTRAN ADDRESS  of Test Value
FORTRAN ADDRESS of Increment Value
.-4
```

FORTRAN Statements

```
DIMENSION   TEST1 (1ø)
.
.
.
READ (11,1) ( TEST1(I),  I = 1,7)
```

FREAD, FWRIT (Continued)

FORTRAN Object Code

```
JSR    @.FREA
.C3
L2.
3
L3.
1
3
V.+Ø            ;TEST 1
Ø
V.+3            ;I
4
V.+3
.C2
.C4         }   Significant right parenthesis sequence
.C2
.-4
5
```

In the above sequence, .-4 is the address containing the FORTRAN ADDRESS of the Indexing Variable.

The ELEMENT DESCRIPTOR SEQUENCE for ASCII string elements is straightforward, as shown below.

String Element Descriptor Sequence

```
6
TEXT STRING
(terminated by a null)
```

FORTRAN Statement

```
WRITE (1Ø) "MESSAGE"
```

FORTRAN Object Code

```
JSR    @.FWRI
.C1
Ø
6
.TXT            /MESSAGE/
```

FREAD, FWRIT (Continued)

It is possible for program control to branch from reading or writing sequence upon receipt of an end-of-file. This procedure is illustrated in the END-OF-FILE ELEMENT DESCRIPTOR SEQUENCE below.

End-of-File Element Descriptor Sequence

7
FORTRAN ADDRESS of EOF Return

FORTRAN Statement

READ (11,1,END=7) A

FORTRAN Object Code

```
JSR   @.FREA
.C1
L2.
7  ⎫
L3.⎭  ◄──── EOF Sequence
Ø
2
V.+Ø          ;A
5
```

Finally, if a user wishes to gain program control after an I/O error at the device level (parity, record size) has been detected, the ERROR ELEMENT DESCRIPTION SEQUENCE must be employed.

Error Element Descriptor Sequence

8
FORTRAN ADDRESS of Error Return

FORTRAN Statement

READ (11,1,ERR=7) A

FREAD, FWRIT (Continued)

FORTRAN Object Code

```
JSR   @.FREA
.C1
L2.
1Ø  }        Error Return sequence
L3. }
Ø
2
V.+Ø         ;4
5
```

The following illustration shows the detailed structure of a call to FREAD generated by the following test FORTRAN program:

```
               READ (11,1) TEST
1              FORMAT (1HØ, E5.1)
               END
```

. FREAD, . FWRIT, . BRD, and . BWR have the following respective FCALL entry points: FREAD, FWRIT, BRD, and BWR. FREAD, FWRIT, BRD, and BWR must each be referenced in an . EXTN statement.

```
A 0002   .MAIN
                        ;         READ      (11,1)   TEST    FORTRAN I/O Statement, generating call to
                        ; 1       FORMAT    (1H0,E5.1)       FREAD
                        ;         END

                        ;         READ      (11,1)   TEST
                                  .NREL
                                  .TITL     .MAIN
                                  .ENT      .MAIN
                                  .NREL
            000001                .TXTM     1
                                  .EXTU
                                  .EXTN     .I
                        .F1:
  00000'000000          .F2:      0
            000000                .CSIZ     0
  00001'000002                    FS.
                        .MAIN:                                       Call to FREAD
  00002'002401                    JMP       @.+1
  00003'000004'                   L1.
                        L1.:
  00004'006001$                   JSR       @.FREA
  00005'000024'                   .C1    ←  FORTRAN ADDRESS of Logical Channel Number
  00006'000013'                   L2.    ←  FORTRAN ADDRESS of Format Text String less two
  00007'000000                    0      ←  ELEMENT DESCRIPTOR SEQUENCE Tag
  00010'000002                    2      ←  Real Variable code
  00011'000011                    V.+0   ←  FORTRAN ADDRESS of variable TEST
  00012'000005                    5      ←  end of FREAD sequence

                        ; 1       FORMAT    (1H0,E5.1)
  00013'002401          L2.:      JMP       @.+1
  00014'000023'                   L3.
                                  .TXT      ◊(1H0,E5.1)1◊
  00015'024061
  00016'044060
  00017'026105                                             Format string
  00020'032456
  00021'030451
  00022'020400

                        ;         END
                        L3.:
  00023'006002$                   JSR       @.FRET
  00024'000013          .C1:      000013    ←  Logical Channel Number
            000002                FS.=2
            000000                SFS.=0
            177611                T.=-167
            000011                V.=200+T.
            177612                TS.=T.+1
            177611                FTS.=T.+0
            000012                VS.=V.+1
            000011                FVS.=V.+0
            000011  TEST=         V.+0   ←  TEST will be read into MAIN's stack frame,
            000013'N1=            L2.       at the first available temporary FTSTR
                                  .END
```

FSEEK

Purpose: To access a particular record on a random access file.

Calling Sequence:

```
JSR    @.FCAL   (or FCALL)
FSEEK
Integer 2 ( since two arguments follow)
FORTRAN ADDRESS of FORTRAN logical channel number
FORTRAN ADDRESS of the record number to be accessed
```

Supporting Routines: FRET, DVD, MPY; .CPYL, .IOCAT .

Subroutine Size: 56 octal locations for DOS and 55 for RDOS of NREL memory.

Notes to User: When more than one record is required to be written or read without intervening calls to FSEEK, records will be written or read sequentially. A random file is positioned initially to the beginning of record Ø by the following:

```
FCALL
FSEEK
2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of integer Ø
```

This routine requires the support of the disk operating system.

Original contents of accumulators, carry are restored upon exit from this routine.

FSEEK must be referenced by an .EXTN statement.

The file (with the given channel number) is positioned at the first byte of the first random record whose length was specified by FOPEN.

A run time error is given if the file is not randomly organized or if the file is not opened.

RDFLD, RDFCH

Purpose:

To read and transfer a portion of an ASCII string from one buffer to another, either by counting characters in the transferred field (RDFLD) or by reading to a specified character (RDFCH).

Calling Sequences:

```
(AC2 contains the number of characters in the field
to be read.)

JSR     @.RDFLD
FORTRAN ADDRESS of "FROM" string byte pointer
FORTRAN ADDRESS of "TO" string byte pointer
abnormal return (character count retained in AC1.
   See Notes to User.)
normal return

(Both the "FROM" and the "TO" string pointers are
updated upon exit.)
```

```
(AC2 contains the terminal field character.)

JSR     @.RDFCH
FORTRAN ADDRESS of "FROM" string byte pointer
FORTRAN ADDRESS of "TO" string byte pointer
abnormal return (character count retained in AC1.  See
   Notes to User.)
normal return

(Both the "FROM" and the "TO" string pointers are
updated upon exit.)
```

Supporting Routines :

FSAV, FRET; .FARG, .LDBT, .STBT, .RTER .

Subroutine Size:

Two page zero locations and 115 octal locations of normally relocatable memory.

RDFLD, RDFCH (Continued)

Notes to User:

Original contents of accumulators and carry are restored upon exit. .RDFLD and .RDFCH must be referenced by an .EXTD statement.

A fatal error message will be output upon overflow of the "TO" buffer only if the last buffer location contains a word consisting of two ASCII rubouts, Ø77577. FREAD will ensure that such a buffer terminator exists in every case where it issues a call to RDFLD or RDFCH.

Both RDFLD and RDFCH examine each character that is transferred. If a null is detected before the scheduled end of the field, a branch is made to the abnormal return. AC1 is then set to the number of characters (excluding the null) which were read and transferred before the branch.

Additionally, if a carriage return or form feed character is detected by RDFLD a branch will be made to the abnormal return location.

.RDFLD and .RDFCH have FCALL entry points RDFLD and RDFCH respectively. RDFLD and RDFCH must be referenced by an .EXTN statement.

READL, WRITL

Purpose:

To perform line input of ASCII (.READL) or binary (.REDS) data strings, or line output of ASCII (.WRITL) or binary (.WRTS) data strings on a FORTRAN logical channel.

Calling Sequences:

(ACØ contains a byte pointer to the beginning of the output string. AC1 contains a pointer to the end of the output string. AC2 contains the FORTRAN logical channel number.)

```
JSR    @.WRITL
FORTRAN ADDRESS of format flag
error return  (System error code returned in AC2)
normal return
```

(ACØ contains a byte pointer to the beginning of the output string. AC1 contains a pointer to the end of the output string. AC2 contains the FORTRAN logical channel number.)

```
JSR    @. WRTS
error return (System error code returned in AC2)
normal return
```

(ACØ contains a byte pointer to the beginning of the input string buffer. AC2 contains the FORTRAN logical channel number.)

```
JSR    @.READL
error return  (System error code returned in AC2)
normal return
```

READL, WRITL (Continued)

```
(ACØ contains a byte pointer to the beginning of the
input string buffer.  AC1 contains a pointer to the end
of the input string buffer.  AC2 contains the FORTRAN
logical channel number.)

JSR    @.REDS
error return (System error code returned in AC2.)
normal return
```

Supporting Routines : FSAV, FRET; .FOPEN, .FARG, .LDBT, .STBT, .IOCAT, .SOSW .

Subroutine Size: Four page zero locations and 175 octal locations of normally relocatable memory under DOS;212 under RDOS.

Notes to User: Contents of accumulators and state of carry are restored upon exit from this routine. Descriptions of the system error codes mentioned above can be found in the DOS User's Manual, Chapter 4, "Input Output Commands," or the RDOS User's Manual, Chapter 5. Leading nulls are ignored and a trailing null is recognized as a terminator under RDOS.

.WRITL, .WRTS, .READL, and .REDS must be referenced by an .EXTD statement.

The format flag, given as a calling parameter for ASCII Write, .WRITL, is simply a one word flag used to indicate whether the data string will be output in free format or not. If the flag is non-zero, formatted output is indicated and a carriage return will be appended to the output string. If the flag is all-zero, free format is indicated and a null will be appended to the end of the string.

If formatted output is indicated, the first character in the output string will then be examined. If this character is found to be ASCII Ø, this zero will be replaced by a carriage return. If the first character is found to be ASCII 1, it will be replaced by a Form Feed character. All first characters which are neither ASCII Ønor 1 will be dropped from the output string.

WRCH

Purpose: To print a string of ASCII characters on a teletype printer.

Calling Sequence: (ACØ contains the byte pointer to the beginning of the byte string.)

```
JSR    @.WRCH
```

(Upon exit from the routine, AC1 contains the number of characters in the string.)

Supporting Routines: FSAV, FRET; .LDBT, .COUT .

Subroutine Size: One page zero location and 15 octal locations of normally relocatable memory are required.

Notes to User: Original states of accumulators (except AC1)and carry are restored upon exit from this routine. The contents of AC1 will be as noted above.

ASCII characters in the string must be packed left to right, 2 characters per word.

This routine can only be used with either the stand-alone or disk operating systems.

.WRCH must be referenced by an .EXTD statement.

This routine has an FCALL entry point, WRCH .

MISCELLANEOUS FORTRAN SUPPORT

AFRTN

Purpose: To provide an abnormal means of return from a FORTRAN subroutine. Return is to an address specified on the called subroutine's stack instead of the first location following the caller's parameter list.

Calling Sequence:

```
JSR    @.AFRTN
FORTRAN ADDRESS of variable containing the return address
```

Supporting Routines: FRET; .FRGØ .

Subroutine Size: One page zero location and 5 locations of normally relocatable memory.

Notes to User: No error messages are generated; accumulators and carry are restored.

This subroutine has no FCALL entry point.

.AFRTN must be referenced by an .EXTD statement.

CGT

Purpose: To implement the FORTRAN "Computed GO TO" facility.

Calling Sequence:

JSR @.CGT
N, The number of statement numbers which can be gone to
FORTRAN ADDRESS of the non-subscripted integer variable, V
Effective address N_1
Effective address N_2
.
.
.
Effective address N_n

Supporting Routines: FRET, FSAV; .RTER, .FRGL .

Subroutine Size: One page zero location and 23 octal locations of normally relocatable memory.

Notes to User: The above assembly language calling sequence is generated by the FORTRAN statement GO TO ($n_1, n_2, \ldots, n_n$) V .

Accumulators and carry are restored upon exit from this subroutine. A fatal error message is generated if the integer variable V is less than 1 or greater than N, and program control remains in the error message subroutine.

.CGT must be referenced by an .EXTD statement.

This routine has an FCALL entry point, CGT . CGT must be referenced by an .EXTN statement.

FINIT

Purpose: To allocate unlabeled common storage.

Calling Sequence:

```
JSR   @.FINI
Absolute address of L1
Absolute address of L2
```

(L1 and L2 are the first and last entries respectively in the blank common displacement table generated by the FORTRAN Compiler. The last entry in the table, L2, is zero unless blank common storage has been requested more than once.)

Supporting Routines: FSAV, FRET; SUCOM .

Subroutine Size: One page zero location and 24 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored .

This routine is of limited usefulness to assembly language programmers. It is mentioned here only for the sake of completeness.

.FINI must be referenced by an .EXTD statement.

This routine has an FCALL entry point, FINIT . FINIT must be referenced by an .EXTN statement.

GT, GE, LT, LE

Purpose:

To perform signed comparisons between the contents of registers ACØ through AC2.

$R1 > R2$ -- GT
$R1 \geq R2$ -- GE
$R1 < R2$ -- LT
$R1 \leq R2$ -- LE

Calling Sequences:

(The contents of the first register, R1, is multiplied by 400_8, and the contents of the second register, R2, is added to that product. The product must be stored in the next sequential location following the call before issuing the call.)

```
      JSR   @.GT (.GE, .LT, .LE)
CODE: 400₈*  R1 + R2
```

(If it is true that (R1) is greater than -- greater than or equal to, less than, or less than or equal to -- (R2), -1 is loaded into R2. Otherwise, Ø is loaded into R2).

Supporting Routines: FRET, FSAV ; none.

Subroutine Size: Four page zero locations plus 76 locations of normally relocatable memory.

Notes to User:

Original states of all accumulators but AC3 and R2 are restored, and the entry state of carry is also restored.

No error messages are generated.

.GT, .GE, .LT, and .LE must be referenced by an .EXTD statement.

.GT, .GE, LE, and .LT have the following respective FCALL entry points: GT, GE, LE, and LT. GT, GE, LE, and LT must be referenced by an .EXTN statement.

NFRTN

Purpose: To provide a called subroutine with a means of return to the first location following the caller's parameter list.

Calling Sequence: JMP @.NFRTN

Supporting Routine: FRET; none.

Subroutine Size: One page zero location and 10 octal locations of normally relocatable memory.

Notes to User: This subroutine assumes that FRTN points to N, the first item in the caller's parameter list.

Accumulators and carry are restored, no error messages are generated.

.NFRT must be referenced by an .EXTD statement.

This routine has no FCALL entry point.

OVFLO

Purpose: To provide a means of abnormal return from a subroutine by checking for the occurrence of non-integer arithmetic overflow.

Calling Sequence:

FCALL
OVERFLOW
2 OR 3
FORTRAN ADDRESS of return upon overflow
FORTRAN ADDRESS of return if no overflow
FORTRAN ADDRESS of string literal "S" or "N"

(The last argument is optional. If "S," overflow error messages are suppressed; if "N," overflow messages are not suppressed. "S" is the default value if no string literal argument address is given.)

Supporting Routines: none; .AFRTN, .CPYL, .OVFL .

Subroutine Size: 23 octal locations of NREL memory.

Notes to User: Accumulators and state of carry are not restored.

The string literal argument consists of an ASCII S or N, left justified and followed by a null byte.

OVERF must be referenced by an .EXTN statement.

RTEØ, RTER, RTES

Purpose:

To indicate that a run time error has occurred, either by specifying an error code (RTER) and the program counter contents, or by specifying an error code and the location from which a call was issued upon detection of an error (RTEØ, RTES). In all cases, the message will specify whether the error is fatal or non fatal.

Calling Sequences:

(ACØ set to called-from address.)
ERROR CODE
JSR @.RTEØ

(Latest entry in SP stack is the called-from address.)
ERROR CODE
JSR @.RTES

ERROR CODE
JSR @.RTER

(The value of the program counter just prior to the call to .RTER will be printed, alongwith the appropriate error code.)

Supporting Routines:

FCALL, FRET, FSAV, .FSAV, .BASC, .BDAS, .I; .OVFL, SP, .WRCH . (FERT1 is also used under RDOS.)

Subroutine Size:

Three page zero locations and 221 octal locations of normally relocatable memory under DOS: 214 locations under RDOS.

Notes To User:

Original states of accumulators and carry will be restored upon exit.

The structure of the ERROR CODE word is as follows:

```
      | 1 | f |                  C                  | 1 | Ø  Ø  1 |
Bit     Ø   1   2                                 11  12        15
```

Field f will be set to a 1 if and only if the error code signifies a fatal run time error, and a "fatal run time error" message will be output by this routine. Field C is the field containing an octal value which will be converted to decimal and output as the specific error code by this routine. A list of all run time error codes is given in the FORTRAN Manual, Appendix A. The definition of the ERROR CODE structure and mnemonic error code assignments are defined on the PARF tape.

Notice that bits zero, twelve, and fifteen are always set to a one, and bits thirteen and fourteen are always set to zero. These fixed bit assignments cause all ERROR CODES to be effective skips. Thus the call to the error routine can be made conditional on the result of a skip test, skipping to the error code if no error message should be output. The code will then be executed as an arithmetic/logic no load, skip instruction skipping over the call to the error processing routine.

The non-fatal error messages output by these routines are of the form:

```
RUNTIME ERROR NN AT LOC.xxxxxx, CALLED FROM
        LOC. yyyyyy
```

wher NN is the decimal run time error code (a complete list of error codes is found in the FORTRAN IV User's Manual, 093-000053). xxxxxx is the NREL starting address of the subroutine detecting the error. yyyyyy is the address (+1) in the main program (or user subroutine) of the assembly language instruction causing the error to occur.

Fatal error messages will be of the same form as non-fatal error messages with the specifier FATAL appended to the message.

.RTEØ is used by the FLINK module, .RTES by the signed integer and single precision and double precision real arithmetic modules, and .RTER by the remainder of the run time routines.

.RTER, .RTES, and .RTEØ must be referenced by an .EXTD statement.

All fatal error conditions cause program control to return to the Debugger (if it is loaded), or otherwise to the operating system under DOS. Under RDOS, control is returned to the Debugger, multitask scheduler, or CLI via the initializer.

STOP, PAUSE

Purpose: To implement the FORTRAN STOP and PAUSE functions.

Calling Sequences:

```
JSR    @.STOP
TEXT
```

(The message "STOP"↵ is output on the TTY printer, then the text message is output with a terminating carriage return and control returns to the operating system.)

```
JSR    @.STOP
-1
```

(The message "STOP" ↵ is output, then control returns to the operating system under DOS or to the CLI or multitask scheduler via the initializer under RDOS.)

```
JSR    @.PAUSE
TEXT
NSI
```

(The message "PAUSE"↵ is output on the TTY printer, then the text message is output, followed by a carriage return. Control reverts to the operating system until any key is struck, when control then returns to the Next Sequential Instruction.)

```
JSR    @.PAUSE
-1
NSI
```

(The message "PAUSE"↵ is output on the TTY printer, and control reverts to the operating system. Control returns to the Next Sequential Instruction as soon as any

Supporting Routines: FRET, FSAV; .WRCH . (FERTN is also used under RDOS.)

Subroutine Size: Two page zero locations and 52 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit from these routines. .STOP and .PAUSE must be referenced by an .EXTD statement.

THREAD, ALLOC

Purpose: To transfer the latest five word element of one list to a second list (THREAD) or to examine a list and -- if it is a null list -- create a five word element and transfer it to a second list (ALLOC).

Calling Sequence:

```
JSR     @.THREAD (or @.ALLOC)
FORTRAN ADDRESS of "FROM" list pointer
FORTRAN ADDRESS of "TO" list pointer
```

(See Notes to User for a detailed explanation of .THREAD and .ALLOC operation.)

Supporting Routines: FSAV, FRET; .CPYARG .

Subroutine Size: Two page zero locations and 44 octal locations of normally relocatable memory.

Notes to User: Contents of accumulators and carry are restored upon exit from this routine. No error messages are generated.

.THREAD and .ALLOC must be referenced by an .EXTD statement.

The five word elements which are list numbers are composed of blocks of five sequential locations. The first location (i.e., the one having the lowest core address) is the link word; the remaining four words are reserved for list data storage:

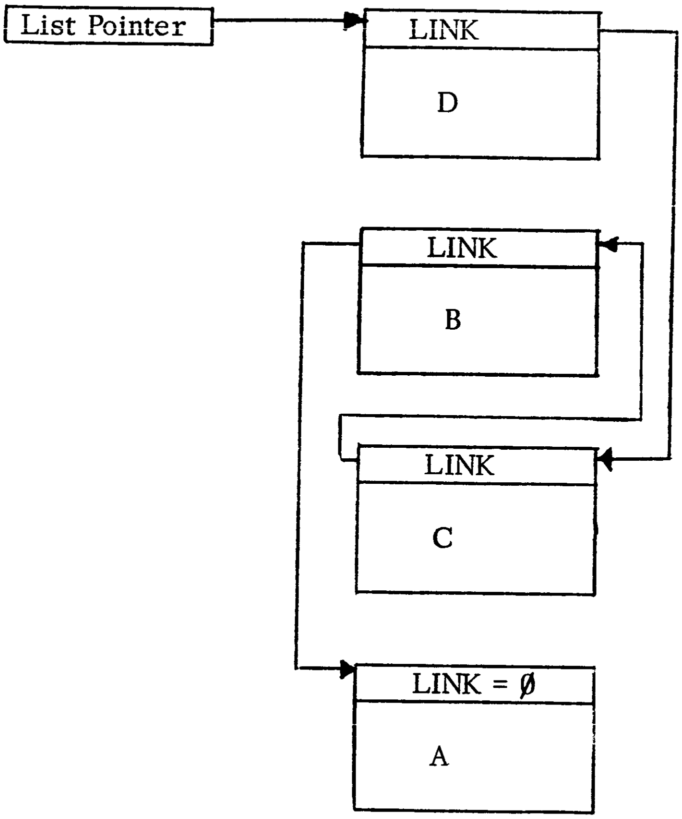
LINK
data
data
data
data

List Element

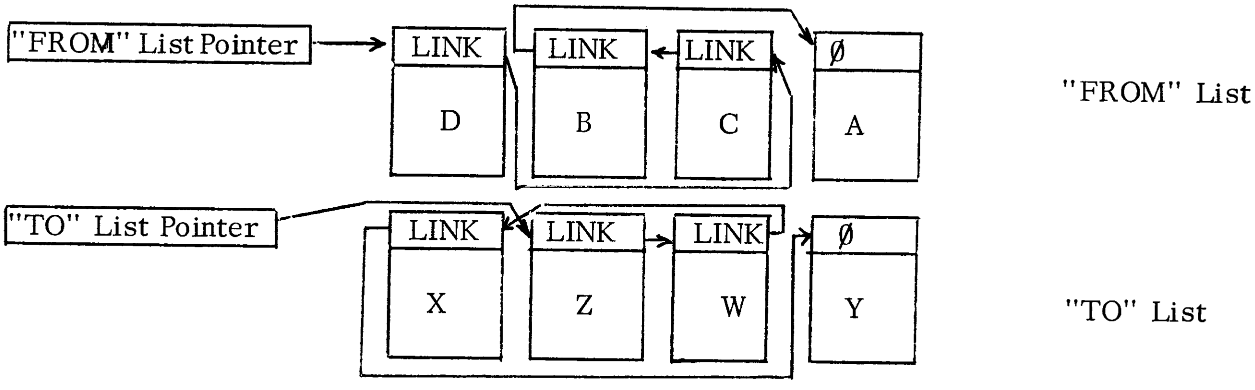
Lists are variable in length, and list elements may be found in scattered locations throughout available core memory. The oldest member of a list has a LINK of

THREAD, ALLOC (Continued)

zero; each successive list element has a LINK which points to the next earlier element. Finally, each list has a pointer to the most recent list element.

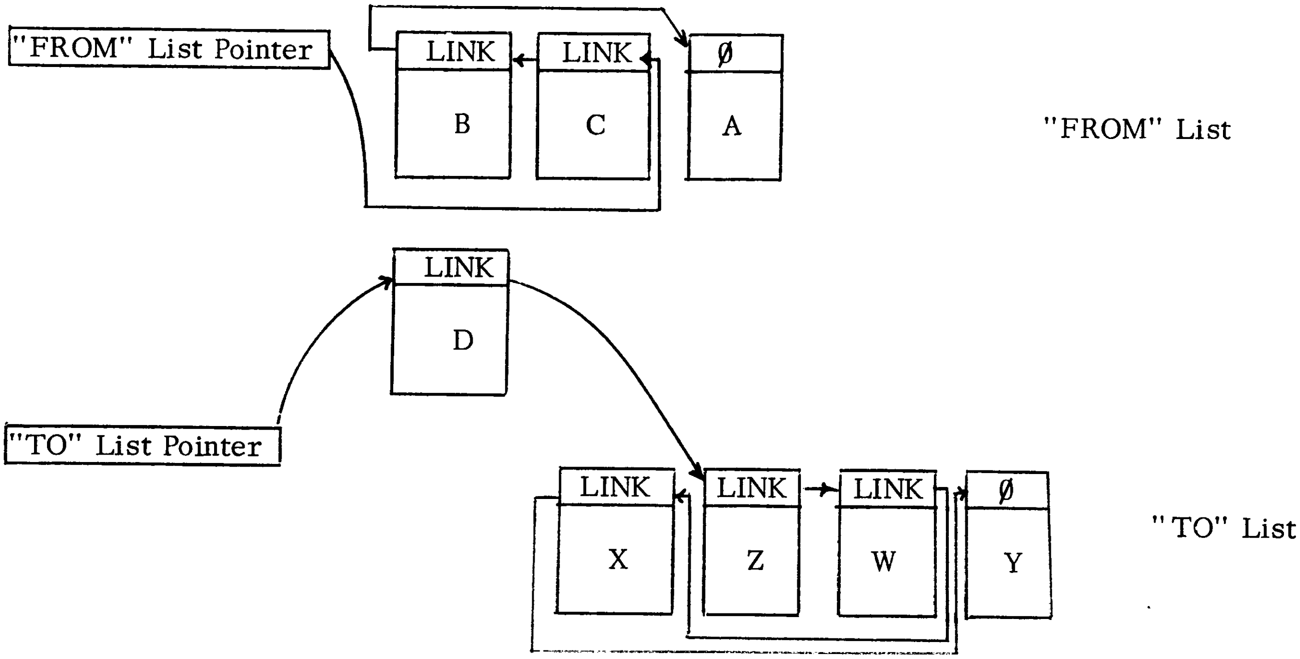


. THREAD takes the most recent element from one list, the "FROM" list, and attaches it to a second list, the "TO" list,where it then becomes the most recent entry in the list.



BEFORE THREAD OPERATION

THREAD, ALLOC (Continued)



AFTER THREAD OPERATION

.ALLOC, on the other hand, first examines the FROM list pointer; if it is non-zero then the list has at least one element, and .ALLOC calls .THREAD. If the pointer contains zero, then the FROM list is a null list. In this case, .ALLOC creates a five word list element, appending it to the stack frame of the routine (or .MAIN) which called .ALLOC. This new element is preserved by adjusting the caller's FLGT, and the new element is added to the TO list by .THREAD.

The routines have FCALL entry points, ALLOC and THREAD. ALLOC and THREAD must be referenced by an .EXTN statement.

ARRAY HANDLING ROUTINES

ARYSZ

Purpose: To determine the size of an array in terms of both elements in the array and core locations needed to contain the array.

Calling Sequence: (ACØ contains the starting address of the subscript bound specifier.)

```
JSR    @.ARYSZ
```

(ACØ contains the total number of elements in the array, AC1 contains the total number of words in the array.)

Supporting Routines: FRET, FSAV, MPYØ ; none .

Subroutine Size: One page zero location and 20 octal .NREL locations.

Notes to User: Accumulators and carry are restored upon exit from this routine. No error messages are generated.

.ARYS must be referenced by an .EXTD statement.

This routine has an FCALL entry point, ARYSZ .

FALOC

Purpose:	To allocate an array on a caller's stack.
Calling Sequence:	JSR @. FALOC FORTRAN ADDRESS of subscript bound specifier FORTRAN ADDRESS of array specifier Integer value of array size in words (not elements).
Supporting Routines :	FSAV, FRET; .CPYARG, .RTER, AFSE .
Subroutine Size:	One page zero location and 33 octal .NREL locations.
Notes to User:	Accumulators and carry are restored upon exit. A fatal error message is generated if there is insufficient run time stack area for allocation of the array. Caller's FLGT is adjusted to include array size so that newly created stacks will not overwrite the array. .FALOC must be referenced by an .EXTD statement. This routine has an FCALL entry point, FALOC . FALOC must be referenced by an .EXTN statement.

FREDI

Purpose:

To permit the redefinition of array subscript values when arrays are passed as dummy arguments.

Calling Sequence:

```
JSR           @.FRED
FORTRAN ADDRESS of special subscript bound specifier
                                   (built by compilter)
FORTRAN ADDRESS of array data address
FORTRAN ADDRESS of area reserved for 3-word array
                                               specifier
```

(A new three word specifier and subscript bound specifier are constructed. The SBS is appended to the caller's stack -- see Appendix D illustration.)

Supporting Routines:

FRET, FSAV, MPYØ, .OFLO; AFSE, .CPYAR, .FRG1 .

Subroutine Size:

One page zero location and 111 octal locations of normally relocatable memory.

Notes to User:

Accumulators and carry are restored upon exit. Upon stack overflow, the contents of the caller's FRTN are printed as an error message.

The call to FREDI is generated by FORTRAN statements of this general form:

```
SUBROUTINE              TESTSUB (x,y,z,a,...)
                        DIMENSION (x(i), y(m), z(n))
```

If there is insufficient run time stack area for the creation of a new SBS (see Appendix D, Array Structure and Handling) is made to .OFLO. .OFLO is an entry in the FLINK module, used by FLINK to collapse run time stack frames to permit the issuing of a stack overflow message. Except for the FLINK subroutines, only FREDI needs to use the .OFLO entry. This is true since at run time only FSAV and FREDI allocate storage on the run time stack. FREDI must be referenced by an .EXTD statement. This routine has the FCALL entry point FREDI.

FSBR, FSUB

Purpose:

To calculate the effective address of an array element for the compiled program (FSUB), or for subroutine FREAD in a formatted I/O entry (FSBR).

Calling Sequences:

JSR @.FSUB
Integer number of arguments
FORTRAN ADDRESS of 3 word address specifier
FORTRAN ADDRESS of result
FORTRAN ADDRESS of subscript 1
FORTRAN ADDRESS of subscript 2
.
.
.
FORTRAN ADDRESS of last subscript

(The effective address of the array element selected by the input subscript choices is placed in the FORTRAN ADDRESS of the result.)

(ACØ contains pointer to FREAD argument list with Element Descriptor List = 1)

JSR @.FSBR

(The effective address of the selected array element is returned in AC1.)

Supporting Routines:

MPY; .MADO, .RTES, SP .

Subroutine Size:

Two page zero locations and 175 octal locations of normally relocatable memory.

Notes to User:

Accumulators and carry are not restored upon exit from this routine. Subscript calculation errors will be flagged by a fatal error message. .FSBR and .FSUB must be referenced by an .EXTD statement.

APPENDIX A

RUN TIME ROUTINE TITLES AND NREL ENTRIES

To aid the debugging of FORTRAN programs and facilitate the interpretation of loader symbol tables, the following list of run time subroutines' NREL entry points is given. Subroutines in this list are common to both the DOS and RDOS FORTRAN Run Time Libraries. This information can be obtained by the user by running an LFE analysis of library programs. However, the alternate names listed here are confined to those that represent meaningful entry points.

Subroutine Title	NREL Entry Point
ABSLT	ABS
AFRTN	AFRTN
AINT	.AINT
ALG	ALG .ALG1Ø
AMNXØ	AMAXØ AMINØ AMXØ
AMNX1	AMAX1 AMIN1 .AMX1 .AMN1
AMOD	.AMOD
ARCTAN	DATN2 DATN
ARGUM	FARGU
ARYSZ	ARYSZ
ATN	ATN2 ATN
BASC	.BASC
BDASC	.BDASC
BREAK	BRK
CABS	.CABS
CADD	CSUB CAD
CATIN	CATIN IMIO
CCEQ	CEQ1
CCOS	CCOS
CDIV	CDV
CEXPO	CEXPO
CGT	CGT
CHSAV	CHSAV CHRST
CLIP	CLP
CLOAD	CFLD
CLOG	CLOG
CMPLX	CMPLX
CMUL	CMUL
CNEG	CNEG
COMP	COMP
CONJG	CONJ
COS	CS SN
COSIN	DCS DSN
COUT	COUT CIN
CPWR	CPW1
CPYAR	CPYARG CPYLS
CRCX1	CRX2
CRCX2	DCRX2
CSIN	CSIN
CSQRT	CSQRT
CSTOR	CFST
CXFL1	CIX
CXFL2	DCIX
DBREAK	DBRK
DCABS	.DCAB
DCADD	DCSUB DCAD
DCCEQ	CEQ2
DCCOS	DCCOS

Subroutine Title	NREL Entry Point
DCDIV	DCDV
DCEXPO	CEXP
DCLOD	DCFLD
DCMPLX	DCMPLX
DCMUL	DCMUL
DCPWR	CPW2
DCSIN	DCSIN
DCSQR	DCSQR
DCSTR	DCFST
DDCLO	DCLOG
DELET*	DELET
DEXPC	DXPC
DEXPO	DEXP
DFL	DFL DFS DFA DFB DFM DFD DFXL DFLX DFSG DFLE DFLT DFGE DFGT DFEQ DFNG
DIM	.DIM
DIPWR	DIPWR
DLOG	DLOG .DLG1
DMNX1	DMAX1 DMIN1 .DMN1 .DMX1
DMOD	.DMOD
DPOLY	DPLY2
DPWER	DPW
DREAL	DREAL DAIMA
DSIGN	DSIGN DSYGN
DSINH	DSHIN DSNH
DSQRT	DSQR
DTANH	DTNH
EXP	EXPO
EXPC	XPC
FALOC	FALOC
FARGØ	FRG1 FRGØ
FCLOS*	FCLOS
FFILE	FFIL
FINIT	FINIT
FL	FL FS FA FB FM FD FXL FLX FSG FNG FLE FGT FLT FGE FEQ
FLINK	SAVØ SAV2 SAV3 RSTR QRSTR
FLIP	FLP FLPØ
FOPEN	FOPEN
FPWER	FPW
FREAD	FREAD FRWRIT BRD BWR
FREDI	FREDI
FRGLD	FRGLD
FSBR	FSBR FSUBA
FSEEK	FSEEK
I	.I
IABS	.IABS

* DOS only.

Subroutine Title	NREL Entry Point
IDIM	.IDIM
IDINT	.IDIN
IFIX	.IFIX
INT	.INT
IPWER	IPWR
ISIGN	.ISIG
LDØ	LDØ LD1 LD2 STØ ST1 ST2
LDREG	LDR1 LDR2
LDSTN	LDB STB
LE	LE LT GE GT
MAD	MAD MADO
MNMXØ	MAXØ MINØ
MNMX1	MAX1 MIN1 .MX1 .MN1
MOD	.MOD
MOVE	MOVE CMOVE
MOVEF	MOVEF
MVBT	MVBT MVBC
MVF	MVF
MVZ	MVZ
NFRTN	NFRTN
NPTR1	NR
NPTR3	NR3
NRPTR	NR2
OVFLO	OVERF
PLY1	PLY1
RATN1	RATN
RATN2	RTN2
RCABS	CABS
RDCABS	RCAB
RDFLD	RDFLD RDFCH
READL	READL WRITL REDS WRITS
REAL	.REAL .AIMA
RIPWR	RIPWR
RTER	RTER RTEØ RTES
SDVD	SDVD
SIGN	SIGN SYGN
SINH	SHIN SNH
SMPY	SMPY
SQRT	SQR
STOP	STOP PAUSE
STREG	ST1 ST2
TAN	TN
TANGENT	DTN
TANH	TNH
THREA	ALLOC THREAD
WRCH	WRCH

Following is a list of loader titles and NREL entry points for subroutines found only in the RDOS FORTRAN Run Time Library.

Subroutine Title	NREL Entry Point
CFILW	CFILW
CLOSE	CLOSE FCLOS
DATE	DATE
DFILW	DELET DFILW
DIR	DIR
FACAL	AKILL ARDY ASUSP
FDELY	FDELY
FINTD	FINRV FINTD
FKILL	KILL QUIT
FOVLY	FOVEN FOVLD FOVRL FQTRL
FPEND	PEND SUSP
FPRI	PRI
FQTASK	FQTASK FQTCK
FSTAT	FSTAT
FSWAP	FBACK FCHAN FSWAP
FTASK	FTASK
FTIME	FGTIM FSTIM
FTMAX	TMAX .IXMT LNKPR LQTSC QTCNT SVVAR
FXMT	REC XMT XMTW
GTATR	GTATR
INIT	INIT
ITEST	ICLR ISET ITEST
MTI	FERTØ FERT1 FERTN .I
OPEN	APPEN OPEN OVOPN
RESET	RESET
TIME	TIME

APPENDIX B

USING EITHER A SIMPLIFIED INITIALIZER OR .I

The following program illustrates a simplified version of the non-real time FORTRAN initializer, .I . This simplified routine does not interface with an operating system and performs no blank common allocation. It is modular in structure, permitting only those portions to be used which are required by a particular assembly language program. The program EXAMPLE given in Appendix C illustrates just such a use of this simplified version of .I; EXAMPLE uses no number stack, so the number stack portion of .I is omitted in that program.

```
        .TITLE  INITIALIZER

;   THIS ROUTINE DOES NO CHECKING FOR AVAILABILITY OF
;   ADEQUATE MEMORY FOR STACK ALLOCATION...IT PRESUMES THERE IS
;   ENOUGH.  NEITHER DOES IT CALL UPON AN OPERATING SYSTEM.

        .ENT    INIT            ;   INITIALIZATION ENTRY
        .ENT    SP              ;   DEFINE THE "SP" STACK POINTER
        .ENT    NSP             ;   DEFINE THE "NSP" STACK POINTER
        .ENT    .NDSP           ;   DEFINE THE END OF THE NUMBER
                                ;   STACK
        .ENT    AFSE            ;   DEFINE THE END OF THE SUB-
                                ;   ROUTINE LINKAGE STACK
        .EXTN   FCALL           ;   LINKAGE CALL
        .EXTN   .MAIN           ;   ENTRY POINT OF MAIN PROGRAM
        .EXTD   QSP


        .ZREL                   ;   PAGE ZERO POINTERS
SP:     .BLK    1               ;   "SP" STACK POINTER
NSP:    .BLK    1               ;   NUMBER STACK POINTER
.NDSP:  .BLK    1               ;   END OF NUMBER STACK INDICATION
AFSE:   .BLK    1               ;   END OF SUBROUTINE STACK
                                ;   INDICATION
        .NREL

;  INITIALIZE THE "SP" STACK FOR "SPSIZ" WORDS

INIT:   LDA     Ø,@.USTHU         ;  FIRST AVAILABLE ADDRESS
        STA     Ø,SP              ;  ABOVE LOADED ROUTINES
        LDA     1,SPSIZ           ;  SIZE OF "SP" STACK
        ADD     1,Ø

;  INITIALIZE THE NUMBER STACK FOR "NSSIZ" WORDS

        STA     Ø,NSP             ;  FIRST WORD OF NUMBER STACK
        LDA     1,NSSIZ           ;  SIZE OF NUMBER STACK
        ADD     1,Ø
        STA     Ø,.NDSP           ;  END OF NUMBER STACK

;  INITIALIZE THE SUBROUTINE LINKAGE STACK
;  "FSP" IS CENTERED ABOUT AN INDEXABLE FRAME

        LDA     3,CENTER          ;  INDEXABLE CENTER
        ADD     Ø,3               ;  FIRST FRAME'S STACK POINTER
        STA     3,FSP             ;  ("FSP" IS DEFINED BY THE
                                  ;  FORTRAN PARAMETER TAPE)

;  ALLOCATE THE SIZE OF THE LINKAGE STACK

        LDA     1,LSSIZ           ;  INDICATION OF LINKAGE STACK
        ADD     Ø,1               ;  SIZE (IT IS COMPARED TO
        STA     1,AFSE            ;  "FSP" FOR OVERFLOW)

;  "QSP" MUST ALWAYS REMAIN IN A FIXED RELATION TO
;  "FSP" AND IS POSITIONED SUCH THAT ACCUMULATOR 2
;  CAN BE IMMEDIATELY FREED IF NECESSARY, E.G.
;               STA     2,@QSP

        LDA     1,QSPDS
        ADD     Ø,1               ;
        STA     1,QSP             ;

;  INITIALIZE THE FIRST STACK FRAME

        SUB     Ø,Ø               ;  VARIABLE LENGTH OF THIS
        STA     Ø,FLGT,3          ;  FRAME IS ZERO
        ADC     Ø,Ø               ;  SET THE PREVIOUS FRAME
        STA     Ø,FOSP,3          ;  TO -1 (INDICATES NO PREVIOUS
        FCALL                     ;  FRAME)
        .MAIN                     ;  CALL THE MAIN PROGRAM
        JMP     .                 ;  JMP TO SELF IF RETURN IS EVER
                                  ;  MADE
```

```
LNKSZ=   1ØØØ                        ;  LINKAGE STACK TOTAL SIZE
                                     ;  (CAN USE ALL AVAILABLE
                                     ;  MEMORY IF THE OPERATING
                                     ;  SYSTEM CALL .MEM IS USED)
SPSIZ:   4Ø                          ;  SIZE OF "SP" STACK
NSSIZ:   6ØØ                         ;  SIZE OF NUMBER STACK
                                     ;  (GOOD FOR 64. SINGLE
                                     ;  PRECISION FLOATING POINT
                                     ;  VARIABLES)
LSSIZ:   2ØØ-FFEL+LNKSZ              ;  LINKAGE STACK SIZE
QSPDS:   2ØØ+FAC2                    ;  QSP DISPLACEMENT FROM START
                                     ;  OF FRAME
.USTHU:   UST+USTHU                  ;  FIRST WORD AVAILABLE
                                     ;  ("UST" DEFINED BY THE
                                     ;  USER PARAMETER TAPE)
CENTER:  2ØØ                         ;  CENTER OF A STACK FRAME
         .END      INIT
```

If instead of writing a simplified initializer routine for a non-real time program, a user wishes to load .I and leave the details of allocating the run time stack to that program, there are several features which must be incorporated into the main assembly language program. First of all, .I must be referenced in an .EXTN statement. This will cause .I to be loaded and for it to gain control when the program is run.

Secondly, the main program must be .ENTered as .MAIN, and the label of its entry point must also be .MAIN . This is due to the fact that .I transfers control to the main program by means of

```
FCALL
.MAIN
```

.MAIN must be preceded by a non-negative integer stack length word describing the number of run time stack temporaries required by the program. If none are required, integer 0 should precede .MAIN so that it can call out to one or more run time routines.

Finally, .MAIN must be terminated by a simple .END statement (as opposed to an .END statement with an argument starting address).

The following assembly language program is one which causes a memory address to be printed out on the TTY printer. This address represents the highest memory address, HMA, available to the user or the lowest address in the symbol table, EST, in the case where the debugger is loaded with .MAIN. For a complete understanding of these terms and the system calls issued by this program, see the DOS User's Manual.

Those features which must be incorporated into any assembly language program using .I are enclosed with rectangles for emphasis. If an assembly language program requires run time stack storage, the appropriate integer (instead of zero as shown) must precede the beginning of .MAIN code.

```
  0001  MEMS
01
02                              ; PROGRAM TO DETERMINE THE HIGHEST AVAILABLE MEMORY
03                              ; ADDRESS, HMA, OR THE START OF THE SYMBOL TABLE
04                              ;
05                              ; THIS ASSEMBLY LANGUAGE  PROGRAM ISSUES RDOS SYSTEM CAL
06                              ; (SEE THE RDOS USER'S MANUAL)
07                              ; AND  IT ALSO ISSUES FORTRAN RUNTIME LIBRARY CALLS
08                              ;
09
10
11                              .TITLE  MEMS
12                              .EXTD   .WRCH
13                              .ENT    .MAIN
14                              .EXTN   FCALL,.BDASC,.I
15                              .NREL
16
17 00000'000000                 0
18 00001'006017        .MAIN:   .SYST
19 00002'001400                 .MEM            ; SYSTEM CALL TO GET THE HMA
20 00003'000400                 JMP     .       ; ERROR RETURN
21 00004'105000                 MOV     0,1
22 00005'020410                 LDA     0,BUF
23 00006'177777                 FCALL
24 00007'177777                 .BDASC          ; CHANGE BINARY TO DECIMAL IN ASCII
25
26 00010'020405                 LDA     0,BUF
27 00011'006001$                JSR     @.WRCH  ; TYPE THE VALUE ON THE TTY
28
29 00012'006017                 .SYST
30 00013'004400                 .RTN            ; SYSTEM CALL TO RETURN TO THE CLI
31 00014'000400                 JMP     .
32 00015'000034"BUF:            2*.BUF
33         000100 .BUF:         .BLK    100
34                              .END
```

```
    0002  MEMS

  BUF       000015'      1/22     1/26     1/32
  FCALL     000006'X     1/23
  .BDAS     000007'X     1/24
  .BUF      000016'      1/32     1/33
  .I        177777 X
  .MAIN     000001'      1/18
  .WRCH     000001$X     1/27
```

APPENDIX C

ILLUSTRATIVE PROGRAMS

Appendix C contains a series of assembly language routines which illustrate the use of the library. The first two are subroutines found in the library. BREAK separates an SPFL number into its integral and fractional components, and demonstrates the use of the number stack. IDINT truncates a DPFL number, and expresses the result as an integer. IDINT utilizes the FSAV/FRET stack managers and it calls FARG, the argument fetching routine.

The illustrative program entitled EXAMPLE computes the product of any two positive integers provided the product does not exceed 65535. The multiplier and multiplicand, entered via the TTY keyboard, are separated by a comma and are followed by a carriage return. The product is returned on the printer, followed by a carriage return and line feed, after which the program is ready to accept new data. Typical program output is as follows:

```
2,3
6
28,9
252
31500,2
63000
```

No error checking is done, so that input data which is non-numeric or causes a product outside the acceptable range results in spurious results being given.

EXAMPLE also illustrates the use of parts of the simplified stack initializer, .I, given in Appendix B.

A flowchart of EXAMPLE follows the program listing to clarify the program coding.

```
  0001  BREAK
01                          ;BREAK UP SINGLE PRECISION FLOATING POINT NUMBER
02                          ;INTO AN INTEGER AND A FRACTIONAL PART
03                          ;INPUT: ARGUMENT ON TOP OF NUMBER STACK
04                          ;OUTPUT:        FRACTION REPLACES INPUT
05                          ;       INTEGER IN AC1
06
07                          .TITLE  BREAK
08
09                          .ENT    FBRK1
10                          .EXTD   NSP,SP
11                          .EXTD   FXFL1,FLFX1,FRLD1,FSB1
12
13                          .ZREL
14 00000-000000'.BRK1:      BRK
15        006000-           FBRK1 = JSR @.BRK1
16
17                          .NREL
18 00000'056002$BRK:        STA     3,@SP
19 00001'010002$            ISZ     SP
20 00002'020001$            LDA     0,NSP
21 00003'000005$            FRLD1                   ;COPY ARGUMENT
22 00004'000004$            FLFX1
23 00005'000005             FAC0-FZD                ;FLOAT TO FIX IT
24 00006'000003$            FXFL1
25 00007'000005             FAC0-FZD                ;FIX TO FLOAT INTEGER
26 00010'000006$            FSB1                    ;GET FRACTION = ARG - FLOAT(I)
27 00011'021605             LDA     0,FAC0,3
28 00012'014002$            DSZ     SP
29 00013'032002$            LDA     2,@SP
30 00014'001000             JMP     0,2
31
32
33                          .END    ;END OF BREAK ROUTINE
```

```
    0002  BREAK

   BRK      000000'     1/14    1/18
   FBRK1    006000-     1/15
   FLFX1    000004$X    1/22
   FRLD1    000005$X    1/21
   FSB1     000006$X    1/26
   FXFL1    000003$X    1/24
   NSP      000001$X    1/20
   SP       000002$X    1/18    1/19    1/28    1/29
   .BRK1    000000-     1/14    1/15
```

```
  0001  IDINT
01                                   ;TRUNCATE D.P NUMBER AT SECOND FORTRAN ADDRESS
02                                   ;TO NEAREST INTEGER IN MAGNITUDE
03                                   ;AND LEAVE RESULT AT FORTRAN ADDRESS
04                                   ;BELOW CALL
05
06                                           .TITLE  IDINT
07                                           .ENT    ID.NT,XI.
08                                           .EXTN   FSAV,FRET
09                                           .EXTD   FLFX2,FFLD2
10                                           .EXTD   .FARG
11                                   .ZREL
12                        XI.:
13 00000-000000'ID.NT:    .IDIN-2
14                                   .NREL
15 00000'177777                      FSAV
16 00001'000002 C2:                  2
17 00002'020777 .IDIN:               LDA     0,C2
18 00003'006003$                     JSR     @.FARG
19 00004'000002$                     FFLD2
20 00005'100012                      @TMP+1-FZD
21 00006'000001$                     FLFX2
22 00007'100011                      @TMP-FZD
23 00010'177777                      FRET
24
25                                   .END     ;END OF TRUNCATION OF D.P NUMBER
```

```
  0002  IDINT

C2        000001'      1/16     1/17
FFLD2     000002$X     1/19
FLFX2     000001$X     1/21
FRET      000010'X     1/23
FSAV      000000'X     1/15
ID.NT     000000-      1/13
XI.       000000-      1/12
.FARG     000003$X     1/18
.IDIN     000002'      1/13     1/17
```

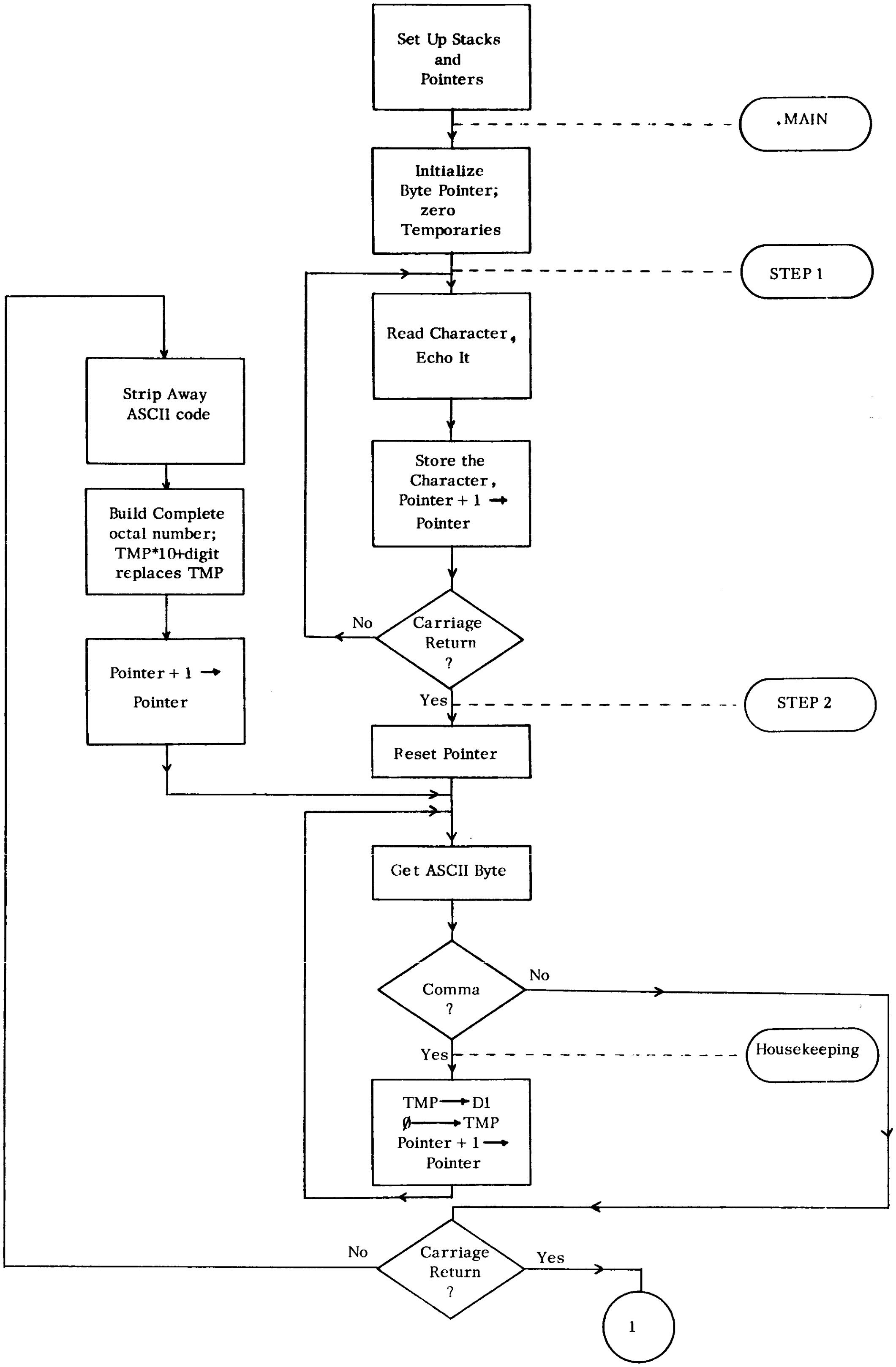
```
---
  ØØØ1   EXAMP
Ø1
Ø2                             .TITLE EXAMPLE
Ø3                ;    ASSEMBLE WITH PARF  AND PARU
Ø4
Ø5
Ø6
Ø7                             .EXTN FCALL,MPY,FRET,.BDASC,MPYØ
Ø8                             .EXTD   .LDB,.COUT,.STB
Ø9                             .EXTD   .CIN,SP,AFSE,QSP
1Ø
11        177612               D1=TMP+1                  ; FIRST INPUT DIGIT
12        177613               BPTR=D1+1                 ; FIXED BYTE POINTER TO BEGIN
13        177614               INPT=BPTR+1               ; MOVING POINTER
14
15
16
17
18
19
2Ø                             .NREL
21 ØØØØØ'Ø22524 .I:            LDA      Ø,@.USTHU        ;SET UP SP
22 ØØØØ1'Ø4ØØØ5$               STA      Ø,SP
23 ØØØØ2'Ø24527                LDA      1,SPSIZ
24 ØØØØ3'123ØØØ                ADD      1,Ø
25 ØØØØ4'Ø34521                LDA      3,CENTER         ; SET UP FSP
26 ØØØØ5'117ØØØ                ADD      Ø,3
27 ØØØØ6'Ø54Ø16                STA      3,FSP
28 ØØØØ7'Ø24517                LDA      1,LSSIZ          ; SET UP AFSE
29 ØØØ1Ø'1Ø7ØØØ                ADD      Ø,1
3Ø ØØØ11'Ø44ØØ6$               STA      1,AFSE
31 ØØØ12'Ø2452Ø                LDA      1,QSPDS          ; SET UP QSP
32 ØØØ13'1Ø7ØØØ                ADD      Ø,1
33 ØØØ14'Ø44ØØ7$               STA      1,QSP
34 ØØØ15'1Ø24ØØ                SUB      Ø,Ø              ; SET UP FIRST STACK FRAME
35 ØØØ16'Ø416ØØ                STA      Ø,FLGT,3
36 ØØØ17'1Ø2ØØØ                ADC      Ø,Ø
37 ØØØ2Ø'Ø416Ø1                STA      Ø,FOSP,3
38 ØØØ21'177777                FCALL
39 ØØØ22'ØØØØ25'               .MAIN
4Ø ØØØ23'ØØØ776                JMP      .-2              ;REPEAT JOB
41 ØØØ24'ØØØØØ4                4
42
43 ØØØ25'Ø2Ø51Ø .MAIN:         LDA      Ø,PINPF          ;CONSTRUCT MOVING PTR
44 ØØØ26'Ø41614                STA      Ø,INPT,3
45 ØØØ27'Ø41613                STA      Ø,BPTR,3         ; CONSTRUCT FIXED POINTER
46 ØØØ3Ø'1Ø24ØØ                SUB      Ø,Ø              ; INITIALIZE FLAGS
47 ØØØ31'Ø41612                STA      Ø,D1,3
48
49
```

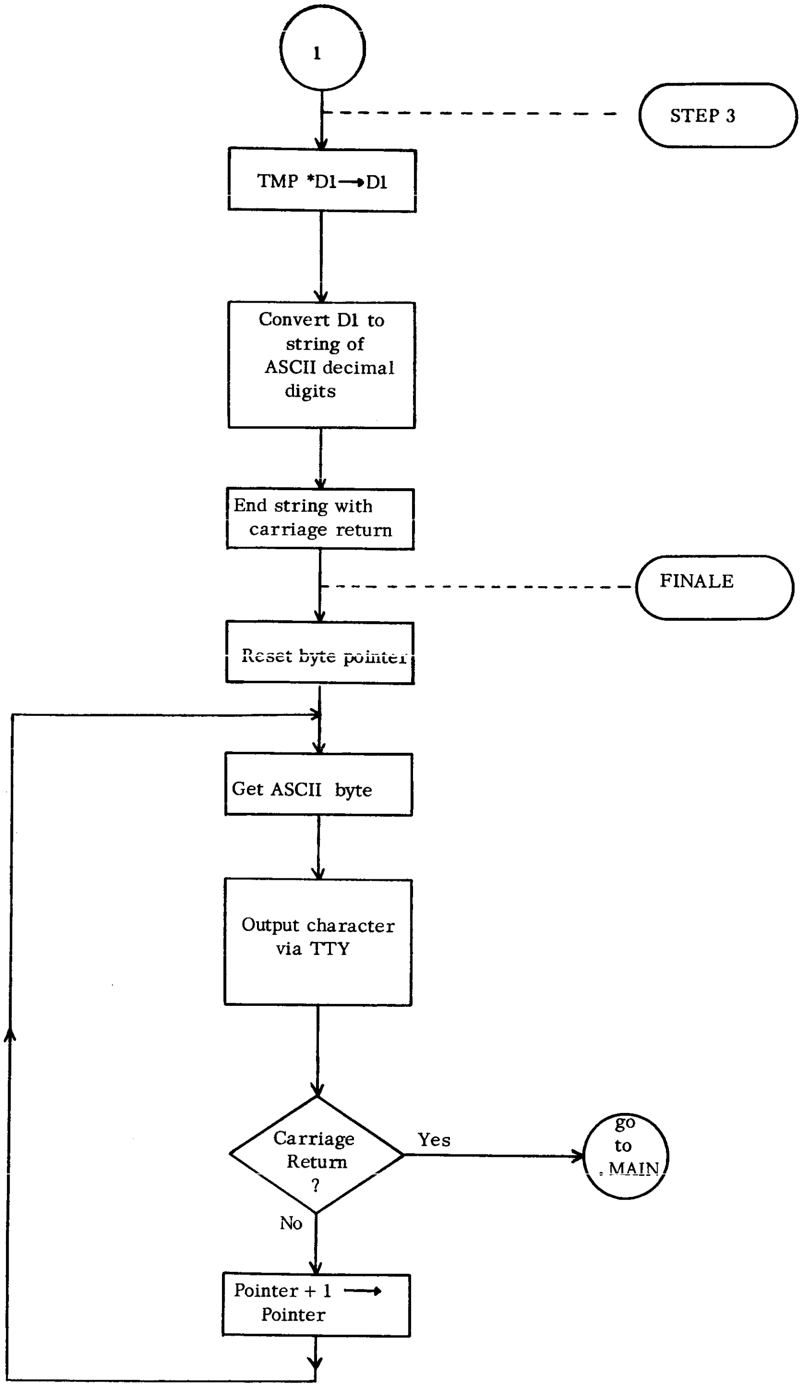
```
---
↑ 0002  EXAMP



00032'006004$STP1:   JSR     @.CIN           ;  INPUT CHARACTER
00033'105000         MOV     0,1
00034'045611         STA     1,TMP,3
00035'021614         LDA     0,INPT,3        ;  GET BYTE POINTER
00036'006003$        JSR     @.STB           ; STORE THE BYTE
00037'011614         ISZ     INPT,3          ; UPDATE POINTER
00040'025611         LDA     1,TMP,3
00041'020467         LDA     0,CRTN
00042'122404         SUB     1,0,SZR         ; IS IT A CR?
00043'000767         JMP     STP1            ; REPEAT
00044'041611         STA     0,TMP,3
00045'021613 STP2:   LDA     0,BPTR,3        ; GET BEGINNING BYTE POINTER
00046'041614         STA     0,INPT,3
00047'006001$        JSR     @.LDB           ; GET TOP DIGIT
00050'030466         LDA     2,CMA           ; IS IT A COMMA?
00051'132415         SUB#    1,2,SNR
00052'000416         JMP     HSKP            ; YES.
00053'030455         LDA     2,CRTN          ;  IS IT A CR?
00054'132415         SUB#    1,2,SNR
00055'000423         JMP     STP3            ;  YES

00056'020455         LDA     0,C017          ; NO, STRIP CODE
00057'123400         AND     1,0
00060'030454         LDA     2,C012
00061'025611         LDA     1,TMP,3
00062'177777         MPY
00063'045611         STA     1,TMP,3
00064'021614         LDA     0,INPT,3        ; BUMP POINTER
00065'101400         INC     0,0
00066'041614         STA     0,INPT,3
00067'000760         JMP     STP2+2
00070'021611 HSKP:   LDA     0,TMP,3
00071'041612         STA     0,D1,3
00072'102400         SUB     0,0
00073'041611         STA     0,TMP,3
00074'021614         LDA     0,INPT,3
00075'101400         INC     0,0
00076'041614         STA     0,INPT,3
00077'000750         JMP     STP2+2
```

```
---
† 0003  EXAMP
01
02
03
04 00100'025612 STP3:   LDA     1,D1,3
05 00101'031611         LDA     2,TMP,3
06 00102'177777         MPY0
07 00103'045612         STA     1,D1,3
08 00104'021613         LDA     0,BPTR,3          ; INITIALIZE BYTE POINTER
09 00105'000021'        FCALL                     ; CONVERT MS WORD TO ASCII
10 00106'177777         .BDASC
11 00107'024421         LDA     1,CRTN
12 00110'006003$        JSR     @.STB
13 00111'021613 FINALE: LDA     0,BPTR,3          ;  INITIALIZE POINTER
14 00112'041614         STA     0,INPT,3
15 00113'021614         LDA     0,INPT,3          ;  GET BYTE POINTER
16 00114'006001S        JSR     @.LDB             ;  AC1 GETS BYTE
17 00115'121000         MOV     1,0
18 00116'006002$        JSR     @.COUT            ; OUTPUT IT FROM AC0
19 00117'030411         LDA     2,CRTN
20 00120'112415         SUB#    0,2,SNR           ;  LAST CHARACTER?
21 00121'177777         FRET                      ;  YES, RETURN
22 00122'011614         ISZ     INPT,3            ;  NO, INCREMENT POINTER
23 00123'000770         JMP     FINALE+2          ;  AND REPEAT
24 00124'000407 .USTHU: UST+USTHU
25 00125'000200 CENTER: 200
26 00126'001167 LSSIZ:  200-FFEL+LNKSZ
27 00127'000007 ZSPDS:  200+FAC2
28 00130'000015 CRTN:   015
29 00131'000040 SPSIZ:  40
30 00132'000007 QSPDS:   200+FAC2
31 00133'000017 C017:   17
32 00134'000012 C012:   12
33       001000 LNKSZ=1000
34 00135'000276"PINPF:    2*INPF
35 00136'000054 CMA:    054
36       000100 INPF:   .BLK 100
37       000000'        .END    .I
```

```
 ØØØ4  EXAMP

AFSE    ØØØØØ6$X    1/3Ø
BPTR    177613      1/12    1/13    1/45    2/15    3/Ø8    3/13
CØ12    ØØØ134'     2/27    3/32
CØ17    ØØØ133'     2/25    3/31
CENTE   ØØØ125'     1/25    3/25
CMA     ØØØ136'     2/18    3/35
CRTN    ØØØ13Ø'     2/11    2/21    3/11    3/19    3/28
D1      177612      1/11    1/12    1/47    2/36    3/Ø4    3/Ø7
FCALL   ØØØ1Ø5'X    1/38    3/Ø9
FINAL   ØØØ111'     3/13    3/23
FRET    ØØØ121'X    3/21
HSKP    ØØØØ7Ø'     2/2Ø    2/35
INPF    ØØØ137'     3/34    3/36
INPT    177614      1/13    1/44    2/Ø7    2/Ø9    2/16    2/31  2/33  2/39
                    2/41    3/14    3/15    3/22
LNKSƵ   ØØ1ØØØ      3/26    3/33
LSSIƵ   ØØØ126'     1/28    3/26
MPY     ØØØØ62'X    2/29
MPYØ    ØØØ1Ø2'X    3/Ø6
PINPF   ØØØ135'     1/43    3/34
QSPDS   ØØØ132'     1/31    3/3Ø
SP      ØØØØØ5$X    1/22
SPSIƵ   ØØØ131'     1/23    3/29
STP1    ØØØØ32'     2/Ø4    2/13
STP2    ØØØØ45'     2/15    2/34    2/42
STP3    ØØØ1ØØ'     2/23    3/Ø4
ƵSPDS   ØØØ127'     3/27
.BDAS   ØØØ1Ø6'X    3/1Ø
.CIN    ØØØØØ4$X    2/Ø4
.COUT   ØØØØØ2$X    3/18
.I      ØØØØØØ'     1/21    3/37
.LDB    ØØØØØ1$X    2/17    3/16
.MAIN   ØØØØ25'     1/39    1/43
.STB    ØØØØØ3$X    2/Ø8    3/12
.USTH   ØØØ124'     1/21    3/24
```

Flow Chart of Illustrative program, "EXAMPLE"




1
STEP 3
TMP *D1→D1
Convert D1 to string of ASCII decimal digits
End string with carriage return
FINALE
Reset byte pointer
Get ASCII byte
Output character via TTY
Carriage Return ?
Yes
go to MAIN
No
Pointer + 1 → Pointer

CHANGES FROM REVISION 2 TO REVISION 3 OF THE FORTRAN IV RUN TIME LIBRARY USER'S MANUAL

Substantive changes are described in the following list. Typographical corrections are not included.

Page 1-26 Comparisons are now given of execution times on the NOVA, NOVA 1200 and NOVA 800 series computers.

Page 1-27 The category formerly entitled "Supporting Routines and Displacements" is now entitled simply "Supporting Routines."

Due to extensive changes throughout this revision of the FORTRAN Run Time Library, the sizes of many routines have changed. The pages containing size changes to previously existing routines are as follows:

2-7, 2-14, 3-5, 3-8, 3-11, 3-15, 3-16, 3-17, 3-19, 3-20, 3-21, 3-22
3-24, 3-28, 4-8, 4-12, 4-13, 4-16, 4-19, 4-20, 4-21, 4-23, 4-24, 4-26,
4-27, 6-5, 7-17, 7-19, 8-4, 10-7, 10-9, 10-10, 10-12, 10-14, 10-16,
11-3, 11-6, 11-9, 11-10, 11-11, 11-13, 11-24, 11-28, 12-9, 13-6,
E-9, E-11, E-15, E-16, E-17, E-18, E-19, E-20, E-22, E-23

Page 3-17 Rounding now occurs when a single precision floating point number is stored.

Page 4-21 Rounding now occurs when a double precision floating point number is stored.

Pages 10-16f The DOS and RDOS initializers have additional entry points to return control to either the CLI, debugger, or task scheduler.

Page 11-6 The channel status array for CHSAV/CHRST has been changed from a six to a two word array.

Page 11-8 If a file is open when DELETE is called, the file is first closed, and only then is it deleted.

Page 11-11 If an attempt is made to open a non-existent file, a new file is created and then is opened.

Page 12-9f The calling sequence for the run time error routines has changed and the error code structure has also been modified. Run time error messages are now more explicit. After a fatal error, control goes to either the single task or multitask initializers which transfer control to either the debugger, the CLI, or to the task scheduler as appropriate.

CHANGES FROM REVISION 2TO REVISION 3 OF THE FORTRAN IV RUN TIME LIBRARY USER'S MANUAL (Continued)

Appendix A To aid in the debugging process, NREL entry points have been assigned to the start of all run time routines. These entry points also facilitate the interpretation of loader symbol tables. This appendix no longer lists summary information for the library, but does list all loader titles and all NREL entry points for routines in the library.

Appendix E Numerous additional real time subroutines are now available in the RDOS FORTRAN Library. These routines are found principally in three new categories: Swap and Overlay Commands, File and I/O Commands, and Bit Manipulation Commands.

Appendix F The FORTRAN Parameter Listing is now given in Appendix F.

This addendum updates and corrects revision 03 of the FORTRAN IV Run Time Library User's Manual so that this manual may be used with revision 02 of the Real Time Disk Operating System. Minor changes are indicated by page number on the following list. This information should be annotated on the appropriate pages of the User's manual. Following the minor changes is a series of new and changed pages which must either replace existing pages in the manual or must be inserted into the manual. Changed information on replacement pages will be indicated by a heavy vertical line in the outside margin. New pages to be inserted into the manual will contain a page number of the form

i-j-k

where i is the chapter or appendix number, j is the page in the chapter which precedes the new page, and k is the number of the page in the insertion series. Thus page E-2 is a replacement page for the current page E-2, while page E-14-1 is a new page which must be inserted after the current page E-14.

Page where Change Occurred	Change
9-5	. SOSW has been removed from the FPZER module. A new flag, . DSI, is used in its place; this flag is defined in the SOS library.
11-3	Channel table assignment initialization is accomplished by the run time stack initializer, . I, under RDOS. There is no CATIN module under RDOS. The sizes of the single and multitask RDOS initializers have changed to 253_8 NREL locations (single task . I) and 402_8 NREL locations (multitask . I). The I/O Channel Assignment Table is built in . I's stack frame.
11-6	A three-word integer array must be allocated for channel status information in CHSAV/CHRST. This module is now 53 octal NREL words in length.
11-8	DELET (now equivalent to DFILW) and RLSE are found in separate modules under RDOS.
11-9, E-55	FCLOS is now equivalent to CLOSE. The supporting routines for CLOSE are as follows: FSAV, FRET; . CPYL, . IOCAT, . RTER . The size of this routine is 47 octal NREL locations.
11-10	Supporting routines for FFILE under RDOS are as follows: FCALL, FCLOS, FRET, FSAV, FSEEK, IOPTR; . CPYA, . FCAL, . IOCAT, . RTER. The size of this routine is 1 ZREL and 64 octal NREL locations.
11-11	Up to 64 FORTRAN channel numbers are allowed, 0 through 63. Supporting routines for FOPEN under RDOS are as follows: . DSI, FRET, FSAV, IOPTR; . CPYL, . IOCAT, . RTER . Subroutine size under RDOS has changed to 1 ZREL and 147 octal NREL locations.
11-24	Supporting routines for FSEEK under RDOS are as follows: FRET, MPY; . CPYL, . IOCAT, . RTER . The size of this routine is 51 octal NREL locations.

Page where Change Occurred	Change
11-27, 11-28	No parameter is input via AC1 to .WRITL. AC1 inputs the number of bytes to be written (or read) in .WRTS(or .REDS). Supporting routines for READL and WRITL under RDOS are as follows: .DSI, FCALL, FRET, FSAV, OPEN; .FARG, .IOCAT, .LDBT, .STBT. The size of the READL/WRITL module is 4 ZREL and 305 octal NREL locations.
13-5	Supporting routines for FREDI under RDOS are as follows: FRET, FSAV, MPY0, .OFLO; AFSE, .CPYA, .FRGL, QSP.
A-1	CATIN is now found in the DOS library only.
A-2	Insert FOVLD (title) OVLOD (entry) before FOVLY. Insert RLSE (title) RLSE (entry) before TIME.
E-17, E-15, E-16	Supporting routines for ASUSP, AKILL, and ARDY are as follows: FRET, TAKIL, TAPEN, TAUNP; .CPYL. The subroutines' module's size is 22 octal NREL locations.
E-18	Supporting Routines for FTASK are as follows: CTASK, FRET; .CPYL. The subroutine size is now 14 octal NREL locations.
E-19	The KILL task call is now part of the multitask scheduler module. Its calling sequence remains the same.
E-20, E-23	Supporting routines for REC, XMT, and XMTW are now as follows: FRET, RECC, XMTT, XMTTW; .CPYL. The subroutines' module's size is 33 octal NREL locations.
E-21	Supporting Routines for SUSP are as follows: FRET, TPEND; .CPYL. The subroutine size is now 5 NREL locations. The suspended task remains suspended until readied by an ARDY or RELSE call.
E-22	Supporting routines for PRI are as follows: FRET, TPRI; .CPYL. Subroutine size is 6 NREL locations.
E-29, E-30	Delete the date parameter from the calling sequences given for FGTIM and FSTIM; change the integer argument count from 4 to 3. The size of this module is reduced from 41 to 24 octal NREL locations.

Page where Change Occurred	Change
E-46, E-43	Supporting routines for FOVRL and FOVLD are as follows: FRET, TOVLD, TOVRL; .CPYL, .IOCAT. The subroutines' module's size is 45 octal NREL locations. The overlay name may be used in place of the overlay number if the name was declared in an .ENTO or OVERLAY statement.
E-50	Supporting Routines for OVOPN are as follows: .DSI, FRET, IOPTR; .CPYL, .IOCAT . The subroutine size is now 174 octal NREL locations.
E-57, E-60, E-65	DIR, INIT, and RLSE may use directory names instead of device names as arguments.
E-64	Supporting Routines for RESET under RDOS are as follows: FRET; .IOCAT . The subroutine's size is 21 octal NREL locations.
E-71	Bit position indicator may be any integer from 0 to 15_{10}.
E-74	Information in the CHANTASK statement may be selectively overridden at relocatable load time by means of RLDR local switches /C and /K. If channel/task number specification information is given at relocatable load time, the CHANTASK statement may be omitted. If the CHANTASK statement is omitted and /C and /K RLDR switches are not used, 1 task and 16_{10} channels are allocated by default. A minimum of 16 logical FORTRAN channels will be allocated, even though fewer may have been specified by the user. However, only the number specified may be used simultaneously.

APPENDIX D

ARRAY STRUCTURE AND HANDLING

Arrays are ordered sets of data, arranged in up to 128 dimensions (see FORTRAN IV User's Manual). The library's allocation of array area presents an exception to the general rules of stack structure given in Chapter 1.

Arrays may be defined to be any size within the limits of available memory storage. Array elements are numbers expressed in packed form, and these are referenced by integer subscripts (one for each dimension). Values are assigned to array elements so that the first subscript varies most rapidly, then the second subscript, and so on.

An array is allocated on a caller's stack by appending the needed number of locations to the current end of the caller's stack. Array allocation is accomplished by FALOC, whose caller's stack is then extended by the size of the designated array. FALOC also adjusts the caller's FLGT so that any further creation of stacks will follow the end of the array.

Elements of an array are not referenced by the conventional FORTRAN addressing scheme. Instead, routines FSBR, FSUB are used to calculate the absolute addresses of an array element. The address in this instance is an absolute NREL address instead of a relative stack displacement.

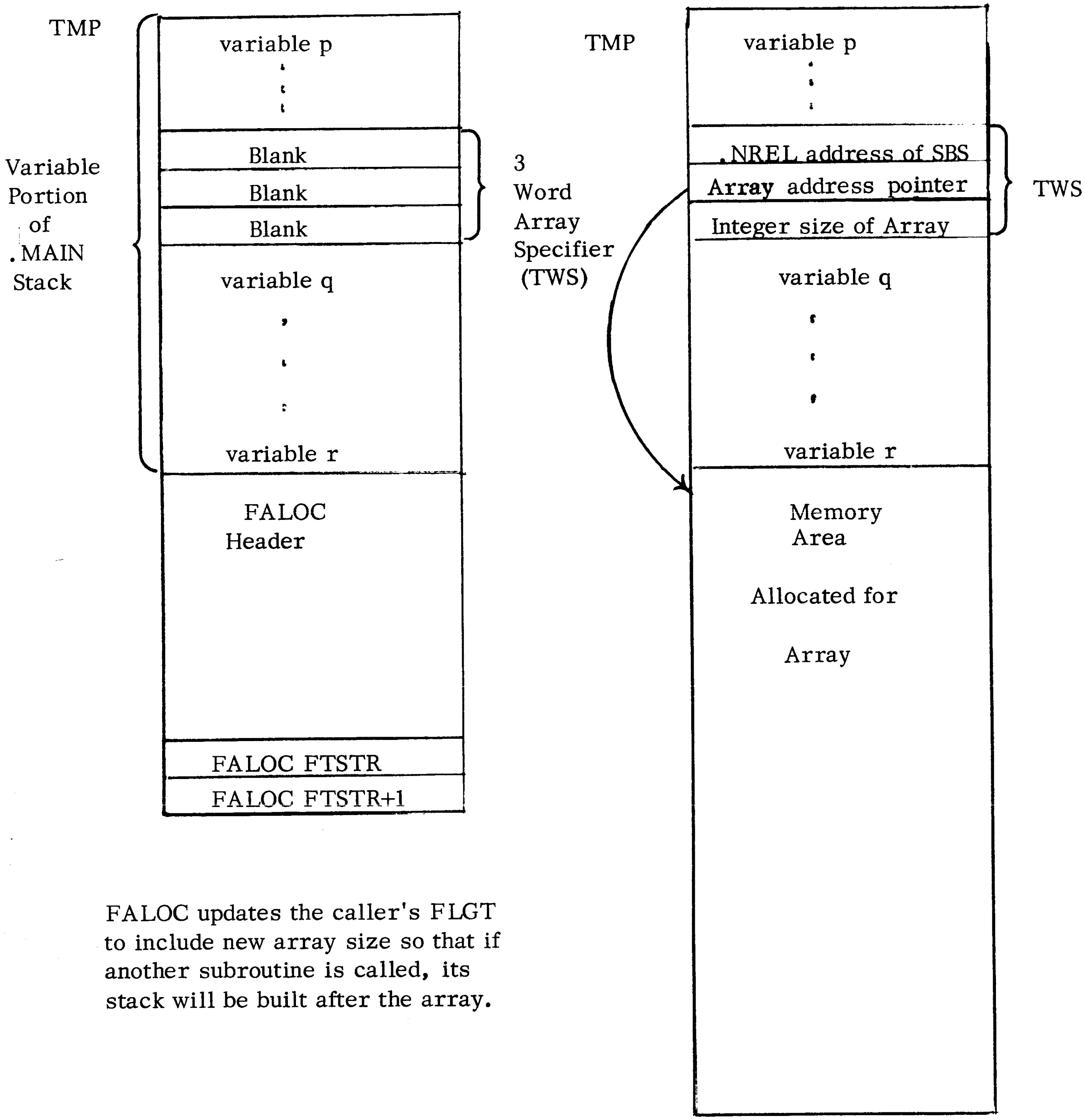
The following picture shows memory maps before and after FALOC execution.

MEMORY MAPS BEFORE AND AFTER FALOC EXECUTION

FORTRAN statement DIMENSION A(x, y, z) generates a call to FALOC.

Map when FALOC is called, its stack is allocated, but before FALOC execution.

Map after FALOC execution.

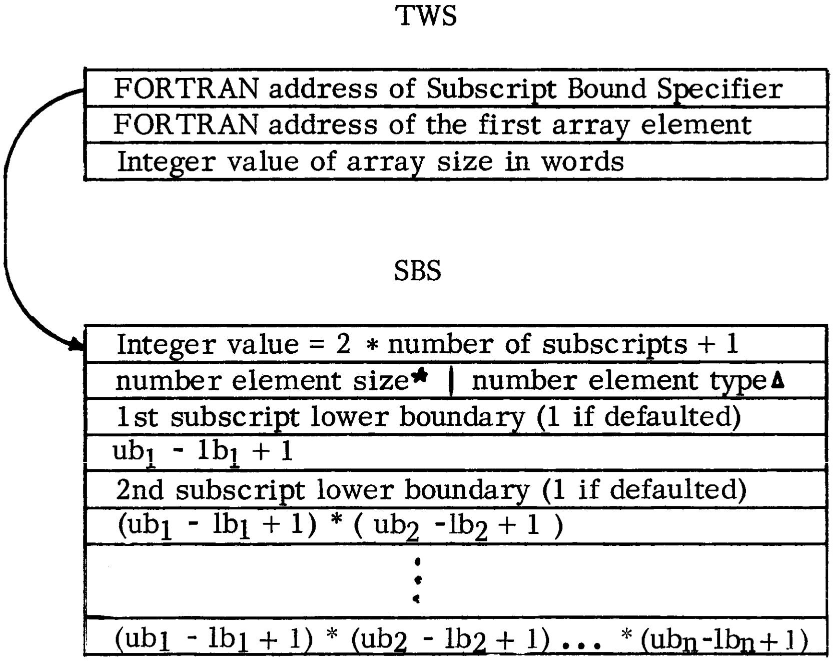


FALOC updates the caller's FLGT to include new array size so that if another subroutine is called, its stack will be built after the array.

Two tables are needed by the array handling routines in order to accomplish the tasks of array allocation and element addressing. A third table is required when an array is to be redimensioned and passed as a dummy argument.

The first of these is the Subscript Bound Specifier (SBS). This table describes each subscript's boundaries and specifies the type of number element stored in the array. Since array indices may begin at values other than 1, both upper and lower index values are specified in the table.

A smaller table, the Three Word Array Specifier (TWS), contains a pointer to the SBS, a pointer to the beginning of the array, and the total number of words (not elements) in the array.



TWS

FORTRAN address of Subscript Bound Specifier
FORTRAN address of the first array element
Integer value of array size in words

SBS

Integer value = 2 * number of subscripts + 1	
number element size★	number element type▲
1st subscript lower boundary (1 if defaulted)	
$ub_1 - lb_1 + 1$	
2nd subscript lower boundary (1 if defaulted)	
$(ub_1 - lb_1 + 1) * (ub_2 - lb_2 + 1)$	
⋮	
$(ub_1 - lb_1 + 1) * (ub_2 - lb_2 + 1) \ldots * (ub_n - lb_n + 1)$	

Three Word Array Specifier and Subscript Bound Specifier Tables for an array of the general form Array A (lower $bound_1$:upper $bound_1$,... lower $bound_n$: upper $bound_n$).

★ i.e., the number of words in the packed form of the number element type: 1 for integers, 2 for SPFL's, 4 for DPFL's and single precision complex number, and 10_8 for double precision complex numbers.

▲ 1↔ integer, 2 ↔ SPFL, 3 ↔ DPFL, 4↔ Single precision complex, 5↔ double precision complex.

Space for the TWS is reserved on the caller's stack before calling FALOC, which then fills in the three word table with the appropriate information. The SBS, on the other hand, is built in NREL memory by the compiler for an array defined in the main program (or in a subroutine if the array is not a passed argument). If an array is to be redimensioned and passed to a subroutine as a dummy argument, a new SBS is created in the run time stack, reflecting the new index values. Given that the array is passed to the subroutine as an argument, the subroutine accesses the array via the new SBS. Array redimensioning and passing is done by FREDI.

As with FALOC, FREDI requires that a 3 word area be reserved on the caller's stack into which it builds the new TWS. FREDI also requires the address of the array being passed, and the address of another table called the Special Subscript Bound Specifier (SSBS). The SSBS is required so that re-dimensioning can be accomplished. SSBS is similar to SBS except that in place of literal values and cumulative partial products for each index, the addresses of the upper and lower bounds for each index are given. The SSBS is built by the compiler in NREL memory.

Special Subscript Bound Specifier

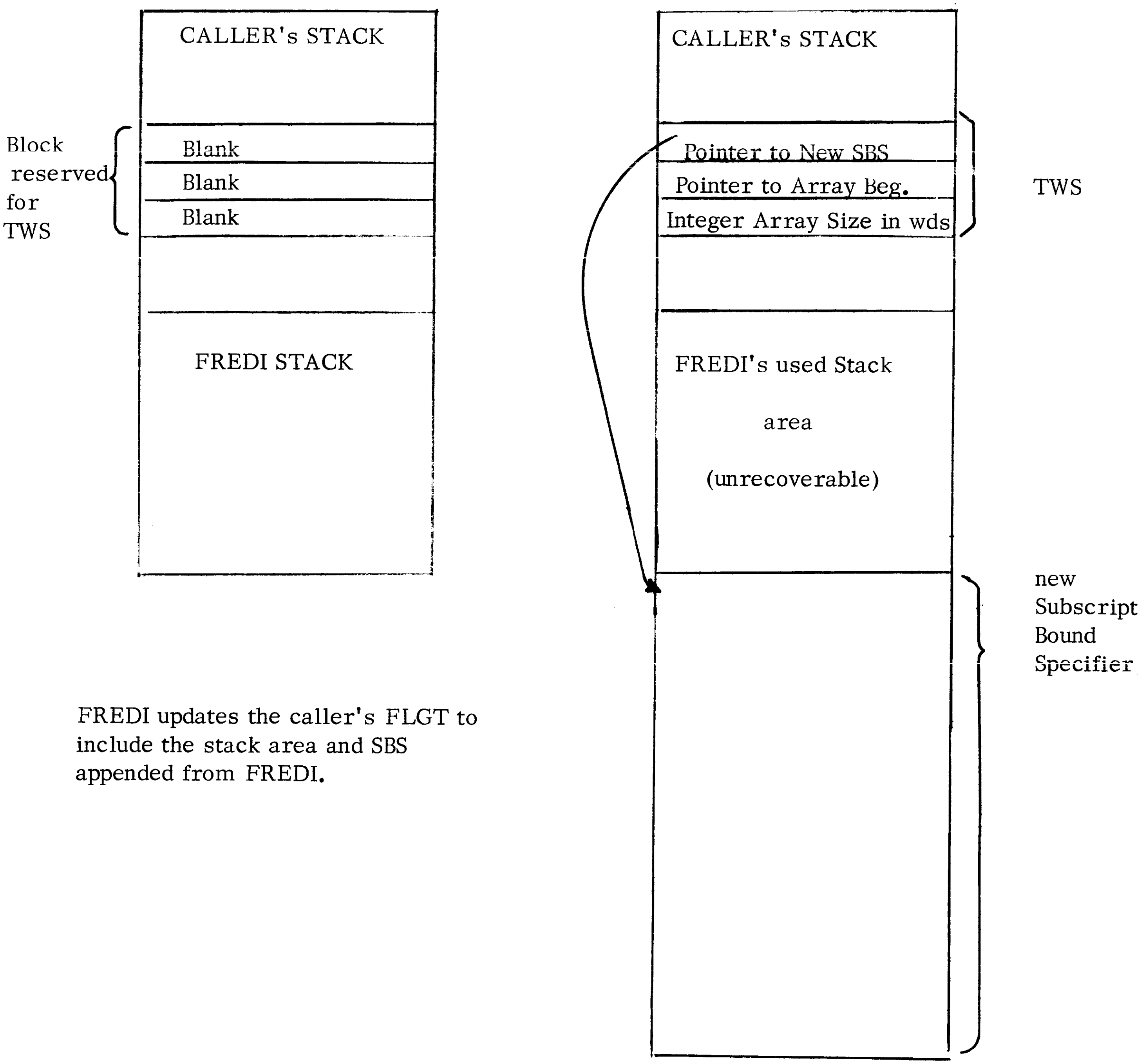
Integer value = 2 * number of subscripts + 1	
number element size★	number element type▲
address of 1st subscript lower bound	
address of 1st subscript upper bound	
address of 2nd subscript lower bound	
address of 2nd subscript upper bound	
⋮	
address of nth subscript upper bound	

The new SBS, built by FREDI for the caller, is appended to its own stack, and the TWS is built into the area of the caller's stack reserved for that purpose. The stack area used by FREDI in its computations becomes a waste area, unused by the caller upon completion of FREDI's operation. FREDI adjusts the caller's FLGT, making the new SBS part of the caller's stack and protecting it from being overwritten by future stacks. The array itself is not appended to the caller's stack, since it is already defined by the calling program.

★ Same as for ordinary Subscript Bound Specifer

▲ Same as for ordinary Subscript Bound Specifier

MEMORY MAPS BEFORE AND AFTER FREDI EXECUTION



After FREDI's stack allocation, before FREDI execution.

After FREDI execution.

FREDI updates the caller's FLGT to include the stack area and SBS appended from FREDI.

APPENDIX E

REAL TIME FORTRAN

DGC Real Time FORTRAN (hereafter called RT FORTRAN) provides programmers with the means to use the computational power of FORTRAN in programs written to control a real time environment. This appendix describes methods for writing RT FORTRAN programs and documents those routines, found only in the RDOS FORTRAN library, which implement the RT FORTRAN capability.

Real Time FORTRAN Concepts

Effective use of RT FORTRAN presumes that users have familiarized themselves thoroughly with DGC Real Time concepts as found in the RDOS User's Manual, 093-000075. This is due to the fact that RT FORTRAN programs will seldom, if ever, be written entirely in FORTRAN. Those segments of control programs handling special user interrupts, for example, must be written in assembly language. Moreover, RT FORTRAN calls parallel closely their assembly language counterparts; a thorough understanding of RDOS will therefore facilitate the use of RT FORTRAN.

The following summarizes DGC Real Time concepts and illustrates the relationship between RT FORTRAN Task calls and RDOS Task calls. A task is a logically complete execution path through a program demanding use of system resources, primarily CPU control. A multitask environment is one in which logically distinct tasks compete simultaneously for the use of system resources; a single task environment in RT FORTRAN is a trivial subset of a multitask environment. By default, RT FORTRAN programs have one task; this task is used to create other tasks if more are needed.

Only one task receives CPU control at any single moment. CPU control is allocated to tasks according to their relative priorities and readiness to use the CPU. Resource allocation is accomplished by the RDOS minimum Task Scheduler, TMIN, in single task environments; the FORTRAN Task Scheduler, TMAX, allocates CPU control to the highest priority ready task in a multitask environment. Note that the RDOS Task Scheduler, TCBMON, differs from the FORTRAN multitask scheduler.

Task priorities range from zero, the highest priority, through 255 decimal. The default task in RT FORTRAN exists at priority zero. Several tasks may exist at the same priority. Among equal priority tasks, the time of a task's creation or task priority modification determines the relative priority of the task within a priority level. The first task created at a given priority has the highest priority within that priority level, etc. There is no practical limit to the number of tasks which may be created within any program. Nonetheless, users are cautioned to request only the minimum number of tasks necessary for the running of an RT FORTRAN program in order to minimize system overhead and to maximize the size of run time stacks allocated for each task.

Tasks may exist in any of four states. Tasks are either ready to perform their functions, they are actually in control of the CPU and are executing their assigned instruction paths, they are suspended and temporarily unable to receive CPU control, or they are dormant, having no priority and no chance of gaining CPU control until readied by an FTASK or ITASK command. Executing, ready, and suspended tasks are linked in a queue called the active chain. Tasks which have been deleted are removed from the active chain and are placed in the inactive chain. The Task Scheduler maintains certain status information about each task. This information is retained within an information structure called a Task Control Block (TCB). There is one TCB for each task. The active chain is in reality the collection of all active TCBs, linked in priority fashion. The inactive chain is merely a pool of empty TCBs which may be used in the creation of new tasks. Whenever a task receives control,that task's state variables (AC's, Carry, etc.) are re-established; these state variables are saved in the task's TCB whenever the task is reduced to the ready or suspended states. Tasks may be assigned unique i.d. numbers in the range 0-255_{10}; this is especially helpful in distinguishing equal priority tasks. Only the highest priority ready task will be given control of the CPU, and other ready tasks await their turn in priority fashion. Suspended tasks are tasks that were once ready. A ready task becomes suspended for one or more of a variety of reasons:

1. It has been suspended by SUSP, ASUSP, or HOLD.
2. It is waiting for a message from another task or awaits the receipt of the message (REC/XMTW).
3. It is awaiting completion of a .SYSTM call.
4. It is waiting for the use of an overlay.

Just as a number of different events may suspend a ready task, several events can cause suspended tasks to become readied:

1. A .SYSTM call has been completed.
2. A message has been posted for a suspended task awaiting its receipt.
3. Another task readied a suspended task via .ARDY or .TIDR.
4. An overlay or overlay area is ready for use.

If a task is suspended for two distinct reasons (e.g., HOLD and awaiting completion of a .SYSTM call), it must be readied by two different events (e.g., .TIDR and completion of .SYSTM call).

Return from a FORTRAN Task call is always either to an error return location (if there is one reserved and an error occurs) or to the next sequential instruction following the call. After the task call has been performed, however, both returns are always routed through the FORTRAN Task Scheduler.

RT FORTRAN permits tasks to communicate with one another by sending and receiving one word non-zero messages. A one word message (i.e., one which can be stored in a single 16-bit cell), is sent to a task in an agreed upon location. The task sending a message may either return control to the Task Scheduler immediately or it may wait and place itself in the suspended state until the receiving task has issued a receive request and has received the message. Upon receipt of the message, the receiving task reverts to the ready state. Interrupt requests from special (i.e., non-SYSGENed) devices do not change the status of tasks in a multitask environment; these events freeze the environment, as will be described later.

Run Time Stack Partitioning

Number formats in RT FORTRAN routines are identical to those given in the Introduction to this manual. Similarly, the SP, Number, and Run Time Stacks are structured and maintained as described for non-real time routines. The major difference in the whole run time stack structure is that the Run Time area is partitioned by the RT initializer into equal segments, one segment for each task specified at the beginning of the FORTRAN program.

Each run time stack segment can be viewed for the most part as a non-real time stack area in miniature. That is, each segment has an SP stack, Number Stack, and Run Time stack in that order. The SP stack is 40 octal words in length, but the two other stacks are necessarily smaller than their non-real time stack counterparts. Each segment Number Stack is 330 octal words long, and each Run Time stack will have the remainder of the segment area. It is not possible to either adjust the number stack size or to omit the allocation of a Number Stack on a selective basis.

Since each segment requires a family of pointers and displacements to describe it uniquely, each segment is preceded by an eight word state save area. In this area are stored values for the following segment stack pointers and flags:

FSP

.NDSP

AFSE

SP

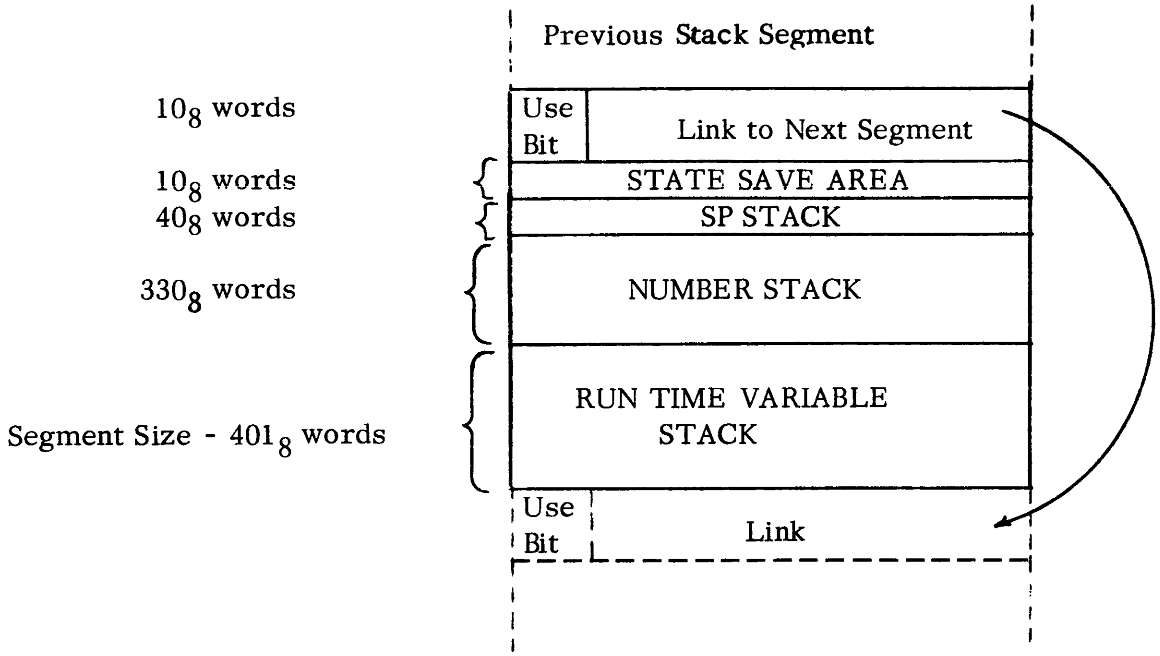
QSP

.OVFL

.SVO

NSP

Immediately preceding each state save area, the first word of each stack serves as both link to the beginning of the next segment and as a flag (bit zero) indicating whether or not a task is currently using the segment.



RUN TIME STACK SEGMENT

User Interrupts

As indicated earlier users wishing to incorporate non-SYSGENed devices into RT FORTRAN programs must provide for the interrupt servicing to be done in assembly language and the creation of a three-word Device Control Table as detailed in Chapter 7 of the RDOS USER'S MANUAL, User Serviced Interrupts. Procedures given throughout this manual will be used to write assembly language modules which will be interfaced with the main RT FORTRAN program.

Interrupt requests from special (i.e., non-SYSGENed) devices do not, for the most part, change the status of tasks in a FORTRAN multitask environment. Instead, user interrupts freeze the environment until servicing of the interrupt is completed and the multitask environment is unfrozen. Likewise, all other tasks will resume their former states when the environment becomes unfrozen unless the user transmits a message to one of them by means of the transmit interrupt message command, .IXMT .

User Interrupts (Continued)

Since control does not go through the FORTRAN Task Scheduler when the environment is unfrozen, .IXMT is not a command which can be issued via FORTRAN source code; rather, .IXMT is a one-word Task call identical to .IXMT discussed in the RDOS User's Manual, Chapter 5. As stated there, if the task for whom the message is intended has issued a receive request for the message, the task state is changed from suspended to ready even though the task environment is frozen. This is the one exception to the rule that user interrupt servicing does not alter the task environment.

It is still necessary, however, to identify the interrupt device to the system by means of a FORTRAN call, and it is possible to remove this device from the system by means of another FORTRAN call.

RT FORTRAN Routine Descriptions

The following seven sections describe all the real time routines of interest to RT FORTRAN programmers. The sections and their contents are:

REAL TIME INITIALIZATION	Initialization and stack segmentation routines.
REAL TIME TASK	Real time single and multitask environment management routines.
REAL TIME CLOCK/CALENDAR	System clock and calendar management routines.
REAL TIME INTERRUPT	Real Time routines used to identify or remove special user devices from the system.
REAL TIME SWAP AND OVERLAY	Routines implementing the RDOS swap and overlay management calls.
REAL TIME FILE AND I/O	File management with block and file I/O RDOS commands.
REAL TIME BIT MANIPULATION	Routines allowing individual bits to be tested, set and cleared within 16 bit words.

Note that not all of the supporting routines listed for .I and ITCB are described. Similarly. other routines used by the FORTRAN Task Scheduler or by other internal routines are mentioned but are not described. Only the two main initialization routines are given since an understanding of these suffices to describe the structuring of the real

RT FORTRAN Routine Descriptions (Continued)

time run stack. Real time programmers wishing to write their own initialization procedures must first consult the program listings of all routines in the initialization package. All real time programs, even those with only one active task, require the support of the real time initialization routines.

All routines described in the REAL TIME TASK section have functions which parallel closely the functions of RDOS task calls. None of these routines is of use in a single task environment. As stated in the RDOS User's Manual, the killing of all tasks causes return of program control to the next higher program level, usually the CLI (Command Line Interpreter). Similarly, depressing either the teletypewriter keys CTRL and A or CTRL and C interrupts the program and causes return to be made to the CLI. The CTRL A break aborts an RT FORTRAN program with no facility for preserving the current environment. CTRL C permits a qualified saving of the real time environment; for more information, see the RDOS User's Manual, Chapter 2.

Routines described in both the REAL TIME CLOCK/CALENDAR and REAL TIME INTERRUPT sections are useful in both single and multitask environments.

RDOS FORTRAN Error Arguments

Several routines in the RDOS FORTRAN library have an error argument which receives a code character at the completion of the routine's execution. This code character describes the success or failure of the routine's execution. The settings of this code are as described below:

Setting	Meaning
0	Indeterminate error.
1	No error occurred.
3...n	RDOS system error code + 3

Error code 2 is generally not used. Thus if the error argument is placed in blank common, the user may define code 2 for use in intertask communication endowing this code with whatever meaning he wishes.

One such possible definition would be the definition given to 2 by FOVLD, i.e., that system action is in progress. FOVLD changes this code to one of the other settings upon completion of the call.

REAL TIME INITIALIZATION

.I

Purpose: To partition the free memory area into equal segments for the creation of each task's run time stacks; to allocate a blank common area if needed; to build an I/O Channel Assignment Table initialized to the default values of the logical FORTRAN channels; to allocate number, SP, and run time stacks and create the associated stack pointers for the first task by means of a call to the TCB initializer.

Calling Sequence: Real Time .I receives control in the same manner that .I used in DOS environments receives control, i.e., by means of an end block which has the starting address .I .

.I invokes the TCB Initializer (.ITCB) , after which it transfers control to the FORTRAN Task Scheduler.

Supporting Routines : DVD, .MAIN, QUIT, SVVAR, TVR; FLSP, .FTSCH, .INHB .IOCAT, .OVFL, .SOSW, SP, SUCOM .

Subroutine Size: 2 page zero locations and 403 octal locations of normally relocatable memory. The Channel Assignment Table, 60 octal locations in length, is written over a portion of .I after that part of the initialization code has been executed.

Notes to User: ITCB, the FORTRAN Task Control Block Initializer, has an alternate entry point in the .I module.

The following describes the functions performed by .I in the sequence that they occur.

A system call, .RESET, is issued to initialize system I/O. USTCS of the User Status Table (UST) is examined to determine the size of blank common. (For a description of the User Status Table see the RDOS User's Manual, Chapter 5.) Blank common is then allocated, if possible, and a pointer to the start of blank common is created. If there is not enough memory available for blank common allocation, an error message is output,

MEMOVFL ↵

and a return to the next higher program level (usually the CLI) is made by means of .SYSTM, .RTN .

.I (Continued)

Notes to User: (Continued)

A temporary SP stack is then created (and will later be overwritten); this SP stack is required for the following operation, which calls DVD. The number of tasks and FORTRAN channels which will be required is determined by examining USTCH of the UST. DVD is then called, and the remaining free memory is partitioned into equal segments for each of the task's later run time use. Each run time segment has a link to the following segment built into its first word, and a flag bit is allocated to indicate whether the segment has yet been assigned to a specific task.

ITCB is then called, setting up stacks and stack pointers in the first run time segment area for the first FORTRAN task. The Channel Assignment Table is then built over the beginning of .I code; this code, having once been executed, is of no further use in a multi-task environment. Upon completion of this last operation, control is given to the FORTRANTask Scheduler.

As with the single task initializers, the multitask initializer also has three additional entries which return control to either the debugger or to the next higher level program (usually the CLI), as in the event of a run time error. These entries are FERTN, FERT1, and FERTØ. FERTN transfers control to the CLI via the call .SYSTM, .RTN . FERTØ transfers control to the CLI via the call .SYSTM, .ERTN . FERT1 transfers control to the debugger.

Note that this version of .I is used only in multitask programs. Single task FORTRAN programs use the single task version of .I given in Chapter 10.

ITCB

Purpose: To allocate number, SP, and run time stacks and stack pointers in a FORTRAN task's run time segment area.

Calling Sequence: (The priority of the task which is to be assigned the stack area segment is input in AC0, AC1 contains the starting address of the task's TCB)

JSR @.ITCB

(The following variables are initialized for the task: SP, NSP, .NDSP, AFSE, .IOCAT, FSP, QSP, .OVFL .)

Supporting Routines : DVD, .MAIN, QUIT, SVVAR, TVR; FLSP, .FTSCH, .INHB, .IOCAT, .OVFL, .SOSW, SP, SUCOM .

Subroutine Size: 2 page zero locations and 403 octal locations of normally relocatable memory.

Notes to User: .I, the Real Time FORTRAN Initializer, has an alternate entry point in this module. .I calls ITCB as part of the initialization process, and .ITCB is called each time a stack segment is to be used by a FORTRAN task for the first time.

REAL TIME TASK

ABORT

Purpose: To kill a task specified by i. d. number.

Calling Sequence:

FCALL
ABORT
Integer 2
FORTRAN ADDRESS of i. d. number
FORTRAN ADDRESS of error code

Supporting Routines: FRET, KTID; .CPYL

Subroutine Size: 15 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB (unless it is the caller who is killed).

ABORT must be referenced in an .EXTN statement.

The calling task itself may be killed by this call.
The TCB which is removed from the active queue is placed in the free element TCB chain. The specified task is not killed immediately only if it is suspended due to an outstanding .SYSTM call, in which case it is killed as soon as the .SYSTM call is completed.

If no task exists with the specified i. d. number, no action is taken, and control goes to the scheduler.

AKILL

Purpose: To delete all tasks of a given priority.

Calling Sequence:

```
FCALL
AKILL
Integer 1
FORTRAN ADDRESS of the task priority
```

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: KILL; .CPYL, .FTSCH, .INHB, .SVALL, SP .

Subroutine Size: 60 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB (unless the caller is also deleted).

AKILL must be referenced by an .EXTN statement.

ARDY and ASUSP have alternate entry points in this module.

The calling task itself may be deleted by means of this command. All TCBs that are removed from the active queue are placed in the free element TCB chain. If a task to be deleted by AKILL is suspended (e.g., the task is awaiting completion of a system call) it will be killed as soon as it becomes ready.

If no task exists at the given priority, this call is an effective no-op, and control goes to the Scheduler.

ARDY

Purpose: To ready all tasks of a given priority.

Calling Sequence:

FCALL
ARDY
Integer 1
FORTRAN ADDRESS of the Task Priority

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: KILL; .CPYL, .FTSCH, .INHB, .SVALL, SP .

Subroutine Size: 60 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB .

ARDY must be referenced by an .EXTN statement.

ASUSP and AKILL have alternate entry points in this module.

This command unconditionally readies all tasks of the given priority. It is the caller's responsibility to insure that the tasks to be readied are not awaiting the occurrence of some other event like the completion of I/O.

CHNGE

Purpose: To change the priority of a task specified by i. d. number.

Calling Sequence:

```
FCALL
CHNGE
Integer 3
FORTRAN ADDRESS of i. d. number
FORTRAN ADDRESS of new priority
FORTRAN ADDRESS of error code
```

Supporting Routines: FRET, TCHNG; .CPYL

Subroutine Size: 16 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - System error code + 3

CHNGE must be referenced in an .EXTN statement.

ASUSP

Purpose: To suspend all tasks of a given priority.

Calling Sequence:

FCALL
ASUSP
Integer 1
FORTRAN ADDRESS of the task priority

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: KILL; .CPYL, .FTSCH, .INHB, .SVALL, SP .

Subroutine Size: 60 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB .

ASUSP must be referenced by an .EXTN statement.

AKILL and ARDY have alternate entry points in this module.

The calling task may itself be suspended by this command. The suspended tasks can be readied only by an .ARDY command. If no tasks exist at the given priority, this call is an effective no-op.

HOLD

Purpose:	To suspend a task specified by identification number.
Calling Sequence:	FCALL HOLD Integer 2 FORTRAN ADDRESS of i. d. number FORTRAN ADDRESS of error code
Supporting Routines:	FRET, STID; .CPYL
Subroutine Size:	15 octal locations of normally relocatable memory.
Notes to User:	Accumulators and carry are saved in the caller's TCB.

HOLD must be referenced in an .EXTN statement.

This call sets bit 1 of the task's priority and status word, TPRST. Thus if the task is already suspended for some other reason (e.g., XMTW, REC, or .SYSTM call), it becomes doubly suspended and can be readied only when all its suspend bits have been set to ready.

The error code word will be set to one of the following states:

0 -	Indeterminate error.
1 -	No error occurred.
3...n -	RDOS system error code + 3.

ITASK

Purpose: To create a task in a real-time FORTRAN environment and assign a unique i.d. to the task.

Calling Sequence:

```
FCALL
ITASK
Integer 4
FORTRAN ADDRESS of Task Entry Point
FORTRAN ADDRESS of Task I. D.
FORTRAN ADDRESS of Task Priority
FORTRAN ADDRESS of Error Code
```

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: CTASK, FRET; .CPYL

Subroutine Size: 24 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are stored in the caller's TCB.

ITASK must be referenced in an .EXTN statement.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - System error code + 3.

FTASK

Purpose: To create a task in a real-time FORTRAN environment.

Calling Sequence:

```
FCALL
FTASK
Integer 3
FORTRAN ADDRESS of Task Code entry point
FORTRAN ADDRESS of Error Return
FORTRAN ADDRESS of Task Priority
```

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: .CPYL, .FTSCH, .INHB, .LNK, .SVALL, .ITCB, CTCB.

Subroutine Size: 35 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

TASK must be referenced by an .EXTN statement.

When the RT FORTRAN program is loaded and first run, only one task exists. This command must be issued to create a multitask environment. The error return is taken if there are no TCBs available, i.e., if the maximum number of tasks specified in CHANTASK was too small.

KILL

Purpose: To kill (delete) the calling task, freeing its TCB so that a new task can be created.

Calling Sequence:

```
FCALL
KILL
0
```

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: .FTSCH, .INHB, .ULNK .

Subroutine Size: 25 octal locations of normally relocatable memory.

Notes to User: KILL must be referenced by an .EXTN statement.

This command deletes the calling task's TCB from the active queue and places it in the free element TCB chain. The calling task is the only task that can be deleted via this command. There is no return from this call. The stack block associated with the deleted task is released.

REC

Purpose: To receive a one-word message from a transmitting task.

Calling Sequence:

FCALL
REC
Integer 2
FORTRAN ADDRESS of the message location ("key location")
FORTRAN ADDRESS to receive the one-word message
(must be different from the key)
(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: .AFRTN, .CPYL, .FTSCH, .INHB, .KSRCH, .SVALL, CTCB.

Subroutine Size: 74 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

REC must be referenced by an .EXTN statement. XMT and XMTW have alternate entry points in this module.

If the contents of the key location are non-zero at the time of this call (i.e., if a message has been sent), the message is passed directly to the receiving task and the contents of the key location are reset to zeroes. If the contents of the key location are zero when this call is issued (i.e., if the message has not yet been sent), the receiving task becomes suspended until the message is sent. When the message is transmitted, it is sent directly to the receiving task, bypassing the key location entirely, and the receiving task becomes readied.

SUSP

Purpose:	To suspend the calling task.
Calling Sequence:	FCALL SUSP 0 (Control returns to the FORTRAN Task Scheduler.)
Supporting Routines:	.FTSCH, .INHB, .SVALL, CTCB.
Subroutine Size:	12 locations of normally relocatable memory.
Notes to User:	Accumulators and carry are saved in the caller's TCB. SUSP must be referenced by an .EXTN statement. The suspended task remains suspended until it is readied by an .ARDY command. PEND is equivalent to SUSP.

RELSE

Purpose: To ready a task specified by i. d. number.

Calling Sequence:

```
FCALL
RELSE
Integer 2
FORTRAN ADDRESS of i. d. number
FORTRAN ADDRESS of error code
```

Supporting Routines: FRET, RTID; .CPYL

Subroutine Size: 13 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

RELSE must be referenced in an .EXTN statement.

This call resets bit 1 of the task's priority and status field word, TPRST. Thus if the task has bits 0 and/or 12 set (e.g., due to an outstanding .SYSTM call or a .REC/.XMTW), these bits would also have to be reset before the task could be raised to the ready state.

The error code word will be set to one of the following states:

0	-	Indeterminate error.
1	-	No error occurred.
3...n	-	RDOS system error code + 3.

STTSK

Purpose: To obtain the status of a task specified by i.d. number.

Calling Sequence:

FCALL
STTSK
Integer 3
FORTRAN ADDRESS of i.d. number
FORTRAN ADDRESS of location to receive task status code
FORTRAN ADDRESS of error code

Supporting Routines: FRET, TIDST; .CPYL

Subroutine Size: 10 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

STTSK must be referenced in an .EXTN statement.

The task status code will be one of the following:

0 - Ready.
1 - Suspended by a .SYSTM call
2 - Suspended by SUSP, ASUSP, or HOLD
3 - Waiting for a message to be transmitted or received.
4 - Waiting for an overlay area to become free.
5 - Suspended by SUSP, ASUSP, or HOLD and by a .SYSTM call.
6 - Suspended by SUSP, ASUSP, or HOLD and by XMTW or REC.
7 - Waiting for an overlay area and suspended by ASUSP, SUSP, or HOLD.
8 - No task exists with this i.d. number.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3.

PRI

Purpose: To change the priority of the calling task.

Calling Sequence:

FCALL
PRI
Integer 1
FORTRAN ADDRESS of the new task priority

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: .CPYL, .FTSCH, .INHB, .SVALL,.LNK,.ULNK , CTCB .

Subroutine Size: 23 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB .

PRI must be referenced by an .EXTN statement.

The calling task is assigned the lowest priority of all tasks within the new priority level. It is permissible to issue a PRI command without changing the caller's present priority level. This will cause the calling task to be assigned the lowest priority of all tasks within the given priority level.

XMT, XMTW

Purpose: To transmit a one-word message (XMT) to a receiving task, then remain ready to resume other task activity, or to transmit a message and wait (XMTW), staying suspended until the message is received.

Calling Sequences:

FCALL
XMT (or XMTW)
Integer 3
FORTRAN ADDRESS of the message location (key location)
FORTRAN ADDRESS of the one-word message
FORTRAN ADDRESS of the error return

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: .AFRTN, .CPYL, .FTSCH, .INHB, .KSRCH, .SVALL , CTCB .

Subroutine Size: 74 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

XMT and XMTW must each be referenced by an .EXTN statement.

REC has an alternate entry point in this module.

A one-word message is replaced in the key location if the task for whom it is intended has not yet requested its receipt. As soon as the receiving task issues a receive request, the message is placed in the address specified by the receiving task, and the contents of the key location are reset to all zeroes. If the receiving task has requested the message before its transmission, the message is sent directly to the receiver's address, bypassing the key location entirely.

The error return is taken if the message address is already in use (i.e., its contents are non-zero) .

REAL TIME CLOCK/CALENDAR

DATE

Purpose: To get the current day of the year.

Calling Sequence:

```
FCALL
DATE
Integer 2
FORTRAN ADDRESS of date array
FORTRAN ADDRESS of error code
```

Supporting Routines: FRET ; .CPYL .

Subroutine Size: 16 octal locations of normally relocatable memory.

Notes to User: Accumulators and Carry are restored upon exit.

This routine issues the RDOS system call, .GDAY. The date is returned as the number of the current day of the year and is stored in the second word of the date array. This array is an integer array of at least three words.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.

FDELY

Purpose: To suspend a FORTRAN Task for a specified period of time.

Calling Sequence:

FCALL
FDELY
Integer 1
FORTRAN ADDRESS of time interval

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: FRET; .CPYL .

Subroutine Size: 7 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are stored in the caller's TCB.

FDELY must be referenced in an .EXTN statement.

This time interval word indicates the number of real time clock pulses during which the task will be suspended. (The real time clock frequency was set at SYSGEN time.)

FGTIM

Purpose: To get the time of day and current date.

Calling Sequence:

```
FCALL
FGTIM
Integer 4
FORTRAN ADDRESS to receive the hour
FORTRAN ADDRESS to receive the minute
FORTRAN ADDRESS to receive the second
FORTRAN ADDRESS to receive the current date
```

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 41 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are stored in the caller's TCB. FGTIM must be referenced in an .EXTN statement.

No error message is possible; the system does not reset the current date to 1 at the end of the year. Instead it continues to increment the date count.

The time of day is given by a 24 hour clock; the date is given as an integer from 1 through 365 (or 366 for leap years).

FSTIM has an alternate entry point in this module.

FSTIM

Purpose: To set the system clock and system calendar.

Calling Sequence:

FCALL
FSTIM
Integer 4
FORTRAN ADDRESS of the current hour
FORTRAN ADDRESS of the current minute
FORTRAN ADDRESS of the current second
FORTRAN ADDRESS of today's date

(Control returns to the FORTRAN Task Scheduler.)

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 41 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are stored in the caller's TCB.

FSTIM must be referenced in an .EXTN statement.

A fatal run time error message, ERTIM, is issued if an attempt is made to set an illegal time or date. Upon issuance of an error message, control returns to either the Debugger or to the CLI.

The system clock is a 24 hour clock; the system calendar is simply an integer from 1 to 365 (or 366 for a leap year). The system does not reset the system calendar to 1 on January 1; instead it continues to increment the date count.

FGTIM has an alternate entry point in this module.

TIME

Purpose: To get the current time of day.

Calling Sequence:

FCALL
TIME
Integer 2
FORTRAN ADDRESS of time array
FORTRAN ADDRESS of error code

Supporting Routines: FRET; .CPYL .

Subroutine Size: 16 octal locations of normally relocatable memory.

Notes to User: Accumulators and Carry are restored upon exit.

This routine issues the RDOS system call, .GTOD . The time is returned in the order: hours, minutes, and seconds, and is stored in the time array. This array is an integer array of at least three words.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.

REAL TIME INTERRUPT

FINRV

Purpose: To remove a non-SYSGENed device, which had been identified by FINTD, from the system's recognition.

Calling Sequence:

```
FCALL
FINRV
Integer 1
FORTRAN ADDRESS of the device code
```

(Control returns to the Task Scheduler.)

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 21 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB. FINRV must be referenced by an .EXTN statement. FINTD has an alternate entry point in this module.

This call removes the device entry from the system interrupt vector table.

A system error code ERDNM is output, with consequent return to the CLI if an illegal device code is given.

FINTD

Purpose: To introduce to the system a non-SYSGENed device capable of generating interrupt requests.

Calling Sequence:

```
FCALL
FINTD
Integer 2
FORTRAN ADDRESS of the device code
FORTRAN ADDRESS of the three word DCT
```

(Control returns to the Task Scheduler).

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 21 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB. FINTD must be referenced in an .EXTN statement. FINRV has an alternate entry point in this module.

This call causes an entry for this device to be placed in the system interrupt vector table.

A system error code ERDNM is output, with consequent return to the CLI, if an illegal device code is given.

IXMT

Purpose: To transmit a message from a user interrupt service routine to a task in the multitasking environment.

Calling Sequence:

FCALL
IXMT
Integer 3
FORTRAN ADDRESS of the message address
FORTRAN ADDRESS of the message
FORTRAN ADDRESS of the error code

Supporting Routines: FRET, IXMTT; .CPYL

Subroutine Size: 16 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit. Return is to the caller, not to the task scheduler. This routine is issued only in a user interrupt routine, outside the multitasking environment. For more information about user interrupt routines, see the RDOS User's Manual, Chapter 7.

IXMT must be referenced in an .EXTN statement.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3.

REAL TIME OVERLAY AND SWAP

FBACK

Purpose: To read in from disk the next higher level program swap.

Calling Sequence:

```
FCALL
FBACK
```

Supporting Routines: FERTØ, FERTN; .CPYL, .FRET, .RTER .

Subroutine Size: 43 octal locations of normally relocatable memory.

Notes to User: The calling program is overwritten and its accumulators and carry are lost. Information can be passed to the higher level program swap via blank common.

When the higher level program swap is read into core, control goes to the highest priority ready task within the swap.

FCHAN and FSWAP have alternate entry points in this routine.

FCHAN

Purpose: To perform a program chain. A new save file is read from disk, overwriting the current core image while not changing program levels.

Calling Sequence:

```
FCALL
FCHAN
Integer 1
FORTRAN ADDRESS of the save file name
```

Supporting Routines: FERTØ, FERTN; .CPYL, .FRET, .RTER .

Subroutine Size: 43 octal locations of normally relocatable memory.

Notes to User: Since the calling program is overwritten, accumulators and carry are not saved. Information can be passed via blank common, since blank common is not overwritten during program swapping or chaining.

When the program chain is read into core, control goes to the highest priority ready task within the new save file.

Control is returned to the higher program level by the FORTRAN call FBACK.

FCHAN and FBACK have alternate entry points in this routine.

Since this routine issues the RDOS system call .EXEC, more information about program swaps can be found in the RDOS User's Manual.

FOVLD

Purpose: To load a FORTRAN overlay into an overlay area. (This routine is for use in multitask environments.)

Calling Sequence:

FCALL
FOVLD
Integer 4
FORTRAN ADDRESS of the channel number upon which the
overlay file has been opened
Overlay number
FORTRAN ADDRESS of the conditional load flag
FORTRAN ADDRESS of the error code

Supporting Routines: FRET, KILL, SVVAR; .IOCAT, .CPYA, .CPYL, .KSRCH, .FTSCH, CTCB, .INHB .

Subroutine Size: 206 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The overlay file which is to be used in this call must have been opened previously by a call to OVOPN. The overlay number is a word which contains the overlay area number in its left byte and the overlay number within its right byte. This number must have been declared either in an .ENTO statement or an OVERLAY statement.

The conditional load flag is a word which is set to be either zero or non-zero. If zero, overlay loading is to be done unconditionally; if non-zero, overlay loading is to be done conditionally.

In conditional loading, if the overlay area is free the overlay is loaded (unless it is already core resident, in which case return is made directly to the Task Scheduler). An area is considered to be free if the overlay use count of the currently resident overlay has gone to zero and if the area has been released by the FOVRL call.

In unconditional loading, if an area is free the requested overlay is loaded regardless whether it is currently core resident or not. If the area is not free, the caller is suspended until the area is released. For more information about conditional and unconditional overlay loading, see the RDOS User's Manual.

FOVLD (Continued)

Notes to User: (Continued)

FOVRL has an alternate entry point in this routine.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
2 - System action in progress.
3...n - RDOS System error code +3 .

This routine is found in the FORTRAN multitask library. To cause this routine to be loaded (instead of FOVLD, the single task overlay load module), the multitask library must precede the RDOS FORTRAN library when relocatable loading is performed.

FOVLY

Purpose:

To load a FORTRAN overlay into an overlay area. (This routine is for use in single task environments.)

Calling Sequence:

FCALL
FOVLD (or OVLOD)
Integer 4
FORTRAN ADDRESS of the channel number upon which the, overlay file has been opened.
Overlay number
FORTRAN ADDRESS of the conditional load flag
FORTRAN ADDRESS of the error code

Supporting Routines:

FRET; .CPYA, . IOCAT

Subroutine Size:

46 octal locations of normally relocatable memory.

Notes to User:

Accumulators and carry are restored upon exit.

The overlay file which is to be used in this call must have been opened previously by a call to OVOPN. The overlay number is a word which contains the overlay area number in its left byte and the overlay number within its right byte. This number must have been declared either by an .ENTO pseudo op or in a FORTRAN OVERLAY statement.

The conditional load flag is a word which is set to be either zero or non-zero. If zero, overlay loading is to be done unconditionally; if non-zero, overlay loading is to be done conditionally.

In conditional loading, the number of the currently loaded overlay is checked. If it is the same as the requested overlay, return is made immediately to the caller. Otherwise the requested overlay is loaded.

In unconditional loading, the requested overlay is loaded regardless of whether it is currently core resident or not. For more information about conditional and unconditional overlay loading, see the RDOS User's Manual.

OVLOD is equivalent to FOVLD.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS System error code + 3.

This routine is found in the RDOS FORTRAN Run Time library.

FOVRL

Purpose:	To release an overlay area.
Calling Sequence:	FCALL FOVRL Integer 2 Overlay number FORTRAN ADDRESS of the error code
Supporting Routines:	FRET, KILL, SVVAR; .IOCAT, .CPYA, .CPYL, .KSRCH, .FTSCH, CTCB, .INHB .
Subroutine Size:	206 octal locations of normally relocatable memory.
Notes to User:	Accumulators and carry are restored upon exit. The overlay number is a word which contains the overlay area number in its left byte and the overlay number within its right byte. The overlay number must have been declared in either a .ENTO or an OVERLAY statement.

This call should be issued each time a user completes his use of a given overlay (i.e., it decrements the overlay use count). When no users remain who wish to use the currently resident overlay (i.e., the overlay use count goes to zero), the overlay area becomes free for the loading of other overlays.

This call must not be issued from within the overlay area which is to be released.

The error code word will be set to one of the following states:

0	-	Indeterminate error.
1	-	No error occurred.
3...n	-	RDOS system error code + 3 .

FOVLD has an alternate entry point in this routine.

FQTASK

Purpose: To load a user overlay and periodically execute a task within the overlay or to periodically execute a core-resident task.

Calling Sequence:

FCALL
FQTASK
Integer number of arguments, 4 or 5
Overlay number or dummy argument
FORTRAN ADDRESS of task entry point
FORTRAN ADDRESS of task queue array
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of constant -1
if a core resident task

Supporting Routines: FRET, TQTSK; .CPYL

Subroutine Size: 42 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB; return is to the task scheduler.

Note that the first argument is either an overlay number (not the FORTRAN ADDRESS of an overlay number) in the case of an overlay task or a dummy argument in the case of a core resident task. The overlay number is a word which contains the overlay area number in its left byte and the overlay number in its right byte.

The task entry point is the entry point within either the overlay or the core resident task where program control is to begin execution; this point must have been globally ENTered.

The task queue array is a 13_8 word integer array, supplied by the user, whose elements contain the following parameters and whose displacements are given the following mnemonic assignments:

Displacement	Contents
QPC	Used by the system.
QNUM	Number of times to execute task.
QTOV	Used by the system.
QSH	Starting hour of task execution.
QSMS	Starting second within the hour QSH.
QPRI	Task priority.

Displacement	Contents
QRR	Rerun increment in seconds.
QTLNK	Used by system.
QOCH	Overlay channel number (dummy for core resident tasks).
QCOND	Overlay conditional load flag (dummy for core resident tasks).
QCOND+1	Task i. d. number.

The error code word will be set to one of the following states:

0 -	Indeterminate error.
1 -	No error occurred.
3...n -	RDOS system error code + 3.

The last parameter must be present and must point to a -1 constant only if the task to be executed is a core resident task.

FQTASK must be referenced in an .EXTN statement.

OVEXT

Purpose: To release an overlay and return control to an address specified by the caller. This call is issued from within the overlay. If several binaries comprise the overlay, this call is issued from within the binary where the overlay name is defined via an OVERLAY or .ENTO statement.

Calling Sequence:

FCALL
OVEXT
Integer 2
overlay number or name
FORTRAN ADDRESS of return

Supporting Routines: FRET, TOVRL; .CPYL, .RTER

Subroutine Size: 22 octal locations of normally relocatable memory.

Notes to User: The overlay name may be used in place of the composite overlay number if the name has been defined in an OVERLAY or .ENTO statement.

This call decrements the overlay use count and releases the overlay area if the count becomes zero.

Accumulators and carry are saved in the caller's TCB.

OVEXX has an alternate entry point in this routine.

OVEXT must be referenced in an .EXTN statement.

OVEXX

Purpose: To release an overlay and return control to an address specified by the caller. This call is issued from within the overlay. If several binaries comprise the overlay, this call is issued from within a binary other than the one which defines the overlay name via an OVERLAY or .ENTO statement.

Calling Sequence:

```
FCALL
OVEXX
Integer 2
overlay number or name
FORTRAN ADDRESS of return
```

Supporting Routines: FRET, TOVRL; .CPYL, .RTER

Subroutine Size: 22 octal locations of normally relocatable memory.

Notes to User: The overlay name may be used in place of the composite overlay number if the name has been defined in an OVERLAY or .ENTO statement.

This call decrements the overlay use count and releases the overlay area if the count becomes zero.

Accumulators and carry are saved in the caller's TCB.

OVEXT has an alternate entry point in this routine.

OVEXX must be referenced by an .EXTN statement.

OVKIL

Purpose: To kill a calling task and release its overlay. This call is issued from within the overlay. If several binaries comprise the overlay, the call is issued from within the binary which defines the overlay name via the OVERLAY or .ENTO statement.

Calling Sequence:

```
FCALL
OVKIL
Integer 1
overlay number or name
```

Supporting Routines: FRET, KILL, TOVRL; .CPYL, .RTER

Subroutine Size: 20 octal locations of normally relocatable memory.

Notes to Users: The overlay name may be used in place of the composite overlay number if the name has been defined in an OVERLAY or .ENTO statement.

This call decrements the overlay use count and releases the overlay area if the count becomes zero.

OVKIX has an alternate entry point in this routine.

OVKIL must be referenced by an .EXTN statement.

OVKIX

Purpose: To kill a calling task and release its overlay. This call is issued from within the overlay. If several binaries comprise the overlay, this call is issued from within a binary other than the one which defines the overlay name via an OVERLAY or .ENTO statement

Calling Sequence:

FCALL
OVKIX
Integer 1
overlay number or name

Supporting Routines: FRET, KILL, TOVRL; .CPYL, .RTER

Subroutine Size: 20 octal locations of normally relocatable memory.

Notes to User: The overlay name may be used in place of the composite overlay number if the name has been defined in an OVERLAY or .ENTO statement.

This call decrements the overlay use count and releases the overlay area if the count becomes zero.

OVKIL has an alternate entry point in this routine.

OVKIX must be referenced by an .EXTN statement.

FSWAP

Purpose: To save the current core image as a disk save file and read in a new save file at a lower program level.

Calling Sequence:

FCALL
FSWAP
Integer 1
FORTRAN ADDRESS of the save file name

Supporting Routines: FERTØ, FERTN; .CPYL, .FRET, .RTER .

Subroutine Size: 43 octal locations of normally relocatable memory.

Notes to User: The calling program is suspended and is saved on disk. The caller's task control block is used to save its accumulators, carry and PC to allow the caller to be resumed when control is transferred back to this level. Control is returned to the caller by the FORTRAN call FBACK.

When the program swap is read into core, control goes to the highest priority ready task within the new save file.

FCHAN and FBACK have alternate entry points in this routine.

Since this routine issues the RDOS system call .EXEC, more information about program swaps can be found in the RDOS User's Manual.

Information can be passed between the caller (the higher level program) and the lower level program via blank common, since blank common is not overwritten during program swapping or chaining.

OVOPN

Purpose: To open an overlay file on a FORTRAN channel.

Calling Sequence:
FCALL
OVOPN
Integer 3
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of error code.

Supporting Routines: FRET; .CPYL, .IOCAT .

Subroutine Size: 147 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

APPEND and OPEN have alternate entry points in this routine.

The file name is an ASCII byte string, including the file .OL extension.

This routine issues the RDOS system call .OVOPN. Thus this routine must be used before FORTRAN overlays can be loaded in either a single or multitask environment.

The FORTRAN routine FCLOS is used to close the overlay file and release its FORTRAN channel.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3 .

REAL TIME FILE AND I/O

APPEND

Purpose: To open a file so that new file information may be appended to that file. An optional blocking factor may be specified for the record size.

Calling Sequence:

```
FCALL
APPEND
Integer number of arguments, 4 or 5
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of open type indicator
FORTRAN ADDRESS of error code
FORTRAN ADDRESS of optional blocking factor
```

Supporting Routines: .DSI, FRET, IOPTR; .CPYL, .IOCAT

Subroutine Size: 174 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

OPEN and OVOPN have alternate entry points in this routine.

The file name is an ASCII byte string.

The open type indicator must be one of the following codes:

2 - shared appending (more than one user)
3 - exclusive appending (only one user)

For a device like the magnetic tape, the file is first opened and spaced to the end-of-file; appending takes place from that point.

The error code word will be set to one of the following:

0 - Indeterminate error
1 - No error occurred
3...n - RDOS system error code+3

The blocking factor constant is an integer indicating the number of bytes/record. For random record I/O, the blocking factor should be 128.

Up to 64 FORTRAN channel numbers are allowed, 0 through 63.

Append must be referenced by an .EXTN statement.

CFILW

Purpose: To create an RDOS disk file.

Calling sequence:

FCALL
CFILW
Integer number of arguments to follow -- 3 or 4
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of file type indicator
optional FORTRAN ADDRESS of file size
FORTRAN ADDRESS of error code

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 46 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The file name is an ASCII byte string.

The file type indicator is an integer. The following integers correspond to the listed file types:

Integer Indicator	File Type
1	Sequentially organized file.
2	Randomly organized file.
3	Contiguously organized file.

The file size argument is used only when a contiguously organized file is being created. The file size is an integer describing the number of disk blocks in the file.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3 .

CLOSE

Purpose: To free a FORTRAN logical channel under RDOS, and close the file associated with that channel.

Calling Sequence:

```
FCALL
CLOSE
Integer 2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of error code
```

Supporting Routines: FSAV, FRET, IMIO; .IOCAT, .RTER, .CPYL, .SOSW

Subroutine Size: 57 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The logical channel number is an integer constant with a value from Ø through 15_{10}.

This routine issues the RDOS system call .CLOSE .

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3... n - RDOS system error code + 3 .

FCLOS has an alternate entry point in this routine.

DFILW

Purpose: To delete a disk file.

Calling Sequence:
FCALL
DFILW
Integer number of arguments, 1 or 2
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of optional error code

Supporting Routines: FRET; .CPYL, .RTER

Subroutine Size: 27 octal locations of normally relocatable memory.

Notes to User: The file name is an ASCII byte string.

This routine issues the RDOS system call .DELET.

If a file requested to be deleted is open on one or more FORTRAN channels, the file will not be deleted. Instead, if no error code argument is supplied, a run time error message will be issued. If the error code argument is supplied, the error code will be set to one of the following states:

0	-	Indeterminate error
1	-	No error occurred
3...n	-	RDOS system error code + 3

Original contents of accumulators and carry are restored.

DELET is equivalent to DFILW.

DFILW must be referenced by an .EXTN statement.

DIR

Purpose: To define a current default directory.

Calling Sequence:
FCALL
DIR
Integer 2
FORTRAN ADDRESS of device name
FORTRAN ADDRESS of error code

Supporting Routines: FRET; .CPYL.

Subroutine Size: 16 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The device name is an ASCII byte string.

This routine issues the RDOS system call .DIR .

The error code word will be set to one of the following states:

0	-	Indeterminate error.
1	-	No error occurred.
3...n	-	RDOS system error code + 3 .

FSTAT

Purpose: To set the attributes of a FORTRAN file (not a device).

Calling Sequence:

FCALL
FSTAT
Integer 3
FORTRAN ADDRESS of the FORTRAN channel number
FORTRAN ADDRESS of the file attributes word
FORTRAN ADDRESS of the error code

Supporting Routines: FRET: .CPYL, .IOCA .

Subroutine Size: 27 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

This routine issues the RDOS system call .CHATR. Thus the bit settings for the file attributes word are as follows. Setting a bit to 1 sets the attribute for a file:

bit 0 - File is read-protected.
bit 1 - File is attribute-protected.
bit 2 - The file is a save file.
bit 15 - The file is write-protected.

The error code word will be set to one of the following states:

0	-	Indeterminate error.
1	-	No error occurred.
3...n	-	RDOS system error code + 3.

GTATR

Purpose: To get the attributes of a FORTRAN file (not a device).

Calling Sequence:

FCALL
GTATR
Integer 3
FORTRAN ADDRESS of the FORTRAN channel number
FORTRAN ADDRESS to receive the attributes word
FORTRAN ADDRESS of the error code

Supporting Routines: FRET; .CPYL, .IOCA .

Subroutine Size: 27 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

This routine issues the RDOS system call .GTATR. Thus the bit settings for the file attributes word are as follows. A logical one in a bit position indicates the file has the given attribute.

bit 0 - File is read-protected.
bit 1 - File is attribute-protected.
bit 2 - The file is a save file.
bit 12 - The file is organized contiguously.
bit 13 - The file is organized randomly.
bit 14 - The file is a permanent file.
bit 15 - The file is write-protected.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3 .

INIT

Purpose: To initialize a directory device or a magnetic tape transport.

Calling Sequence:

FCALL
INIT
Integer 3
FORTRAN ADDRESS of device name
FORTRAN ADDRESS of initialization mode word
FORTRAN ADDRESS of error code

Supporting Routines: FRET; .CPYL .

Subroutine Size: 17 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

This routine issues the RDOS system call .INIT. Thus full, partial, and partial initialization with overlays is permitted. The mode word determines which kind of initialization will occur, and has the following definitions:

-1	-	full initialization
0	-	partial initialion
1	-	partial initialization with overlays

Only full or partial initialization is permitted on magnetic tape transports. Full initialization causes a tape to be rewound and two end-of-file characters to be written. Partial initialization simply rewinds the tape and resets the tape file pointer to file zero.

The device name is an ASCII string consisting of a valid string mnemonic for either a disk or magnetic tape transport. This string is terminated by a null byte.

The error code word will be set to one of the following states:

0	-	Indeterminate error.
1	-	No error occurred.
3...n	-	RDOS system error code + 3 .

OPEN

Purpose: To open a file on a FORTRAN channel, optionally specifying a blocking factor for the file.

Calling Sequence:

FCALL
OPEN
Integer number of arguments, 4 or 5
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of open type indicator
FORTRAN ADDRESS of error code
FORTRAN ADDRESS of optional blocking factor

Supporting Routines: .DSI, FRET, IOPTR; .CPYL, .IOCAT

Subroutine Size: 174 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

APPEND and OVOPN have alternate entry points in this routine.

The file name is an ASCII byte string.

The open type indicator must be one of the following codes:

1 - Open for reading only by one or more users.
2 - Open for reading/writing by one or more users.
3 - Open for reading/writing by only one user.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3.

The file blocking factor constant is an integer indicating the number of bytes/record. For random record I/O, the blocking factor should be 128.

Up to 64 FORTRAN channel numbers are allowed, 0 through 63.

OPEN must be referenced by an .EXTN statement.

RDBLK

Purpose: To read into an array a series of disk blocks from a file that is organized either randomly or contiguously.

Calling Sequence:

FCALL
RDBLK
Integer number of arguments, 5 or 6
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of the starting block number
FORTRAN ADDRESS of the array to receive the block data
FORTRAN ADDRESS of the number of blocks to be read
FORTRAN ADDRESS of the error code
FORTRAN ADDRESS of the optional block count

Supporting Routines: FRET; .CPYL, .IOCAT

Subroutine Size: 72 octal locations of normally relocatable memory

Notes to User: Accumulators and carry are restored upon exit.

The starting block number is the logical (or relative) number of the block within the file which will be read. The first block in the file is logical block number 0, the second block is block number 1, etc.

Since blocks are each 256_{10} words long, the array size must be n*256 where n is the number of blocks to be read. No check is made to determine whether or not the size of the array is adequate. In the case where a premature end of file is detected, the optional block count argument will be set to the number of blocks actually read.

The error code word will be set to one of the following states:

0	-	Indeterminate error
1	-	No error occurred.
3...n	-	RDOS system error code +3

WRBLK has an alternate entry point in this routine.

RDBLK must be referenced by an .EXTN statement.

RENAM

Purpose: To rename a disk file.

Calling Sequence:

```
FCALL
RENAM
Integer 3
FORTRAN ADDRESS of old name
FORTRAN ADDRESS of new name
FORTRAN ADDRESS of error code
```

Supporting Routines: FRET; .CPYL

Subroutine Size: 20 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are saved in the caller's TCB.

Disk file names are byte strings of ASCII characters, packed left to right and terminated by either a carriage return, form feed, space, or null. Allowable ASCII characters in the file name are all upper case alphabetics, numerals, and $. A file name can contain any number of characters, but RDOS considers only the first 10_{10} significant.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3.

RENAM must be referenced by an .EXTN statement.

READR

Purpose: To read a series of records from a file into an array.

Calling Sequence:

FCALL
READR
Integer number of arguments, 5 or 6
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of the starting record number
FORTRAN ADDRESS of the array to receive the records
FORTRAN ADDRESS of the number of records to be read
FORTRAN ADDRESS of the error code
optional FORTRAN ADDRESS of the byte count.

Supporting Routines: FRET, MPY, DVD; .CPYL, .IOCAT, .RTER

Subroutine Size: 100 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The starting record number is the logical (or relative) number of the record within the file which will be read. The first record within the file is logical record number 0, the second is logical record number 1, etc.

The routine performs sequential reads by issuing RDOS system call .RDS . If a premature end-of-file is detected, the routine returns a byte count of all bytes read during the call, and places this count in the FORTRAN ADDRESS of the byte count, if one is provided.

No check is made to determine whether the size of the array is adequate or not.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred
3...n - RDOS system error code + 3.

WRITR has an alternate entry point in this routine.

READR must be referenced by an .EXTN statement.

RESET

Purpose: To close all currently open files and all FORTRAN channels.

Calling Sequence:

```
FCALL
RESET
```

Supporting Routines: FRET, IMIO; .IOCAT .

Subroutine Size: 27 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

This routine issues the RDOS system call .RESET. If this call is issued in a multitask environment, it must be issued only when no other task is performing any channel-related operations.

RLSE

Purpose: To release a previously initialized device or directory from the system.

Calling Sequence:

```
FCALL
RLSE
Integer 2
FORTRAN ADDRESS of device or directory name
FORTRAN ADDRESS of error code
```

Supporting Routines: FRET; .CPYL

Subroutine Size: 16 octal locations of normally relocatable memory.

Notes to User: The device name is an ASCII byte string terminated by a carriage return, null, form feed, or space.

This routine issues the RDOS system call .RLSE .

Original contents of accumulators and carry are saved in the caller's TCB.

The error code word will be set to one of the following states:

```
     0 - Indeterminate error.
     1 - No error occurred.
3...n  - RDOS system error code + 3.
```

RLSE must be referenced by an .EXTN statement.

WRBLK

Purpose: To write a series of 256-word data blocks from an array into an RDOS disk file. The disk file must be organized either randomly or contiguously.

Calling Sequence:

FCALL
WRBLK
Integer number of arguments, 5 or 6
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of the starting block number
FORTRAN ADDRESS of the array transmitting the block data
FORTRAN ADDRESS of the number of blocks to be written
FORTRAN ADDRESS of the error code
FORTRAN ADDRESS of the optional block count

Supporting Routines: FRET; .CPYL, .IOCAT

Subroutine Size: 72 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The starting block number is the logical (or relative) number of the block within the file to which writing will occur. The first block in the file is logical block number 0, the second block is block number 1, etc.

Since disk blocks are each 256_{10} words in length, the array size must be n*256 where n is the number of blocks to be written. No check is made to determine whether or not the size of the array is adequate. In the case where disk overflow occurs, the optional block count argument will be set to the number of blocks actually written.

The error code word will be set to one of the following states:

0 - Indeterminate error.
1 - No error occurred.
3...n - RDOS system error code + 3

RDBLK has an alternate entry point in this routine.

WRBLK must be referenced by an .EXTN statement.

WRITR

Purpose: To write a series of records from an array into a file.

Calling Sequence:

FCALL
WRITR
Integer number of arguments, 5 or 6
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of the starting record number
FORTRAN ADDRESS of the array transmitting the records
FORTRAN ADDRESS of the number of records to be written
FORTRAN ADDRESS of the error code
optional FORTRAN ADDRESS of the byte count

Supporting Routines: FRET, MPY, DVD; .CPYL, .IOCAT, .RTER

Subroutine Size: 100 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

The starting record number is the logical (or relative) number of the block within the file to which writing will occur. The first record within the file is logical record number 0, the second record is logical record number 1, etc.

The routine performs sequential writes by issuing the RDOS system call .WRS . No check is made to determine whether or not the size of the array is adequate.

The error code word will be set to one of the following states:

0 - Indeterminate error
1 - No error occurred.
3...n - RDOS system error code + 3

If disk overflow occurs, RDOS system error code ERSPC will be given.

READR has an alternate entry point in this routine.

WRITR must be referenced by an .EXTN statement.

REAL TIME BIT MANIPULATION

ICLR

Purpose: To clear a bit in a 16-bit word.

Calling Sequence:

FCALL
ICLR
Integer 2
FORTRAN ADDRESS of word with bit position to be cleared
FORTRAN ADDRESS of bit position indicator

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 50 octal locations of normally relocatable memory.

Notes to User: ITEST and ISET have alternate entry points in this routine.

Accumulators and carry are restored upon exit.

This routine clears one bit in a word, the bit selected according to the bit position indicator which may be any integer from 0 to 15_8. The following bit position indicators cause the following bit positions to be cleared:

MSB															LSB
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Bit Position Indicators

ISET

Purpose: To set a bit in a 16-bit word.

Calling Sequence:

```
FCALL
ISET
Integer 2
FORTRAN ADDRESS of word with bit position to be set
FORTRAN ADDRESS of bit position indicator
```

Supporting routines: FRET; .CPYL, .RTER .

Subroutine Size: 50 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

This routine sets one bit position in a word. The bit position set is selected according to the bit position indicator which may be any integer from 0 to 15_8. The following bit position indicators cause the following bit positions to be set:

Bit Position Indicator	Bit Position Set
0	Least significant bit.
1	Next least significant bit.
.	.
.	.
.	.
15	Most significant bit.

ITEST and ICLR have alternate entry points in this routine.

ITEST

Purpose: To examine a bit in a 16-bit word.

Calling Sequence:

FCALL
ITEST
Integer 3
FORTRAN ADDRESS of result
FORTRAN ADDRESS of word to be examined
FORTRAN ADDRESS of bit position indicator

Supporting Routines: FRET; .CPYL, .RTER .

Subroutine Size: 50 octal locations of normally relocatable memory.

Notes to User: Accumulators and carry are restored upon exit.

This routine performs a logical AND between the word to be examined and a bit mask, placing the result in the FORTRAN ADDRESS of the result.

The bit position indicator is an integer from 0 to 15_8. This word is used as a power to which the base 2 is raised, creating a bit mask for the AND operation. Thus the following bit position indicators cause the following bit positions to be tested:

Bit Position Indicator	Bit Position Examined
0	Least significant bit.
1	Next least significant bit.
.	.
.	.
.	.
15	Most significant bit.

ICLR and ISET have alternate entry points in this routine.

Writing a Real Time Program

Since RT FORTRAN is a superset of non-real time FORTRAN, all information given in the DGC FORTRAN IV USER'S MANUAL, 093-000053, applies to the writing of an RT FORTRAN program. Additionally, the following points must be considered when writing an RT FORTRAN program.

The first statement which must be found in the main RT FORTRAN program is the CHANTASK statement. This statement specifies both the maximum number of tasks which will be active at any one moment and the maximum number of RDOS channels which will be required. The CHANTASK statement is a specification statement which is non-executable.

The format of the CHANTASK statement is as follows:

$$\boxed{\text{CHANTASK } \underline{i}_1, \underline{i}_2}$$

where: $\underline{i}_1$ is an integer representing the number of RDOS channels which will be required by the program.

$\underline{i}_2$ is an integer representing the maximum number of active tasks at any one moment.

Users are cautioned to be precise in specifying $\underline{i}_2$, since the use of a value larger than the maximum number of tasks which will be active at any one moment will cause the run time stack area to be segmented into more subdivisions than are required at any one moment, with a consequently smaller size allocated for each segment. The use of a value smaller than this maximum value will cause a fatal run time error to occur.

If the CHANTASK statement is omitted, a default value of 1 task and 8 RDOS channels will be presumed and TMIN (the RDOS minimum Task Scheduler) will be loaded.

The EXTERNAL statement given in non-real time FORTRAN has been expanded for RT FORTRAN to include the names of the entry points of all tasks which will be initiated by the main program. Correspondingly, each Task subprogram must be declared by a TASK statement so that the Task module can become accessible by other programs. The format of the TASK statement is as follows:

$$\boxed{\text{TASK } \underline{\text{taskname}}}$$

Writing a Real Time Program (Continued)

where: taskname is the name given to the Task subprogram.

Since a task is a logically complete execution path through a program, parameters cannot be passed to tasks (although one-word messages can be passed to them as mentioned earlier).

Since RT FORTRAN programs will be written in FORTRAN source code, users should be aware of the FORTRAN equivalence of the assembly language calling sequences given in this appendix to describe each routine. As stated in the introduction to this manual, the FORTRAN statement:

CALL SUBROUTINE (parameter_1, . . . , parameter_n)

is equivalent to the assembly language calling sequence:

```
FCALL
SUBROUTINE
Integer n
.
.
.
FORTRAN ADDRESS of parameter_n
```

Thus, for example, a FORTRAN call to the PRI subroutine would be of the form:

CALL PRI (priority address)

The following two pages summarize the FORTRAN calls found only in the RDOS FORTRAN run time library.

Task Commands

CALL ABORT (i.d. number,error word)	Kill a task specified by i.d. number.
CALL AKILL (priority)	Ready a class of tasks.
CALL ARDY (priority)	Suspend a class of tasks.
CALL ASUSP (priority)	Change the priority of a specific task.
CALL CHNGE (i.d. number, new priority error word)	
CALL FTASK (name, error return,priority)	Create a task.
CALL HOLD (i.d. number, error word)	Suspend a specific task.
CALL ITASK (name,i.d. number,priority, error word)	Create a task and assign it an i.d. number.
CALL KILL	Change a task's priority.
CALL PRI (priority)	Receive a task message.
CALL REC (i.d. number, error word)	Ready a specific task.
CALL RELSE (i.d. number, error word)	Obtain the status of a specific task.
CALL STTSK (i.d. number, task status word, error word)	Suspend the calling task.
CALL SUSP	Kill the calling task.
CALL XMT (message address, message, error return)	Transmit a task message.
CALL XMTW (message address, message, error return)	Transmit a task message and wait for its receipt.

Clock Calendar Commands

CALL DATE (date array, error word)	Get the current date.
CALL FDELY (number of RTC cycles)	Suspend the calling task for a specified interval of time.
CALL FGTIM (hour, minute, second)	Get the current time.
CALL FSTIM (hour, minute, second)	Set the system clock.
CALL TIME (time array, error word)	Get the current time of day.

Interrupt Commands

CALL FINRV (device-code)	Remove a special user device from the system.
CALL FINTD (device-code, user-dct)	Introduce a special user device to the system.
CALL IXMT (message address, message, error word)	Transmit a message from a user interrupt routine.

Swap and Overlay Commands

CALL FBACK	Return to the next higher program level.
CALL FCHAN (save file name)	Perform a program chain.
CALL FOVLD (channel number, overlay name, conditional load flag, error word)	Load a user overlay in a multitask environment.
CALL FOVRL (overlay name,error word)	Release an overlay area.
CALL FQTASK (overlay name or dummy, task name, task queue array, error word, optional core-resident task flag)	Execute a task at periodic intervals. If the task is in an overlay, load the overlay so that the task can be executed.
CALL FSWAP (save file name)	Save the current program level and call in a program swap.

Swap and Overlay Commands: (Cont'd)

CALL OVEXT (overlay name, return address)	Release an overlay and return control to a specified address.
CALL OVEXX (overlay name, return address)	Release an overlay and return control to a specified address.
CALL OVKIL (overlay name)	Kill a calling task and release its overlay.
CALL OVKIX (overlay name)	Kill a calling task and release its overlay.
CALL OVLOD (channel number, overlay name, conditional load flag, error word)	Load a user overlay in a single task environment.
CALL OVOPN (channel number, overlay name, error word)	Open an overlay file.

File and I/O Commands:

CALL APPEND (channel number, file name, open type indicator, error word, optional blocking factor)	Open a file for appending.
CALL CFILW (file name, file type indicator, optional file size, error word)	Create a disk file.
CALL CLOSE (channel number, error word)	Close a file and channel.
CALL DFILW (file name, optional error word)	Delete a disk file.
CALL DIR (device name, error word)	Change default directory or device.
CALL FSTAT (channel number, file attributes, error word)	Set file attributes.
CALL GTATR (channel number, file attributes, error word)	To examine file attributes.
CALL INIT (device name, initialization mode word, error word)	To initialize a magnetic tape cassette, disk device or directory.
CALL OPEN (channel number, file name, open type indicator, error word, optional blocking factor)	Open a file on a FORTRAN channel.
CALL RDBLK (channel number, starting block number, receiving array, number of blocks to be read, error word, optional block count)	Read a series of disk blocks into an array.
CALL READR (channel number, starting record number, receiving array, number of records to be read, error word, optional byte count)	To read a series of file records into an array.
CALL RENAM (old name, new name, error word)	Rename a disk file.
CALL RESET	Close all open files.
CALL RLSE (device name, error word)	Release a previously initialized device or directory.
CALL WRBLK (channel number, starting block number, transmitting array, number of blocks to be written, error word, optional block count)	To write a series of data blocks from an array into a file.
CALL WRITR (channel number, starting record number, transmitting array, number of records to be written, error word, optional byte count)	To write a series of records from an array into a file.

Bit Manipulation Commands:

CALL ICLR (word, bit indicator)	Clear a bit in a 16-bit word.
CALL ISET (word, bit indicator)	Set a bit in a 16-bit word.
ITEST (word to be examined, bit indicator)	Examine a bit in a 16-bit word.

RT FORTRAN Program Example

The following RT FORTRAN program example consists of three program modules, each separately compiled by means of three commands:

```
FORT MAIN )
FORT TIMPLT )
FORT QUAD )
```

After compilation, the binaries are loaded by means of the following command sequence:

```
RLDR MAIN TIMPLT QUAD FMT.LB FORT1.LB FORT2.LB FORT3.LB FORT4.LB )
```

(Note that the multitask library, FMT.LB, is not loaded in single task programs.) The first module is the main program. Its functions are to type the title, ***REAL TIME QUADRATIC EQUATION SOLVER***, on the line printer and then to activate the two tasks whose logic is contained in the two remaining modules.

The first task module, TIMPLT, prints a counter on the line printer, one count per line, 55 lines per page. The counter is incremented once each second, given a real time clock cycle of 100 milliseconds.

The second task module, QUAD, accepts coefficients for a quadratic equation from the teletypewriter keyboard and prints these coefficients on the line printer at the moment the carriage return terminator is detected. If the roots of the equation are complex, a message is output. Otherwise, the two real roots, X_1 and X_2, are also printed on the line printer. The program runs continuously until the user aborts it by means of a CTRL A break.

Some sample output produced by this real time program follows the listing of the source program modules.

EXAMPLE SOURCE LISTING

```
C  MAIN PROGRAM
       CHANTASK 3,3
       EXTERNAL QUAD, TIMPLT
       WRITE (12) " ***REAL TIME QUADRATIC EQUATION SOLVER***"
C  CREATE TIME PLOT TASK AT NEXT HIGHEST PRIORITY
       CALL FTASK (TIMPLT,$1Ø,1)
C  CREATE QUADRATIC SOLVER TASK AT LOWEST PRIORITY
       CALL FTASK (QUAD,$1Ø,1)
       CALL KILL
  1Ø   WRITE (1Ø) " NOT ENOUGH TCB'S"
       END

       TASK TIMPLT
C  SET OUTPUT COUNTER TO ZERO
       N = Ø
  1    LINES = Ø
C  RESET LINE COUNTER TO ZERO, TOP OF PAGE
  2    LINES = LINES + 1
       N = N + 1
       CALL FDELY (1Ø)
C  IF BOTTOM OF PAGE, GO TO TOP OF NEXT PAGE
       IF (LINES.EQ.55) GO TO 1Ø
       WRITE (12) N
       GO TO 2
  1Ø   WRITE (12) N
       WRITE (12,2Ø)
  2Ø   FORMAT (1H1)
       GO TO 1
       END

       TASK QUAD
C  GET QUADRATIC EQUATION COEFFICIENTS
  1ØØ  READ (11) A,B,C
C  F(X) = A*X**2 + B*X +C
C  IF COMPLEX ROOTS, OUTPUT COEFFICIENTS AND FLAG
       IF ((B**2-4*A*C).LT.Ø) GO TO 1Ø
C  FIND THE REAL ROOTS
       X1R = (-B+(B**2-4*A*C)**.5)/(2*A)
       X2R = (-B-(B**2-4*A*C)**.5)/(2*A)
C  OUTPUT  THE COEFFICIENTS AND THE TWO REAL ROOTS
       WRITE (12,1) A,B,C,X1R,X2R
  1    FORMAT (1HØ,"A= ",F1Ø.4," B= ",F1Ø.4," C= ",
     1 F1Ø.4," X1= ",F1Ø.4," X2= ",F1Ø.4)
       GO TO 1ØØ
  1Ø   WRITE (12,2) A,B,C
  2    FORMAT (1HØ,"*** COMPLEX ROOTS***", "A= ",F1Ø.4,"B= "
     1 ,F1Ø.4,"C= ",F1Ø.4)
       GO TO 1ØØ
       END
```

SAMPLE PROGRAM OUTPUT

```
 ***REAL TIME QUADRATIC EQUATION SOLVER***
         1
         2
         3
         4

*** COMPLEX ROOTS***A=      1.0000B=      2.0000C=      3.0000
         5
         6
         7
         8
         9
        10
        11
        12

A=       1.0000 B=       0.0000 C=    -16.0000 X1=      3.9999 X2=     -3.9999
        13
        14
        15
        16
        17
        18
        19
        20
        21
        22
        23
        24
        25
        26
        27

*** COMPLEX ROOTS***A= 12345.6000B= 12345.6000C=  9876.5400
        28
        29
        30
        31
        32
        33
        34
        35
        36
        37
        38
        39
        40
        41
        42
        43
        44
```

Preserving Reentrance During Interrupt Processing

As noted earlier, when a special user interrupt occurs in a real time environment, all task states are frozen. Thus no saving of task states is required when processing interrupts, since interrupt processing occurs apart from task considerations. If, however, users wish to issue FORTRAN calls as part of their interrupt processing, it is imperative that certain stack variables be saved before these calls are made. Failure to preserve stack variables will disrupt management of the run time stack when the multitask environment becomes unfrozen. (Note that the system saves these variables when interrupts are generated by SYSGENed devices.)

When the multitask environment becomes frozen, page zero contains the variables for the stack segment of the FORTRAN task which was in control of the CPU at the time of the interrupt. Therefore if FORTRAN calls are to be issued from interrupt service routines, these routines must utilize the remaining free area in the frozen executing task's stack segment for run time variable storage. Although interrupts must be turned off while the interrupt processing logic is saving the segment variables, interrupts may be enabled as soon as these state variables have been preserved.

The segment stack variables which must be saved by the interrupt processing routine are as follows:

.SV0 *
.OVFL *
FSP
SP *
NSP

Additionally, a new QSP*value must be calculated which corresponds to the new FSP.

Of the five variables which must be saved, .SV0 and .OVFL may be saved in the new stack frame. SP may simply be incremented by one before the first FORTRAN call, and decremented by one after the last FORTRAN call in the interrupt servicing routine. NSP must be incremented by 6 and similarly decremented by 6 after the last FORTRAN call. A convenient location in which to store the old FSP is in the new frame's FOSP. The old FSP must be restored upon exit from the interrupt service routine. In order to create the new frame (and new FSP), the following adjustment must be made to the old FSP:

$$C(FSP') = C(FSP) + FLGT + 2*FFEL$$

A new value for QSP is calculated by adding PARF displacement FAC2 to its

* See the FPZERO module, Chapter 9.

Preserving Reentrance During Interrupt Processing (Continued)

associated FSP:

$$C(QSP') = FAC2 + C(FSP')$$

The old value for QSP must be restored when its associated FSP value is restored.

The following code example adjusts FSP, stores the old FSP in the new frame's FOSP, and stores .SV0 and .OVFL in this frame's two temporaries:

```
         LDA    3,FSP                ;GET THE FROZEN FSP
         MOV    3,2
         LDA    0,FLGT,3             ;ADJUST NEW FSP
         LDA    1,MAGIC              ;ADJUST NEW FSP
         ADD    0,1                  ;ADJUST NEW FSP
         ADD    1,3                  ;ADJUST NEW FSP
         STA    3,FSP                ;INSTALL THE NEW FSP
         LDA    0,TWO                ;RESERVE TWO TEMPORARIES FOR .SV0
         STA    0,FLGT,3             ;AND .OVFL
         STA    2,FOSP,3             ;SAVE THE FROZEN FSP
         LDA    0,.SV0               ;GET THE FROZEN .SV0
         STA    0,SAV0,3             ;SAVE IT
         LDA    0,.OVFL              ;GET THE FROZEN .OVFL
         STA    0,OVFL,3             ;SAVE IT
         .
         .
         .
MAGIC:   2*FFEL                      ;FFEL = 11 OCTAL, FOUND ON PARF
TWO:     2
SAV0 =   FTSTR
OVFL =   SAV0+1
```

APPENDIX F

The FORTRAN parameter tape, PARF, must be assembled with any user-written programs using thc FORTRAN runtime libraries. A listing of the RDOS FORTRAN parameters (090-001000-02) follows.

```
ØØØ1

                      ; COPYRIGHT (C) DATA GENERAL CORPORATION, 1972, 1973
                      ; ALL RIGHTS RESERVED.


                      ; FORTRAN RUN-TIME PARAMETER TAPE


                      ; DEFINE THE CURRENT STACK POINTER LOCATION

         ØØØØ16 .DUSR   FSP=    LSP


                      ; DEFINE THE FIXED STACK DISPLACEMENTS

         1776 1Ø .DUSR  FRTN=   -17Ø     ; DON'T MODIFY THE DISPLACEMENTS
         1776Ø7 .DUSR   FAC2=   -171     ;  FOR FRTN, FAC2, FAC1, FACØ
         1776Ø6 .DUSR   FAC1=   -172     ; AC1
         1776Ø5 .DUSR   FACØ=   -173     ; ACØ
         1776Ø4 .DUSR   FCRY=   -174     ; CARRY
         1776Ø3 .DUSR   FEAD=   -175     ; SUBROUTINE ENTRY ADDRESS
         1776Ø2 .DUSR   FPLP=   -176     ; PARAMETER LIST POINTER
         1776Ø1 .DUSR   FOSP=   -177     ; OLD STACK POINTER
         1776ØØ .DUSR   FLGT=   -2ØØ     ; STACK FRAME LENGTH

         ØØØØ11 .DUSR   FFEL=   11       ; FIXED LENGTH OF THE STACK FRAME
         177611 .DUSR   FTSTR=  -167     ; TEMPORARY STORAGE STARTING DISKP.
         177611 .DUSR   TMP=    FTSTR
         1776ØØ .DUSR   FZD=    -2ØØ     ;FUDGE FACTOR FOR ZEROTH FORTRAN DISPLAC
```

```
A  00002



             ; DEFINE THE RUN-TIME ERROR CODES
             ;       FATAL ERRORS USE @"CODE" WHERE THE "CODES" ARE
             ;        GIVEN BELOW

             ; DEFINE THE ALC MAGIC
      100011 .DUSR   ENOP=    @11              ; ALC NO-OP
      000020 .DUSR   EOS=     1B11             ; ERROR CODE OFFSET
      040000 .DUSR   FATAL=   1B1              ; FATAL ERROR BIT


      100031 .DUSR   FEMOF=   1,*EOS+ENOP ; STACK OVERFLOW
      100051 .DUSR   FECGT=   2,*EOS+ENOP ; COMPUTED GOTO ERROR
      100111 .DUSR   FEDVZ=   4,*EOS+ENOP      ; DIVISION BY ZERO
      100131 .DUSR   FEIOV=   5,*EOS+ENOP      ; INTEGER OVERFLOW
      100151 .DUSR   FEIPR=   6,*EOS+ENOP      ; INTEGER POWER OVERFLOW
      100171 .DUSR   FEFUF=   7,*EOS+ENOP      ; FLOATING POINT UNDERFLOW
      100211 .DUSR   FEFOF=   8,*EOS+ENOP      ; FLOATING POINT OVERFLOW
      100231 .DUSR   FEFMR=   9,*EOS+ENOP      ;ILLEGAL FORMAT SYNTAX
      100271 .DUSR   FELER=   11,*EOS+ENOP     ;LOGIC CONVERSION ERROR
      100331 .DUSR   FENER=   13,*EOS+ENOP     ;NUMBER CONVERSION ERROR
      100351 .DUSR   FEIOR=   14,*EOS+ENOP     ;I/O ERROR
      100371 .DUSR   FEFLD=   15,*EOS+ENOP     ;FIELD ERROR
      100411 .DUSR   FESQR=   16,*EOS+ENOP     ;SQUARE ROOT OF NEGATIVE NUMBER
      100431 .DUSR   FELOG=   17,*EOS+ENOP     ;LOG OF NEGATIVE NUMBER
      100451 .DUSR   FECLS=   18,*EOS+ENOP     ;CHANNEL NOT OPEN
      100471 .DUSR   FEOPN=   19,*EOS+ENOP     ;CHANNEL ALREADY OPEN
      100511 .DUSR   FECHN=   20,*EOS+ENOP     ;NØ CHANNELS AVAILABLE
      100531 .DUSR   FEDOS=   21,*EOS+ENOP     ;DOS EXCEPTIONAL STATUS
      100611 .DUSR   FEEXP=   24,*EOS+ENOP     ; EXPONENTIAL OVER/UNDERFLOW
      100631 .DUSR   FEOOB=   25,*EOS+ENOP     ;ARRAY REFERENCE OUT OF BOUNDS.
      100651 .DUSR   FEPWR=   26,*EOS+ENOP     ;-VE BASE FOR FLOATING POWER
      100671 .DUSR   FENSO=   27,*EOS+ENOP     ;NUMBER STACK OVERFLOW
      100711 .DUSR   FEBNI=   28,*EOS+ENOP     ;BACKSPACE NOT IMPLEMENTED
      100731 .DUSR   FERST=   29,*EOS+ENOP     ; ATTEMPT TO RESTORE CHANNEL
                                               ; STATUS NOT PREVIOUSLY SAVED
      100751 .DUSR   FEQTS =  30,*EOS+ENOP     ;QUEUED TASK ERROR
      100771 .DUSR   FEFSR=   31,*EOS+ENOP     ;SEEK ON NONRANDOM FILE
      101011 .DUSR   FEOVL=   32,*EOS+ENOP     ;OVERLAY ABORTED
      101031 .DUSR   FEARG =  33,*EOS+ENOP     ;ILLEGAL ARGUMENT
      101051 .DUSR   FEDEL =  34,*EOS+ENOP     ;DELETE ERROR(FILE OPEN)
      101071 .DUSR   FEOVK =  35,*EOS+FNOP     ;OVERLAY ERROR IN OVERLAY KILL

                     ;SYSTEM ERROR DISPLACEMENT

      000003 .DUSR   CEDSP = 3
```

```
[illegible] MCV3

                ; DEFINE THE FLOATING POINT INTERPRETER PARAMETERS

        000012 .DUSR    MAXPR=  10.     ; MAXIMUM PRECISION
                                        ; (NO. OF WDS. OF MANTISSA MAX.)




                ; DEFINE THE FLOATING REGISTER EQUIVALENCES

        177775 .DUSR    SGN=    -3      ; SIGN (BIT 15)
                                              ; (LEAVE AS MOST NEG. DISPLACE.)
        177776 .DUSR    EX=     -2      ; EXPONENT (BITS 9-15)
        177777 .DUSR    PRC=    -1      ; REGISTER PRECISION
        000000 .DUSR    MANT=   0       ; HIGH ORDER MANTISSA WORD



                ; DEFINE I/O CHANNEL ASSIGNMENT TABLE.

        177777 .DUSR    CHCNT=-1        ;CHANNEL COUNT
        000000 .DUSR    CATFL=0         ;FLAGS
        000001 .DUSR    CATRL=1         ; RECORD LENGTH (RANDOM ONLY)
        100000 .DUSR    CATOP=1B0       ;OPEN SWITCH
        040000 .DUSR    CATMD=1B1       ;MODE
```

```
0004 .MAIN
                 ;       DEFINE THE NUMBER STACK DISPLACEMENTS

      000000             .DUSR   OP1S=0           ;CURRENT STACK OPERAND - SIGN
      000001             .DUSR   OP1X=1           ;EXPONENT
      000002             .DUSR   OP1M=2           ;MANTISSA

      177772             .DUSR   OP2S=-6          ;LAST OPERAND - SIGN
      177773             .DUSR   OP2X=-5          ;EXPONENT
      177774             .DUSR   OP2M=-4          ;MANTISSA

      000006             .DUSR   OP3S=6           ;NEXT OPERAND - SIGN
      000007             .DUSR   OP3X=7           ;EXPONENT
      000010             .DUSR   OP3M=10          ;MANTISSA

      000006             .DUSR   REGL=6           ;REGISTER LENGTH

                ; TASK EXTENTION PARAMETERS
                ; N.B. THESE MUST CORRESPOND TO THE FPZERO .BLK DEFINITIONS
      000000 .DUSR   TNDSP= 0
      000001 .DUSR   TAFSP= 1
      000002 .DUSR   TSP= 2
      000003 .DUSR   TOVFL= 3
      000004 .DUSR   TSV0= 4
      000005 .DUSR   TQSP= 5
      000006 .DUSR   TFLSP= 6        ;MUST BE LAST DUE TO FLOATING POINT LOAD


      000007 .DUSR   TLEXN= TFLSP-TNDSP+1


                         .EOT    ;END OF PARAMETERS
```

cut along dotted line

DATA GENERAL CORPORATION
PROGRAMMING DOCUMENTATION
REMARKS FORM

DOCUMENT TITLE ______________________________

DOCUMENT NUMBER (lower righthand corner of title page) ______________

Specific Comments. List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

General Comments and Suggestions for Improvement of the Publication.

FROM: Name: ______________________ Date: __________
Title: ______________________
Company: ______________________
Address: ______________________
____

FOLD DOWN FIRST FOLD DOWN

FIRST
CLASS
PERMIT
No. 26
Southboro
Mass. 01772

BUSINESS REPLY MAIL

No Postage Necessary If Mailed In The United States

Postage will be paid by:

Data General Corporation

Southboro, Massachusetts 01772

ATTENTION: Programming Documentation

FOLD UP SECOND FOLD UP

STAPLE