NOVA® -LINE
FORTRAN IV
RUN TIME
LIBRARY

User's Manual

093-000068-05

Ordering No. 093-000068

Printed in the United States of America
Rev. 05, February 1975

Original Release - December 1971
First Revision - March 1972
Second Revision - July 1972
Third Revision - February 1973
Fourth Revision - September 1974
Fifth Revision - February 1975
086-000033-00 - February 1976

This revision of the FORTRAN IV Run Time User's Manual, 093-000068-05, supersedes 093-000068-04 and constitutes a minor revision.

DataGeneral

SOFTWARE ADDENDUM

Addendum to NOVA® -LINE FORTRAN IV RUN TIME LIBRARY User's Manual

086-000033-00

This addendum updates manual 093-000068-05.

See UPDATING INSTRUCTIONS on reverse.

Ordering No. 086-000033

Printed in the United States of America
Rev. 00, February 1976

UPDATING INSTRUCTIONS

This addendum (086-000033-00) to the FORTRAN IV Run Time Library User's Manual (093-000068-05) supplies additions and corrections to run time routine entry points.

To update your copy of 093-000068-05, remove and insert the following pages:

Remove	Insert
Title/Notice	Title/Notice
i/ii	i/ii
v/vi	v/vi
2-3/2-4	2-3/2-4
5-5/5-6	5-5/5-6
6-1/6-2	6-1/6-2
16-3 through 16-6	16-3 through 16-6
.....	16-10.1
17-5/17-6	17-5 through 17-6.1

Insert this sheet immediately behind the new Title/Notice page.

A vertical bar or an asterisk in the margin of a page indicates substantive change or deletion, respectively, from 093-000068-05.

The addendum number appears on the lower inside corner of only those pages changed by this addendum.

CHAPTER 5 - USING RDOS FEATURES

CHAPTER 6 - INPUT/OUTPUT

CHAPTER 7 - USING THE RUN TIME LIBRARY

CHAPTER 8 - INTEGER ROUTINES

CHAPTER 9 - SINGLE PRECISION FLOATING ROUTINES

CHAPTER 9 - SINGLE PRECISION FLOATING ROUTINES (Continued)

CHAPTER 10 - DOUBLE PRECISION FLOATING POINT ROUTINES

CHAPTER 11 - SINGLE PRECISION COMPLEX ROUTINES

CHAPTER 12 - DOUBLE PRECISION COMPLEX ROUTINES

CHAPTER 13 - MIXED MODE ROUTINES

CHAPTER 14 - STRING/BYTE MANIPULATION ROUTINES

CHAPTER 15 - POINTERS AND DISPLACEMENTS

CHAPTER 16 - LINKAGE AND INITIALIZATION ROUTINES

CHAPTER 17 - INPUT/OUTPUT ROUTINES

CHAPTER 18 - MISCELLANEOUS FORTRAN SUPPORT ROUTINES

CHAPTER 19 - ARRAY HANDLING ROUTINES

CHAPTER 20 - DEVICE/DIRECTORY MAINTENANCE ROUTINES

CHAPTER 21 - OVERLAY ROUTINES

CHAPTER 22 - TASK ROUTINES

CHAPTER 23 - FOREGROUND/BACKGROUND ROUTINES

CHAPTER 24 - COMMUNICATION ROUTINES

CHAPTER ONE

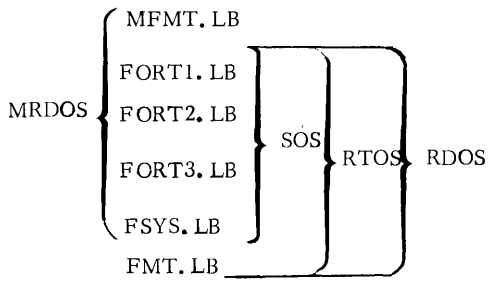
RUN TIME LIBRARY STRUCTURE

The FORTRAN IV Run Time Library allows the user to execute FORTRAN programs in either a single-task or multitask real time environment. In addition to the 8K Stand Alone FORTRAN, operating systems supported are Data General Corporation's SOS (Stand Alone Operating System), RTOS (Real Time Operating System), and RDOS/MRDOS (Real Time Disk Operating System and Mapped Real Time Disk Operating System.)

All routines in the Run Time Library are reentrant and relocatable, making them suitable for use in time-sharing environments. The stack structure of the library is fundamental to its reentrancy, as all variables not declared as being in COMMON storage are stored on the stacks.

The appropriate version of the Run Time Library must be configured with the operating system being used. Certain routines, found in FORT1.LB, FORT2.LB, and FORT3.LB are common to all versions of the Run Time Library. The diagram below illustrates which libraries are used with which operating system. Note that RDOS and RTOS use FMT.LB and that MRDOS is supplied with MFMT.LB instead. Additional libraries containing the multiply/divide routines may be supplied according to customer configuration requirements. These libraries are: SOFTMPYD.LB, HMPYD.LB, NMPYD.LB.

For the purpose of discussing the FORTRAN IV Run Time Library, the SOS single task environment may be considered a subset of the RTOS and RDOS single task environments. Except where noted, discussions of RDOS and RTOS single task FORTRAN are also applicable to SOS.



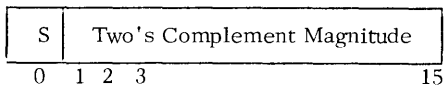
DATA STORAGE

Fixed Point Numbers (Integers)

A fixed point number is represented by a 16-bit word, with bit 0 containing the sign (0 for positive, 1 for negative). Bits 1-15 express the magnitude of the number in two's complement notation. The magnitude ranges from $-(2^{15}-1)$ to $(2^{15}-1)$, or $-32,767_{10}$ to $32,767_{10}$. The representation for zero is a word consisting of all zeroes. (100000 is an illegal number.)

* Note that the FMT libraries for RDOS and RTOS are different. Other libraries are the same.

Fixed Point Numbers (Integers) (Continued)



where S = 0 for positive, 1 for negative

Real Numbers

Real numbers may be either single or double precision, in packed or unpacked form. Numbers on the number stack are always unpacked; all other numbers are usually in packed form.

Single precision floating point numbers (SPFL) in packed format are stored in two sequential sixteen bit words, with the high order word preceding the low order word in memory. Bit 0 contains the sign (0 for positive, 1 for negative); bits 1-7 represent the exponent, and bits 8-31 are the mantissa.

The exponent is expressed in excess 64 form; the exponent is to the base 16, to which is added an offset of 100_8. A negative number is formed by setting the sign bit of the corresponding positive number.

Zero is represented as two sequential all zero words, but any SPFL number input to a routine will be considered zero if it has an all zero mantissa.

All SPFL numbers input to the library routines must be in normalized form; in addition, all floating point numbers in computations are maintained in normalized form. An SPFL number is considered normalized if there is at least one binary 1 in any of the first four bit positions of the mantissa (bits 8-11). 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 are stored in four sequential sixteen bit words. The sign and exponent are stored in the same manner as for packed SPFL numbers. The remaining 56 bits 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 values 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.

Complex Numbers

Single precision complex numbers in packed format consist of two sequential packed SPFL numbers, the high order portion representing the real portion of the number and the second representing the imaginary part of the complex number. Four sequential memory locations are required, and the bit definitions, range and significance of the real and imaginary parts are the same as those for SPFL numbers.

Double precision complex numbers in packed format consist of two sequential packed DPFL numbers, the high order portion representing the real portion of the number and the second representing the imaginary part of the complex number. Eight sequential memory locations are required, and the bit definitions, range and significance of the real and imaginary parts are the same as those for DPFL numbers.

BYTE MANIPULATION

For processing 8-bit bytes, a byte pointer is used in which bits 0-14 are the address of the memory location that contains or will receive the byte, and bit 15 specifies which half (0 left, 1 right). Note that this is exactly the reverse of the convention described in "How To Use the Nova Computers". In order to insure that packing occurs left to right in accordance with the pointer structure, ". TXTM 1" should be included in any source program which generates ASCII text messages.

CHAPTER TWO

STACK STRUCTURE AND LINKAGE

In a multitasking environment, the Run Time (RT) initializer partitions the Run Time Area into equal segments, one segment for each task specified at the beginning of the FORTRAN program. 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. Note that the segmenting of the Run Time Area is determined by the number of tasks specified, not by the number of tasks actually active at any given time.

Each segment has an SP Stack, Number Stack, and Run Time Stack. The SP Stack is 40_8 words in length, and the Number Stack is 330_8 words; the Run Time Stack occupies the rest of the segment area. The Number Stack size may not be adjusted, and the allocation of a selected Number Stack may not be omitted.

Each segment is preceded by an eight word state save area, containing a family of pointers and displacements to describe it uniquely. These stack pointers and flags are FSP, .NDSP, AFSE, SP, QSP, .OVFL, .SVO, and NSP.

SP STACK

The SP Stack is a block of 40_8 sequential locations with a page zero pointer, SP; it is used for general purpose temporary storage, 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
```

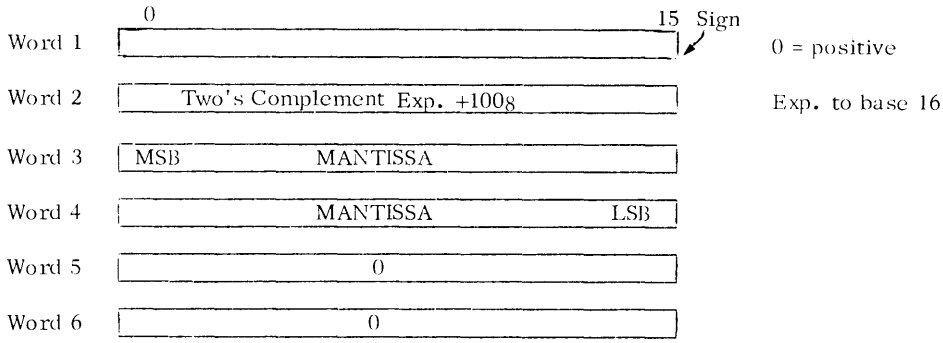
SP Stack overflow is a fatal error and is undetected by the library routines.

NUMBER STACK

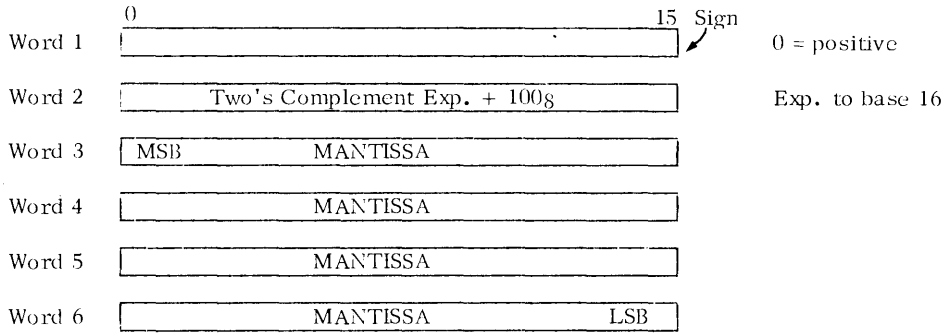
The Number Stack is a 330_8 word block of locations reserved for the storage of numeric values. The numbers may be I/O for the arithmetic routines or temporary storage for computations from the routines. The entire storage area of the number stack need not be used. The number stack expands dynamically as numbers are loaded onto it and contracts as they are removed; however, it never exceeds its maximum of 330_8 words. The stack is built in the direction of increasing addresses, with .NDSP pointing to the end of the stack and NSP pointing to the current top of the stack (the most recently loaded number).

All numbers on the stack are stored in sequential six word frames (or multiples of six word frames), illustrated in the following diagrams. An attempt to load a number onto an already filled number stack will result in error message FENSO. No check is made for stack underflow, an attempt to load a number below the first frame on the 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 in core) SPFL number represents the imaginary portion, and the next SPFL number represents the real position of the complex number.

RUN TIME STACK

Each Run Time Stack consists of one or more FORTRAN stack frames. The Run Time Stack expands and contracts dynamically with the execution of a main program, expanding when more nests of subroutines are called or as subroutines are called which demand temporary storage. As the Run Time Stack within a given segment expands, any FORTRAN stacks created earlier for subroutines already executed are overwritten by the new stacks. The stack is built in the direction of increasing memory addresses, and stack overflow occurs if more storage area than memory available at run time is demanded. .AFSE is a word used to determine the end or uppermost memory location available for the Run Time 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 within a run time stack consists of an initial 11_8 word header; most routines also require a varying length series of temporary locations following each header.

RUN TIME STACK (Continued)

Each header location is used to store a specific type of information pertaining to the subroutine that 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 location 16_8, and also defines the fixed displacements and mnemonic assignments of each location in the stack header. These mnemonic assignments and fixed displacements from FSP are listed below:

FLGT -200	Length of variable portion of the stack frame.
FOSP -177	Old FSP.
FPLP -176	Number of arguments copied, if .FARL or .FARG copied the argument list.
FEAD -175	Entry address to the last routine called by this routine.
FCRY -174	State of carry at the time this routine issues a subroutine call. Bit 15=0 if carry was 0.
FAC0 -173	Contents of AC0 when this routine issues a subroutine call.
FAC1 -172	Contents of AC1 when this routine issues a subroutine call.
FAC2 -171	Contents of AC2 when this routine issues a subroutine call.
FRTN -170	Address of next sequential instruction when the routine issues a subroutine call.

The parameter stored in location 16_8 is actually an address which is the FSP pointer of the current routine. It is this address from which the fixed displacements are calculated in order to determine the effective addresses of the header locations.

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 address are always recoverable from its stack header at locations FCRY through FRTN. FPLP is not currently used by the library routines.

There is a page zero pointer, called QSP, which may reside anywhere from 20_8 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.

The stack mnemonics are really 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.

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.

STACK ALLOCATION IN A SINGLE TASK ENVIRONMENT

In single task environments there is a single SP Stack, a variable length Number Stack, and a single Run Time Stack. The stack structures and maintenance are essentially the same as those in a multitask environment. However, since there is only one run time stack segment, it is not necessary to have the eight word save area or a link to the next segment.

The Number Stack in a single task environment may be variable length. The default length is 630_8 locations, but the size may be redefined by the user at assembly time by means of the following statements:

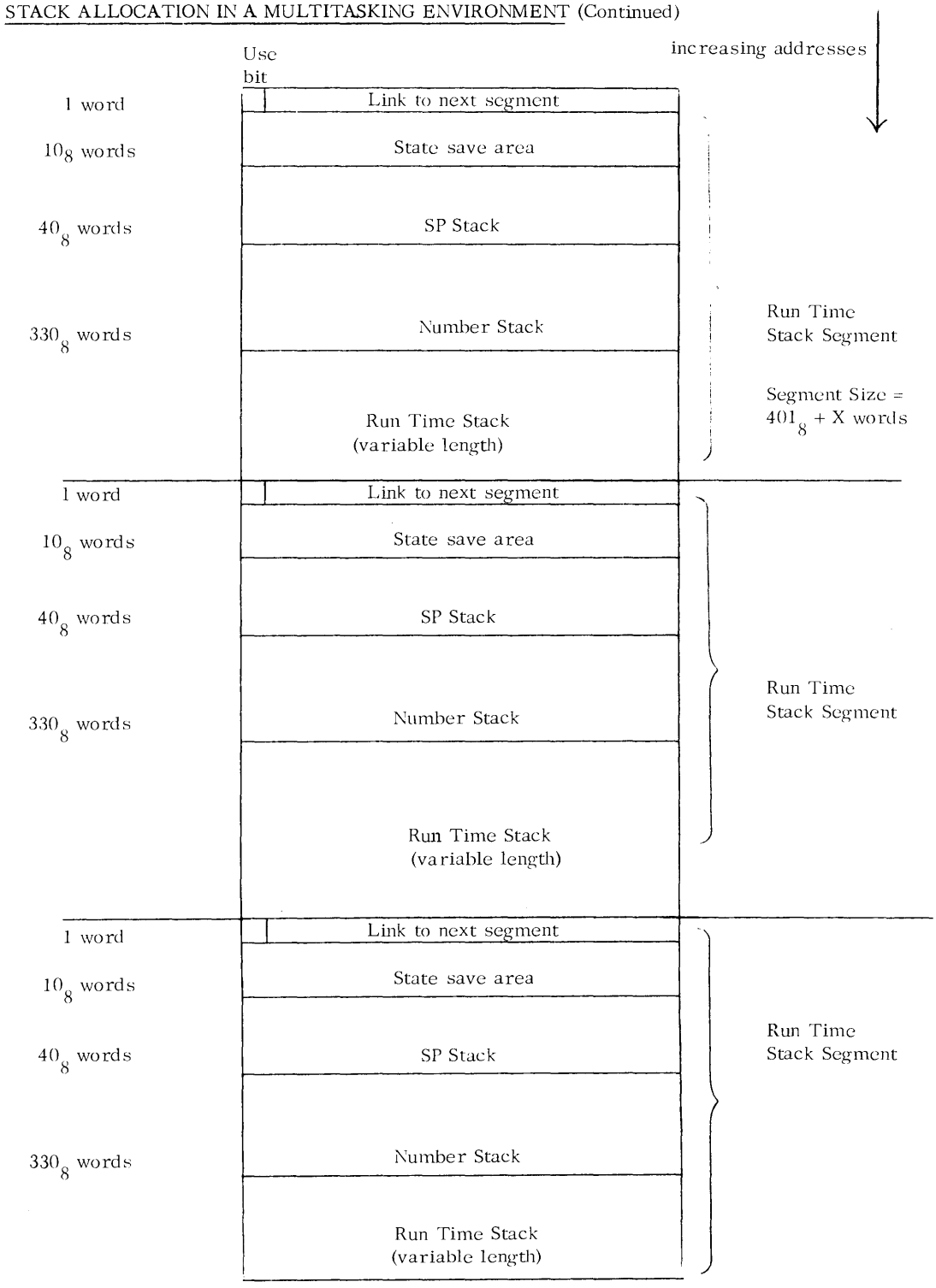
```
.ENT        .FLSZ
.FLSZ = n
```

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.)

STACK ALLOCATION IN A MULTITASKING ENVIRONMENT.

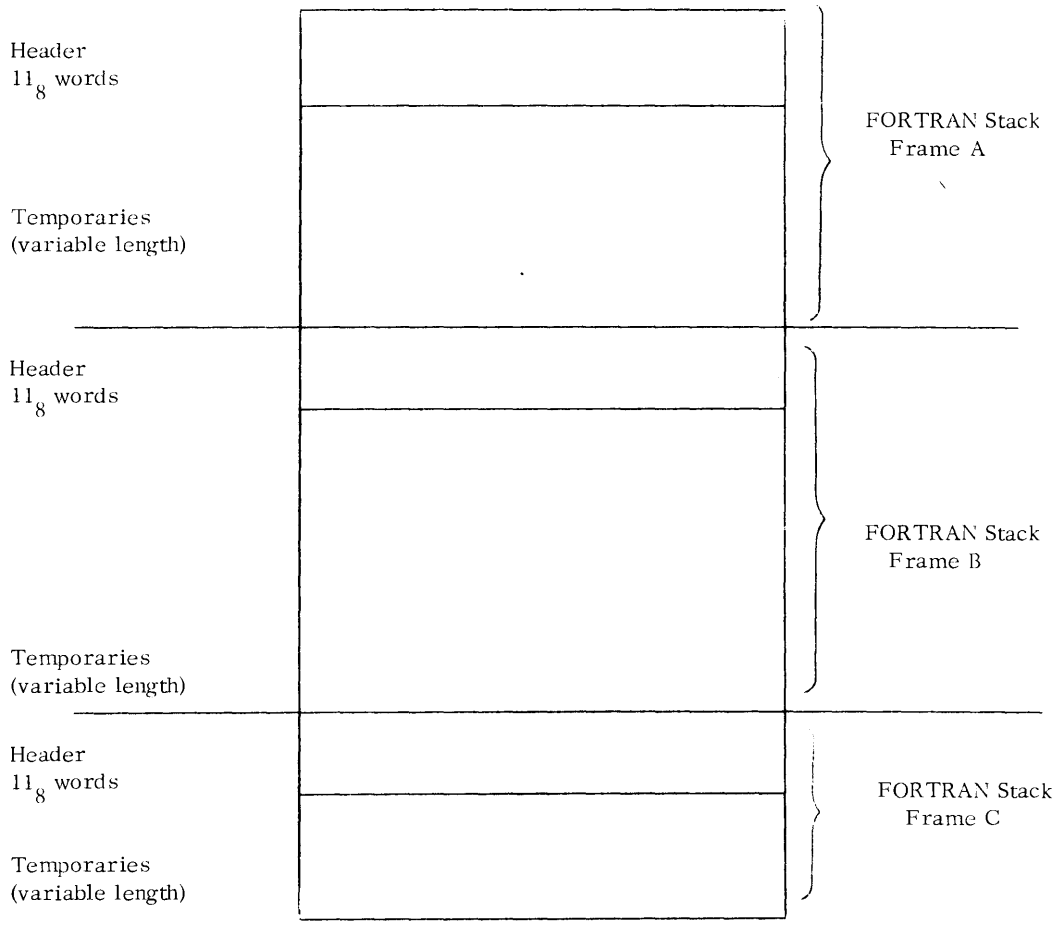
Stack allocation in a multitasking environment is illustrated on the following page.

STACK ALLOCATION IN A MULTITASKING ENVIRONMENT (Continued)



Run Time Stack

(Multitasking Environment)



INTER-SUBROUTINE LINKAGE, FLINK

The FLINK module of the run time library contains entries which enable the calling of other library routines and perform all required stack frame creation/deletion and maintenance functions. FLINK forms the nucleus of the run time subroutines' communications facility.

FLINK Entries

Library routines, including FLINK, have two types of entry points: page zero (.ZREL) and 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. Following is a list of the mnemonic entries of the FLINK module:

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

FRCAL and FQRET also have page zero entries, but these have not been specified with an .ENT statement and are not available for programming use.

The purpose of each FLINK entry is outlined below:

Entry	Purpose
FCALL (or JSR @.FCALL)	Calls a library routine by its .NREL entry point. Also performs FSAV functions.
FRCAL	Calls a library routine with its .NREL entry contained in AC2. Also performs FSAV functions.
FSAV (or JMP @.FSAV)	Maintains the caller's header, allocates a frame for the called subroutine, and updates FSP.
FRET (or JSR @.FRET)	Restores control to the caller, restores the caller's registers and carry, updates FSP.
FQRET	Provides a quick return to a caller when the called subroutine has no stack frame; restores 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 two less than the entry point:

```
                .ZREL
.SBR:           SBR-2

                .NREL
(page zero call)JSR @.SBR
                .
                .
                .
                FSAV
                N                   ; Stack Length Word
.SBR:                               ; True beginning of the subroutine
                .
                .
                .
                FRET (or FQRET)
```

Stack Length Word

The Stack Length Word (SLW) immediately precedes the true beginning of a called subroutine and may be any positive integer, zero or -1. If the SLW is -1, no stack header or temporary storage locations will be allocated for the called subroutine, and 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 location after 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 subroutineFRET must be used to provide a return of program control to the caller. If the SLW is zero, only a stack header is created, providing for the storage and restoration of the values in accumulators AC0 through AC3 and the state of carry should this subroutine make a call to another routine.

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

Subroutine Calls

Whenever one subroutine with a stack allocated for it calls another subroutine with a stack, the contents of AC0 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 subroutine have no stack allocated for it, 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, the FCALL entry of the FLINK module may be used to perform the subroutine call:

```
FCALL
SBR
```

where SBR represents the .NREL entry point to the routine. Subroutines called by FCALL need not be preceded by FSAV, as FCALL performs the functions of FSAV; however, 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 however that this type of call requires 2 words as opposed to 1 word for the alternate call.

Subroutine Calls (Continued)

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 SLW's 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, and then FRCAL is issued. FRCAL accomplishes the same functions as FCALL.

FORTRAN ADDRESSING

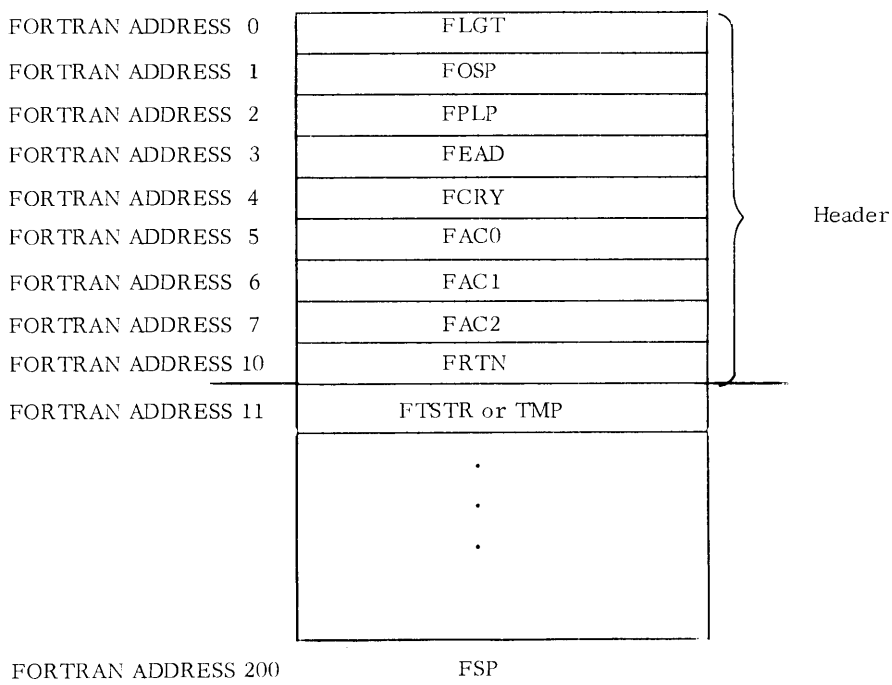
The placing of current FSP values in AC3 and next-to-most-recent values of FSP in AC2 by FLINK, permits an addressing scheme called FORTRAN ADDRESSING, which is used by the library and the FORTRAN Compiler.

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

1. Variables on the stack are referenced relative to that stack's FSP.

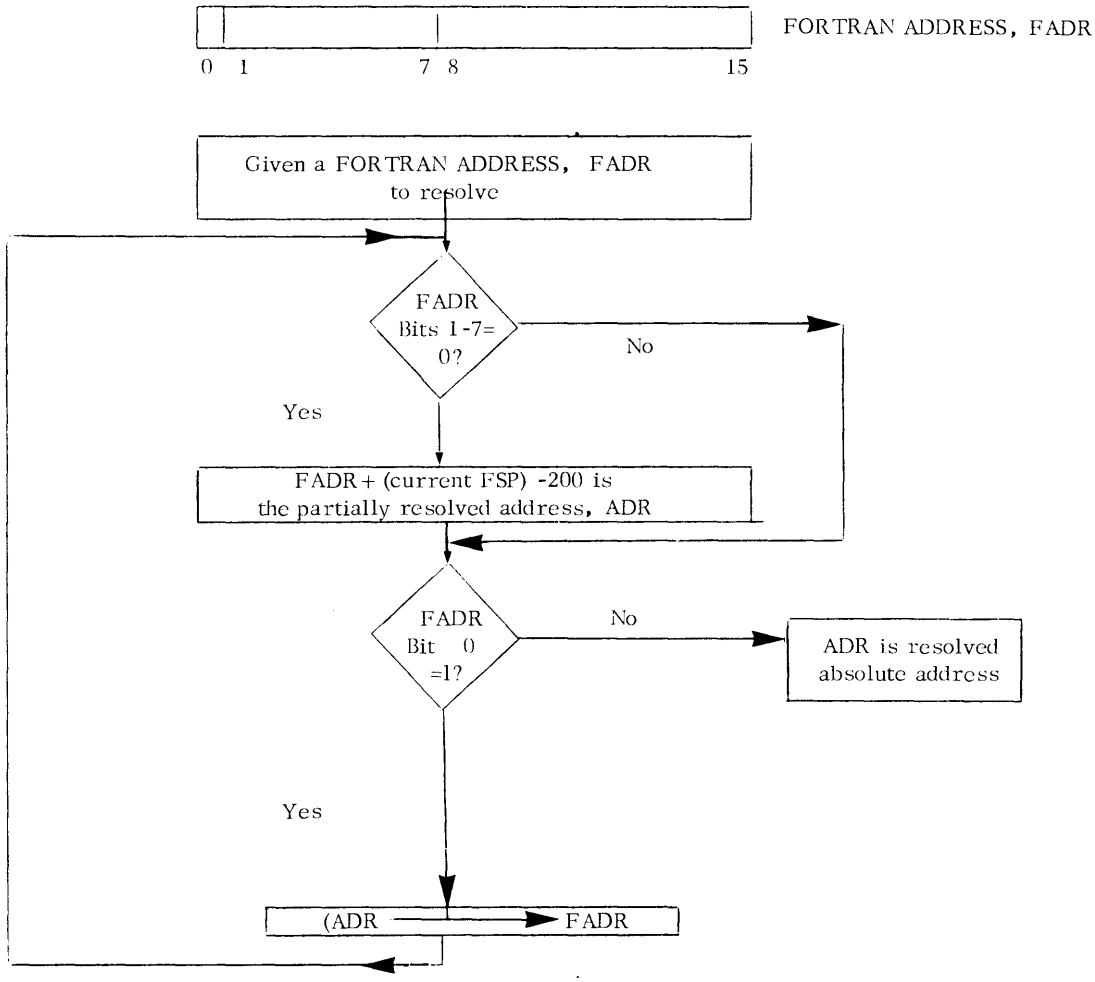
2. Full word addressing for all absolute addresses is effected by subroutines .LD0 and .ST0.

Since NOVA family computers can address 256_{10} words in an indexed instruction, by 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 0 and 377_8 inclusive, or as between 100000_8 and 100377_8 (indirect addressing; 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 0, since FLGT - FZD = -200-(-200) = 0. Using similar reasoning, all direct FORTRAN stack ADDRESSes are positive, with a range of 0 through 377_8 inclusive.



FORTRAN ADDRESSING (Continued)

FORTRAN ADDRESSes greater than 377_8 are treated as absolute .NREL addresses. The following chart illustrates the decisions made by library routines in interpreting FORTRAN ADDRESSes:



Since the most recent FSP is always placed in AC3 by the linkage routines (FLINK), any of 377_8 locations on a frame can be addressed in such instructions as:

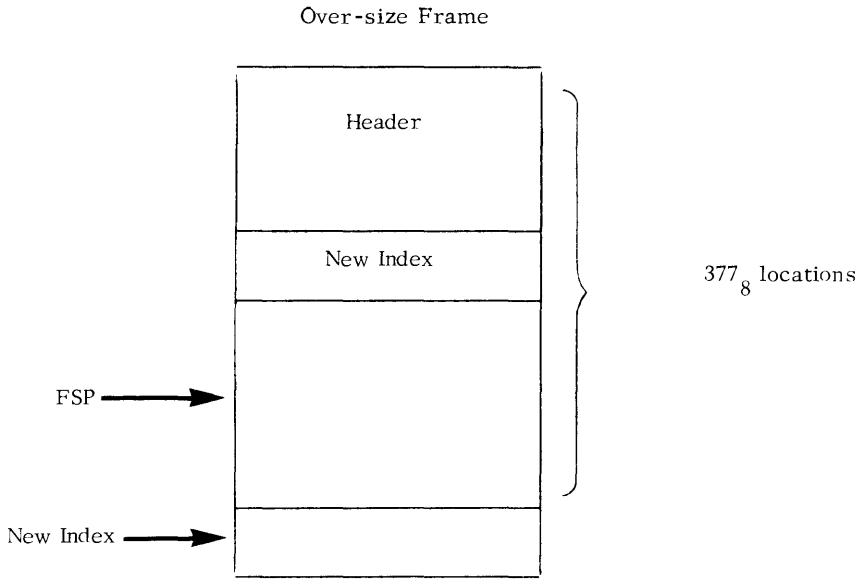
```
LDA 0, -167, 3
```

which is equivalent to

```
LDA 0, TMP, 3
```

The variable nature of the length of a stack frame does not preclude the possibility of a stack frame length exceeding 377_8 locations. If frames exceed 377_8 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:

FORTRAN ADDRESSING (Continued)



FORTRAN array handling presents another means of accessing locations on a stack (see Chapter 3).

The following examples illustrate 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 0,@FRTN,2       ; Parameter ---> AC0
                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:

Without Using QSP

```
                .ZREL
TEMP:           .BLK 1
                .NREL
FSAV:           STA 2, TEMP
                LDA 2, FSP
                STA 0, FAC0, 2
                STA 1, FAC1, 2
                STA 3, FRTN, 2
                LDA 0, TMP
                STA 0, FAC2, 2
```

Using QSP

```
FSAV:           STA 2, @QSP
                LDA 2, FSP
                STA 0, FAC0, 2
                STA 1, FAC1, 2
                STA 3, FRTN, 2
```

FORTRAN ADDRESSING (Continued)

QSP may 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
```

Despite 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          ; The result is returned in
            STA 1,FAC0,2           ; the caller's AC0
            FRET
```

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

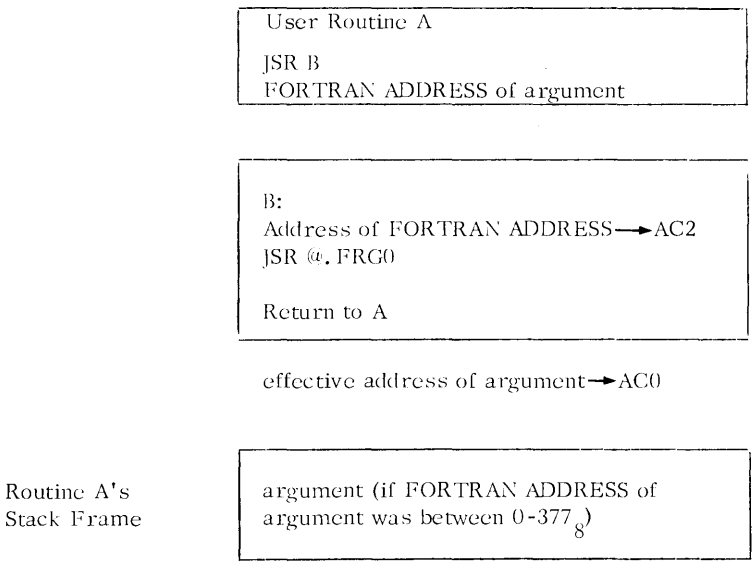
```
            FCALL
            NAME
            Return if condition 1 satisfied
            Return if condition 2 satisfied
            .
            .
            .
            SLW
NAME:       .
            .
            .
            LDA 2,FOSP,3
            SUB 0,1,SZR            ; Condition 2 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 when they reference stack displacements: FRG0/FRG1, MAD/MADO, FRGLD, CPYARG/CPYLS, and FARG. In addition to performing an effective address calculation, when the FORTRAN ADDRESS references a stack displacement, FRGLD loads the contents of this address in AC0; if the FORTRAN ADDRESS exceeds 377_8, it is treated as an absolute address and FRGLD loads the contents of this address in AC0.

CPYARG/CPYLS and FARG transfer effective addresses to the caller's stack. FRG0 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.

FRG0 Operation



Subroutine B must not specify a stack frame.

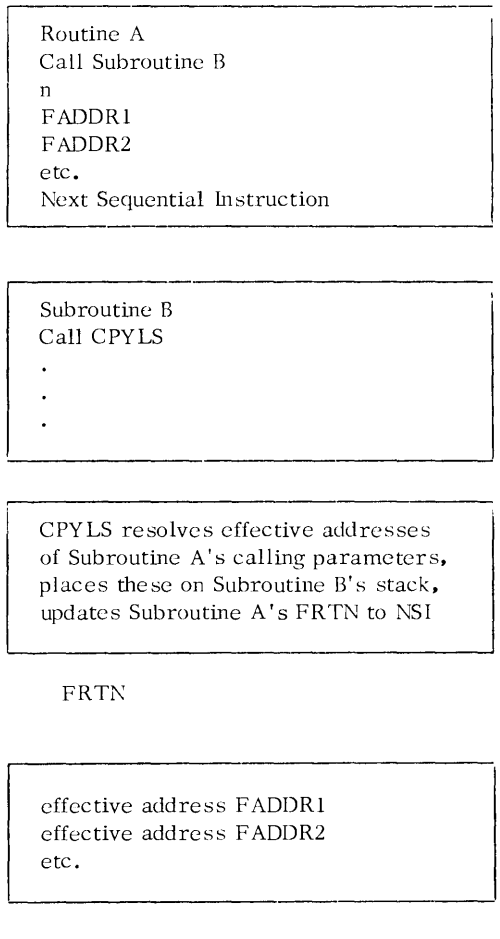
FRGLD computes the effective address of an argument stored at a FORTRAN ADDRESS, and then loads the contents of that address in AC0. If the address is a stack frame displacement, it must be a displacement on the next-most-recently created frame. FRGLD calls FRG1 to resolve the effective address of the 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 2-10. 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 in the library available for resolving FORTRAN ADDRESSes passed by a caller (either referencing stack displacements or absolute addresses of parameters) 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 is in the calling sequence which each accepts.



Note that the parameters to be passed will be on Subroutine A's stack only if the FORTRAN ADDRESSes (FADDRn) in the call to Subroutine B were between 0 and 377_8.

Returning Results to the Caller

The order of the calling sequence generated by the FORTRAN statement

CALL SUB2 (P_1,...,P_N)

is as follows:

```
JSR @.FCAL
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). It can do this by first calling CPYARG (or CPYLS), using the effective addresses it has received, and then returning the result to the specified address. Remember that the FORTRAN ADDRESSes of the caller's parameters can be either displacements on a stack frame or actual pointers to the parameters. One way of returning the result is to load it into an accumulator and then store it:

```
STA @0, 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 was transferred in order onto SUB2's stack. The effective address of the Result 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

Single Task Environment

Certain initialization procedures, performed by single task .I at the beginning of run time, are necessary before a main FORTRAN program may be run.

Single task .I is an entry in the library routine l. .I allocates a stack for itself consisting of 60_8 locations plus header, where the Channel Assignment Table is placed. .I is called by the operating system 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.

Under SOS a system call is issued at the start, .SYSI, which initializes system I/O (this call is a no-op to RDOS). Then 40_8 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.

Single Task Environment (Continued)

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_8 words long, or 30_8 plus twice whatever value a user has specified in a .FLSZ statement. The default value sets aside enough storage for 68 single precision floating point numbers, or 34 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 enough room 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. A simplified version of .I, which illustrates the elements of .I as they might be used in an assembly language program, is given in Appendix B. AFSE, referenced in both versions of .I, is a pointer used to determine whether stack overflow has occurred.

Multitask Environment

Multitask .I is an entry in the multitasking library routine MTI. The Channel Assignment Table, 60_8 locations in length, is written over a section of .I after that part of the initialization code has been executed.

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. 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, error message MEMOVFL is output and a return to the next higher program level (usually CLI under RDOS) is made by means of .SYSTM, .RTN. A temporary SP stack is then created (and will later be overwritten). 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, one for each 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, which is of no further use in a multitasking environment after its initial execution. Control is then given to the FORTRAN Task Scheduler.

CHAPTER THREE

ARRAY STRUCTURE AND HANDLING

An array is an ordered set of data of one or more dimensions; up to 128 dimensions are allowed. Array structure and handling present an exception to the discussion of the library's stack allocation presented earlier.

An array may be defined to be any size in up to 128 dimensions, provided it does not exceed the limits of available memory. Array elements are referenced by integer subscripts, one for each dimension. Element values are assigned so that the first subscript varies most rapidly, then the second subscript, etc.; the element values themselves are numbers in packed format.

An array is allocated by a call to FALOC. The required number of locations is appended to the current end of the caller's stack, thus extending the stack by the size of the designated array. FALOC also adjusts the caller's FLGT to ensure that any further creation of stacks will follow the end of the array.

Array elements are not referenced by the FORTRAN ADDRESSing scheme of relative stack displacements. Instead, routines FSBR, FSUB are used to calculate the absolute address of an array element, with the result an absolute NREL address instead of a relative stack displacement.

The array handling routines require two tables for array allocation and element addressing. An additional table is required when an array is to be redimensioned and passed as a dummy argument.

The first table is the Subscript Bound Specifier (SBS). It describes each subscript's boundaries (both upper and lower index values are included, since array indices may begin at values other than 1) and specifies both the type and size of the array elements (i.e., integers, 1 word; SPFL's, 2 words; DPFL's and SPFL complex, 4 words; and DPFL complex, 10_8 words). The element type coding is 1, integer; 2, SPFL; 3, DPFL; 4, SPFL complex; and 5, DPFL complex.

If N is the number of dimensions of an array of the general form Array A(lower bound$_1$: upper bound$_1$,..., lower bound$_n$: upper bound$_n$), then the SBS table will have the following form:

Word 1	Integer value = 2N-1	
Word 2	element size (words)	element type
Word 3	1st subscript lower boundary (lb_1)	
Word 4	$ub_1 - lb_1 + 1$	
Word 5	2nd subscript lower boundary (lb_2)	
Word 6	$(ub_1 - lb_1 + 1) * (ub_2 - lb_2 + 1)$	
	:	
Word 2N+1	nth subscript lower boundary	
Word 2N+2	$(ub_1 - lb_1 + 1) * \ldots * (ub_n - lb_n + 1)$	total number of elements in array

Subscript Bound Specifier (SBS) Table

ARRAY STRUCTURE AND HANDLING (Continued)

The Three Word Array Specifier (TWS) is a smaller table which is set up within the caller's Run Time Stack. It 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:

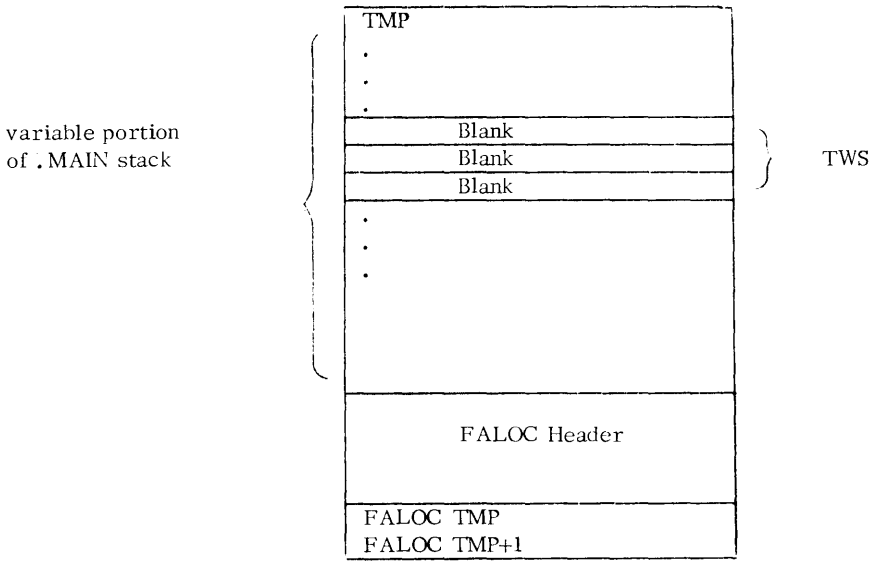
Word 1	FORTRAN ADDRESS of SBS
Word 2	FORTRAN ADDRESS of first array element
	Array size in words

Three Word Array Specifier (TWS)

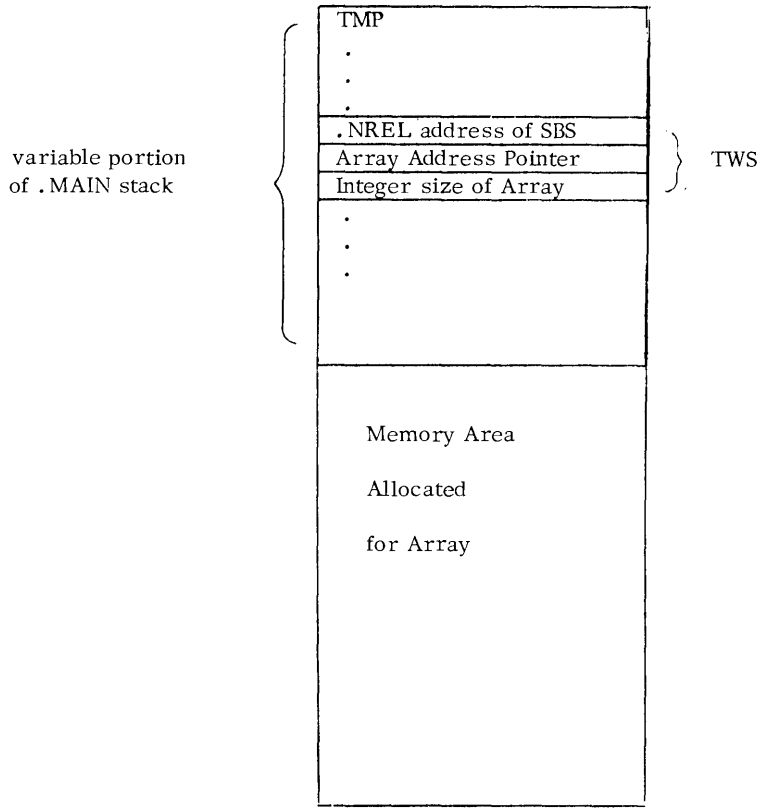
Space for the TWS is reserved on the caller's stack before calling FALOC, which then fills in the three word table with the appropriate information. The SBS is built in NREL memory by the compiler for an array defined in the main program (or in a subroutine if the array is not a passed argument).

The following diagrams illustrate memory maps after a FORTRAN statement DIMENSION A (x,y,z) generates a call to FALOC:

Map after FALOC is called and its stack is allocated, but before FALOC is executed:



Map After FALOC Execution



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. When the array is passed to the subroutine as an argument, the subroutine accesses the array via the new SBS. Array redimensioning and passing are done by FREDI.

As with FALOC, FREDI requires a 3 word area 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) so that re-dimensioning can be accomplished. SSBS is similar in structure to SBS; in place of literal values and cumulative partial products for each index, the addresses of the upper and lower bounds of each index are given. The SSBS is built by the compiler in NREL memory.

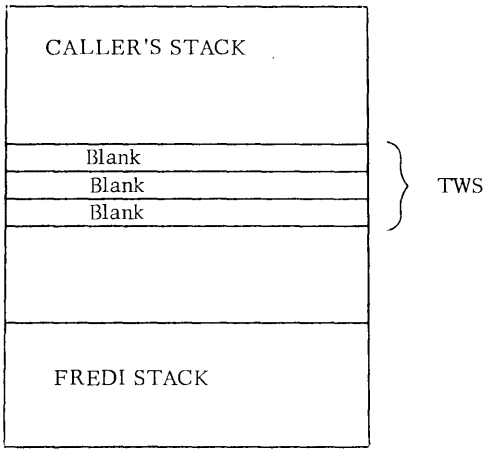
Integer value = 2N+1	
element size (words)	element type
address of first subscript lower bound	
.	
.	
.	
address of nth subscript upper bound	

Special Subscript Bound Specifier (SSBS)

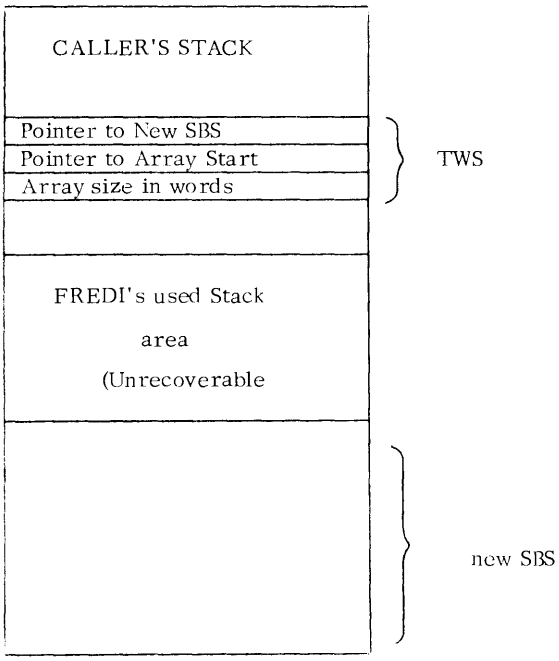
The new SBS, built by FREDI for the caller, is appended to FREDI's own stack. The TWS is built into the area of the caller's stack reserved for that purpose. The stack area used by FREDI in its computations becomes a waste area, unused by the caller upon completion of FREDI's operation. FREDI adjusts 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.

The following diagrams illustrate memory maps after a call to FREDI:

Map after FREDI is called and its stack
is allocated, but before FREDI is executed



After FREDI execution, FREDI updates the caller's FLGT to include the stack area and SBS appended from FREDI.



CHAPTER FOUR

PROGRAM SEGMENTATION

TASKS

A task is a logically complete execution path through a user program that demands use of system resources (I/O devices, system or user overlays, CPU control, etc.). Its execution may be independent of and occur asynchronously with other tasks. A FORTRAN IV program in a multitasking environment may consist of any number of tasks.

Tasks compete simultaneously for the use of system resources in a multitasking environment, although only one task may receive CPU control and the desired resources at a given moment. A task scheduler allocates control to each task, based on task priority and readiness to use system resources.

FORTRAN IV uses the multitask programming facilities available under RDOS, allowing execution of various routines to be performed asynchronously as distinct tasks.

Task Priorities

Task priorities are assigned when tasks are activated and may range from 0 (highest priority) to 255_{10} (lowest priority). Several tasks may exist at the same priority level; the time of a task's creation or priority modification determines its initial rank within a given priority level--the first task assigned to a given level has the highest priority within it.

The relative priority of ready tasks within a common priority level is determined by the relative positions of their respective task control blocks (TCB's) on the active chain maintained by the task scheduler. Each time a task receives control of the processor, its TCB is moved to the end of the active chain. This gives ready tasks within a common priority level approximately equal access to the processor.

There is no limit placed on the number of tasks which may be created within a program. However, each task requested causes a decrease in the size of the run time stacks allocated for each task; thus only the minimum number of tasks necessary for the running of a program should be requested.

Task States

At any given point during the execution of a multitask program, a task may be in any one of four states: dormant, suspended, ready or executing. A dormant task has not yet been activated or has been terminated. A ready task can proceed as soon as it is given control of the processor. A suspended task is not ready to receive control of the processor yet but has not been terminated. An executing task is in control of the processor.

Executing, ready and suspended tasks are said to be active and are linked together in a queue called the active chain; tasks which have been deleted are removed from the active chain and placed in the inactive chain.

When a task is activated (through an FTASK or ITASK call) it becomes ready and competes with other ready tasks, on the basis of its assigned priorities, for control of the processor. When a task receives control of the processor from the scheduler, it goes into the executing state and retains control until it has been completed or some event forces it to relinquish control. A task is suspended when it cannot proceed until the occurrence of some event, such as an appropriate system call or a hardware (I/O) interrupt. The task remains suspended until the required event occurs.

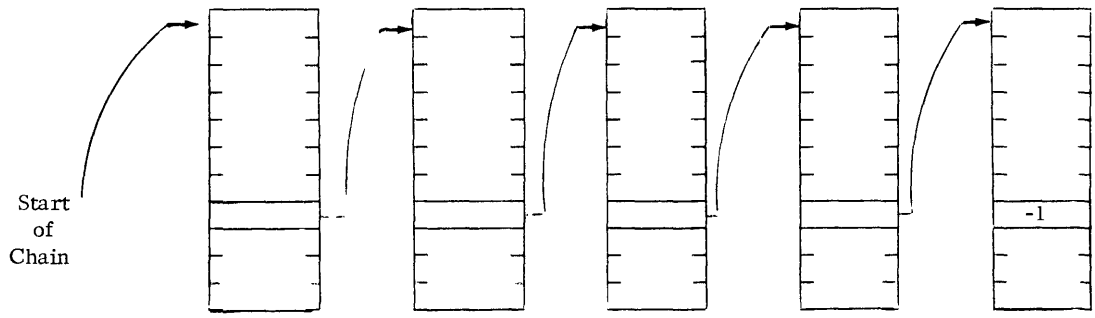
Task Control Blocks

A task control block (TCB) consists of eleven locations used to store the status of an active task. The scheduler maintains a separate TCB for each active task; the TCB's are linked together to form an active chain. TCB's are created at load time, as specified in the CHANTASK statement at the beginning of a multitask program.

Task Control Block

Word 1	Program Counter and Carry
Word 2	Task's AC0
Word 3	Task's AC1
Word 4	Task's AC2
Word 5	Task's AC3
Word 6	Status Bits and Priority
Word 7	System Call Word
Word 10	Link to next TCB in Chain
Word 11	FSP
Word 12	Pointer to Run Time Stack Segment for this Task
Word 13	Temporary location for use by RDOS

TCB Chain



Task Control Blocks (Continued)

During execution, those TCB's that are not being used are linked together to form the free chain.

When a task is activated, a free TCB is removed from the free chain and linked to the active chain; status information for the newly activated task is then placed in this TCB.

Task Scheduler

During the execution of a multitask program, the task scheduler receives control of the processor after each system call. The scheduler then scans the active chain, searching for the TCB of the ready task with the highest priority; this task then receives control of the processor. In the case where several tasks are at the same priority level, the scheduler moves the TCB of the first ready task encountered along the active chain to the end of the chain, and gives control of the processor to the corresponding task. If there are no ready tasks in the active chain, the scheduler will wait until some event causes a task to become ready.

Multitasking Commands and Subroutines

The main FORTRAN program and one or more task subprograms are written as program units of a multitasking program. The main program is at priority level 0 initially and thus receives control at the initiation of execution. If the main FORTRAN unit is killed, it cannot be reactivated unless the program is restarted. The main FORTRAN program cannot be activated by a call to FTASK or ITASK.

The next statement after COMPILER DOUBLE PRECISION, COMPILER NOSTACK, and OVERLAY statements (if any are used) in the main FORTRAN program of a multitask program is a CHANTASK statement:

CHANTASK number of channels, number of tasks

where: number of channels is an integer constant in the range of 1 to 64_{10} representing the number of channels that can be open at any one time.

number of tasks is an integer decimal constant representing the maximum number of tasks that can be simultaneously active at a time during program execution.

If the CHANTASK statement is omitted, the default values are 8 channels and only one active task during program execution.

The specification of number of tasks could also be made in an assembly language program loaded just before the run time library. The program is named FRTSK and has the following source code:

```
.TITL FRTSK
.ENT FRTSK
FRTSK = (n)  where n is the number of tasks
.END
```

FRTSK, if it appears in the RLDR line, will also override a task specification made in the CHANTASK statement.

The CHANTASK statement (or the specification of number of tasks in FRTSK) may be overridden at load time by using the /C and /K local switches of the RLDR command. The /C switch specifies that the octal value immediately preceding the switch is the number of required channels. The /K switch specifies that the octal value immediately preceding is the number of required tasks.

Multitasking Commands and Subroutines (Continued)

The number of channels specified in CHANTASK (or in RLDR) determines the number of FORTRAN logical channels available for concurrent use. A minimum of 16_{10} FORTRAN logical channels will be allocated even though fewer channels may be specified; in this case the number of channels available for concurrent use will be limited to the number specified.

The number of tasks in the CHANTASK specification determines the number of TCB's that will be available at run time. If an attempt is made to activate a task when all available TCB's are in use, an error condition results.

At load time, the total run time stack area available is divided equally into as many parts as there are tasks specified by CHANTASK. Each task has only the space of one of these equal segments available to it at run time. If a task needs more space than is available to it at run time, a fatal error results. Thus the user should not specify the number of tasks to be any greater than he actually needs.

Each task subprogram must begin with a TASK statement and end with an END statement. The TASK statement is of the form

TASK taskname

where taskname is the name assigned to the task program unit. This name must be unique in its first five characters with respect to all function and subroutine names as well as all other task names and overlay names.

A task name must be declared as EXTERNAL in each external program unit that references it. Each task subprogram may be executed an arbitrary number of times during a single execution of the multitasking program.

Activating a Task

All tasks except the main FORTRAN program are activated by executing a call to either the FTASK, FQTASK, ITASK or ASSOC routine. FTASK activates a task by its taskname; ITASK associates an identifier with the task by which it can later be referenced. FQTASK causes periodic execution of a task contained in a user overlay.

Intertask Communication

Active tasks may communicate with each other through shared COMMON (either labeled or blank). Information generated by one executing task can be retained in data or subprogram units until one or more other tasks are executing and can access this information. However, no synchronization of creation and use of information is implicit in this scheme. Unless precautions are taken attempts may be made to use information that has not yet been generated.

Synchronized transmission of one word messages between active tasks can be accomplished by use of the FXMT library subroutine. Entry points for this routine are XMT, XMTW, and REC. Through calls to these entry points a task may transmit a message to another task and continue executing (XMT), transmit a message to another task and wait for its receipt (XMTW), or receive a message from another task (REC). In the case of calls to XMTW or REC, the calling task is suspended immediately following the call until the transmission is complete.

Suspending a Task

There are seven conditions which can cause the suspension of a task:

Suspending a Task (Continued)

1. A CALL SUSP is executed.
2. A CALL HOLD is executed.
3. The task must await some I/O event.
4. A CALL FDELY is executed.
5. A CALL ASUSP is executed to suspend all tasks of the same priority as the executing task.
6. A CALL REC is executed to receive a message not yet sent.
7. A CALL XMTW is executed to transmit a message for which a corresponding CALL REC has not yet been executed.

A task may be doubly suspended, as by a call to ASUSP and I/O completion. In such a case two separate suspend bits are set and both must be reset before the task may become ready.

Readying a Task

A task is put into the ready state when it is activated; while active, it remains in either the ready state, the executing state or the suspended state. There are five conditions which may cause a task to become ready:

1. A task suspended by execution of a call to SUSP, HOLD, or ASUSP may be readied by execution of a call to ARDY or RELSE.
2. A task suspended for performance of I/O is readied automatically upon completion of I/O.
3. A task suspended by execution of a call to FDELY is readied by execution at the the of time period specified.
4. A task suspended by execution of a call to REC is readied by execution of a corresponding call to XMT or to XMTW.
5. A task suspended by execution of a call to XMTW is readied by a corresponding call to REC.

As noted previously, a task that has been doubly suspended must be doubly readied.

Changing Task Priority

When a task is activated it is assigned a priority number. Execution of a call to PRI causes the priority number of the executing task to be changed. A task may change its priority number any number of times while it is active.

Execution of a call to CHNGE causes the priority number of the task having the identification number given in the call to be changed to the specified priority number.

Killing a Task

A task may be killed by a call to KILL (kills the executing task), AKILL (immediately terminates all ready or executing tasks of the priority number specified; suspended tasks with the specified priority number are killed immediately after they become ready), or ABORT (terminates the task with the specified i.d. number).

Obtaining Task Status

The user can obtain the current status of a given task (ready, suspended, inactive) by a call to the STTSK routine. There are nine possible status codes:

Obtaining Task Status (Continued)

0 Ready
1 Suspended by a .SYSTM call.
2 Suspended by an ASUSP call.
3 Wait due to XMTW or REC.
4 Wait for an overlay node.
5 Suspended by ASUSP, SUSP or HOLD and by a .SYSTM call.
6 Suspended by XMT/REC and by ASUSP, SUSP or HOLD.
7 Wait for overlay node and suspended by ASUSP, SUSP, or HOLD.
8 No task exists for this i.d.

OVERLAYS

Separately compilable FORTRAN program units (subroutine, function, block data, and task subprograms) may be divided into root programs and overlay programs. When a given program is executing, its root will be core resident; overlay programs will be disk resident and will be brought into core as they are needed for continued execution.

Several overlay programs may be grouped into overlays, and a number of overlays may time share an area of core (overlay-area). Segments of programs that need not be simultaneously core resident can be made into disk resident overlays, and, when required, can be loaded into a common overlay area in core. Obviously, only one of the overlays assigned to an overlay area can be resident in it at a time.

At run time, all overlays are maintained as core images in a single file on disk; these disk resident core images are never altered during program execution. Each time an overlay is loaded into core, it is in its original form, regardless of whether or not it contains a non-reentrant routine. No part of an overwritten overlay is saved.

After compilation, each program unit contained in an overlay or root program (including a single main program unit) is contained in a separate file in relocatable binary form. The RDOS extended relocatable loader accepts these relocatable binary files and builds the root and overlays in core-image executable form. Two files are produced: a save file containing the root, and an overlay file containing all the overlays. The overlays may be divided into overlay areas, each assigned to a different area of core. Each overlay area starts at a different point, called a node point; there may be up to 128 node points. When the save file is in core for execution, the overlay file remains on disk. Overlays may be loaded during execution as they are needed.

A single overlay area is a multiple of 400_8 locations and must be large enough to accommodate the largest overlay assigned to it. Overlays may exist in either single or multiple task environments. In either case the overlay must be assigned a name and the overlay file must be opened before the overlay is loaded into core; the overlay file must also be closed when it is no longer needed.

In a multitasking environment, overlays and their corresponding overlay area may be shared by several tasks. It is therefore necessary that a determination of whether or not a given overlay area is already in use be made before an overlay is loaded into it.

Different loading routines are called in single and multitasking environments; in a multitasking environment an overlay must be released after use. A task waiting for an overlay area that is in use must be suspended until the overlay area is released.

Overlay Statements and Routines

In both single and multiple task environments each overlay must be assigned a name in an OVERLAY statement. OVERLAY is a non-executable specification statement:

Overlay Statements and Routines (Continued)

OVERLAY overlayname

where overlayname is the name of the overlay. This name must be unique in its first five characters with respect to all function and subroutine names as well as all other overlay names and task names.

An overlay name must be declared as EXTERNAL in any external program unit that references it.

An OVERLAY specification statement must appear as the first statement in one of the program units belonging to the overlay. If more than one subprogram of an overlay contains an OVERLAY statement, each overlayname specified in these statements may be used interchangeably to reference the overlay.

Opening and Closing an Overlay File

In both single and multiple task environments, the overlay file associated with a program using overlays must be opened by execution of a call to the OVOPN routine before any overlays can be loaded.

In both single and multiple task environments, each overlay file is closed by a call to CLOSE. An overlay used by multiple programs in a multitasking environment must be closed by any program that opens it before another program can use the overlay.

Loading an Overlay--Single Task Environment

An overlay is loaded in a single task environment by execution of a call to the OVLOD routine. The resulting load may be either conditional or unconditional. In the case of an unconditional load, the user overlay is loaded regardless of whether it is already in core. This allows for initialization of non-reentrant code.

A request for a conditional overlay load causes the user overlay to be loaded only if it is not already present in core. This may save time in some cases, but conditional loading should be specified only when overlays are reentrant.

Each overlay has a corresponding overlay use count (OUC) that indicates whether or not the overlay is core resident. Specification of a conditional load results in a check of the OUC; an OUC of one indicates that the overlay is core-resident, zero indicates that the overlay is not resident in core. Following are the conditions determining the loading of an overlay:

1. If the load request is conditional (flag≠0) and the area is free (OUC=0), the OUC is incremented, the overlay is loaded, and the error return is set to 1 to indicate the overlay has been loaded.

2. If the load request is conditional and the overlay area already contains the requested overlay (OUC=1), the overlay remains in the area, the OUC is decremented, and the error return is set to 1 to indicate that the overlay has been loaded.

3. If the load request is unconditional, the OUC is set to 1, the overlay is loaded, and the error return is set to 1.

4. If for any reason the overlay cannot be loaded, an appropriate error return is set.

Loading an Overlay--Multitask Environment

An overlay is loaded in a multiple task environment by execution of a call to the FOVLD routine.

The loading of an overlay depends upon the state of the conditional flag and the OUC. However, since overlays and overlay areas may be shared by two or more tasks, the conditions for loading are more complex than in a single task environment.

When a task causes an overlay to be loaded, the task is suspended until the loading process is completed. When a task tries to load an overlay into an overlay area and cannot because the overlay area is already in use, the task is suspended until the overlay area is freed and the desired overlay is successfully loaded. If more than one task is suspended waiting for an overlay area to be freed, the task with the highest priority waiting for the overlay area has its desired overlay loaded when the overlay area is freed, and this task is readied when loading is completed.

The overlay use count is incremented each time a task causes an overlay to be loaded and is decremented each time a task causes the overlay to be released. Since more than one task can use an overlay, the OUC may be greater than 1. An overlay area is free only when the OUC goes to 0. Following are the conditions for loading an overlay in a multitask environment:

1. If the load request is conditional (flag≠0) and if the area is free (OUC=0), then the OUC is incremented, the overlay is loaded, and the error return is set to 1 to indicate the overlay has been loaded.

2. If the load request is conditional and if the area is not free but already contains the requested overlay, the overlay remains in the area, the OUC is decremented, and the error return is set to 1 to indicate that the overlay has been loaded.

3. If the load request is conditional and the area is not free and does not contain the requested overlay, the caller is suspended until the area is freed.

4. If the load request is unconditional (flag=0) and if the area is free (OUC), the OUC is incremented and the overlay is loaded regardless of whether it is core resident or not.

5. If the load request is unconditional and if the OUC has not gone to zero freeing the area, the calling task is suspended until the area becomes free.

6. If for any reason the overlay cannot be successfully loaded, the error specification in the call is set to the appropriate error code.

When a task causes an overlay to be loaded, the task is suspended during the loading process as it would be for any other I/O operation. For those cases in which no loading occurs, and the task does not have to wait for an overlay area to become available, the task is not suspended.

Releasing an Overlay Area

All overlay loads (FOVLD) in the multitask environment must eventually be paired with overlay releases or overlay areas will be reserved indefinitely. An overlay area can be released from outside the overlay by execution of a call to FOVRL; this causes the OUC for the specified overlay area to be decremented. (The overlay area is freed only when the OUC goes to zero.) If an overlay other than the one resident in the specified overlay area is named in the subroutine call, an error condition results and the overlay area is not released.

Releasing an Overlay Area (Continued)

Overlays may be released from inside the overlay area, either from the routine in which the overlay was named or from some other routine within the overlay.

A call to OVKIL can be made from the routine in which the overlay was named in an OVERLAY statement and causes the overlay to be released and the task containing the overlay to be killed.

A call to OVKIX is made from a routine outside that in which the overlay was named. The OVKIX routine causes the overlay to be released and the task containing the overlay to be killed.

A call to OVEXT can be made from the routine in which the overlay is named. It causes the overlay to be released and provides a return location.

A call to OVEXX is made from outside the routine in which the overlay is named. OVEXX causes the overlay to be released and provides a return location.

Periodic Execution of Overlay or Core Resident Tasks

Real time overlay tasks are periodic executions of a task contained in a user overlay. Execution of a call to FQTASK causes the overlay containing the specified task to be loaded so that the task can be executed. Provision is made in the call in case the task is already core resident.

Before the call to FQTASK is executed, the file containing the specified overlay must have been opened (OVOPN). When the call is executed, the specified overlay is loaded at the specified time and the required task is first executed. The task is executed periodically after each specified increment until the task has been executed the required number of times. If the task is core resident, a dummy name is specified instead of the overlay name.

If the necessary overlay area for the specified overlay is not available, or if there is no TCB available for the required task, task execution is postponed until the necessary resource is available. An error code of 1 represents successful completion of the FQTASK call.

PROGRAM SWAPPING AND CHAINING

During run time, programs may be chained or swapped. In chaining, the currently executing program issues a call to FCHAN that causes the program to be overwritten in core by another program loaded from disk. The core image of the calling program is not saved. In swapping, the currently executing program issues a call to FSWAP that causes its core image to be saved on disk and a new program to be loaded from disk for execution. The saved program can later be restored by a call to FBACK.

When a program calls another in a swap, the calling program is said to execute at a higher level than the called program. The RDOS CLI executes at level 0 and user programs execute at levels 1-4. When a program issues a call to FSWAP, the execution level is incremented, and when a program issues a call to FBACK the level is decremented. If an attempt is made to nest swaps to a level deeper than 5, an RDOS error results.

When a program issues a call to FCHAN, the called program replaces the calling program at the same execution level. Thus, any number of chaining calls may be made.

Program chaining can be used to subdivide a program that would exceed the limits of core into sequentially executable units. Program swapping allows core images of programs to be saved and called for execution more than once during program execution.

PROGRAM SWAPPING AND CHAINING (Continued)

Each save file that is swapped or chained must contain a single complete FORTRAN IV program consisting of a main program unit and all subprograms whether directly or indirectly linked to it. Each swapped or chained program can be independent of all others except for linkage through swapping or chaining statements and shared data. Each swapped or chained program can have its own independent task or overlay structure.

Swapped or chained programs can communicate through commonly accessed files and unlabeled COMMON areas in core. The user can insure that data necessary for swapped or chained programs in unlabeled COMMON is not overwritten by an incoming core image by declaring the same size area. Additionally, all participating FORTRAN programs must be either single tasking or multi-tasking, not a combination of the two.

CHAPTER FIVE

USING RDOS FEATURES

FILE MAINTENANCE AND I/O

FORTRAN IV Run Time routines provide an interface to RDOS file maintenance features. The following functions may be performed by use of the indicated calls:

1. Create an RDOS disk file. (CFILW)
2. Delete an RDOS disk file. (DFILW)
3. Open an RDOS file. (OPEN)
4. Open an RDOS file for appending. (APPEND)
5. Rename an RDOS disk file. (RENAME)
6. Close an RDOS file. (CLOSE)
7. Close all open files. (RESET)
8. Examine the attributes of an RDOS file (GTATR)
9. Set the attributes of an RDOS file. (FSTAT)
10. Read a series of blocks. (RDBLK)
11. Read a series of records. (READR)
12. Write a series of blocks. (WRBLK)
13. Write a series of records. (WRITR)

Calling sequences, parameter definitions, error codes and other pertinent information are given in the routine descriptions.

DIRECTORY/DEVICE MAINTENANCE

FORTRAN IV run time routines also interface with RDOS directory/device maintenance features. The following functions may be performed with the indicated calls:

1. Change the current directory. (DIR)
2. Initialize a directory. (INIT)
3. Release a directory. (RLSE)

Consult the routine descriptions.

INTERRUPTS

User Interrupts

Users wishing to incorporate non-SYSGENed devices into real time FORTRAN programs must provide for the interrupt servicing to be done in assembly language and for the creation of a three word device control table (DCT) as detailed in the RDOS USER'S MANUAL (093-000075).

Interrupts from special (non-SYSGENed) devices do not generally change the status of tasks in a FORTRAN multitask environment. Instead, such interrupts freeze the environment until servicing of the interrupt is completed and the multitask environment is unfrozen. 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. If the task for which the message is intended has issued a REC call for the message, the task state is changed from suspended to ready even though the task activity is in suspension.

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 Task Call identical to .IXMT described in the RDOS USER'S MANUAL. This is the only situation in which user interrupt servicing may alter the task environment.

Identifying a User Interrupt Device

It is still necessary, however, to identify the interrupt device to the system by means of a FORTRAN call to FINTD, and it is possible to remove this device by means of another FORTRAN call to FINRV.

FINTD is used to identify to the system a device capable of generating interrupt requests but which was not SYSGENed. FINTD causes an entry for the specified device code to be placed in the system interrupt vector table.

There is a special usage of a call to FINTD to provide for automatic restart of user-defined devices after a power failure. Users having hardware option 4006 (power monitor/automatic restart hardware) may make use of the call to provide power-up service for user devices in a user-written routine, as indicated in the routine description.

Removing a Service Interrupt Device

A previously added (FINTD) user interrupt device can be removed from the system interrupt vector table by a call to FINRV. If an attempt is made to remove a SYSGENed device or if the device code argument is not within legal range of user interrupt devices (≤ 76), a fatal run time error occurs and execution is terminated.

Preserving Reentrance During Interrupt Processing

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 is 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 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 .SV0, .OVFL, FSP, SP and NSP. Additionally, a new QSP 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 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:

Preserving Reentrance During Interrupt Processing (Continued)

$$C(QSP') = F(FSP) + FLGT + 2FFEL$$

where C(FSP) indicates the contents of FSP and FFEL is as defined in the description of the FPZERO module. A new value for QSP is calculated by adding PARF dispacement FAC2 to its associated FSP:

$$C(QSP') = FAC2 + C(FSP')$$

The old value for QSP must be restored when its associated FSP value is restored.

The following 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
```

FOREGROUND/BACKGROUND PROGRAMMING

Throughout the preceding discussion of a multitasking environment the environment was understood to be confined to a single program, although tasks within that program could execute asynchronously.

A dual programming environment allows two separate and distinct programs, either or both of which may be multitasking, to share system resources simultaneously in much the same manner as tasks in a multitasking environment.

In dual programming, one program is designated as the foreground program; the other is designated as operating in the background. Each program is independent of the other, and each maintains its own task scheduler. The two programs may have the same priority, or the foreground program may operate at a higher priority than the background program. If the foreground program has a higher priority, the background program will receive control only when there are no ready tasks in the foreground.

Foreground and background programs may communicate with each other despite their independence. This is accomplished through definition of a communication area within each program to be used for transmitting and receiving messages.

FOREGROUND/BACKGROUND PROGRAMMING (Continued)

A user-written module, FHMA, specifies the highest memory address for the background to ensure room for the foreground. FHMA must be loaded immediately before the Run Time Library and has the following structure:

```
.TITLE  FHMA
.ENT  FHMA
FHMA = (n)
.END
```

Mapped and Unmapped Environments

A mapped environment includes the Memory Management and Protection Unit (MMPU); the foreground and background programs are separated by absolute hardware protection. Foreground save and overlay files are built in the same way as for a single program environment; an entire page zero and NREL memory is reserved for the foreground.

In an unmapped environment (I.E., lacking the MMPU), the foreground and background areas are separated by software partitions; the boundaries are set by the user at relocatable load time.

OPERATION PROCEDURES UNDER RDOS

The user receives the FORTRAN IV compiler as two dumped tapes. Before the compiler can be used, save files must be created from the tapes with the LOAD command. After the compiler has been loaded, the FORTRAN library tapes must be transferred to disk using the XFER command. The library tapes to be transferred are

```
FMT.LB        (FORTRAN multitask library)
FORT1.LB
FORT2.LB
FORT3.LB
SOFTMPYD.LB  }
NMPYD.LB     }  one will be supplied.
HMPYD.LB     }
```

Once the compiler and library are properly loaded onto disk, the FORTRAN IV compiler can be invoked with a FORT command followed by appropriate arguments, as described in the RDOS USER'S MANUAL.

Each FORTRAN main program, external subroutine, or external function is separately compiled. When the main program and its external subroutines and functions have been successfully compiled, the programs are loaded using the RLDR command. The FORTRAN libraries must be loaded with the programs.

The following example illustrates a series of commands for compiling, loading and running a FORTRAN program:

```
FORT MAIN )
FORT XSUB1 )
FORT XFUN )
FORT XSUB2 )

RLDR/D MAIN XSUB1 XFUN XSUB2 FORT1.LB FORT2.LB FORT3.LB FORT4.LB )
```

In the above example the /D switch will also cause loading of the symbolic debugger DEBUG III.

Loading - Single Task Environment

The RLDR command is used to load the relocatable binary output of compilation. The main program is loaded first, followed by any external subroutines and then the FORTRAN library files, FORT1.LB, FORT2.LB, FORT3.LB and FORT4.LB:

RLDR mainfile subfiles libraryfiles

To load main program MAIN and subprograms SBR1, SBR2, and SBR3, the user would issue the command:

RLDR MAIN SBR1 SBR2 FORT1.LB FORT2.LB FORT3.LB FORT4.LB

Loading- Multitask Environment

In a multitask environment, the multitask library FMT.LB must be loaded before the other libraries. To load a multitasking FORTRAN program consisting of MAIN, TASK1 and TASK2, issue the command:

RLDR MAIN TASK1 TASK2 FMT.LB FORT1.LB FORT2.LB FORT3.LB FORT4.LB

When overlays are used, the overlays that belong to a given overlay area are enclosed in square brackets in the command line:

RLDR MAIN [OV1,OV2 OV3] TASK1 TASK2 [OV4 OV5,OV6,OV7] [OV8,OV9]
FMT.LB FORT1.LB FORT2.LB FORT3.LB FORT4.LB

In this example,

1. MAIN, the main program, and task subprograms TASK1 and TASK2 are loaded into save file MAIN.SV. Space is allotted in the save file for three overlay areas.
2. Overlay-area$_0$ is shared by two overlays, the first containing OV1 and the other OV2 and OV3.
3. Overlay-area$_1$ is shared by three overlays, the first containing OV4 and OV5, the second OV6, and the third OV7.
4. Overlay-area$_2$ contains two overlays, OV8 and OV9.
5. The overlays, divided into overlay areas, are loaded into MAIN.OL.

*

LIMITATIONS OF RTOS

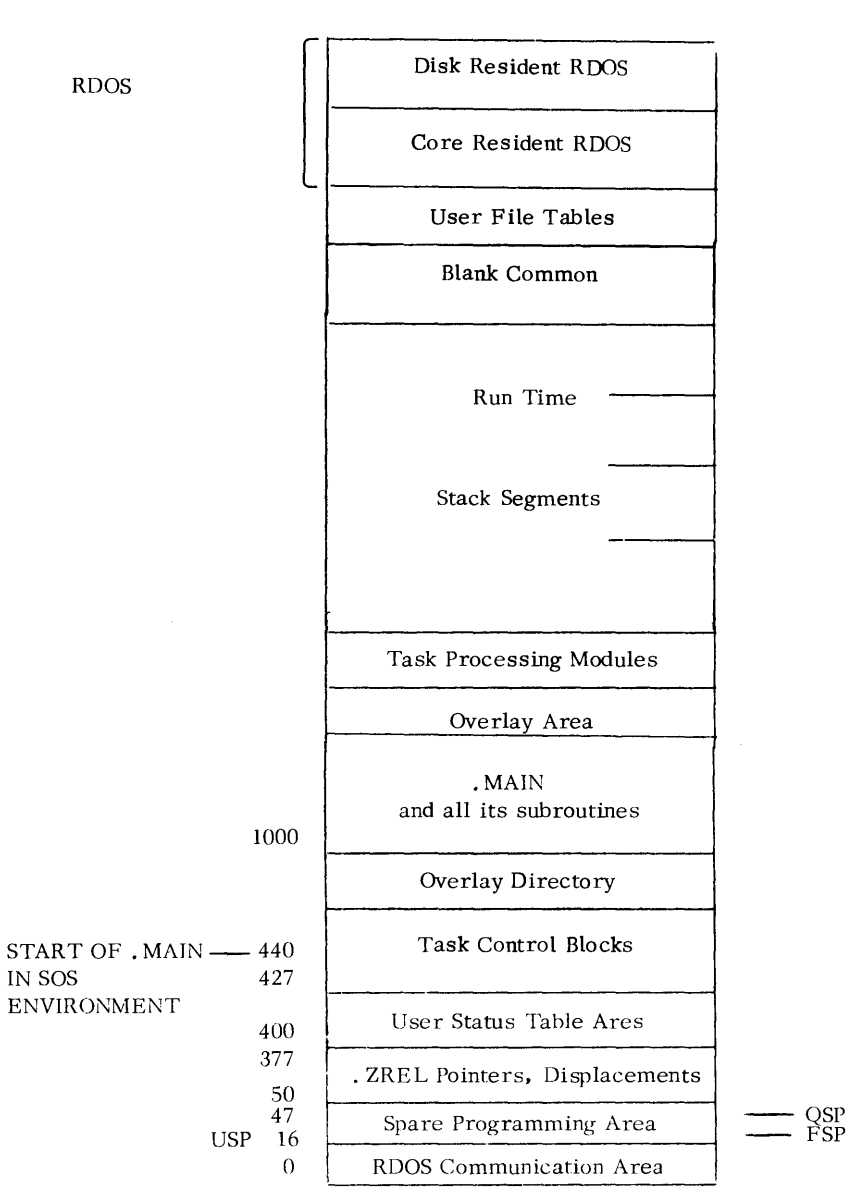
Since RTOS is a compatible subset of RDOS, RTOS will support a subset of DGC Real Time FORTRAN IV. The only restriction on use of RT FORTRAN IV under RTOS is that only those real time calls which have corresponding system and task calls implemented in RTOS may be used. Thus CHANTASK and OVERLAY statements may not be used, and the following calls are not allowed:

BACK	CPART	FBACK	FSTAT	IOPROG	OVLOD	SWAP
BOOT	DFILW	FCHAN	FSWAP	MDIR	OVOPN	UPDATE
CDIR	DIR	FCLOS	GCIN	ODIS	RDCMN	WRCMN
CFILW	DLINK	FGND	GCOUT	OEBL	RDOPR	WRITR
CHAIN	DULNK	FOVLD	GDIR	OVEXT	RDRW	WROPR
CHLAT	EBACK	FOVRL	GSYS	OVEXX	READR	WRTR
CHSAV	EXBG	FQTASK	ICMN	OVKIL	RENAM	
CHSTS	EXFG	FSEEK	IOPC	OVKIX	STAT	

LIMITATIONS OF RTOS (Continued)

In general, no disk I/O-related calls are available, with the exception of WRBLK and RDBLK for disk I/O.

Memory Map of .MAIN at Run Time



CHAPTER SIX

INPUT/OUTPUT

The FORTRAN statements READ and WRITE generate calls to entries in the FREAD routine of the Run Time library. These entries perform formatted or free form FORTRAN I/O of ASCII data or FORTRAN I/O of binary data.

When formatted data is to be read or written, the general form of the call is

```
JSR @<SUBR>
FORTRAN address of the logical channel number
FORTRAN address of the format statement text string (formatted I/O only)
ELEMENT DESCRIPTOR SEQUENCE (S)
5
```

where SUBR is .FREAD for ASCII read, .FWRIT for ASCII write, .BRD for binary read, and .BWR for binary write. The integer 5 indicates the end of the calling sequence. This is necessary because the ELEMENT DESCRIPTOR SEQUENCES may be variable length.

The ELEMENT DESCRIPTOR SEQUENCES describe fully the nature of each data type in the I/O list. One sequence is selected to describe each data element type in the I/O list. For example, the FORTRAN statement

```
WRITE (10) 'REAL RESULT IS' ,X
```

contains two data types: string and variable. Thus there will be two ELEMENT DESCRIPTOR SEQUENCES in the call to .FWRIT which is generated.

The first word of each ELEMENT DESCRIPTOR SEQUENCE is an integer tag which labels the type of sequence which is to follow. There are eight possible tag descriptions:

Tag	Data Element Type
0	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

VARIABLE DATA ELEMENT

A variable data ELEMENT DESCRIPTOR SEQUENCE consists of three words. The first is the tag 0, as indicated above. The second is a code for the variable type:

Code	Variable Type
1	Integer, Logical, Alphabetic/Hollerith
2	SPFL
3	DPFL
4	SPCX
5	DPCX

VARIABLE DATA ELEMENT (Continued)

The third word is the FORTRAN ADDRESS of the variable. Thus the FORTRAN statement

```
READ (11, 1)TEST
```

generates the following complete call to FREAD

```
JSR @. FREA
.C1          ;FORTRAN ADDRESS of logical channel number
L2.          ;FORTRAN ADDRESS of format statement text string
0            ;Data element type - variable
2            ;SPFL
V.+0         ;FORTRAN ADDRESS of variable
5            ;Terminator
```

ARRAY ELEMENT

The second ELEMENT DESCRIPTOR SEQUENCE describes an Array Element in the I/O list and consists of a variable number of words. The first is the tag 1. The second is an integer indicating the number of parameters to follow, excluding the calling sequence delimiter (5). The third word is the FORTRAN ADDRESS of the Three Word Specifier, as described on page 3-2. The fourth word is 0. The fifth and subsequent words are the FORTRAN ADDRESSes of the subscripts in the I/O list. Thus the FORTRAN statements

```
DIMENSION NAME (2)
.
.
.
READ (11, 100)NAME(1)
```

generates the following call to FREAD:

```
JSR @. FREA
.C3          ;FORTRAN ADDRESS of logical channel number
L2.          ;FORTRAN ADDRESS of format statement text string
1            ;Data element type - array element
3            ;Number of parameters following
V.+0         ;FORTRAN ADDRESS of Three Word Specifier
0            ; Predefined
.C2          ;FORTRAN ADDRESS of subscript
5            ;Terminator
```

COMPLETE ARRAY

A complete array ELEMENT DESCRIPTOR SEQUENCE consists of two words. The first is the tag 2. The second word is the FORTRAN ADDRESS of the Three Word Specifier. The FORTRAN statements

```
DIMENSION A(10)
.
.
.
READ BINARY (13)A
```

--- ROUTINE: AFRTN

Supporting Subroutines: FRET; .FRG0

Subroutine Size: 1 page zero location and 5 locations of normally relocatable memory.

Entry: .AFRTN

Function: Provides 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
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .AFRTN must be referenced by an .EXTD statement.

(3) No error messages are generated.

--- ROUTINE: ARGUM

Supporting Subroutines: ; SP

Subroutine Size: 1 page zero location and 34_8 locations of normally relocatable memory.

Entry: .FARG

Function: Fetches a called subroutine's argument addresses when these are stored as FORTRAN ADDRESSes immediately following the caller.

Calling Sequence:

(AC0 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FARG must be referenced by an .EXTD statement.

(3) The following example illustrates the use of .FARG:

ROUTINE: ARGUM (Continued)

Notes: (Continued)

```
                .ZREL
AL.G0           .ALG10-2
.NREL           .NREL
.MAIN           .
                .
                .
CAL1:           JSR @AL.G0 ; THIS IS THE CALLING ROUTINE
                FORTRAN ADDRESS of argument
                .
                .
                .
                FSAV
                3
.ALG10          SUBZL 0, 0 ; PUT 1 IN AC0, SINCE THERE IS ONLY
                           ; ONE ARGUMENT FOLLOWING MAIN CALLER
.CAL2:          JSR @.FARG ; ARGUMENT ADDRESS IS STORED ON
                           ;ALG10'S STACK.
```

Entry: .FARL

Function: Transfers effective addresses of a caller's argument list to its called subroutine's stack.

Calling Sequence:

(FRTN points to argument list: $\left\{\begin{matrix}N\\ARG1, \text{ etc.}\end{matrix}\right\}$)

JSR @.FARL

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FARL must be referenced with an .EXTD statement.

(3) Both .FARL and .FARG set FPLP on the current stack to be the number of arguments copied.

--- ROUTINE: CPYARG

Supporting Subroutines: FRET, FSAV; .MADO

Subroutine Size: 2 page zero locations and 42_8 locations of normally relocatable memory.

Entry: .CPYA

Function: Transfers effective addresses of a caller's argument list to its called subroutine's stack.

Calling Sequence:

```
        FCALL
        SUBR
        N           ; NUMBER OF ARGUMENTS IN LIST
        FADDR
        FADDR
        .
        .
        .
SUBR:   .
        .
        .
```

(AC0 contains the number of arguments to be passed.)

```
        JSR @.CPYA  ; ADDRESSES OF CALLER'S ARGUMENTS
                    ; ARE NOW ON SUBR STACK
```

ROUTINE: CPYARG (Continued)

Entry: .CPYL

Function: Transfers effective addresses of a caller's argument list to its called subroutine's stack.

Calling Sequence:

```
        FCALL
        SUBR
        N               ; NUMBER OF ARGUMENTS IN LIST
        FADDR
        FADDR
        .
        .
SUBR:   .
        .
        .
        .
```

(AC0 contains the number of arguments to be passed.)

```
        JSR @.CPYL      ; ADDRESS OF CALLER'S ARGUMENTS ARE
                        ; NOW ON SUBR STACK
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .CPYL and .CPYA must be referenced by .EXTD statements. The FCALL entry point CPYAR must be referenced by an .EXTN statement.

(3) This routine is more generalized than FARG.

(4) .CPYL updates the caller's return address (stored in FRTN) to the next sequential instruction following the call.

--- ROUTINE: FARG0

Supporting Subroutines: ; SP

Subroutine Size: 2 page zero locations and 24_8 locations of normally relocatable memory.

Entry: .FRG0

Function: Calculates the effective address of an argument on the current stack frame, given its FORTRAN ADDRESS pointed to by AC2.

Calling Sequence:

(FORTRAN ADDRESS is pointed to by AC2.)

JSR @.FRG0

(The address is returned in AC0.)

ROUTINE: FARG0 (Continued)

Entry: .FRG1

Function: Calculates the effective address of an argument on thr next-most-current stack frame, given its FORTRAN ADDRESS pointed to by AC2.

Calling Sequence:

(FORTRAN ADDRESS is pointed to by AC2.)

JSR @.FRG1

(The address is returned in AC0.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FRG0 and .FRG1 must be referenced by .EXTD statements.

(3) 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.

--- ROUTINE: FLINK

Supporting Subroutines: .I; AFSE, .RTE0, QSP

Subroutine Size: 5 page zero locations and 140_8 locations of normally relocatable memory.

Entry: FCALL

Function: Used to call a subroutine which has no page zero entry, or to call a subroutine which has a page zero entry without using its page zero entry.

Calling Sequence:

```
        FCALL
        SUBRT       ; NAME OF SUBROUTINE WHICH HAS NO
                    ; PAGE ZERO ENTRY
   or,

        .ZREL
.SUBR:  SUBR-2      ; BOTH SUBR and .SUBR HAVE BEEN ENTERED
        .NREL
        FCALL
        SUBR        ; SUBR HAS A ZREL ENTRY
NSI:    ....        ; NEXT SEQUENTIAL INSTRUCTION
                    ; FOLLOWING CALL TO SUBR
        FSAV
SLW:    Integer 1
SUBR:   ...         ; FIRST TRUE INSTRUCTION OF CALLED SUBR
```

--- ROUTINE: ISARET

Supporting Subroutines: FRET

Subroutine Size: 2 ZREL

Entry: .ISANORM, ISA.NORM

Provides ISA normal return by setting the argument [indicated] by FPLP to a 1. (.FARL or .FARG sets FPLP.)

Call: ISA.NORM

Entry: .ISAERR, ISA.ERR

Provides ISA error return by setting the argument [indicated] by FPLP to the value in AC2, plus 3. (.FARL or .FARG sets FPLP.) Bit 0 of the resulting error is always set to zero.

Call: ISA.ERR

Both routines then return using FRET.

CHAPTER SEVEN

USING THE RUN TIME LIBRARY

STRUCTURE OF SUBROUTINE DESCRIPTIONS

Not all subroutines in the FORTRAN IV Run Time Library are fully described in the following pages. Documentation is not included for those routines which the user is not likely to need to access, such as the multitask scheduler, FTMAX. However, Appendix A includes a complete list of routine titles and meaningful NREL entries.

Each subroutine description is divided into the following categories: Title, Supporting Subroutines (and Displacements), Subroutine Size, Entry Points, Function, Calling Sequence(s), and Notes.

The "Title" is a name selected to describe the subroutine being discussed, corresponding to loader-recognized titles. Appendix A summarizes loader-recognized titles and NREL entry points.

The "Function" section is followed by "Calling Sequence" for each distinct functional entry point. The "Calling Sequence" illustrates the applicable 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 (such as 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".

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 routine, including all entries to the routine. 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 the module.

Selected subroutines have been measured to determine their typical execution times, and these are given in the "Notes". The following comparisons of typical execution times of single precision real arithmetic functions 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 With Software Multiply/Divide	Typical Execution Time 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

INTERFACE BETWEEN ASSEMBLY LANGUAGE AND FORTRAN PROGRAMS

Several rules must be followed when writing a function or subroutine in assembly language which will be called by a FORTRAN program, or which will call FORTRAN programs or subprograms:

1. The first five letters in the 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 (arg1, arg2,...,argn) is as follows:

```
.EXTN NAME
JSR @.FCAL
NAME
n                                   ; number of arguments
FORTRAN ADDRESS of arg1
FORTRAN ADDRESS of arg2
.
.
.
FORTRAN ADDRESS of argn
```

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 (primarily FCALL entries) are specified by .EXTN.

n 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, regardless of how many times NAME is called from within the program.

Corresponding to the calling sequence generated by the FORTRAN CALL statement is the receiving sequence in the called subroutine. This is the means by which the called subroutine fetches the calling parameters. The form of the receiving sequence generated by use of the FORTRAN statement SUBROUTINE is as follows:

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

where FS is the number of temporary locations required by the subroutine NAME in the FORTRAN stack. FS must be large enough to provide the maximum number of arguments expected by the routine. .CPYL converts the n argument addresses to effective addresses and places these addresses in locations TMP, TMP+1,..., TMP+n-1 on the subroutine's 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.

The assembly language code generated by the FORTRAN RETURN statement is JSR @.FRET. This routine restores accumulators, carry, the contents of FSP, and places the current FSP in AC3.

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

INTERFACE BETWEEN ASSEMBLY LANGUAGE AND FORTRAN PROGRAMS (Continued)

SUBROUTINE name (arg1, arg2,...,argn)

the assembly language code required to call this routine would be

```
.EXTN name
FCALL
name
N
FORTRAN ADDRESS 1        ⎫
.                        ⎪
.                        ⎬   Argument Addresses
.                        ⎪
FORTRAN ADDRESS N        ⎭
```

Similarly, given the following FORTRAN function,

FUNCTION name (arg1,arg2,...,argn)

the assembly language code required to call this function would be

```
.EXTN name
FCALL
name
N+1
FORTRAN ADDRESS of result
FORTRAN ADDRESS 1        ⎫
.                        ⎪
.                        ⎬   Argument Addresses
.                        ⎪
FORTRAN ADDRESS N        ⎭
```

If there is no argument list in either a subroutine or function definition, N=0 must be specified explicitly.

If any text strings are to be passed to FORTRAN routines, the first must be preceded by a statement to force the string of text in left to right order: .TXTN 1 .

RUN TIME ERROR MESSAGES

At the initiation of execution, if there is not enough space to allocate unlabeled COMMON and the initial stack frame, the message

MEMOVFL

will be typed.

All other non-fatal runtime error messages are printed as follows:

RUNTIME ERROR nn AT LOC. xxxxxx CALLED FROM LOC. yyyyyy

where nn is a decimal number representing a non-fatal error code.
xxxxxx is the address where the error was detected.
yyyyyy is the address of the first machine instruction of the FORTRAN statement that was being executed when the failure occurred. The address will be within the user's main program or one of his subroutines.

RUN TIME ERROR MESSAGES (Continued)

Execution will continue for non-fatal errors. If the error is fatal the message will read:

FATAL RUNTIME ERROR nn AT LOC. xxxxxx CALLED FROM LOC. yyyyyy

and execution will halt.

The decimal error codes and their meanings are

Code	Meaning
1	Stack overflow.
2	Computed GO TO error.
4	Division by zero.
5	Integer overflow.
6	Integer power error (illegal or overflow).
7	Floating point underflow.
8	Floating point overflow.
9	Illegal format syntax.
11	Logical conversion error.
13	Number conversion error.
14	I/O error.
15	Field error (i.e., F5.10, E5.4, etc.).
16	Square root of negative number.
17	Log of negative number.
18	Channel not open.
19	Channel already open.
20	No channels available.
21	RDOS/SOS exceptional status.
24	Exponential over/underflow.
25	Array element out of bounds.
26	Negative base for floating-point power.
27	Number Stack overflow.
28	BACKSPACE not implemented.
29	Attempt to restore status of channel when the status was not saved.
30	Queued task error.
31	Seek on a non-random file.
32	Overlay aborted.
33	Illegal argument.
34	Delete error (file open).
35	Overlay error in overlay kill.
36	Undefined entry. **

** Occurs when an attempt is made to call a subroutine that was not loaded.

INTEGER ROUTINES

--- ROUTINE: BASC

Supporting Subroutines: FRET, FSAV; .STBT

Subroutine Size: No page zero locations and 35_8 locations of normally relocatable memory.

Entry: .BASC

Function: Converts an unsigned fixed point number to an ASCII string of six digits.

Calling Sequence:

(AC0 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, AC0 contains updated pointer to null byte.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .BASC must be referenced by an .EXTN statement.

(3) The input fixed point number is of the form

$$N_6N_5N_4N_3N_2N_1$$

where N is an octal digit.

(4) The output ASCII string is of the form

A_6A_5
A_4A_3
A_2A_1
0 0

where A_n corresponds to N_n.

(5) No error messages are output

--- ROUTINE: BDASC

Supporting Subroutines: FRET, FSAV; .STBT

Subroutine Size: No page zero locations and 62_8 locations of normally relocatable memory.

Entry: .BDASC

Function: Converts an unsigned fixed point number to a string of ASCII decimal characters.

ROUTINE: BDASC (Continued)

Entry: .BDASC (Continued)

Calling Sequence:

(AC0 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. AC0 contains the updated pointer to the null byte.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .BDASC must be referenced by an .EXTN statement

(3) No error messages are generated.

--- ROUTINE: IABS

Supporting Subroutines: FRET, FSAV; .CPYARG

Subroutine Size: 1 page zero location and 11_8 locations of normally relocatable memory.

Entry: IA.S

Function: Computes the absolute value of an integer argument.

Calling Sequence:

```
JSR @IA.S
FORTRAN ADDRESS of result
FORTRAN ADDRESS of argument
```

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) IA.S must be referenced by an .EXTD statement. The FCALL entry point .IABS must be referenced by an .EXTN statement.

(3) No error messages are generated.

--- ROUTINE: IDIM

Supporting Subroutines: FRET, FSAV; .FARG

Subroutine Size: 1 page zero location and 13_8 locations of normally relocatable memory.

Entry: ID.M

Function: Computes 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
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) ID.M must be referenced by an .EXTD statement. The FCALL entry .IDIM must be referenced by an .EXTN statement.

(3) No error messages are generated.

(4) If $I-J \leq 0$, the result is 0; otherwise, the result is the difference I-J.

--- ROUTINE: IPWER

Supporting Subroutines: MPY; SP, .RTES

Subroutine Size: 1 page zero location and 53_8 locations of normally relocatable memory.

Entry: .IPWR

Function: Raises an integer to an integer power, with an integer result.

Calling Sequence:

(The integral base is in AC1, and the integral power is in AC0.)

JSR @.IPWR

(The result is in AC1.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) An error message is issued if overflow occurs or if a zero base was input.

(3) .IPWR must be referenced by an .EXTD statement.

--- ROUTINE: ISIGN

Supporting Subroutines: FRET, FSAV; .FARG

Subroutine Size: 1 page zero location and 14_8 locations of normally relocatable memory.

Entry: IS.GN

Function: Transfers 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 to transfer
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) IS.GN must be referenced by an .EXTD statement. The FCALL entry .ISIGN must be referenced by an .EXTN statement.

(3) No error messages are generated.

--- ROUTINE: MNMX0

Supporting Subroutines: FRET, FSAV; .FARG

Subroutine Size: 2 page zero locations and 44_8 locations of normally relocatable memory.

Entry: MA.0

Function: Selects the largest member from a set of integers, expressing the selection as an integer.

Calling Sequence:

```
JSR @MA.0
N+1 (where N is the number of members in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of I0
FORTRAN ADDRESS of I1
:
FORTRAN ADDRESS of In-1
```

(The integer result is stored at the FORTRAN ADDRESS of the result given in the calling sequence.)

Entry: MI.0

Function: Selects the smallest member from a set of integers, expressing the selection as an integer.

ROUTINE: MNMX0 (Continued)

Entry: MI.0 (Continued)

Calling Sequence:

JSR @MI.0
N+1 (where N is 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 integer result is stored at the FORTRAN ADDRESS of the result given in the calling sequence.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) MA.0 and MI.0 must be referenced by .EXTD statements. The FCALL entry points, MAX0 and MIN0, must be referenced by .EXTN statements.

--- ROUTINE: MOD

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

Subroutine Size: 1 page zero location and 11_8 locations of normally relocatable memory.

Entry: MO.

Function: Fetches 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 I_1
FORTRAN ADDRESS of integer I_2

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) MO. must be referenced by an .EXTD statement. The FCALL entry .MOD must be referenced by an .EXTN statement.

(3) In the case of an illegal division, an error return will be made by .SDVD, and a zero result will be returned.

--- ROUTINE: MULT

Supporting Subroutines: ; .SVO

Subroutine Size: 3 page.zero locations and 33_8 locations of normally relocatable memory.

Entry: MPY

Function: Performs unsigned integer multiplication on NOVA family machines lacking the hardware multiply/divide.

Calling Sequence:

(AC1 and AC2 contain the multiplier and multiplicand upon entry to the routine; contents of AC0 will be added to the product.)

MPY

(The product of AC1 and AC2 is computed, and the entry contents of AC0 is added to the product. This sum is returned with the more significant half in AC0, the less significant half in AC1. AC3 contains the caller's FSP upon exit.)

Entry: MPY0

Function: Performs unsigned integer multiplication on NOVA family machines lacking the hardware multiply/divide.

Calling Sequence:

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

MPY0

(The product of AC1 and AC2 is returned with the less significant half in AC1, and the more significant half in AC0. AC3 contains the caller's FSP on exit.)

Entry: DVD

Function: Performs unsigned integer division on NOVA family machines lacking the hardware multiply/divide.

Calling Sequence:

(The high and low parts of the dividend are in AC0 and AC1, the divisor is in AC2.)

DVD

(The remainder is in AC0, the quotient is in AC1, AC2 is unchanged, and carry is cleared; AC3 is set to FSP. Upon overflow, carry is set, FSP is placed in AC3, and return is made with the accumulators unchanged.)

ROUTINE: MULT (Continued)

Notes:

(1) MPY, MPY0, and DVD must each be referenced by an .EXTN statement.

(2) Typical execution times for MPY0 are 74 μs on the SUPERNOVA and 349 μs on the NOVA.

(3) Typical execution times for MPY are 73 μs on the SUPERNOVA and 343 μs on the NOVA.

(4) Typical execution times for DVD are 96 μs on the SUPERNOVA and 491 μs on the NOVA.

--- ROUTINE: SDVD

Supporting Subroutines: DVD; .RTES, SP

Subroutine Size: 1 page zero location and 46_8 locations of normally relocatable memory.

Entry: .SDVD

Function: Performs a division of two signed integers.

Calling Sequence:

(AC0 contains the signed divisor, AC1 contains the signed dividend.)

JSR @.SDVD

(AC0 contains the signed remainder, AC1 contains the signed quotient.)

Notes:

(1) Accumulators and carry are restored except as noted.

(2) .SDVD must be referenced by an .EXTD statement.

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

--- ROUTINE: SMPY

Supporting Subroutines: MPY; .RTES, SP

Subroutine Size: 1 page zero location and 24_8 locations of normally relocatable memory.

Entry: .SMPY

Function: Performs a multiplication of two signed integers.

Calling Sequence:

(AC0 contains the signed multiplicand, AC1 contains the signed multiplier.)

JSR @.SMPY

(AC1 contains the signed result; the result is 0 if overflow occurs.)

Notes:

(1) Accumulators and carry are restored except as noted.

(2) .SMPY must be referenced by an .EXTD statement.

(3) An error message is output if overflow occurs.

SINGLE PRECISION FLOATING ROUTINES

--- ROUTINE: ABS

Supporting Subroutines: FSAV, FQRET; NSP

Subroutine Size: 1 page zero location and 6 locations of normally relocatable memory.

Entry: .ABS

Function: Computes 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.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) ABS., XAS., and DABS. are all equivalent to .ABS . .ABS must be referenced by an .EXTD statement, and ABS., XAS., and DABS. by .EXTN statements. The FCALL entry, ABS, must be referenced by an .EXTN statement.

(3) No error messages are generated.

--- ROUTINE: AINT

Supporting Subroutines: ;.FRG0, FFLDI, NSP, SP

Subroutine Size: 1 page zero location and 60_8 locations of normally relocatable memory.

Entry: AI.T

Function: Truncates 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XA.T is equivalent to AI.T and both must be referenced by .EXTD statements.

(3) No error messages are generated.

---ROUTINE: ALG

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

Subroutine Size: 2 page zero locations and 205_8 locations of normally relocatable memory.

Entry: ALOG.

Function: Computes the single precision real natural logarithm of a single precision real positive argument x.

Calling Sequence:

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

ALOG.

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

Entry: AL.G0

Function: Computes the single precision real base 10 logarithm of a single precision real argument x.

Calling Sequence:

```
JSR @AL.G0
FORTRAN ADDRESS of x
```

(Output result is placed on top of number stack.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) ALOG. and its FCALL entry, ALG, must both be referenced by .EXTN statements. AL.G0 must be referenced by an .EXTD statement and its FCALL entry, .ALG10, must be referenced by an .EXTN statement.

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

(4) In the case of a negative argument, an error message is given and the logarithm of the argument is computed.

---ROUTINE: AMNX1

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

Subroutine Size: 2 page zero locations and 74_8 locations of normally relocatable memory.

Entry: AM.X1, AMAX1

Function: Selects the largest member from a set of single precision real numbers, expressing the result as a single precision real number.

Calling Sequence:

JSR @AM.1
N (the number of members in the set)
FORTRAN ADDRESS of R_0
FORTRAN ADDRESS of R_1
:
FORTRAN ADDRESS of R_{n-1}

(The result is placed on the number stack.)

O R

FCALL
AMAX1
N+1 (where N is the number of elements in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of R_0
FORTRAN ADDRESS of R_1
:
FORTRAN ADDRESS of R_{n-1}

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

Entry: AM.N1, AMIN1

Function: Selects the smallest member from a set of single precision real numbers, expressing the selection as a single precision real number.

Calling Sequence:

JSR @ AM.1
N (the number of members in the set)
FORTRAN ADDRESS of R_0
FORTRAN ADDRESS of R_1
:
FORTRAN ADDRESS of R_{n-1}

(The result is placed on the number stack)

O R

ROUTINE: AMNX1 (Continued)

Calling Sequence: (Continued)

FCALL
AMIN1
N+1 (where N is the number of elements in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of R_0
FORTRAN ADDRESS of R_1
⋮
FORTRAN ADDRESS of R_{n-1}

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) AM.X1 and AM.N1 must be referenced by .EXTD statements. AMAX1 and AMIN1 must be referenced by .EXTN statements.

(3) No error messages are generated.

--- ROUTINE: AMOD

Supporting Subroutines: ; .FRG0, NSP, FML1, FDV1, FFLD1

Subroutine Size: 1 page zero location and 100_8 locations of normally relocatable memory.

Entry: AM.D

Function: Fetches 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.)

Notes:

(1) Contents of accumulators and carry are lost.

(2) AM.D must be referenced by an .EXTD statement.

(3) If the quotient causes overflow or underflow, an error message is output by FDV1 and no meaningful result is obtained.

--- ROUTINE: ATN

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

Subroutine Size: 2 page zero locations and 222_8 locations of normally relocatable memory.

Entry: ATAN.

Function: Computes the real arctangent of a real argument

Calling Sequence:

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

ATAN.

(Output argument replaces x on the number stack.)

Entry: AT.N2

Function: Computes the real arctangent of the quotient of two real arguments, y/x.

Calling Sequence:

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

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) ATAN. must be referenced by an .EXTN statement.
AT.N2 must be referenced by an .EXTD statement.
ATAN. has an FCALL entry point, ATN, which must be referenced by an .EXTN statement.

(3) 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.

(4) The routine will accept input arguments of any size, but results computed by the routine will fall within the following ranges:

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

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

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

--- ROUTINE: COS

Supporting Subroutines: FRET, FSAV; NSP, FPLY1, FRLD1, FDV1, FML1, FSB1, FNEG1, FBRK1

Subroutine Size: 2 page zero locations and 145_8 locations of normally relocatable memory.

Entry: COS.

Function: Computes the real cosine of an argument x expressed as a single precision real number.

Calling Sequence:

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

COS.

(Output result replaces x on the number stack.)

Entry: SIN.

Function: Computes the real sine of an argument x expressed as a single precision real number.

Calling Sequence:

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

SIN.

(Output result replaces x on the number stack.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) COS. and SIN. must be referenced by .EXTN statements.

(3) 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.

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

---ROUTINE: DIM

Supporting Subroutines: ; .FRG0, NSP, FFLD1, FRLD1, FCLT1, FSB1

Subroutine Size: 1 page zero location and 32_8 locations of normally relocatable memory.

Entry: DI.

Function: Computes the positive difference of two SPFL real numbers, R and S.

ROUTINE: DIM (Continued)

Entry: DI. (Continued)

Calling Sequence:

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XD. is equivalent to DI. and both must be referenced by .EXTD statements.

(3) If R-S $\leq$ 0 the result is zero; otherwise, the result is the difference R-S. The result is placed on the number stack.

---ROUTINE: EXP

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

Subroutine Size: 1 page zero location and 160_8 locations of normally relocatable memory.

Entry: EXP.

Function: Computes the real value of e^x for x any single precision floating point argument.

Calling Sequence:

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

EXP.

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) EXP. must be referenced by an .EXTN statement. The FCALL entry point EXPO must also be referenced by an .EXTN statement.

(3) 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 $1 \leq 175_8$.

(4) 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.

ROUTINE: EXP (Continued)

Notes: (Continued)

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

--- ROUTINE: EXPC

Supporting Subroutines: : FRLD1, FML1, FLIP1, FPLY1, FSB1, FDV1, .NIR, SP

Subroutine Size: 1 page zero location and 100_8 locations of normally relocatable memory.

Entry: EXPC

Function: Calculates the value e^x-1 for x a single precision real number.

Calling Sequence:

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

JSR @EXPC

(Result replaces x on number stack.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) EXPC must be referenced by an .EXTD statement.

(3) x must be selected such that $0 \leq x*\log_{10}e < 1/2$. No error message is issued if x is selected to yield a value outside the acceptable range.

---ROUTINE: FL

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

Subroutine Size: 17_8 page zero locations and 754_8 locations of normally relocatable memory.

Entry: FFLD1

Function: Unpacks and loads a single precision real number onto the number stack.

Calling Sequence:

FFLD1
FORTRAN ADDRESS of packed number

Entry: FFST1

Function: Packs and stores a single precision real number from the number stack into a FORTRAN ADDRESS

ROUTINE: FL (Continued

Entry: FFST1 (Continued)

Calling Sequence:

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.)

Entry: FAD1

Function: Adds two single precision real numbers.

Calling Sequence:

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.)

Entry: FSB1

Function: Subtracts two single precision real numbers.

Calling Sequence:

FSB1

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

Entry: FCLT1, FCLE1, FCEQ1, FCGE1, FCGT1

Function: Compares the size and sign of two single precision real numbers, and sets 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 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.)

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.)

ROUTINE: FL (Continued)

Entry: FML1

Function: Multiplies two single precision real numbers.

Calling Sequence:

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.)

Entry: FDV1

Function: Divides two single precision real numbers.

Calling Sequence:

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 on the stack.)

Entry: FNEG1

Function: Changes the sign of a single precision 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.)

Entry: FSGN1

Function: Examines 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

(AC0 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 number stack.)

Entry: FXFL1

Function: Converts a fixed point number to an unpacked single precision real and loads it on the number stack.

ROUTINE: FL (Continued)

Entry: FXFL1 (Continued)

Calling Sequence:

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.)

Entry: FLFX1

Function: Pops a single precision real number from the number stack, converts it to fixed point format, and stores it at a specified FORTRAN ADDRESS.

Calling Sequence:

FLFX1
FORTRAN ADDDRESS to receive I

(The top number on 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FFLD1, FFST1, FAD1, FSB1, FCLT1, FCLE1, FCEQ1, FCGE1, FCGT1, FML1, FDV1, FNEG1, FXFL1 and FLFX1 must all be referenced by .EXTN statements.

(3) JSR @DB.E is equivalent to FFLD1 and DB.E must be referenced by an .EXTD statement. JSR @FL.AT is equivalent to FLFX1 and must be referenced by an .EXTD statement.

(4) Error messages are generated on stack overflow or underflow. Error messages are also generated whenever a truncation of significant exponent digits occurs as the result of packing an unpacked number. An error message will also be issued if the input argument to FLFX1 falls outside the range $[-2^{15}+1, +2^{15}-1]$, and 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.

(5) Numbers to be placed on the number stack are normalized after arithmetic operations.

--- ROUTINE: FPWER

Supporting Subroutines:: FLIP1, NSP, .RTES, SP, FRLD1, FML1, ALOG. EXP.

Subroutine Size: 1 page zero location and 53_8 locations of normally relocatable memory.

ROUTINE: FPWER (Continued)

Entry: FPWR1

Function: Raises a non-negative single precision real base to a single precision real power.

Calling Sequence:

(The real power is loaded on 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FPWR1 must be referenced by an .EXTN statement.

(3) This routine generates an error message on receipt of a negative base argument, and returns the negative base as the result; error messages are generated by the supporting routines if stack underflow or overflow occurs.

(4) 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.

--- ROUTINE: PLY1

Supporting Subroutines: ; FRST1, FRLD1, FML1, FAD1, NSP, SP

Subroutine Size: 1 page zero location and 33_8 locations of normally relocatable memory.

Entry: FPLY1

Function: Computes a polynomial P(x) for x a single precision real argument.

Calling Sequence:

(The input argument x is at the top of the number stack. AC0 contains the start address of LIST + 1. See Notes.)

FPLY1

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FPLY1 must be referenced by an .EXTN statement.

ROUTINE: PLY1 (Continued)

Notes: (Continued)

(3) P(x) is of the form $P(x) = C_0 + C_1x + C_2x^2 + \ldots + C_nx^n$ where $C_0 \ldots C_n$ are single precision real coefficients and all powers of x are positive integers.

The order of the coefficients list is

LIST: Single precision fixed point order of the polynomial.
LIST+1: Real coefficient C_n in unpacked form.
⋮
LIST+7: Real coefficient C_0 in unpacked form.
⋮
LIST+1+6n: Real Coefficient C_0 in unpacked form.

(4) The coefficient list requires 6n + 2 locations of normally relocatable memory.

---ROUTINE: RATN1

Supporting Subroutines: ; ATAN., NSP, FDV1, SP, FRLD1, FSB1

Subroutine Size: 1 page zero location and 35_8 locations of normally relocatable memory.

Entry: RATN1

Function: Calculates the arctangent of the quotient of two single precision real arguments on the number stack.

Calling Sequence:

(The argument divisor, OP1, is at the top of the number stack. The argument dividend, 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) RATN1 must be referenced by an .EXTN statement.

--- ROUTINE: SIGN

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

Subroutine Size: 1 page zero location and 33_8 locations of normally relocatable memory.

ROUTINE: SIGN (Continued)

Entry: SI.N, SIGN

Function: Transfers the sign of one single precision real number to another single precision real number.

Calling Sequence:

```
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.)

OR

```
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.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) No error messages are generated.

(3) SI.N must be referenced by an .EXTD statement and SIGN by an .EXTN statement.

--- ROUTINE: SINH

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

Subroutine Size: 2 page zero locations and 66_8 locations of normally relocatable memory.

Entry: .SHIN

Function: Computes the hyperbolic sine of a single precision real number.

Calling Sequence:

(Input argument is on top of number stack.)

```
JSR @.SHIN
```

(Result is placed on the number stack.)

ROUTINE: SINH (Continued)

Entry: SI.H

Function: Computes the hyperbolic sine of a single precision real number.

Calling Sequence:

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

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) .SHIN and SI.H have FCALL entry points SHIN and SNH. SHIN and SNH must be referenced by .EXTN statements; .SHIN and SI.H must be referenced by .EXTD statements.

(3) No error messages are generated.

--- ROUTINE: SQRT

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

Subroutine Size: 1 page zero location and 142_8 locations of normally relocatable memory.

Entry: SQRT.

Function: Computes the single precision real square root of a 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.)
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) Both SQRT. and the FCALL entry point SQR must be referenced by .EXTN statements.

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

(4) If the input argument is negative, an error message is output and the square root of the absolute value of the argument is calculated.

SPFL

--- ROUTINE: TAN

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

Subroutine Size: 1 page zero location and 116_8 locations of normally relocatable memory.

Entry: TAN.

Function: Computes the single precision real tangent of a single precision real argument x.

Calling Sequence:

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

TAN.

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) Both TAN. and the FCALL entry point TN must be referenced by .EXTN statments.

(3) 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.

--- ROUTINE: TANH

Supporting Subroutines: FRET, FSAV; .FARG, NSP, FRLD1, FRST1, FSB1, FAD1, FML1, FDV1, FFLD1, FCLE1, FLIP1, EXP., FNEG1

Subroutine Size: 1 page zero location and 126_8 locations of normally relocatable memory.

Entry: TA.H

Function: Calculates the hyperbolic tangent of a single precision real number.

Calling Sequence:

JSR @TA.H
FORTRAN ADDRESS of the argument

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) TA.H must be referenced by an .EXTD statement. The FCALL entry point TNH must be referenced by an .EXTN statement.

(3) Error messages may be generated by the supporting routines.

DOUBLE PRECISION FLOATING POINT ROUTINES

--- ROUTINE: ARCTA

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

Subroutine Size: 2 page zero locations and 301_8 locations of normally relocatable memory.

Entry: DATA.

Function: Calculates the arctangent of a double precision real number.

Calling Sequence:

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

DATA.

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

Entry: DA.N2

Function: Calculates the arctangent of the quotient of two double precision real numbers.

Calling Sequence:

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

(The arctangent of the quotient of the input arguments is loaded on the number stack.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) XA.N2 and DA.A2 are equivalent to DA.N2 and all must be referenced by .EXTD statements. XAAN. is equivalent to DATA. and both must be referenced by .EXTN statements.

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

(4) Typical execution times for DA.N2 are 120 ms for the NOVA with software multiply/divide and 14 ms for the SUPERNOVA with hardware fixed point multiply/divide. Timings for DATA. are 74 ms for the NOVA with software multiply/divide and 15 ms for the SUPERNOVA with hardware fixed point multiply/divide.

--- ROUTINE: COSIN

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

Subroutine Size: 2 page zero locations and 161_8 locations of normally relocatable memory.

Entry: DCOS.

Function: Calculates the cosine of a double precision real number.

Calling Sequence:

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

DCOS.

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

Entry: DSIN

Function: Calculates the sine of a double precision real number.

Calling Sequence:

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

DSIN.

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) XCS. is equivalent to DCOS. and XSN. is equivalent to DSIN. All must be referenced by .EXTN statements.

(3) Typical execution times for DCOS. are 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.

--- ROUTINE: DEXPC

Supporting Subroutines: ; FRLD2, FML2, FLIP2, FPLY2, FSB2, FDV2, NSP, SP, .NR1

Subroutine Size: 1 page zero location and 137_8 locations of normally relocatable memory.

Entry: DEXPC

Function: Calculates the value e^x-1 for x a double precision real number.

Calling Sequence:

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

JSR @DEXPC

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) DEXPC must be referenced by an .EXTD statement.

(3) Error messages may be generated by the supporting routines.

(4) The range of value for input arguments is restricted such that $0 \leq \log_{10} e * x < 1/2$.

--- ROUTINE: DEXPO

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

Subroutine Size: 1 page zero location and 232_8 locations of normally relocatable memory.

Entry: XEP.

Function: Calculates the value e^x for x any double precision real number.

Calling Sequence:

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

XEP.

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

ROUTINE: DEXPO (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) DEXP. is equivalent to XEP. and both must be referenced by .EXTN statements. The FCALL entry point DEXP must also be referenced by an .EXTN statement.

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

(4) 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.

--- ROUTINE: DFL

Supporting Subroutines: DVD, MPY; SP, FLSP, .RTES, .NDSP, .SV0

Subroutine Size: 17_8 page zero locations and 1233_8 locations of normally relocatable memory.

Entry: FAD2

Function: Adds two double precision real numbers.

Calling Sequence:

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.)

Entry: FSB2

Function: Subtracts two double precision real numbers.

Calling Sequence:

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.)

ROUTINE: DFL (Continued)

Entry: FCLT2, FCLE2, FCEQ2, FCGE2, FCGT2

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

OP2	$<$	OP1 --	FCLT2
OP2	$\leq$	OP1 --	FCLE2
OP2	=	OP1 --	FCEQ2
OP2	$\geq$	OP1 --	FCGE2
OP2	$>$	OP1 --	FCGT2

where OP1 is the top number on the number stack 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, FCEQ2, 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.)

Entry: FFLD2

Function: Unpacks and loads a double precision real number onto the number stack.

Calling Sequence:

```
FFLD2
FORTRAN ADDRESS of packed number
```

Entry: FFST2

Function: Packs and stores a single precision real number from the number stack into a FORTRAN ADDRESS.

Calling Sequence:

```
FFST2
FORTRAN ADDRESS of destination
```

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

Entry: FML2

Function: Multiplies two double precision real numbers.

Calling Sequence:

```
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.)

ROUTINE: DFL (Continued)

Entry: FDV2

Function: Divides two double precision real numbers.

Calling Sequence:

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 on the stack.)

Entry: FNEG2

Function: Changes the sign of a double precision real number at the top of the number stack.

Calling Sequence:

FNEG2

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

Entry: FSGN2

Function: Examines 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

Entry: FSGN2 (Continued)

(AC0 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 number stack.)

Entry: FXFL2

Function: Converts a fixed point number to an unpacked double precision real, and loads it on the number stack.

Calling Sequence:

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.)

ROUTINE: DFL (Continued)

Entry: FLFX2

Function: Pops a double precision real number from the number stack, converts it to fixed point format, and stores it at a specified FORTRAN ADDRESS.

Calling Sequence:

```
FLFX2
FORTRAN ADDRESS to receive I
```

(The top member from 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FFLD2, FFST2, FAD2, FSB2, FCLT2, FCLE2, FCEQ2, FCGE2, FCGT2, FML2, FDV2, FNEG2, FXFL2, and FLFX2 must all be referenced by .EXTN statements.

(3) JSR @XD.E is equivalent to FFLD2 and XD.E must be referenced by an .EXTD statement. JSR @DF.OT is equivalent to FLFX2 and DF.OT must be referenced by an .EXTD statement.

(4) Error messages are generated on stack overflow or underflow. Error messages are also generated whenever a truncation of significant exponent digits occurs as the result of packing an unpacked number. An error message will also be issued if the input argument to FLFX2 falls outside the range $[-2^{15}+1, +2^{15}-1]$, and 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.

(5) Numbers to be placed on the number stack are normalized after arithmetic operations.

--- ROUTINE: DINT

Supporting Subroutines: ;.FRGO, FFLD2, NSP, SP

Subroutine Size: 1 page zero location and 105_8 of normally relocatable memory.

Entry: DI.T

Function: Truncates a double precision real number.

Calling Sequence:

```
JSR @CI.T
FORTRAN ADDRESS of number to be truncated
```

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

ROUTINE: DINT (Continued)

Notes:

(1) Accumulators and carry are not restored on EXIT.

(2) XDIT is equivalent to DI.T and both must be referenced by .EXTD statements.

(3) no error messages are generated.

--- ROUTINE: DLOG

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

Subroutine Size: 2 page zero locations and 275_8 locations of normally relocatable memory.

Entry: DLOG.

Function: Calculates the natural logarithm of a double precision real number.

Calling Sequence:

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

DLOG.

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

Entry: DL.G0, XA.G0

Function: Calculates the logarithm to the base 10 of a double precision real number.

Calling Sequence:

JSR @DL.G0(or @XA.G0)
FORTRAN ADDRESS of argument whose base 10 logarithm is to be calculated.

(The result is loaded on the number stack.)

ROUTINE: DLOG (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) DLOG. and XAOG. are equivalent and must be referenced by .EXTN statements. DL.G0 and XA.G0 are equivalent and must be referenced by .EXTD statements.

(3) If the input argument is negative an error message is issued and the logarithm of the absolute value is calculated. On receipt of a zero argument the largest possible negative number will be returned.

(4) Typical execution times for the natural logarithm function are 99 ms for the NOVA with software multiply/divide and 13 ms for the SUPERNOVA with hardware fixed point multiply/divide. Typical timings 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.

---ROUTINE: DMNMX

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

Subroutine Size: 2 page zero locations and 12_8 locations of normally relocatable memory.

Entry: DM.X1, DMAX1

Function: Selects the largest member from a set of double precision real numbers, expressing the result as a double precision real number.

Calling Sequence:

```
JSR @DM.X1
N (number of members in the set)
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
:
FORTRAN ADDRESS of DRN
```

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

O R

ROUTINE: DMNMX (Continued)

Calling Sequence: (Continued)

```
FCALL
DMAX1
N+1 (N is the number of members in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
:
FORTRAN ADDRESS of DRN
```

Entry: DM.N1, DMIN1

Function: Selects the smallest member from a set of double precision real numbers, expressing the result as a double precision real number.

Calling Sequence:

```
JSR @DM.N1
N (number of members in the set)
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
:
FORTRAN ADDRESS of DRN
```

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

O R

```
FCALL
DMIN1
N+1 (N is the number of members in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
:
FORTRAN ADDRESS of DRN
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) No error messages are generated.

(3) XA.X1 is equivalent to DM.X1, and XA.N1 is equivalent to DM.N1. All must be referenced by .EXTD statements.

(4) DMAX1 and DMIN1 must be referenced by .EXTN statments.

--- ROUTINE: DMOD

Supporting Subroutines: ; FFLD2, FDV2, FML2, .FRG0, NSP, SP, FFST2

Subroutine Size: 1 page zero location and 127_8 locations of normally relocatable memory.

Entry: DM.D

Function: Fetches the remainder of a double precision real quotient when DR1 is divided by DR2.

Calling Sequence:

```
JSR @DM.D
FORTRAN ADDRESS of DR1
FORTRAN ADDRESS of DR2
```

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

Notes:

(1) XA.D is equivalent to DM.D and both must be referenced by .EXTD statements.

(2) If the quotient DR1/DR2 causes overflow or underflow, an error message will be output by FDV2 and no meaningful result will be returned.

--- ROUTINE: DPOLY

Supporting Subroutines: ; NSP, SP, FRLD2, FML2, FAD2, FRST2

Subroutine Size: 1 page zero location and 33_8 locations of normally relocatable memory.

Entry: FPLY2

Function: Computes a polynomial P(x) for x a double precision real number.

Calling Sequence:

(The input argument x is at the top of the number stack. AC0 contains the start address of LIST +1. See Notes.)

```
FPLY2
```

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

ROUTINE: DPOLY: (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FPLY2 must be referenced by an .EXTN statement.

(3) P(x) is of the form $P(x)=C_0 + C_1x + C_2x^2 +...+ C_nx^n$ where $C_0...C_n$ are double precision real coefficients and all powers of x are positive integers.

The order of the coefficients list is:

LIST: Single precision fixed point order of the polynomial
LIST +1: DPFL real coefficient C_n in unpacked form
:
LIST +7: DPFL real coefficient C_{n-1} in unpacked form
:
LIST+1+6n: DPFL real coefficient C_0 in unpacked form.

(4) The coefficient list requires 6n+2 locations of normally relocatable memory.

---ROUTINE: DPWER

Supporting Subroutines: FLIP2, NSP, .RTES, SP, FRLD2, FML2, DLOG., DEXP.

Subroutine Size: 1 page zero location and 55_8 locations of normally relocatable memory.

Entry: FPWR2

Function: Raises 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 loaded onto the number stack, and the result replaces the base at the top of the stack.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FPWR2 must be referenced by an .EXTN statement.

(3) This routine generates an error message and returns the base as the result upon receipt of a negative base argument; the supporting routines generate error messages if underflow or overflow occurs.

(4) 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.

--- ROUTINE: DSIGN

Supporting Subroutines: FRET, FSAV; FFLD2, NSP, .FARG, FFST2

Subroutine Size: 1 page zero location and 33_8 locations of normally relocatable memory.

Entry: DS.GN, DSIGN

Function: Transfers the sign of one double precision real number to another double precision real number.

Calling Sequence:

```
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.)

O R

```
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.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) No error messages are generated.

(3) XS.N is equivalent to DS.GN and both must be referenced by .EXTD statements.

(4) DSIGN must be referenced by an .EXTN statement.

--- ROUTINE: DSINH

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

Subroutine Size: 2 page zero locations and 72_8 locations of normally relocatable memory.

Entry: .DSHIN

Function: Calculates 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.)

Entry: DS.NH

Function: Calculates the hyperbolic sine of a double precision real number.

Calling Sequence:

JSR @DS.NH
FORTRAN ADDRESS of argument

(The result is placed on the number stack.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) XS.H is equivalent to .DSHIN. DS.NH, .DSHIN, and XS.H must all be referenced by .EXTD statements. .DSHIN and DS.NH have FCALL entry points DSINH and DHNH. DSINH and DSNH must be referenced by .EXTN statements.

--- ROUTINE: DSQRT

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

Subroutine Size: 1 page zero location and 127_8 locations of normally relocatable memory.

Entry: DSQR.

Function: Calculates the square root of a double precision real number.

Calling Sequence:

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

DSQR.

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

ROUTINE: DSQRT (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) XSRT. is equivalent to DSQR. and both must be referenced by .EXTN statements. The FCALL entry point DSQR must also be referenced by an .EXTN statement.

(3) An error message is output on receipt of a negative argument. In this case the square root of the absolute value of the argument is calculated.

(4) 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.

--- ROUTINE: DTANH

Supporting Subroutines: FRET, FSAV; FAD2, FML2, FDV2, FFLD2, DEXP., FSB2, FLIP2, FCLT2, FRST2, FRLD2

Subroutine Size: 1 page zero location and 136$_8$ locations of normally relocatable memory.

Entry: DT.NH

Function: Calculates 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.)
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) XT.H is equivalent to DT.NH and both must be referenced by .EXTD statements.

(3) Error messages may be generated by the supporting routines.

(4) The FCALL entry point must be referenced by an .EXTN statement.

(5) 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.

--- ROUTINE: RATN2

Supporting Subroutines: ; DATA., NSP, FDV2, SP, FRLD2, FSB2

Subroutine Size: 1 page zero location and 37_8 locations of normally relocatable memory.

Entry: RATN2

Function: Calculates the arctangent of the quotient of two double precision real arguments on the number stack.

Calling Sequence:

(The argument divisor, OP1, is at the top of the number stack. The argument dividend, OP2, is at the frame following OP1 on the number stack.)

RATN2

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) RATN2 must be referenced by an .EXTN statement.

--- ROUTINE: TANGE

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

Subroutine Size: 1 page zero location and 165_8 locations of normally relocatable memory.

Entry: DTAN.

Function: Calculates the tangent of a double precision real number.

Calling Sequence:

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

DTAN.

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) XTN. is equivalent to DTAN. and both must be referenced by .EXTN statements. The FCALL entry DTN must also be referenced by an .EXTN statement.

(3) 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.

SINGLE PRECISION COMPLEX ROUTINES

--- ROUTINE: CABS

Supporting Subroutines: ; .FRG0, FFLD1, RCABS, SP

Subroutine Size: 1 page zero location and 17_8 locations of normally relocatable memory.

Entry: CA.S

Function: Obtains 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CA.S must be referenced by an .EXTD statement.

(3) Stack overflow messages may be issued by the supporting routines.

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

--- ROUTINE: CADD

Supporting Subroutines: ; FAD1, FRST1, FRLD1, .NR2, SP, NSP

Subroutine Size: 2 page zero locations and 22_8 locations of normally relocatable memory.

Entry: CAD1

Function: Adds the topmost two single precision complex numbers on the number stack.

Calling Sequence:

(The two arguments are loaded on the number stack.)

```
CAD1
```

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

Entry: CSB1

Function: Subtracts the top single precision complex number on the number stack from the next-to-top single precision complex number on the stack.

ROUTINE: CADD (Continued)

Entry: CBS1 (Continued)

Calling Sequence:

(The two arguments are loaded on the number stack.)

CSB1

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CAD1 and CSB1 must be referenced by .EXTN statements.

(3) Error messages are generated by supporting routines if overflow or underflow occurs.

--- ROUTINE: CCEQ

Supporting Subroutines: ; FCEQ1, .NR2, .NR1, .NR3, FRST1, FRLD1, NSP, SP.

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: CCEQ1

Function: Compares two single precision complex numbers for equivalence.

Calling Sequence:

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

CCEQ1

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CCEQ1 must be referenced by an .EXTN statement.

--- ROUTINE: CCOS

Supporting Subroutines: ; COS., SIN., .SHIN, EXP., .NR2, FAD1, FML1, FRST1, FRLD1, FLIP1, SP, NSP

Subroutine Size: 1 page zero location and 43_8 locations of normally relocatable memory.

Entry: CCOS.

Function: Computes the cosine of a single precision complex number.

Calling Sequence:

(A complex number 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CCOS. must be referenced by an .EXTN statement.

(3) Error messages are generated by EXP. or FML1 on underflow or overflow.

(4) 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.

--- ROUTINE: CDIV

Supporting Subroutines: ; FRLD1, FCLE1, FDV1, .NR2, CML1, FLIP1, FML1, .NR3, FAD1, FRST1, .NR1, SP, NSP

Subroutine Size: 1 page zero location and 75_8 locations of normally relocatable memory.

Entry: CDV1

Function: Divides 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.)

ROUTINE: CDIV (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CDV1 must be referenced by an .EXTN statement.

(3) Error messages are generated by supporting routines if overflow or underflow occurs.

--- ROUTINE: CEXPO

Supporting Subroutines: ; EXP., SIN., .NR2, FLIP1, FRLD1, FRST1, FML1, SP, NSP

Subroutine Size: 1 page zero location and 24_8 locations of normally relocatable memory.

Entry: CEXP.

Function: Computes the value of e^C for 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CEXP. must be referenced by an .EXTN statement.

(3) Error messages are generated on underflow or overflow.

(4) 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.

--- ROUTINE: CLIP

Supporting Subroutines: ; .NR3, SP, NSP, .NR1, .NR2, .FLIP

Subroutine Size: 1 page zero location and 15_8 locations of normally relocatable memory.

Entry: CLIP1, CLIP2

Function: Swaps positions of the two topmost complex numbers on the number stack (whether single precision, double precision, or both).

Calling Sequence:

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

CLIP1 (or CLIP2)

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CLIP1 and CLIP2 are equivalent and both must be referenced by .EXTN statements.

--- ROUTINE: CLOAD

Supporting Subroutines: ; .FRG0, FFLD1, SP

Subroutine Size: 1 page zero location and 16_8 locations of normally relocatable memory.

Entry: CFLD1

Function: Unpacks and loads a single precision complex number onto the number stack.

Calling Sequence:

CFLD1
FORTRAN ADDRESS of the packed complex number.

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

ROUTINE: CLOAD (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CFLD1 must be referenced by an .EXTN statement.

(3) An error message is issued by FFLD1 if number stack overflow occurs.

(4) The FORTRAN ADDRESS of the packed complex number points to four sequential 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.

--- ROUTINE: CLOG

Supporting Subroutines: ; .NR2, RATN1, FRLD1, FRST1, ALOG., CLIP1, RCABS, SP, NSP

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: CLOG.

Function: Computes the natural logarithm of a single precision complex number.

Calling Sequence:

(The single precision argument is loaded on the number stack.)

CLOG.

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CLOG. must be referenced by an .EXTN statement.

(3) Error messages are generated on underflow or overflow by the supporting routines.

(4) 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.

--- ROUTINE: CMUL

Supporting Subroutines: ; FML1, FRLD1, FAD1, FRST1, FSB1, .NR1, .NR2, .NR3, SP, NSP

Subroutine Size: 1 page zero location and 47_8 locations of normally relocatable memory.

Entry: CML1

Function: Multiplies a single precision complex number by another single precision complex number.

Calling Sequence:

(The two arguments are loaded on the number stack.)

CML1

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CML1 must be referenced by an .EXTN statement.

(3) Error messages are issued by supporting routines if overflow or underflow occurs.

--- ROUTINE: CNEG

Supporting Subroutines: ; NSP

Subroutine Size: 1 page zero location and 6 locations of normally relocatable memory.

Entry: CNEG1, CNEG2

Function: Negates 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.)

ROUTINE: CNEG (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CNEG1 and CNEG2 must be referenced by .EXTN statements.

(3) No error messages are generated.

--- ROUTINE: CONJG

Supporting Subroutines: ; NSP

Subroutine Size: 1 page zero location and 5 locations of normally relocatable memory.

Entry: CONJ.

Function: Produces 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.)

Notes:

(1) Accumulators and carry are not restored and AC3 contains FSP on exit.

(2) XCNJ. and DCON. are both equivalent to CONJ. and all must be referenced by .EXTN statements.

(3) Either single or double precision complex numbers may be used as input arguments.

--- ROUTINE: CPWR

Supporting Subroutines: CLOG., CEXP., CML1, .NR3, .NR2, FRLD1, FRST1, SP.

Subroutine Size: 1 page zero location and 20_8 locations of normally relocatable memory.

Entry: CPWR1

Function: Raises a single precision complex number to a single precision complex power.

Calling Sequence:

(The complex power is on 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CPWR1 must be referenced by an .EXTN statement.

(3) Error messages can arise from the supporting routines.

--- ROUTINE: CSIN

Supporting Subroutines: ; SP, NSP, COS., SIN., .SHIN, EXP., .NR2, FAD1, FML1, FRST1, FRLD1, FLIP1

Subroutine Size: 1 page zero location and 42_8 locations of normally relocatable memory.

Entry: CSIN.

Function: Computes the sine of a single precision complex number.

Calling Sequence:

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

CSIN.

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CSIN. must be referenced by an .EXTN statement.

(3) Error messages may be generated by the supporting routines.

--- ROUTINE: CSQRT

Supporting Subroutines: ; .NR2, FRLD1, RATN1, FLIP1, CLIP1, FML1, SQRT., RCABS, SIN., COS., FRST1, SP, NSP

Subroutine Size: 1 page zero location and 47_8 locations of normally relocatable memory.

Entry: CSQR.

Function: Computes 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CSQR. must be referenced by an .EXTN statement.

(3) 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.

--- ROUTINE: CSTOR

Supporting Subroutines: ; .FRG0, FFST1, SP

Subroutine Size: 1 page zero location and 17_8 locations of normally relocatable memory.

Entry: CFST1

Function: Packs and stores 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CFST1 must be referenced by an .EXTN statement.

(3) No error messages are generated.

ROUTINE: CSTOR (Continued)

Notes: (Continued)

(4) 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.

--- ROUTINE: RCABS

Supporting Subroutines: ; SP, NSP, .NR2, .NR1, SQRT, FLIP1, FML1, FDV1 FAD1, FRLD1, FCLE1

Subroutine Size: 1 page zero location and 42_8 locations of normally relocatable memory.

Entry: RCABS

Function: Obtains the absolute value of a single precision complex number located on the number stack.

Calling Sequence:

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

JSR @RCABS

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) RCABS must be referenced by an .EXTD statement.

(3) Error messages are generated by supporting routines.

--- ROUTINE: REAL

Supporting Subroutines: ; .FRG0, FFLD1, SP

Subroutine Size: 2 page zero locations and 22_8 locations of normally relocatable memory.

Entry: RE.L

Function: Fetches the real part of a single precision complex number.

Calling Sequence:

JSR @RE.L
FORTRAN ADDRESS of complex number

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

ROUTINE: REAL (Continued)

Entry: AI.AG

Function: Fetches the imaginary part of a single precision complex number.

Calling Sequence:

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

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) RE.L and AI.AG must be referenced by .EXTD statements.

(3) No error messages are generated.

DOUBLE PRECISION COMPLEX ROUTINES

--- ROUTINE: DCABS

Supporting Subroutines: ; .FRG0, FFLD2, RDCABS, SP

Subroutine Size: 1 page zero location and 22_8 locations of normally relocatable memory.

Entry: DC.BS

Function: Obtains 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XC.S is equivalent to DC.BS and both must be referenced by .EXTD statements.

(3) Stack overflow messages may be issued by the supporting routines.

(4) The result obtained by this routine is a real number and occupies only one 6-word frame on the number stack.

--- ROUTINE: DCADD

Supporting Subroutines: ; FAD2, FRST2, FRLD2, .NR2, SP, NSP

Subroutine Size: 2 page zero locations and 22_8 locations of normally relocatable memory.

Entry: CAD2

Function: Adds the topmost two double precision complex numbers on the number stack.

Calling Sequence:

(The two arguments are loaded on the number stack.)

```
CAD2
```

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

ROUTINE: DCADD (Continued)

Entry: CSB2

Function: Subtracts the top double precision complex number on the stack from the next-to-top double precision complex number on the stack.

Calling Sequence:

(The two arguments are loaded on the number stack.)

CSB2

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CAD2 and CSB2 must be referenced by .EXTN statements.

(3) Error messages are generated by supporting routines if overflow or underflow occurs.

--- ROUTINE: DCCEQ

Supporting Subroutines: FCEQ2, .NR2, .NR1, .NR3, FRST2, FRLD2, SP, NSP

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: CCEQ2

Function: Compares two double precision complex numbers for equivalence.

Calling Sequence:

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

CCEQ2

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CCEQ2 must be referenced by an .EXTN statement.

--- ROUTINE: DCCOS

Supporting Subroutines: ; SP, NSP, DCOS., DSIN., .DSHI, DEXP., .NR2, FAD2, FML2, FRST2, FRLD2, FLIP2

Subroutine Size: 1 page zero location and 45_8 locations of normally relocatable memory.

Entry: DCCO.

Function: Computes 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XCOS. is equivalent to DCCO. and both must be referenced by an .EXTN statement.

(3) Error messages may be generated by the supporting subroutines.

(4) 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.

--- ROUTINE: DCDIV

Supporting Subroutines: ; FRLD2, FCLE2, FDV2, CML2, FLIP2, FML2, .NR3, .NR2, .NR1, FAD2, FRST2, SP, NSP

Subroutine Size: 1 page zero location and 101_8 locations of normally relocatable memory.

Entry: CDV2

Function: Divides 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CDV2 must be referenced by an .EXTN statement.

(3) Error messages are generated by supporting routines if overflow or underflow occurs.

--- ROUTINE: DCEXP

Supporting Subroutines: ; DEXP., DCOS., DSIN., .NR2, FLIP2, FRLD2, FRST2, FML2, SP, NSP

Subroutine Size: 1 page zero location and 24_8 locations of normally relocatable memory.

Entry: DCEX.

Function: Computes the value of e^C for 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XCXP. is equivalent to DCEX. and both must be referenced by .EXTN statements.

(3) Error messages are generated on underflow or overflow.

(4) 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.

--- ROUTINE: DCLOD

Supporting Subroutines: ; .FRG0, FFLD2, SP

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: CFLD2

Function: Unpacks and loads a double precision complex number onto the number stack.

Calling Sequence:

CFLD2
FORTRAN ADDRESS of the packed complex number

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CFLD2 must be referenced by an .EXTN statement.

(3) An error message is issued by FFLD2 if number stack overflow occurs.

ROUTINE: DCLOD (Continued)

Notes: (Continued)

(4) The FORTRAN ADDRESS of the packed complex number points to eight sequential 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.

--- ROUTINE: DCMUL

Supporting Subroutines: ; FML2, FRLD2, FAD2, FRST2, SP, NSP, .NR2, FSB2, .NR1, .NR3

Subroutine Size: 1 page zero location and 47_8 locations of normally relocatable memory.

Entry: CML2

Function: Multiplies one double precision complex number by another double precision complex number.

Calling Sequence:

(The two arguments are loaded on the number stack.)

CML2

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CML2 must be referenced by an .EXTN statement.

(3) Error messages are generated by supporting routines if overflow or underflow occurs.

--- ROUTINE: DCPWR

Supporting Subroutines: ; DCLO., DCEX., CML2, .NR2, .NR3, FRLD2, SP, FRST2.

Subroutine Size: 1 page zero location and 20_8 locations of normally relocatable memory.

Entry: CPWR2

Function: Raises a double precision complex number to a double precision complex power.

ROUTINE: DCPWR (Continued

Entry: CPWR2 (Continued)

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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CPWR2 must be referenced by an .EXTN statement.

(3) Error messages can arise from the supporting routines.

--- ROUTINE: DCSIN

Supporting Subroutines: ; DCOS., .DSHI, DSIN., DEXP., .NR2, FAD2, FML2, FRST2, FRLD2, FLIP2, SP, NSP

Subroutine Size: 1 page zero location and 44_8 locations of normally relocatable memory.

Entry: DCSI.

Function: Computes the sine of a double precision complex number.

Calling Sequence:

(The double precision complex argument is placed at the top of the number stack.)

DCSI.

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XCIN. is equivalent to DCSI. and both must be referenced by .EXTN statements.

(3) Error messages may be generated by the supporting routines.

(4) 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.

--- ROUTINE: DCSQR

Supporting Subroutines: ; SP, NSP, .NR2, FRLD2, RATN2, FLIP2, CLIP2, FML2, DSQR., RDCAB, DSIN., DCOS., FRST2

Subroutine Size: 1 page zero location and 51_8 locations of normally relocatable memory.

Entry: DCSQ

Function: Computes 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XCQR. is equivalent to DCSQ. and both must be referenced by .EXTN statements.

(3) 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.

--- ROUTINE: DCSTR

Supporting Subroutines: ; .FRG0, FFST2, SP

Subroutine Size: 1 page zero location and 20_8 locations of normally relocatable memory.

Entry: CFST2

Function: Packs and stores 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CFST2 must be referenced by an .EXTN statement.

(3) No error messages are generated.

ROUTINE: DCSTR (Continued)

Notes: (Continued)

(4) 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.

--- ROUTINE: DDCLO

Supporting Subroutines: ; .NR2, RATN2, FRLD2, FRST2, DLOG., CLIP2, RDCABS, SP, NSP

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: XCOG.

Function: Computes 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) DCLO. is equivalent to XCOC. and both must be referenced by .EXTN statments.

(3) Error messages are generated on underflow or overflow by the supporting routines.

(4) 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.

--- ROUTINE: DREAL

Supporting Subroutines: .FRG0, FFLD2, SP

Subroutine Size: 2 page zero locations and 24_8 locations of normally relocatable memory.

Entry: DR.AL

Function: Fetches the real part of a double precision complex number.

ROUTINE: DREAL (Continued)

Entry: DR.AL (Continued)

Calling Sequence:

JSR @DR.AL
FORTRAN ADDRESS of complex number

(The real portion of the complex number is loaded on the number stack.)

Entry: DA.MG

Function: Fetches the imaginary part of a double precision complex number.

Calling Sequence:

JSR @DA.MG
FORTRAN ADDRESS of complex number

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

Notes:

(1) Accumulators and carry are not restored on exit.

(2) XR.L is equivalent to DR.AL, and XA.AG is equivalent to DA.MG. All must be referenced by .EXTD statements.

(3) No error messages are generated.

--- ROUTINE: RDCAB

Supporting Subroutines: ; SP, NSP, .NR2, .NR1, DSQR, FLIP2, FML2, FDV2, FAD2, FRLD2, FCLE2

Subroutine Size: 1 page zero location and 44_8 locations of normally relocatable memory.

Entry: RDCAB

Function: Obtains 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.)

ROUTINE: RDCAB (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) RDCAB must be referenced by an .EXTD statement.

(3) Error messages are generated by supporting routines.

(4) The result obtained is a real number occupying only one 6-word frame on the number stack.

MIXED MODE ROUTINES

Mixed Mode

---ROUTINE: AMNX0

Supporting Subroutines: FRET, FSAV; .FARG, FXFL1, FFST1

Subroutine Size: 2 page zero locations and 76_8 locations of normally relocatable memory.

Entry: AM.N0, AMIN0

Function: Selects the smallest member from a set of integers, expressing the selection as a single precision real value.

Calling Sequence:

JSR @ AM.N0
N (integer representing the number of members in 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.)

OR

FCALL
AMIN0
N+1 (where N is 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 stored at the FORTRAN ADDRESS of the result given in the calling sequence.)

Entry: AM.X0, AMAX0

Function: Selects the largest member from a set of integers, expressing the selection as a single precision real value.

Calling Sequence:

JSR @AM.X0
N (integer representing the number of members in 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.)

OR

ROUTINE: AMNX0 (Continued)

Calling Sequence: (Continued)

```
FCALL
AMAX0
N+1 (where N is the number of members in the set)
FORTRAN ADDRESS of I0
FORTRAN ADDRESS of I1
:
FORTRAN ADDRESS of IN-1
```

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

Notes:

(1) Accumulators and carry are restored on exit.

(2) AM.N0 and AM.X0 must be referenced by .EXTD statements. AMAX0 and AMIN0 must be referenced by .EXTN statemtnts.

(3) No error messages are generated.

---ROUTINE: BREAK

Supporting Subroutines: ; NSP, SP, FXFL1, FLFX1, FRLD1, FSB1

Subroutine Size: 1 page zero location and 15_8 locations of normally relocatable memory.

Entry: FBRK1

Function: Separates a single precision real number x into its integer and fractional components.

Calling Sequence:

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

FBRK1

(Output fractional result replaces x on the number stack; integer result is placed in AC0.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FBRK1 must be referenced by an .EXTN statement.

(3) FLFX1 will generate an error message whenever the integer portion of the argument exceeds the range $\pm (2^{15}-1)$.

--ROUTINE: CMPLX

Supporting Subroutines: FSAV, FRET; FFST1, .FARG, FFLD1

Subroutine Size: 1 page zero location and 40_8 locations of normally relocatable memory.

Entry: CM.LX, CMPLX

Function: Constructs a single precision complex number from two single precision real numbers.

Calling Sequence:

```
JSR @CM.LX
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion

(A complex number is formed and loaded on the number stack.)

OR

FCALL
CMPLX
Integer 3
FORTRAN ADDRESS of result
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) CM.LX must be referenced by an .EXTD statement and CMPLX by an .EXTN statement.

(3) No error messages are generated.

---ROUTINE: CRCX1

Supporting Subroutines: ;.FRG0, FFLD1, FRLD1, SP

Subroutine Size: 1 page zero location and 22_8 locations of normally relocatable memory.

Entry: CRCX1

Function: Converts a packed single precision real number R to a single precision complex number of the form R+0i .

Calling Sequence:

```
CRCX1
FORTRAN ADDRESS of single precision real argument R

(The real argument R is expanded to a complex number of the form
R+0i, which is loaded on the number stack.)
```

ROUTINE: CRCX1 (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CRCX1 must be referenced by an .EXTN statement.

(3) Error messages may be generated by the supporting subroutines.

---ROUTINE: CRCX2

Supporting Subroutines: ; .FRG0, FFLD2, FRLD2, SP

Subroutine Size: 1 page zero location and 24_8 locations of normally relocatable memory.

Entry: CRCX2

Function: Converts a packed double precision real number D to a double precision complex number of the form D+0i .

Calling Sequence:

```
CRCX2
FORTRAN ADDRESS of double precision real argument D
```

(The real argument D is expanded to a complex number of the form D+0i, which is loaded onto the number stack.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CRCX2 must be referenced by an .EXTN statement.

(3) An error message will be generated by FFLD2 or FRLD2 if stack overflow occurs.

---ROUTINE: CXFL1

Supporting Subroutines: ; FXFL1, FRLD1, .FRG0, SP

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: CXFL1

Function: Converts an integer I to a single precision complex number of the form I+0i .

Calling Sequence:

```
CXFL1
FORTRAN ADDRESS of the integer argument I
```

(The integer argument I is expanded to a complex number of the form I+0i which is loaded onto the number stack.)

ROUTINE: CXFL1 (Continued)

Notes:

(1) Accumulators and carry are not stored on exit.

(2) CXFL1 must be referenced by an .EXTN statement.

(3) An error message is issued by a supporting routine on stack overflow.

---ROUTINE: CXFL2

Supporting Subroutines: ; FXFL2, FRLD2, .FRG0, SP

Subroutine Size: 1 page zero location and 23_8 locations of normally relocatable memory.

Entry: CXFL2

Function: Converts an integer I to a double precision complex number of the form I+0i .

Calling Sequence:

CXFL2
FORTRAN ADDRESS of integer argument I

(The integer argument I is expanded to a double precision complex number of the form I+0i which is loaded onto the number stack.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) CXFL2 must be referenced by an .EXTN statement.

(3) An error message is issued by a supporting routine if stack overflow occurs.

---ROUTINE: DBREAK

Supporting Subroutines: ; NSP, SP, FXFL2, FRLD2, FSB2, FLFX2

Subroutine Size: 1 page zero location and 15_8 locations of normally relocatable memory.

Entry: FBRK2

Function: Separates a double precision real number into its integer and fractional components.

ROUTINE: DBREAK (Continued)

Entry: FBRK2 (Continued)

Calling Sequence:

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

FBRK2

(The integer portion is expressed as a single precision fixed point value which is loaded into AC0. The fractional component replaces the input argument on the number stack.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FBRK2 must be referenced by an .EXTN statement.

(3) Error messages may be generated by the supporting routines.

---ROUTINE: DCMPL

Supporting Subroutines: FSAV, FRET; .FARG, FFLD2, FFST2

Subroutine Size: 1 page zero location and 43_8 locations of normally relocatable memory.

Entry: DC.PX, DCMPL

Function: Constructs a double precision complex number from two double precision real numbers.

Calling Sequence:

JSR @DC.PX
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion

(A complex number is formed and loaded on the number stack.)

OR

FCALL
DCMPL
Integer 3
FORTRAN ADDRESS of result
FORTRAN ADDRESS of real portion
FORTRAN ADDRESS of imaginary portion

Notes:

(1) Accumulators and carry are restored on exit.

(2) XC.LX is equivalent to DC.PX and both must be referenced by .EXTD statements. DCMPL must be referenced by an .EXTN statement.

(3) No error messages are generated.

--- ROUTINE: DIPWR

Supporting Subroutines: FRET, FSAV; .FARG, NSP, FLIP2, FDV2, FML2, FRST2, FRLD2, FFLD2

Subroutine Size: 1 page zero location and 52_8 locations of normally relocatable memory.

Entry: FIPR2

Function: Raises a double precision real number to an integer power.

Calling Sequence:

FIPR2
FORTRAN ADDRESS of the integer power, I
FORTRAN ADDRESS of the real base

(The real result is loaded on the number stack.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) FIPR2 must be referenced by an .EXTN statement. The FCALL entry point, DIPWR, must also be referenced by an .EXTN statement.

(3) Typical execution times on the NOVA with software multiply/divide are 5 ms*(I-1) when I≥1, or 5 ms when I=0. When I≤-1, the execution time is 17.5 ms + (-I-1)*5.5 ms.

(4) Typical execution times on the SUPERNOVA with hardware multiply/divide are 425 μs when I=0, and 1 ms + (I-1)*.6 ms when I≥1. Execution times when I≤-1 are correspondingly larger.

(5) Each of the above execution times includes the time required for one floating point store operation.

--- ROUTINE: FLIP

Supporting Subroutines: ; SP, NSP

Subroutine Size: 2 page zero locations and 26_8 locations of normally relocatable memory.

Entry: .FLIP

Function: Interchanges two single or double precision real numbers.

Calling Sequence:

(AC0 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.)

Mixed Mode

ROUTINE: FLIP (Continued)

Entry: FLIP1, FLIP2

Function: Interchanges two single or double precision real numbers on the number stack.

Calling Sequence:

(The two topmost frames on the number stack contain variables which will be interchanged.)

FLIP1 or FLIP2

(The two topmost variables on the number stack are swapped.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FLIP must be referenced by an .EXTD statement. FLIP1 and FLIP2 are equivalent and both must be referenced by .EXTN statements.

(3) No error messages are generated.

---ROUTINE: IDINT

Supporting Subroutines: FSAV, FRET; FLFX2, FFLD2, .FARG

Subroutine Size: 1 page zero location and 11_8 locations of normally relocatable memory.

Entry: ID.NT

Function: Truncates a double precision real number and expresses the result as a fixed point number.

Calling Sequence:

```
JSR @ID.NT
FORTRAN ADDRESS of result
FORTRAN ADDRESS of real DR to be truncated
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) DR is truncated, converted to a fixed point number, and is stored at the FORTRAN ADDRESS of the result.

(3) XI. is equivalent to ID.NT and both must be referenced by .EXTD statements. The FCALL entry .IDIN must be referenced by an .EXTN statement.

(4) Error messages are generated if the truncated real number is greater than $2^{15}-1$ or less than $-(2^{15}-1)$.

--- ROUTINE: IFIX

Supporting Subroutines: FSAV, FRET; .FARG, FFLD1, FLFX1, NSP

Subroutine Size: 1 page zero location and 21_8 locations of normally relocatable memory.

Entry: IF.X

Function: Truncates a single precision real number and expresses it as a fixed point number.

Calling Sequence:

```
JSR @IF.X
FORTRAN ADDRESS of integer result
FORTRAN ADDRESS of real value to be truncated
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) XI.X is equivalent to IF.X and both must be referenced by .EXTD statements. The FCALL entry point .IFIX must be referenced by an .EXTN statement.

--- ROUTINE: INT

Supporting Subroutines: FSAV, FRET; .FARG, FFLD1, FLFX1

Subroutine Size: 1 page zero location and 11_8 locations of normally relocatable memory.

Entry: IN.

Function: Truncates a single precision real and expresses the result as the nearest integer.

Calling Sequence:

```
JSR @IN.
FORTRAN ADDRESS of result
FORTRAN ADDRESS of real R to be truncated.
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) R is truncated, converted to a fixed point number and is stored at the FORTRAN ADDRESS indicated in the calling sequence.

(3) If the truncated real is greater than $2^{15}-1$ or less than $-(2^{15}-1)$, FLFX1 will generate an error message.

(4) Result = Sign of argument * largest integer $\leq$ |argument| .

(5) IN. must be referenced by an .EXTD statement. The FCALL entry point .INT must be referenced by an .EXTN statement.

---ROUTINE: LDREG

Supporting Subroutines: ; NSP, SP, .NDSP, .RTER

Subroutine Size. 2 page zero locations and 32$_8$ locations of normally relocatable memory.

Entry: FRLD1

Function: Loads an unpacked single precision real number onto the number stack.

Calling Sequence:

(AC0 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.)

Entry: FRLD 2

Function: Loads an unpacked double precision real number onto the number stack.

Calling Sequence:

(AC0 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FRLD1 and FRLD2 must be referenced by .EXTN statements.

(3) A fatal error message is generated on stack overflow.

(4) An unpacked single precision real number in normally relocatable memory occupies four sequential memory locations. 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.

---ROUTINE: MNMX1

Supporting Subroutines: FSAV, FRET; .FARG, FLFX1, FFLD1, FCLT1

Subroutine Size: 2 page zero locations and 46$_8$ locations of normally relocatable memory.

Entry: MA.1

Function: Selects the largest member from a set of single precision real numbers and expresses the result as a fixed point number.

ROUTINE: MNMX1 (Continued)

Entry: MA.1 (Continued)

Calling Sequence:

```
JSR @MA.1
N+1 (where N is the number of members in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of R1
:
FORTRAN ADDRESS OF RN
```

Entry: MI.1

Function: Selects the smallest member from a set of single precision real numbers and expresses the result as a fixed point number.

Calling Sequence:

```
JSR @MI.1
N+1 (where N is the number of members in the set)
FORTRAN ADDRESS of result
FORTRAN ADDRESS of R1
:
FORTRAN ADDRESS of RN
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) XA.1 is equivalent to MA.1 and XI.1 is equivalent to MI.1 . All must be referenced by .EXTD statements.

(3) FCALL entries MAX1 and MIN1 must be referenced by .EXTN statements.

(4) An error message is generated if the truncated real exceeds $2^{15}-1$ or is less than $-(2^{15}-1)$.

---ROUTINE: RIPWR

Supporting Subroutines: FRET, FSAV; .FARG, NSP, FLIP1, FDV1, FML1, FRST1, FRLD1, FFLD1

Subroutine Size: 1 page zero location and 50_8 locations of normally relocatable memory.

Entry: FIPR1

Function: Raises a single precision real base to an integer power.

Calling Sequence:

```
FIPR1
FORTRAN ADDRESS of the integer power, I
FORTRAN ADDRESS of the real base
```

(The real result is loaded on the number stack.)

ROUTINE: RIPWR (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) FIPR1 must be referenced by an .EXTN statement. The FCALL entry RIPWR must also be referenced by an .EXTN statement.

(3) Error messages are issued by supporting routines whenever appropriate.

(4) Typical execution times on the NOVA with software multiply/divide are 1.45 ms when I=0, and 3 ms + (I-1)*1.7 ms when I > 1. When I < -1, NOVA execution times are 5.3 ms + (-I-1)* 1.6 ms.

(5) Typical execution times on the SUPERNOVA with hardware multiply/divide are 360 µs when I=0, and 550 µs + (I-1)* 180 µs when I > 1. Execution times when I < -1 are correspondingly larger.

(6) Each of the above execution times includes the time required for one floating store operation.

---ROUTINE: STREG

Supporting Subroutines: ; SP, NSP

Subroutine Size: 2 page zero locations and 258 locations of normally relocatable memory.

Entry: FRST1

Function: Stores a single precision real number located on the number stack at a specified address, in unpacked form.

Calling Sequence:

(Address to receive sign word of single precision real number is contained in AC0.)

FRST1

(The single precision number is stored, unpacked, at the four sequential addresses specified, and the number is popped from the number stack.)

Entry: FRST2

Function: Stores a double precision real number located on the number stack at a specified address, in unpacked form.

Calling Sequence:

(Address to receive sign word of double precision real number is contained in AC0.)

FRST2

(The double precision real number is stored, unpacked, at the six sequential addresses specified, and the number is popped from the number stack.)

ROUTINE: STREG (Continued)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) FRST1 and FRST2 must be referenced by .EXTN statements.

(3) No error messages are generated and no check is made by this routine to ascertain whether or not there really is a number on the number stack.

STRING/BYTE MANIPULATION ROUTINES

--- ROUTINE: COMP

Supporting Subroutines: FSAV, FRET; .LDBT

Subroutine Size: 1 page zero location and 34_8 locations of normally relocatable memory.

Entry: .COMP

Function: Compares two character strings for identity.

Calling Sequence:

(String byte pointers in AC0 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.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .COMP must be referenced by an .EXTD statement. The FCALL entry COMP must be referenced by an .EXTN statement.

(3) No error messages are generated.

(4) Each string must be terminated with a null byte.

--- ROUTINE: LD0

Supporting Subroutines: ; QSP

Subroutine Size: 6 page zero locations and 17_8 locations of normally relocatable memory.

Entry: .LD0, .LD1, .LD2

Function: Permits the loading of any accumulator except AC3 from any absolute address.

Calling Sequence:

JSR @.LD0 (.LD1, .LD2)
Any Absolute Address

(AC0 -- or AC1, AC2 -- is loaded with the contents of the absolute address.)

Entry: .ST0, .ST1, .ST2

Function: Permits the storing of any accumulator except AC3 into any absolute address.

ROUTINE: LD0 (Continued)

Entry: .ST0, .ST1, .ST2 (Continued)

Calling Sequence:

JSR @.ST0 (.ST1, .ST2)
Any absolute address

(The contents of AC0 -- or AC1, AC2 -- is stored at the absolute address.)

Notes:

(1) The value of FSP contained in AC3 prior to the call is restored in AC3 upon exit from the routine.

(2) .LD0, .LD1, .LD2, .ST0, .ST1, and .ST2 must be referenced by .EXTD statements.

(3) No error messages are generated if an attempt is made to reference a non-existent location.

(4) This routine uses QSP for temporary storage, so the existence of at least one FORTRAN stack frame is required for operation of the routine.

--- ROUTINE: LDSTB

Supporting Subroutines: ; .SV0

Subroutine Size: 2 page zero locations and 30_8 locations of normally relocatable memory.

Entry: .LDBT

Function: Loads a byte by means of a byte pointer.

Calling Sequence:

(AC0 contains the byte pointer.)

JSR @.LDBT

(AC1 contains the byte, right justified.)

Entry: .STBT

Function: Stores a byte by means of a byte pointer.

Calling Sequence:

(AC1 contains the word whose right byte is to be stored. AC0 contains the byte pointer.)

JSR @.STBT

ROUTINE: LDSTB (Continued)

Notes:

(1) Accumulators and carry are not restored on exit, except for AC0. AC3 contains FSP on exit.

(2) .LDBT and .STBT must be referenced by .EXTD statements.

(3) No error messages are generated.

(4) The byte pointer is left unchanged on exit.

--- ROUTINE: MOVE

Supporting Subroutines: FRET, FSAV

Subroutine Size: No page zero locations and 44$_8$ locations of normally relocatable memory.

Entry: MOVE

Function: Moves all of a byte string.

Calling Sequence:

(AC0 contains the byte pointer to the beginning of the source string. AC1 contains a 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.)

Entry: CMOVE

Function: Moves part of a byte string.

Calling Sequence:

(AC0 contains the byte pointer to the beginning of the source string. AC1 contains a 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.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) MOVE and CMOVE must be referenced by .EXTN statements.

ROUTINE: MOVE: (Continued)

Notes: (Continued)

(3) 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.

--- ROUTINE: MOVEF

Supporting Subroutines: ; SP, .MAD, QSP

Subroutine Size, 1 page zero location and 23_8 locations of normally relocatable memory.

Entry: .MOVE

Function: Moves a block of words.

Calling Sequence:

```
JSR @.MOVE
Word Count
FORTRAN ADDRESS of word block
FORTRAN ADDRESS of word block destination
```

Notes:

(1) Accumulators and carry are not restored on exit.

(2) Upon completion of this routine, the word block is found both at its original location and at the destination location.

--- ROUTINE: MVBT

Supporting Subroutines: FRET, FSAV; .LDBT, .STBT

Subroutine Size: 2 page zero locations and 37_8 locations of normally relocatable memory.

Entry: .MVBC

Function: Moves a byte string.

Calling Sequence:

(AC0 contains a byte pointer to the byte string to be moved;
AC1 contains a byte pointer to the destination of the string;
and AC2 contains the number of bytes in the string.)

```
JSR @.MVBC
```

ROUTINE: MVBT (Continued)

Entry: .MVBT

Function: Moves a byte string.

Calling Sequence:

(AC0 contains a byte pointer to the byte string to be moved; AC1 contains a byte pointer to the destination of the string; and AC2 contains the terminal character in the byte string.)

JSR @.MVBT

Notes:

(1) Accumulators and carry are restored on exit.

(2) .MVBC and .MVBT must be referenced by .EXTD statements. The FCALL entry points MVBC and MVBT must be referenced by .EXTN statements.

(3) On exit from the routine, the byte string is found both at the specified destination and at its original location.

(4) Bytes are packed left to right. Bits 0-14 of the byte pointer specify a memory address, and bit 15 is 0 for left and 1 for the right byte at this address.

--- ROUTINE: MVF

Supporting Subroutines: FQRET, FSAV

Subroutine Size: 1 page zero location and 16_8 locations of normally relocatable memory.

Entry: .MVF

Function: Moves blocks of whole words within core memory.

Calling Sequence:

(Beginning address of the word block to be moved is in AC0; the destination address is in AC1; the number of words in the block is a positive integer in AC2.)

JSR @.MVF

Notes:

(1) Accumulators and carry are restored on exit.

(2) .MVF must be referenced by an .EXTD statement. The FCALL entry point MVF must be referenced by an .EXTN statement.

(3) No error messages are generated.

(4) The original word block is unchanged.

--- ROUTINE: MVZ

Supporting Subroutines: FQRET, FSAV

Subroutine Size: 1 page zero location and 15_8 locations of normally relocatable memory.

Entry: .MVZ

Function: Clears blocks of memory words.

Calling Sequence:

(Beginning address of block in AC1, number of words in the block to be zeroed is in AC0.)

```
JSR @.MVZ
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .MVZ must be referenced by an .EXTD statement. The FCALL entry point MVZ must be referenced by an .EXTN statement.

(3) No error messages are generated.

POINTERS AND DISPLACEMENTS

--- ROUTINE: ARDUM

Supporting Subroutines: None

Subroutine Size: 1 page zero location

Entry: .FLSP

Function: Enables .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.

Definition:

```
        .NREL
.FLSP:  FLSP
        .END
```

Notes:

(1) .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.

(2) .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, and either allocate space for the number stack or not, depending upon the result of this test.

--- ROUTINE: FPTRS

Supporting Subroutines: None

Subroutine Size: 1 page zero location

Function: Defines 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.

Definition:

```
        .ZREL
FLSP:   0
NSP=    FLSP
        .END
```

Notes:

(1) NSP and FLSP are synonyomous 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 .

ROUTINE: FPTRS (Continued)

Notes (Continued)

(2) FLSP is the label of a ZREL location other than 377$_8$. This label is tested by .I which then either allocates a number stack or not, depending on the result of this test.

(3) FLSP and NSP must both be referenced by .EXTD statements.

---ROUTINE: FPZERO

Supporting Subroutines: None

Subroutine Size: 11$_8$ page zero locations

Function: These page zero locations are reserved for use by run time routines.

Definitions:

SP	-	Pointer to the Return Address Stack, which is a stack located after the .I stack, and whose size is determined by .T. 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.
.DSI	-	Flag indicating whether or not the Stand-Alone-Operating System has been loaded. If non-zero, SOS was loaded.
.SV0	-	Return save for zero level routines like MPY.
QSP	-	Pointer to FAC2.

Notes:

(1) 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.

--- ROUTINE: NPTR1

Supporting Subroutines: ; NSP

Subroutine Size: 1 page zero location and 5 locations of normally relocatable memory.

Entry: .NR1

Function: Obtains a pointer to the first frame below the top frame of the number stack.

Calling Sequence:

JSR @.NR1

(Pointer is returned in AC0.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .NR1 must be referenced by an .EXTD statement.

(3) No error messages are generated.

(4) A frame is understood to be a block of six consecutive locations on the number stack.

--- ROUTINE: NPTR3

Supporting Subroutines: ; NSP

Subroutine Size: 1 page zero location and 5 locations of normally relocatable memory.

Entry: .NR3

Function: Obtains a pointer to the third frame below the top frame of the number stack.

Calling Sequence:

JSR @.NR3

(Pointer is returned in AC0.)

Notes:

(1) Accumulators and carry are not restored and AC3 loses FSP on exit.

(2) .NR3 must be referenced by an .EXTD statement.

(3) No error messages are generated.

(4) A frame consists of a block of six consecutive locations on the number stack.

--- ROUTINE: NRPTR

Supporting Subroutines: ; NSP

Subroutine Size: 1 page zero location and 5 locations of normally relocatable memory.

Entry: .NR2

Function: Obtains a pointer to the second frame below the top frame of the number stack.

Calling Sequence:

```
JSR @.NR2
```

(Pointer is returned in AC0.)

Notes:

(1) Accumulators and carry are not restored and AC3 loses FSP on exit.

(2) .NR2 must be referenced by an .EXTD statement.

(3) No error messages are generated.

(4) A frame consists of a block of six consecutive locations on the number stack.

LINKAGE AND INITIALIZATION ROUTINES

--- ROUTINE: AFRTN

Supporting Subroutines: FRET; .FRG0

Subroutine Size: 1 page zero location and 5 locations of normally relocatable memory.

Entry: .AFRTN

Function: Provides 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
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .AFRTN must be referenced by an .EXTD statement.

(3) No error messages are generated.

--- ROUTINE: ARGUM

Supporting Subroutines: ; SP

Subroutine Size: 1 page zero location and 34_8 locations of normally relocatable memory.

Entry: .FARG

Function: Fetches a called subroutine's argument addresses when these are stored as FORTRAN ADDRESSes immediately following the caller.

Calling Sequence:

(AC0 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.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FARG must be referenced by an .EXTD statement.

(3) The following example illustrates the use of .FARG:

ROUTINE: ARGUM (Continued)

Notes: (Continued)

```
                    .ZREL
AL.G0               .ALG10-2
.NREL               .NREL
.MAIN               .
                    .
                    .
CAL1:               JSR @AL.G0 ; THIS IS THE CALLING ROUTINE
                    FOR FRAN ADDRESS of argument
                    .
                    .
                    .
                    FSAV
                    3
.ALG10              SUBZL 0, 0 ; PUT 1 IN AC0, SINCE THERE IS ONLY
                               ; ONE ARGUMENT FOLLOWING MAIN CALLER
.CAL2:              JSR @.FARG ; ARGUMENT ADDRESS IS STORED ON
                               ;ALG10'S STACK.
```

--- ROUTINE: CPYARG

Supporting Subroutines: FRET, FSAV; .MADO

Subroutine Size: 2 page zero locations and 42_8 locations of normally relocatable memory.

Entry: .CPYA

Function: Transfers effective addresses of a caller's argument list to its called subroutine's stack.

Calling Sequence:

```
                FCALL
                SUBR
                N          ; NUMBER OF ARGUMENTS IN LIST
                FADDR
                FADDR
                .
                .
                .
SUBR:           .
                .
                .
```

(AC0 contains the number of arguments to be passed.)

```
JSR @.CPYA  ; ADDRESSES OF CALLER'S ARGUMENTS
            ; ARE NOW ON SUBR STACK
```

ROUTINE: CPYARG (Continued)

Entry: .CPYL

Function: Transfers effective addresses of a caller's argument list to its called subroutine's stack.

Calling Sequence:

```
        FCALL
        SUBR
        N               ; NUMBER OF ARGUMENTS IN LIST
        FADDR
        FADDR
        .
        .
SUBR:   .
        .
        .
        .
```

(AC0 contains the number of arguments to be passed.)

```
JSR @.CPYL      ; ADDRESS OF CALLER'S ARGUMENTS ARE
                ; NOW ON SUBR STACK
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .CPYL and .CPYA must be referenced by .EXTD statements. The FCALL entry point CPYAR must be referenced by an .EXTN statement.

(3) This routine is more generalized than FARG.

(4) .CPYL updates the caller's return address (stored in FRIN) to the next sequential instruction following the call.

--- ROUTINE: FARG0

Supporting Subroutines: ; SP

Subroutine Size: 2 page zero locations and 24_8 locations of normally relocatable memory.

Entry: .FRG0

Function: Calculates the effective address of an argument on the current stack frame, given its FORTRAN ADDRESS pointed to by AC2.

Calling Sequence:

(FORTRAN ADDRESS is pointed to by AC2.)

JSR @.FRG0

(The address is returned in AC0.)

ROUTINE: FARG0 (Continued)

Entry: .FRG1

Function: Calculates the effective address of an argument on the next-most-current stack frame, given its FORTRAN ADDRESS pointed to by AC2.

Calling Sequence:

(FORTRAN ADDRESS is pointed to by AC2.)

JSR @.FRG1

(The address is returned in AC0.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FRG0 and .FRG1 must be referenced by .EXTD statements.

(3) 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.

--- ROUTINE: FLINK

Supporting Subroutines: .I; AFSE, .RTE0, QSP

Subroutine Size: 5 page zero locations and 140_8 locations of normally relocatable memory.

Entry: FCALL

Function: Used to call a subroutine which has no page zero entry, or to call a subroutine which has a page zero entry without using its page zero entry.

Calling Sequence:

```
        FCALL
        SUBRT       ; NAME OF SUBROUTINE WHICH HAS NO
                    ; PAGE ZERO ENTRY
   or,

        .ZREL
.SUBR:  SUBR-2      ; BOTH SUBR and .SUBR HAVE BEEN ENTERED
        .NREL
        FCALL
        SUBR        ; SUBR HAS A ZREL ENTRY
 NSI:   ....        ; NEXT SEQUENTIAL INSTRUCTION
                    ; FOLLOWING CALL TO SUBR
        FSAV
 SLW:   Integer 1
 SUBR:  ...         ; FIRST TRUE INSTRUCTION OF CALLED SUBR
```

ROUTINE: FLINK (Continued)

Entry: FRCAL

Function: Calls a subroutine whose address is contained in AC2, and creates a stack for this subroutine if needed.

Calling Sequence:

```
          .ZREL
.SUBR:    SUBR
          .
          .
          .
          .NREL
          LDA 2, .SUBR
          FRCAL
          .
          .
          .
          SLW             ; STACK LENGTH WORD
SUBR:     .
          .
          .
```

Entry: FSAV

Function: Saves a caller's accumulators and state of carry upon a subroutine page zero call, creates a new stack frame with temporary storage allocated if needed, and checks for stack overflow.

Calling Sequence:

```
          FSAV
          I               ; STACK LENGTH WORD

          .ZREL
.SUBR:    SUBR-2
          .NREL
          .
          .
          .
          JSR @.SUBR      ; NEXT SEQUENTIAL INSTRUCTION FOLLOWING
NSI:      ...             ; RETURN FROM SUBR
          .
          .
          .
          FSAV            ; SAVE ACCUMULATORS, CARRY
SLW:      I               ; STACK LENGTH WORD
SUBR:     ...             ; FIRST TRUE CALLED INSTRUCTION
```

Entry: FRET

Function: Restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

ROUTINE: FLINK (Continued)

Entry: FRET (Continued)

Calling Sequence:

```
          .ZREL
.SUBR:    SUBR-2
          .NREL
          JSR @.SUBR
NEXT:     ...
          FSAV
          SLW
SUBR:     ...
          .
          .
          .
          FRET          ; RESTORE CALLER'S ACCUMULATORS AND RETURN
```

Entry: FQRET

Function: Provides return from a called subroutine which neither requires temporary storage nor calls other subroutines.

Calling Sequence:

```
          FSAV
          -1            ; NO TEMPORARY STORAGE
SUBR      ...
          .
          .
          .
          FQRET
```

Notes:

(1) Accumulators (except AC3) and carry are restored on exit; AC3 contains the caller's FSP.

(2) JSR @.FCALL is equivalent to FCALL. JSR @.FSAV is equivalent to FSAV. FRET is equivalent to JSR @.FRET.

(3) FCALL, FRCAL, FSAV, FRET, and FQRET must all be referenced by .EXTN statements.

(4) .FCALL, .FSAV, .FRET must all be referenced by .EXTD statements.

(5) A fatal error message is generated if insufficient core is available for creation of the called routine's stack frame.

(6) On entry to SUBR, AC0 and 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.

(7) 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.

--- ROUTINE: FRGLD

Supporting Subroutines: ; SP, .FRG1

Subroutine Size: 1 page zero location and 10_8 locations of normally relocatable memory.

Entry: .FRGLD

Function: Fetches 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 AC0.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FRGLD must be referenced by an .EXTD statement.

(3) If the FORTRAN ADDRESS is a stack displacement, it is resolved with respect to the next-most-current stack frame, the caller's stack frame.

--- ROUTINE: I

Supporting Subroutines: .FLSZ, .FLSP, FCALL, .MAIN; QSP, SUCOM, AFSE, .NDSP, SP, .STOP

Subroutine Size: 1 page zero location and 241_8 locations of normally relocatable memory.

Entry: .I

Function: Allocates number and SP stacks and blank and unlabeled common for FORTRAN compiled programs, initializes the Run Time Stack, and constructs pointers in a SOS or single task RDOS environment.

Calling Sequence:

Instead of being called, .I receives program control when the loaded program is started, since the .END statement in this routine has the argument .I, whereas each other library routine is terminated by a simple .END statement. .I invokes the TCB initializer, after which it transfers control to the FORTRAN task scheduler.

ROUTINE: I (Continued)

Entry: I (Continued)

Description:

A system call, .SYSI, is issued to initialize system I/O under SOS, 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 created. The SP stack is simply 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_8 words long or 30_8 plus twice whatever a user has specified in a .FLSZ statement. The default value creates enough room for 68 real numbers or 34 complex numbers, either single or double precision. After the allocation of the number stack (or after the allocation of the SP stack if no number stack is called for), .I's stack with 60 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 Assignement Table is initialized and placed in the .I stack.

Upon completion of the initialization procedure, .I issues an FCALL to the assembly language routine having the entry .MAIN, generated from the main FORTRAN program. At the completion of .MAIN, return is made to .I . Under RDOS, .I calls STOP, which then transfers control back to .I after out-putting STOP 999 on the console. The system performs an effective halt, JMP . , under SOS.

Entry: FERTN

Function: Transfers control to the CLI via the call .SYSTM, .RTN .

Entry: FERT0

Function: Transfers control to the CLI via the call .SYSTM, .ERTN .

Entry: FERT1

Function: Transfers control to the debugger.

--- ROUTINE: MAD

Supporting Subroutines: None

Subroutine Size: 2 page zero locations and 25_8 locations of normally relocatable memory.

Entry: .MAD

Function: Resolves an effective address from a given FORTRAN ADDRESS.

Calling Sequence:

(Input FORTRAN ADDRESS in AC2; current (i.e., caller's) FSP is base used in calculation.)

JSR @.MAD

(AC2 contains effective address on exit; AC3 does not contain caller's FSP on exit.)

Entry: MADO

Function: Resolves an effective address from a given FORTRAN ADDRESS.

Calling Sequence:

(Input FORTRAN ADDRESS in AC2; base FSP in AC1.)

JSR @.MADO

(AC2 contains effective address on exit; AC3 does not contain caller's FSP on exit.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .MAD and .MADO must be referenced by .EXTD statements.

(3) No error messages are generated.

---ROUTINE: MTI

Supporting Subroutines: INHB, FRTSK, FHMA, DVD, FCALL, KILL, SVVAR, TMAX2, .BASC, .MAIN; SUCOM, FLSP, .OVFL, SP, .WRCH, .IOCAT

Subroutine Size: No page zero locations and 426_8 locations of normally relocatable memory. The Channel Assignment Table, 60_8 locations, overwrites part of .I after that section of initialization code has been executed.

Entry: .I

Function: (1) Performs initialization functions in a multitasking environment.

(2) Partitions the free memory area into equal segments for the creation of each task's run time stacks.

(3) Allocates a blank common area if needed.

(4) Builds an I/O Channel Assignment Table initialized to the default values of the logical FORTRAN channels.

(5) Allocates number, SP, and run time stacks, and creates the associated stack pointers for the first task by means of a call to the TCB initializer.

Calling Sequence:

Instead of being called, .I receives program control when the loaded program is started, since the .END statement in this routine has the argument .I, whereas each other library routine is terminated by a simple .END statement. .I invokes the TCB initializer, after which it transfers control to the FORTRAN task scheduler.

Description:

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. 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, error message MEMOVFL is output and a return to the next higher program level (usually the CLI) is made by means of .SYSTM, .RTN .

A temporary SP stack is then created (and will later be overwritten). 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, one for each 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. For a more detailed discussion of run time segments, see Chapter 2.

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, which is of no further use in a multi-tasking environment after its initial execution. Control is then given to the FORTRAN Task Scheduler.

ROUTINE: MTI (Continued)

Entry: FERTN

Function: Transfers control to the CLI via the call .SYSTM, .RTN .

Entry: FERT0

Function: Transfers control to the CLI via the call .SYSTM, .ERTN .

Entry: FERT1

Function: Transfers control to the debugger.

Entry: ITCB

Function: Allocates number, SP, and run time stacks and stack pointers in a FORTRAN task's run time segment area.

Calling Sequence:

Input:
AC0 - priority of the task which is to be assigned the stack area segment
AC1 - starting address of the task's TCB

JSR @.ITCB

Output:
SP, NSP, .NDSP, AFSE, .IOCAT, FSP, QSP, .OVFL are initialized

Notes:

.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.

--- ROUTINE: NFRTN

Supporting Subroutines: FRET

Subroutine Size: 1 page zero location and 10_8 locations of normally relocatable memory.

Entry: .NFRTN

Function: Provides a called subroutine with a means of return to the first location following the caller's parameter list.

ROUTINE: NFRTN (Continued)

Entry: .NFRTN (Continued)

Calling Sequence:

```
JMP @.NFRTN
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .NFRT must be referenced by an .EXTD statement.

(3) No error messages are generated.

(4) This routine assumes that FRTN points to N, the first item in the caller's parameter list.

--- ROUTINE: OVFLO

Supporting Subroutines: ; .AFRT, .OVFL, .CPYL

Subroutine Size: No page zero locations and 23_8 locations of normally relocatable memory.

Entry: OVERFLOW

Function: Provides a means of abnormal return from a subroutine by checking for the occurrence of non-integer arithmetic overflow.

Calling Sequence:

```
FCALL
OVERFLOW
Integer 2 or 3
FORTRAN ADDRESS of return on overflow
FORTRAN ADDRESS of return if no overflow
optional FORTRAN ADDRESS of string literal "S" or "N"
```

Notes:

(1) Accumulators and carry are not restored on exit.

(2) OVERFLOW must be referenced by an .EXTN statement.

(3) The string literal argument consists of an ASCII S or N, left justified and followed by a null byte. If the argument is S, overflow messages are suppressed; if N, overflow messages are not suppressed. S is the default value if no string literal argument address is given.

INPUT/OUTPUT ROUTINES

--- ROUTINE: COUT

Supporting Subroutines: FQRET, FSAV

Subroutine Size: 2 page zero locations and 23_8 locations of normally relocatable memory.

Entry: .COUT

Function: Outputs a character on a teletype.

Calling Sequence:

(AC0 contains the character to be output, right justified.)

JSR @.COUT

(The character is output to a TTY printer/punch.)

Entry: .CIN

Function: Inputs a character on a teletype.

Calling Sequence:

JSR @.CIN

(AC0 contains a character input from a TTY reader/keyboard.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .COUT and .CIN must be referenced by .EXTD statements.

(3) The FCALL entry points COUT and CIN must be referenced by .EXTN statements.

(4) No error messages are generated.

(5) Characters input via .CIN will also be echoed on the TTY printer/punch.

(6) If the character output was a carriage return, a line feed will also be output.

---ROUTINE: FREAD

Supporting Subroutines: FSAV, FRET, MPY, DVD, FERT0; .WRTS, .REDS, .ALLOC, .THREAD, .FRG1, .FRGLD, .READL, .WRITL, .RDFCH, .RTER, .RDFLD, .STBT, .LDBT, .MVBC, .ARYSZ, .FSBR, .WRCH, SP, .SV0

Subroutine Size: 24_8 page zero locations and 3670_8 locations of normally relocatable memory.

Entry: .BRD

Function: Performs FORTRAN input of binary data.

ROUTINE: FREAD (Continued)

Entry: .BRD (Continued)

Calling Sequence:

```
JSR @ .BRD
FORTRAN ADDRESS of the logical channel number
0
ELEMENT DESCRIPTOR SEQUENCE (S) (see Chapter 6)
5
```

Entry: .BWR

Function: Performs FORTRAN output of binary data.

Calling Sequence:

```
JSR @.BWR
FORTRAN ADDRESS of the logical channel number
0
ELEMENT DESCRIPTOR SEQUENCE (S) (see Chapter 6)
5
```

Entry: .FREAD

Function: Performs formatted or free form FORTRAN input of ASCII data.

Calling Sequence:

```
JSR @ .FREAD
FORTRAN ADDRESS of the logical channel number
FORTRAN ADDRESS of the format statement text string
ELEMENT DESCRIPTOR SEQUENCE (S) (see Chapter 6)
5
```

Entry: .FWRIT

Function: Performs formatted or free form FORTRAN output of ASCII data.

Calling Sequence:

```
JSR @.FWRIT
FORTRAN ADDRESS of the logical channel number
FORTRAN ADDRESS of the format statement text string
ELEMENT DESCRIPTOR SEQUENCE (S) (see Chapter 6)
5
```

Notes:

(1) Contents of accumulators and carry are restored on exit.

(2) .FREAD, .FWRI, .BRD, and .BWR must all be referenced by .EXTD statements.

(3) If free form FORTRAN I/O of ASCII data is to be performed, the FORTRAN ADDRESS of the format statement text string in the calling sequence must be 0.

ROUTINE: FREAD (Continued)

Notes: (Continued)

(4) If the contents of the first word in the format text string are 002401, 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, which assembles as 002401.

(5) 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 I/O list. Refer to Chapter 6 for a detailed description.

(6) There are four FCALL entry points corresponding to .FREAD, .FWRIT, .BRD and .BWR: FREAD, FWRIT, BRD and BWR, which must be referenced by .EXTN statements.

--- ROUTINE: IOPRE

Supporting Subroutines: .SV0, .RTER

Subroutine Size: 2 ZREL

Entry: .IOPR

Translates a FORTRAN channel number into the corresponding I/O channel assignment table (IOCAT) entry.

Call:
```
AC2 = FORTRAN channel number
JSR     @.IOPR
 <channel is closed>        ;AC2 = @ERFOP
 <channel is open >         ;AC2 = SYSTEM CHANNEL NUMBER

AC1 = address of IOCAT entry
AC0 = unchanged
```

--- ROUTINE MTDIO

Supporting Subroutines: FRET; .CPYL, .IOCAT

Subroutine Size: No page zero locations and 55_8 locations of normally relocatable memory.

Entry: MTDIO

Function: Allows machine level I/O on magnetic tape and cassette units as described in the RDOS and RTOS manuals.

Calling Sequence:

```
FCALL
MTDIO
Integer 5 or 6
FORTRAN ADDRESS of channel number
FORTRAN ADDRESS of command word
FORTRAN ADDRESS of I/O array specifier
FORTRAN ADDRESS of status word
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of WRD/REC count
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) MTDIO must be referenced by an .EXTN statement.

(3) The command word specifies the operation to be performed:

Bits	Meaning
0	Parity bit (1=even, 0=odd)
1-3	0 - read 1 - rewind the tape 3 - space forward (over record or over file of any size) 4 - space backward (over record or over file of any size) 5 - write (words) 6 - write end of file 7 - read device status word
4-15	Word or record count. If a 0 on a space forward or backward, the tape is positioned to the beginning of the next or previous file on the tape. If 0 on a read or write command, 4096 words are read or written unless an end of record is detected.

ROUTINE: MTDIO (Continued)

Notes: (Continued)

(4) The I/O array specifier points to an integer array used for I/O. This I/O array will be used to return the status information of the transfer according to the following definition:

Bit	Meaning
0	Error (bits 1, 3, 5, 6, 7, 8, 10, or 14 set)
1	Data late
2	Tape is rewinding
3	Illegal command
4	High density if =1 (always 1 for cassettes)
5	Parity error
6	End of tape
7	End of file
8	Tape is at load point
9	9 track if 1, 7 track if 0 (always 1 for cassettes)
10	Bad tape or write failure
11	Send clock (always zero for cassettes)
12	First character (always 0 for cassettes)
13	Write protected or write locked
14	Odd character (always 0 for cassettes)
15	Unit ready

(5) In addition to being returned in the I/O array if the command word contained a 7 in bits 1-3, the status information is always returned in the status word defined in the calling sequence. The bit meanings are defined in (4) above.

WARNING !!	WARNING !!	(6) CAUTION: THIS ROUTINE BYPASSES THE NORMAL ERROR CHECKING AND RECOVERY FOR I/O. IT IS POSSIBLE TO DESTROY THE SYSTEM.

ROUTINE: MTDIO (Continued)

Notes: (Continued)

Only very limited error checking is performed. The error code will be set to one of the following states.

0 - Indeterminate error
1 - No error occurred or error ignored
3...n - RDOS system error code +3

An indeterminate error indicates that bit 0 of the device status word was set. However, an indeterminate error will not be indicated if this error bit was set as a result of end of tape or end of file.

The RDOS error codes will provide limited system and file error information, but fatal device errors can go undetected.

(7) The optional WRD/REC count will return the actual number of words or records processed.

(8) This routine issues the RDOS system call .MTDIO.

---ROUTINE: RDBLK

Supporting Subroutines: FRET; .CPYL, .IOCAT

Subroutine Size: No page zero locations and 72_8 locations of normally relocatable memory.

Entry: RDBLK

Function: Reads 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 starting block number
FORTRAN ADDRESS of array to receive block of data
FORTRAN ADDRESS of number of blocks to be read
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of block count
```

Entry: WRBLK

Function: Writes a series of 256-word blocks from an array into an RDOS disk file. The disk file must be organized either randomly or contiguously.

ROUTINE: RDBLK (Continued)

Entry: WRBLK (Continued)

Calling Sequence:

```
FCALL
WRBLK
Integer number of arguments, 5 or 6
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of starting block number
FORTRAN ADDRESS of array transmitting block of data
FORTRAN ADDRESS of number of blocks to be written
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of block count
```

Notes:

(1) Accumulators and carry are restored upon exit.

(2) RDBLK and WRBLK must be referenced by .EXTN statements.

(3) The starting block number is the logical (or relative) number of the block within the file to or from which reading or writing will occur. The first block in the file is logical block 0, the second is block 1, etc.

(4) 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 read or written. No check is made to determine whether the size of the array is adequate. In the case where a premature end of file is detected on a read, or disk overflow occurs on a write, the optional block count argument will be set to the number of blocks actually read or written.

(5) 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
```

--- ROUTINE: RDFLD

Supporting Subroutines. FSAV, FRET; .FARG, .LDBT, .STBT, .RTER

Subroutine Size: 2 page zero locations and 115_8 locations of normally relocatable memory.

Entry: .RDFLD

Function: Reads and transfers a portion of an ASCII string from one buffer to another by counting characters in the transferred field.

ROUTINE: RDFLD (Continued)

Entry: .RDFLD (Continued)

Calling Sequence:

(AC2 contains number of characters 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)
normal return

(Both the "FROM" and "TO" string pointers are updated upon exit.)

Entry: .RDFCH

Function: Reads and transfers a portion of an ASCII string from one buffer to another by reading to a specified character.

Calling Sequence:

(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)
normal return

(Both the "FROM" and "TO" string pointers are updated upon exit.)

Notes:

(1) Original contents of accumulators and carry are restored upon exit.

(2) .RDFLD and .RDFCH must be referenced by .EXTD statements. RDFLD and RDFCH are FCALL entry points which must be referenced by .EXTN statements.

(3) A fatal error message will be output on overflow of the "TO" buffer only if the last buffer location contains a word consisting of two ASCII rubouts, 077577. FREAD will ensure that such a buffer terminator exists in every case where it issues a call to RDFLD or RDFCH.

(4) 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.

(5) If a carriage return or form feed character is detected by RDFLD, a branch will be made to the abnormal return location.

--- ROUTINE: READL

Supporting Subroutines: FSAV, FRET, FCALL, OPEN; .STBT, .IOCA, .LDBT, .SOSW, .FARG

Subroutine Size: 4 page zero locations and 304_8 locations of normally relocatable memory.

Entry: .WRITL

Function: Performs line output of ASCII data strings on a FORTRAN channel.

Calling Sequence:

(AC0 contains a byte pointer to the beginning 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
```

Notes:

(1) The format flag 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 nonzero, 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.

(2) If formatted output is indicated, the first character in the output string will be examined. If this character is ASCII 0, this zero will be replaced by a carriage return. If the first character is ASCII 1, it will be replaced by a form feed character. All other first characters which are neither ASCII 0 nor 1 will be dropped from the output string.

Entry: .WRTS

Function: Performs line output of binary data strings on a FORTRAN channel.

Calling Sequence:

(AC0 contains a byte pointer to the beginning of the output string. AC1 contains the number of bytes to be written. AC2 contains the FORTRAN logical channel number.)

```
JSR @.WRTS
error return (System error code returned in AC2)
normal return
```

Entry: .READL

Function: Performs line input of ASCII data strings on a FORTRAN channel.

ROUTINE: READL (Continued)

Entry: .READL (Continued)

Calling Sequence:

(AC0 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

Entry: .REDS

Function: Performs line input of binary data strings on a FORTRAN channel.

Calling Sequence:

(AC0 contains a byte pointer to the beginning of the input string buffer. AC1 contains the number of bytes to be read. AC2 contains the FORTRAN logical channel number.)

JSR @.REDS
error return (System error code returned in AC2)
normal return

Notes:

(1) Contents of accumulators and carry are restored upon exit.

(2) Leading nulls are ignored and a trailing null is recognized as a terminator under RDOS.

(3) .WRITL, .WRTS, .READL, and .REDS must be referenced by .EXTD statements.

---ROUTINE: READR

Supporting Subroutines: FRET, MPY, DVD; .CPYL, .IOCAT, .RTER

Subroutine Size: No page zero locations and 76_8 locations of normally relocatable memory.

Entry: READR

Function: Reads a series of records from a file into an array.

ROUTINE: READR (Contineud)

Entry: READR (Continued)

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 array to receive records
FORTRAN ADDRESS of number of records to be read
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of byte count
```

Entry: WRITR

Function: Writes 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 array transmitting records
FORTRAN ADDRESS of number of records to be written
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of byte count
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) The starting record number is the logical (or relative) number of the block within the file to which writing will occur or which will be read. The first record within the file is logical record number 0, the second is logical record number 1, etc.

(3) The routine performs sequential reads or writes by issuing RDOS system calls .RDS or .WRS. If a premature end-of-file is detected on a read, the routine returns a byte count for 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.

(4) 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 on a write, RDOS system error code ERSPC will be given.

(5) READR and WRITR must be referenced by .EXTN statements.

--- ROUTINE: WRCH

Supporting Subroutines: FRET, FSAV; .LDBT, .COUT

Subroutine Size: 1 page zero location and 15_8 locations of normally relocatable memory.

Entry: .WRCH

Function: Prints a string of ASCII characters on a teletype printer.

Calling Sequence:

(AC0 contains a 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.)

Notes:

(1) Accumulators and carry are restored on exit, except for AC1. The contents of AC1 will be as noted above.

(2) .WRCH must be referenced by an .EXTD statement. The FCALL entry point, WRCH, must be referenced by an .EXTN statement.

(3) ASCII characters in the string must be packed left to right, 2 characters per word.

MISCELLANEOUS FORTRAN SUPPORT ROUTINES

--- ROUTINE: BSTRING

Supporting Subroutines: .CPYL, FRET

Subroutine Size: 63_8 locations of normally relocatable memory.

Entry: IOR (m,n)

Function: Inclusive OR where m and n designate arguments that are logically added, bit by bit.

Calling Sequence:

```
FCALL
IOR
N (number of arguments)
FORTRAN address of result
FORTRAN address of M
FORTRAN address of N
```

Entry: IAND (m,n)

Function: Logical product where m and n designate arguments that are logically manipulated.

Calling Sequence:

```
FCALL
IAND
N (number of arguments)
FORTRAN address of result
FORTRAN address of M
FORTRAN address of N
```

Entry: NOT (m)

Function: Logical complement where m designates an argument that is logically complemented.

Calling Sequence:

```
FCALL
NOT
N (number of arguments)
FORTRAN address of result
FORTRAN address of M
```

ROUTINE: BSTRING (Continued)

Entry: IEOR (m,n)

Function: Exclusive OR where m and n designate arguments that are exclusively added.

Calling Sequence:

FCALL
IEOR
N (number of arguments)
FORTRAN address of result
FORTRAN address of M
FORTRAN address of N

Entry: ISHFT (m,n)

Function: Shifts argument m, n positions. The direction of the shift is:

n < 0 shift right
n = 0 no shift
n > 0 shift left

Calling Sequence:

FCALL
ISHFT
N (number of arguments)
FORTRAN address of result
FORTRAN address of M
FORTRAN address of N

--- ROUTINE: CGT

Supporting Subroutines: FRET, FSAV; .RTER, .FRGL

Subroutine Size: 1 page zero location and 23_8 locations of normally relocatable memory.

Entry: .CGT

Function: Implements 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 N1
Effective address N2
:
:
Effective address Nn
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) .CGT must be referenced by an .EXTD statement. The FCALL entry, CGT, must be referenced by an .EXTN statement.

(3) The above calling sequence is generated by the FORTRAN statement

GO TO $(n_1, n_2, \ldots, n_n)V$

(4) 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 routine.

--- ROUTINE: CHSAV

Supporting Subroutines: FRET; .IOCAT, .RTER, .CPYL

Subroutine Size: No page zero locations and 53_8 locations of normally relocatable memory.

Entry: CHSAV

Function: Saves the status of a FORTRAN channel to permit rereading or rewriting of FORTRAN records on disk.

Calling Sequence:

```
FCALL
CHSAV
2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of 3-word integer array for saved channel status data
```

ROUTINE: CHSAV (Continued)

Entry: CHRST

Function: Restores the original status of the channel after a call to CHSAV has been issued to permit rereading or rewriting of FORTRAN records on disk.

Calling Sequence:

```
FCALL
CHRST
2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of 3-word integer array containing previously
saved channel status data.
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) A non-fatal error message is issued 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 in a call to CHSAV.

(3) The status of more than one channel may be saved in the same array. For example, an array declared as I (3, 100) can be used to save up to 100 blocks of channel status information.

(4) The method of using this routine 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.

--- ROUTINE: FINIT

Supporting Subroutines: FRET, FSAV; SUCOM

Subroutine Size: 1 page zero location and 24_8 locations of normally relocatable memory.

Entry: .FINI

Function: Allocates 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.)

ROUTINE: FINIT (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .FINI must be referenced by an .EXTD statement. The FCALL entry point must be referenced by an .EXTN statement.

(3) This routine is of limited usefulness to assembly language programmers and is only described here for completeness.

--- ROUTINE: FOPEN

Supporting Subroutines: FRET, FSAV, IOPTR; .IOCAT, .RTER, .CPYL, .SOSW

Subroutine Size: 1 page zero location and 123_8 locations of normally relocatable memory.

Entry: .FOPEN

Function: Opens a FORTRAN channel.

Calling Sequence:

```
JSR @.FOPEN
Integer number of arguments - 2, 3, or 4
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.)
```

Notes:

(1) .FOPEN must be referenced by an .EXTD statement. The FCALL entry point FOPEN must be referenced by an .EXTN statement.

(2) Up to 64 FORTRAN channel numbers are allowed. Channel numbers are represented by integer constants with values 0 through 63_{10}.

(3) The file name is an ASCII byte string terminated by a null byte. Similarly, 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, such as a rubout character following a tab character output by a paper tape punch.

(4) 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 randomly under RDOS.

--- ROUTINE: ITEST

Supporting Subroutines: FRET; .CPYL, .RTER

Subroutine Size: No page zero locations and 50_8 locations of normally relocatable memory.

Entry: ITEST

Function: Examines a bit in a 16-bit word; performs a logical AND between the word to be examined and a bit mask, placing the result in the FORTRAN ADDRESS of the result.

Calling Sequence:

```
FCALL
ITEST
Integer 3
FORTRAN ADDRESS of result
FORTRAN ADDRESS of word to be examined
FORTRAN ADDRESS of bit position indicator
```

Entry: ISET

Function: Sets 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
```

Entry: ICLR

Function: Clears a bit in a 16-bit word.

Calling Sequence:

```
FCALL
ICLR
Integer 2
FORTRAN ADDRESS of word with bit position to clear
FORTRAN ADDRESS of bit position indicator
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) The bit position indicator is an integer from 0 to 15. The following bit position indicators cause the following bit positions to be acted on:

Bit Position Indicator	Bit Acted Upon
0	Least significant bit
:	
15	Most significant bit

ROUTINE: LE

Supporting Subroutines: FRET, FSAV

Subroutine Size: 4 page zero locations and 76_8 locations of normally relocatable memory.

Entry: .GT, .GE, .LT, .LE

Function: Performs signed comparisons between the contents of registers AC0 through AC2.

R1 > R2 -- GT
R1 ≥ R2 -- GE
R1 < R2 -- LT
R1 ≤ R2 -- LE

Calling Sequence:

(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: 400_8 * 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, 0 is loaded into R2.)

Notes:

(1) Original states of all accumulators but AC3 and R2 are restored on exit, and the entry state of carry is also restored.

(2) .GT, .GE, .LT, and .LE must all be referenced by .EXTD statements. The FCALL entry points GT, GE, LE, and LT must all be referenced by .EXTN statements.

(3) No error messages are generated.

ROUTINE: RTER

Supporting Subroutines: FCALL, FERT1, FRET, FSAV, .BASC, .BDAS, .I; .WRCH, .OVFL, SP, .FSAV

Subroutine Size: 3 page zero locations and 214_8 locations of normally relocatable memory.

Entry: RTE0, RTER, RTES

Function: Indicates 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 (RTE0, RTES). In all cases, the message will specify whether the error is fatal or non fatal.

ROUTINE: RTER (Continued

Entry: RTE0, RTER, RTES (Continued)

Calling Sequence:

(AC0 set to called-from address.)

```
ERROR CODE
JSR @.RTE0
```

(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, along with the appropriate error code.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .RTER, .RTES, and .RTE0 must be referenced by .EXTD statements.

(3) .RTE0 is used by the FLINK module, .RTES by the signed integer and single and double precision real arithmetic routines, and .RTER by the remainder of the run time routines.

(4) The ERROR CODE word has the following structure:

Bit	Meaning
0	Always 1
1	1 for fatal error, 0 for non-fatal
2-11	Specific error code
12	Always 1
13-14	Always 0
15	Always 1

(5) The specific error code will be converted to decimal and output by this routine. A list of all run time error codes is given in the FORTRAN Manual, Appendix B. The definition of the error code structure and mnemonic error code assignments are defined on the PARF tape.

(6) The fixed values of bits zero, twelve, thirteen, fourteen, and fifteen 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/logical no load, skip instruction, skipping over the call to the error processing routine.

ROUTINE: RTER (Continued)

Notes: (Continued)

(7) The non-fatal error messages output by these routines are of the form

RUNTIME ERROR NN AT LOC. xxxxxx, CALLED FROM LOC. yyyyyy

where NN is the decimal run time error code (a complete list of error codes is found in the FORTRAN IV User's Manual). 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.

(8) Fatal error messages will be of the same form as non-fatal error messages with the specifier FATAL appended to the message.

(9) All fatal error conditions cause program control to return to the Debugger (if it is loaded), or otherwise to the operating system.

--- ROUTINE: STOP

Supporting Subroutines: FSAV, FRET, FERTN; .WRCH

Subroutine Size: 2 page zero locations and 52_8 locations of normally relocatable memory.

Entry: .STOP

Function: Implements the FORTRAN STOP function.

Calling Sequence:

```
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.)

Entry: .PAUSE

Function: Implements the FORTRAN PAUSE function.

Calling Sequence:

```
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.)

ROUTINE: STOP (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .STOP and .PAUSE must be referenced by .EXTD statements.

(3) If no text output is desired, a -1 should replace the text string in the calling sequence.

--- ROUTINE: THREAD

Supporting Subroutines: FSAV, FRET; .CPYARG

Subroutine Size: 2 page zero locations and 44_8 locations of normally relocatable memory.

Entry: THREAD

Function: Transfers the latest five word element of one list to a second list.

Calling Sequence:

```
JSR @.THREAD
FORTRAN ADDRESS of "FROM" list pointer
FORTRAN ADDRESS of "TO" list pointer
```

Notes:

(1) Contents of accumulators and carry are restored on exit.

(2) No error messages are generated.

(3) .THREAD must be referenced by an .EXTD statement. THREAD is an FCALL entry point and must be referenced by an .EXTN statement.

(4) Each five word element of a list consists of a block of five sequential locations. The first location (i.e., the one with the lowest core address) is the link word; the remaining four words are reserved for list data storage:

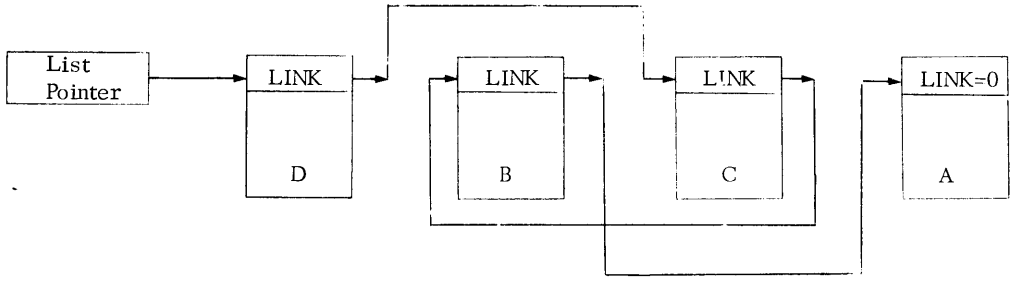
LINK
data
data
data
data

List Element

Lists are variable in length, and list elements may be found in scattered locations throughout available core. The oldest member of a list has a LINK of 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:

ROUTINE: THREAD (Continued)

Notes: (Continued)



(5) .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.

Entry: .ALLOC

Function: Examines a list and, if it is a null list, creates a five word element and transfers it to a second list.

Calling Sequence:

```
JSR @.ALLOC
FORTRAN ADDRESS of "FROM" list pointer
FORTRAN ADDRESS of "TO" list pointer
```

Notes:

(1) Contents of accumulators and carry are restored on exit.

(2) No error messages are generated.

(3) .ALLOC must be referenced by an .EXTD statement. ALLOC is an FCALL entry point and must be referenced by an .EXTN statement.

(4) .ALLOC operates on lists such as those described above. .ALLOC 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 .

ARRAY HANDLING ROUTINES

--- ROUTINE: ARYSZ

Supporting Subroutines: FRET, FSAV, MPY0

Subroutine Size: 1 page zero location and 20_8 locations of normally relocatable memory.

Entry: .ARYSZ

Function: Determines the size of an array in terms of both elements in the array and core locations needed to contain the array.

Calling Sequence:

(AC0 contains the starting address of the subscript bound specifier.)

JSR @.ARYSZ

(AC0 contains the total number of elements in the array, and AC1 contains the total number of words in the array.)

Notes:

(1) Accumulators and carry are restored on exit.

(2) .ARYSZ must be referenced by an .EXTD statement. The FCALL entry point, ARYSZ, must be referenced by an .EXTN statement.

(3) No error messages are generated.

--- ROUTINE: FALOC

Supporting Subroutines: FSAV, FRET; .CPYARG, .RTER, AFSE

Subroutine Size: 1 page zero location and 34_8 locations of normally relocatable memory.

Entry: .FALOC

Function: Allocates 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)

Notes:

(1) Accumulators and carry are restored upon exit.

(2) .FALOC must be referenced by an .EXTD statement.

(3) A fatal error message is generated if there is insufficient run time stack area for allocation of the array.

ROUTINE: FALOC (Continued)

Notes: (Continued)

(4) The caller's FLGT is adjusted to include array size so that newly created stacks will not overwrite the array.

(5) The FCALL entry point, FALOC, must be referenced by an .EXTN statement.

--- ROUTINE: FREDI

Supporting Subroutines: FRET, FSAV, MPY0, .OFLO; AFSE, .CPYAR, .FRGL, QSP

Subroutine Size: 1 page zero location and 111_8 locations of normally relocatable memory.

Entry: .FREDI

Function: Permits 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 compiler)
FORTRAN ADDRESS of area reserved for 3-word array specifier

(A new three word specifier and subscript bound specifier are consutructed. The SBS is appended to the caller's stack. See Chapter 3.)

Notes:

(1) Accumulators and carry are restored upon exit.

(2) .FREDI must be referenced by an .EXTD statement. The FCALL entry point, FREDI, must be referenced by an .EXTN statement.

(3) Upon stack overflow, the contents of the caller's FRTN are printed as an error message.

(4) The call to FREDI is generated by FORTRAN statements of the general form

```
SUBROUTINE      TESTSUB (x,y,z,a,...)
                DIMENSION (x(i),y(m),z(n) )
```

(5) If there is insuffient run time stack area for the creation of a new SBS, a call 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 because at run time only FSAV and FREDI allocate storage on the run time stack.

--- ROUTINE: FSBR

Supporting Subroutines: MPY; .MADO, .RTES, SP

Subroutine Size: 2 page zero locations and 175_8 locations of normally relocatable memory.

Entry: .FSUB

Function: Calculates the effective address of an array element for the compiled program.

Calling Sequence:

```
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.)

Entry: .FSBR

Function: Calculates the effective address of an array element for subroutine FREAD in a formatted I/O entry.

Calling Sequence:

(AC0 contains a pointer to FREAD argument list with ELEMENT DESCRIPTOR SEQUENCE = 1)

```
JSR @.FSBR
```

(The effective address of the selected array element is returned in AC1.)

Notes:

(1) Accumulators and carry are not restored on exit.

(2) .FSBR and .FSUB must be referenced by .EXTD statements.

(3) Subscript calculation errors will be flagged by a fatal error message.

DEVICE/DIRECTORY MAINTENANCE ROUTINES

--- ROUTINE: CDIR

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: CDIR

Function: Creates a subdirectory with a specified name.

Calling Sequence:

```
FCALL
CDIR
Integer 2
FORTRAN ADDRESS of subdirectory name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) CDIR must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .CDIR .

(4) The name of the subdirectory to be created is an ASCII string.

(5) 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
```

--- ROUTINE: CHSTS

Supporting Subroutines: FRET; .CPYL, .IOCAT

Subroutine Size: No page zero locations and 26_8 locations of normally relocatable memory.

Entry: CHSTS

Function: Returns a copy of the current directory status information for a file on a specified channel.

Calling Sequence:

```
FCALL
CHSTS
Integer 3
FORTRAN ADDRESS of channel number
FORTRAN ADDRESS of array to receive status information
FORTRAN ADDRESS of error code
```

ROUTINE: CHSTS (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) CHSTS must be referenced by an .EXTN statement

(3) This routine issues the RDOS system call CHSTS .

(4) This routine differs from STAT in that it obtains the current directory status information for whatever file is open on the specified channel; STAT obtains status information about a specified file. The array size and definition are the same as those for STAT:

Word	Contents
1 - 5	File name
6	Extension
7	File attributes
10	Link access attributes
11	Number of the last block in the file
12	Number of bytes in the last block
13	Starting logical block address of the file (the random file index for random files)
14	Year/day last accessed
15	Year/day created
16	Hour/minute created
17	UFD temporary
20	UFD temporary
21	User count
22	DCT link

(5) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3...n - RDOS system error code + 3

-- ROUTINE: CPART

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 17_8 locations of normally relocatable memory.

Entry: CPART

Function: Creates a secondary partition and enters the name of the secondary partition in the primary partition's directory.

ROUTINE: CPART (Continued)

Entry: CPART (Continued)

Calling Sequence:

```
FCALL
CPART
Integer 3
FORTRAN ADDRESS of name of secondary partition to be created
FORTRAN ADDRESS of size of partition
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) CPART must be referenced by an .EXTN statement.

(3) The name of the secondary partition is an ASCII string.

(4) The size of the secondary partition is the number of contiguous blocks in the partition.

(5) This routine issues the RDOS system call .CPAR .

(6) 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
```

--- ROUTINE: DIR

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: DIR

Function: Defines a current default directory.

Calling Sequence:

```
FCALL
DIR
Integer 2
FORTRAN ADDRESS of device name or directory name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) The device or directory name is an ASCII byte string.

ROUTINE: DIR (Continued)

Notes: (Continued)

(3) This routine issues the RDOS system call .DIR .

(4) 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

--- ROUTINE: DULNK

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: DULNK

Function: Deletes a link entry in the current directory.

Calling Sequence:

```
FCALL
DULNK
Integer 2
FORTRAN ADDRESS of link entry name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) DULNK must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .ULNK .

(4) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3 ... n - RDOS system error code + 3

--- ROUTINE: EQUIV

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 20_8 locations of normally relocatable memory.

Entry: EQUIV

Function. Assigns a temporary name to the master file device.

ROUTINE: EQUIV (Continued)

Entry: EQUIV (Continued)

Calling Sequence:

```
FCALL
EQUIV
Integer 3
FORTRAN ADDRESS of master file device name
FORTRAN ADDRESS of temporary name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) EQUIV must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .EQIV .

(4) 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
```

--- ROUTINE: FSPOL

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 36_8 locations of normally relocatable memory.

Entry: SPEBL

Function: Enables spooling on a specified device.

Calling Sequence:

```
FCALL
SPEBL
Integer 2
FORTRAN ADDRESS of device name
FORTRAN ADDRESS of error code
```

Entry: SPDIS

Function: Disables spooling on a specified device.

Calling Sequence:

```
FCALL
SPDIS
Integer 2
FORTRAN ADDRESS of device name
FORTRAN ADDRESS of error code
```

ROUTINE: FSPOL (Continued)

Entry: SPKIL

Function: Stops a spooling operation which is currently in progress.

Calling Sequence:

```
FCALL
SPKIL
Integer 2
FORTRAN ADDRESS of device name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) SPEBL, SPDIS, and SPKIL must all be referenced by .EXTN statements.

(3) This routine issues the RDOS system calls .SPEA , .SPDA , and .SPKL .

(4) The error code will be set to one of the following states:

```
    0 - Indeterminate error
    1 - No error occurred
3 ... n - RDOS system error + 3
```

--- ROUTINE: GCIN

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 10_8 locations of normally relocatable memory.

Entry: GCIN

Function: Obtains the name of the current console input device.

Calling Sequence:

```
FCALL
GCIN
Integer 1
FORTRAN ADDRESS of array to receive name
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) GCIN must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .GCIN .

(4) No error messages are generated.

--- ROUTINE: GCOUT

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 10_8 locations of normally relocatable memory.

Entry: GCOUT

Function: Obtains the name of the current console output device.

Calling Sequence:

```
FCALL
GCOUT
Integer 1
FORTRAN ADDRESS of array to receive device name
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) GCOUT must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .GCOUT .

(4) No error messages are issued.

--- ROUTINE: GDIR

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: GDIR

Function: Gets the name of the current default directory/device.

Calling Sequence:

```
FCALL
GDIR
Integer 2
FORTRAN ADDRESS of array to receive name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) GDIR must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .GDIR .

(4) The array to receive the current default directory/device name must be large enough to accomodate 13 bytes.

ROUTINE: GDIR (Continued)

Notes: (Continued)

(5) 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

--- ROUTINE: INIT

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 17_8 locations of normally relocatable memory.

Entry: INIT

Function: Causes a device to be initialized.

Calling Sequence:

```
FCALL
INIT
Integer 3
FORTRAN ADDRESS of device name
FORTRAN ADDRESS of mode indicator
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) INIT must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .INIT .

(4) The mode indicator is one of the following values:

1 - Partial initialization with overlays
0 - Partial initialization
-1 - Full initialization

(5) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3 ... n - RDOS system error code + 3

(6) 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.

(7) The device name is an ASCII string consisting of a valid string mnemonic for either a disk or magnetic tape transport and is terminated by a null byte.

---ROUTINE: MDIR

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: MDIR

Function: Obtains the logical name of the current master device.

Calling Sequence:

```
FCALL
MDIR
Integer 2
FORTRAN ADDRESS of array to receive name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) MDIR must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .MDIR .

(4) 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
```

--- ROUTINE: RENAM

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 20_8 locations of normally relocatable memory.

Entry: RENAM

Function: Renames a disk file.

Calling Sequence:

```
FCALL
RENAM
Integer 3
FORTRAN ADDRESS of old name string
FORTRAN ADDRESS of new name string
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

ROUTINE: RENAM (Continued)

Notes: (Continued)

(2) RENAM must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .RENAM .

(4) 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

(5) Each name string is an ASCII byte string terminated by a carriage return, null, form feed, or space.

--- ROUTINE: RLSE

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: RLSE

Function: Releases 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
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) RLSE must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .RLSE .

(4) The device name is an ASCII byte string terminated by a carriage return, null, form feed, or space.

(5) 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

OVERLAY ROUTINES

--- ROUTINE: FOVLD

Supporting Subroutines: FRET; .CPYA, .IOCAT

Subroutine Size: No page zero locations and 46_8 locations of normally relocatable memory.

Entry: FOVLD

Function: Loads a FORTRAN overlay into an overlay area in a single task environment.

Calling Sequence:

```
FCALL
FOVLD (or OVLOD)
Integer 4
FORTRAN ADDRESS of channel number on which overlay has been
opened.
Overlay number
FORTRAN ADDRESS of conditional load flag
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) OVLOD and FOVLD are equivalent and must be referenced by .EXTN statements.

(3) The overlay file which is specified must have been previously opened by a call to OVOPN. The overlay number parameter contains the overlay area number in its left byte and the overlay number in its right byte. This number must have been declared either by an .ENTO pseudo-op or in a FORTRAN OVERLAY statement.

(4) If the conditional load flag is zero, overlay loading is to be done unconditionally; if non-zero, overlay loading is to be done conditionally. Refer to Chapter 4.

(5) 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
```

--- ROUTINE: FOVLY

Supporting Subroutines: FRET, TOVLD, TOVRL; .CPYL, .IOCA

Subroutine Size: 45_8 locations of normally relocatable memory.

Entry: FOVLD

Function: Loads a FORTRAN overlay into an overlay area in a multitask environment.

ROUTINE: FOVLY (Continued)

Entry: FOVLD (Continued)

Calling Sequence:

```
FCALL
FOVLD
Integer 4
FORTRAN ADDRESS of the channel number upon which the overlay
file has been opened
Overlay number (overlay name may be used if previously declared in
an .ENTO or OVERLAY statement)
FORTRAN ADDRESS of the conditional load flag
FORTRAN ADDRESS of the error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) The overlay file which is to be used 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 in its right byte. This number must have been declared in .ENTO or OVERLAY statement.

(3) The conditional load flag is a word which is set to zero if overlay loading is to be done unconditionally; a nonzero value indicates that overlay loading is to be done conditionally.

(4) 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.

(5) In unconditional loading, if an area is free the requested overlay is loaded regardless of whether it is currently core resident or not. If the area is not free, the caller is suspended until the area is released. Consult the RDOS User's Manual for more information about conditional and unconditional loading.

(6) 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
```

(7) 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.

Entry: FOVRL

Function: To release an overlay area.

ROUTINE: FOVLY (Continued)

Entry: FOVRL (Continued)

Calling Sequence:

```
FCALL
FOVRL
Integer 2
Overlay number (overlay name may be used if previously declared in
an .ENTO or OVERLAY statement)
FORTRAN ADDRESS of the error code
```

Notes:

(1) Accumulators and carry are restored upon exit.

(2) Same as (2) for FOVLD.

(3) This call should be issued each time a user completes his use of a given overlay, in order to decrement the overlay use count. When no users remain who wish to use the currently resident overlay, the overlay use count goes to zero and the overlay area becomes free for the loading of other overlays.

(4) This call must not be issued from within the overlay area which is to be released.

(5) 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
```

--- ROUTINE: OVEXT

Supporting Subroutines: FRET, TOVRL; .CPYL, .RTER

Subroutine Size: No page zero locations and 22_8 locations of normally relocatable memory.

Entry: OVEXT

Function: Releases an overlay from the routine in which the overlay is named in an OVERLAY statement and provides a return location.

Calling Sequence:

```
FCALL
OVEXT
Integer 2
FORTRAN ADDRESS of Overlay Name
FORTRAN ADDRESS of Return
```

Notes:

OVEXT must be referenced by an .EXTN statement.

ROUTINE: OVEXT (Continued)

Entry: OVEXX

Function: Releases an overlay from a routine outside that in which the overlay was named in an OVERLAY statement and provides a return location.

Calling Sequence:

```
FCALL
OVEXX
Integer 2
FORTRAN ADDRESS of Overlay Name
FORTRAN ADDRESS of Return
```

Notes:

OVEXX must be referenced by an .EXTN statement.

--- ROUTINE: OVKIL

Supporting Subroutines: FRET, KILL, TOVRL; .CPYL, .RTER

Subroutine Size: No page zero locations and 20_8 locations of normally relocatable memory.

Entry: OVKIL

Function: Releases an overlay from within the routine in which the overlay was named in an OVERLAY statement; also causes the task containing the overlay to be killed.

Calling Sequence:

```
FCALL
OVKIL
Integer 1
FORTRAN ADDRESS of Overlay Name

(Control returns to the FORTRAN Task Scheduler)
```

Notes:

(1) OVKIL causes the task issuing the call to be killed.

(2) OVKIL must be referenced by an .EXTN statement.

Entry: OVKIX

Function: Releases an overlay from a routine outside that in which the overlay was named in an OVERLAY statement; also causes the task containing the overlay to be killed.

ROUTINE: OVKIL (Continued)

Entry: OVKIX (Continued)

Calling Sequence:

```
FCALL
OVKIX
Integer 1
FORTRAN ADDRESS of Overlay Name

(Control returns to the FORTRAN Task Scheduler)
```

Notes:

OVKIX must be referenced by an .EXTN statement.

TASK ROUTINES

--- ROUTINE: ABORT

Supporting Subroutines: FRET, KTID; .CPYL

Subroutine Size: 15_8 locations of normally relocatable memory

Entry: ABORT

Function: 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
```

Notes:

(1) Accumulators and carry are saved in the caller's TCB unless it is the caller who is killed.

(2) ABORT must be referenced in an .EXTN statement.

(3) The calling task itself may be killed by this call.

(4) The TCB which is removed from the active queue is placed in the free element TCB chain. If the specified task is suspended due to an outstanding .SYSTM call, it is killed as soon as the .SYSTM call is completed.

(5) If no task exists with the specified i.d. number, no action is taken and control goes to the scheduler.

--- ROUTINE: ASSOC

Supporting Subroutines: CTASK, FRET, FRCAL, TPEND; .CPYL

Subroutine Size: No page zero locations and 65_8 locations of normally relocatable memory.

Entry: ASSOC

Function: Associates a unique i.d. with a FORTRAN task, without activating the task.

Calling Sequence:

```
FCALL
ASSOC
Integer 4 or 5
FORTRAN ADDRESS of task name
FORTRAN ADDRESS of i.d.
FORTRAN ADDRESS of priority
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of no-stack flag
```

ROUTINE: ASSOC (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) ASSOC must be referenced by an .EXTN statement.

(3) The specified task is associated with the specified i.d. and put into the suspended state.

--- ROUTINE: CHNGE

Supporting Subroutines: FRET, TCHNG; .CPYL

Subroutine Size: 16_8 locations of normally relocatable memory.

Entry: CHNGE

Function: Changes the priority of a task specified by i.d.

Calling Sequence:

```
FCALL
CHNGE
Integer 3
FORTRAN ADDRESS of i.d.
FORTRAN ADDRESS of new priority
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) CHNGE must be referenced by an .EXTN statement.

(3) The error code is set to one of the following states:

```
   0  - Indeterminate error
   1  - No error occurred
3... n - RDOS system error code + 3
```

--- ROUTINE: FACAL

Supporting Subroutines: FRET, TAKIL, TAPEN, TAUNP; .CPYL

Subroutine Size: 22_8 locations of normally relocatable memory.

Entry: ASUSP

Function: Suspends all tasks of a given priority.

ROUTINE: FACAL (Continued)

Entry: ASUSP (Continued)

Calling Sequence:

FCALL
ASUSP
Integer 1
FORTRAN ADDRESS of the task priority

(Control returns to the FORTRAN Task Scheduler.)

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) ASUSP must be referenced by an .EXTN statement.

(3) The calling task may itself be suspended by this command.

(4) The suspended tasks can be readied only by an ARDY command.

(5) If no tasks exist at the given priority, this call is an effective no-op.

Entry: ARDY

Function: Readies 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.)

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) ARDY must be referenced by an .EXTN statement.

(3) This command unconditionally readies all tasks of a 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, such as I/O completion.

Entry: AKILL

Function: Deletes 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.)

ROUTINE: FACAL (Continued)

Notes:

(1) Accumulators and carry are saved in the caller's TCB (unless the caller is also deleted).

(2) AKILL must be referenced by an .EXTN statement.

(3) The calling task itself may be deleted by this command.

(4) All TCB's that are removed from the active queue are placed in the free element TCB chain.

(5) If a task to be deleted is already suspended (as when the task is waiting for completion of a system call) it will be killed as soon as it becomes ready.

(6) If no task exists at the given priority level, this call is an effective no-op and control goes to the Scheduler.

--- ROUTINE: FDELY

Supporting Subroutines: FRET; .CPYL

Subroutine Size: 1 page zero location and 7 normally relocatable locations.

Entry: FDELY

Function: Suspends 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.)
```

Notes:

(1) Accumulators and carry are stored in the caller's TCB.

(2) FDELY must be referenced by an .EXTN statement.

(3) The 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.)

--- ROUTINE: FPEND

Supporting Subroutines: FRET, TPEND; .CPYL

Subroutine Size: 5 locations of normally relocatable memory.

Entry: SUSP

Function: Suspends the calling task.

Calling Sequence:

```
FCALL
SUSP
0
```

(Control returns to the FORTRAN Task Scheduler.)

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) SUSP must be referenced by an .EXTN statement.

(3) The suspended task remains suspended until it is readied by an ARDY or RELSE call.

Entry: PEND

Equivalent to FPEND

--- ROUTINE: FPRI

Supporting Subroutines: FRET, TPRI; .CPYL

Subroutine Size: 6 words of normally relocatable memory.

Entry: PRI

Function: 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.)

ROUTINE: FPRI (Continued)

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) PRI must be referenced by an .EXTN statement.

(3) The calling task is assigned the lowest priority of all tasks within the new priority level.

(4) 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 its priority level.

--- ROUTINE: FQTAS

Supporting Subroutines: FRET, TQTSK; .CPYL

Subroutine Size: 51_8 locations of normally relocatable memory.

Entry: FQTAS

Function: Loads a user overlay and periodically executes a task within the overlay, or periodically executes a core-resident task.

Calling Sequence:

```
FCALL
FQTAS
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 task descriptor

(Control returns to the FORTRAN Task Scheduler.)
```

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) 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.

(3) 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.

ROUTINE: FQTAS (Continued)

Notes: (Continued)

(4) The task queue array is a 13_8 word integer array, supplied by a 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 hour QSH
QPRI	Task priority
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

(5) 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

(6) The last parameter (task descriptor) is optional. A -1 indicates that the task is core resident and no overlay load is necessary. -2 indicates that no FORTRAN Run Time Stack is necessary. -3 indicates both that the task is core resident and that no stack is needed.

(7) FQTASK must be referenced in an .EXTN statement.

--- ROUTINE: FTASK

Supporting Subroutines: CTASK, FRET; .CPYL

Subroutine Size: 26_8 locations of normally relocatable memory.

Entry: FTASK

Function: Initiates a task in a real-time FORTRAN environment.

Calling Sequence:

```
FCALL
FTASK
Integer 3 or 4
FORTRAN ADDRESS of Task Entry Point
FORTRAN ADDRESS of Error Return
FORTRAN ADDRESS of Task Priority
optional FORTRAN ADDRESS of no-stack flag

(Control returns to the FORTRAN Task Scheduler.)
```

ROUTINE: FTASK (Continued)

Notes:

(1) Accumulators and carry are stored in the caller's TCB.

(2) FTASK must be referenced by an .EXTN statement.

(3) When the RT FORTRAN program is loaded and first run, only one task exists. This command must be issued to create a multitask environment.

(4) The error return is taken if there are no TCBs available, which occurs if the maximum number of tasks specified in CHANTASK was too small.

(5) A non-zero argument for the optional no-stack flag indicates that no FORTRAN stack is required.

--- ROUTINE: HOLD

Supporting Subroutines: FRET, STID; .CPYL

Subroutine Size: 15_8 locations of normally relocatable memory.

Entry: HOLD

Function: To suspend a task with a specified i.d. number.

Calling Sequence:

```
FCALL
HOLD
Integer 2
FORTRAN ADDRESS of i.d. number
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) HOLD must be referenced by an .EXTN statement.

(3) This call sets bit 1 of the task's priority and status word, TPRST. Thus if the task is already suspended, it becomes doubly suspended and can be readied only when all its suspend bits have been set to ready.

(4) 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
```

--- ROUTINE: IOPC

Supporting Subroutines: FRET, .CPYL, PCTMP, TERCM, TIOPC, TMAX2, TSAVE

Subroutine Size: 134_8 locations of normally relocatable memory.

Entry: IOPC

Function: To initialize the OPCOM package.

Calling Sequence:

```
FCALL
IOPC
FORTRAN address of program array
FORTRAN address of number of programs
FORTRAN address of queue array
FORTRAN address of number of queues
FORTRAN address of overlay channel
FORTRAN address of error
```

Notes:

(1) One TCB must be reserved for the OPCOM package.

(2) When running without a program table (IOPROG) set the first five arguments to zero.

(3) IOPC may be called more than once. Additional calls will remove the previous program array. A new array must be given for each call.

Entry: IOPROG

Function: To build a program table of task information for reference by the OPCOM commands RUN and QUE.

Calling Sequence:

```
FCALL
IOPROG
FORTRAN address of program name
FORTRAN address of program number
FORTRAN address of task identifier
FORTRAN address of task priority
FORTRAN address of overlay node/number
FORTRAN address of conditional load
FORTRAN address of error
FORTRAN address of ASM
```

--- ROUTINE: ITASK

Supporting Subroutines: CTASK, FRET; .CPYL

Subroutine Size: 36_8 locations of normally relocatable memory

Entry: ITASK

Function: Initializes a task in a real-time FORTRAN environment and assigns a unique i.d. to the task.

Calling Sequence:

```
FCALL
ITASK
Integer 4 or 5
FORTRAN ADDRESS of Task Entry Point
FORTRAN ADDRESS of Task I. D.
FORTRAN ADDRESS of Task Priority
FORTRAN ADDRESS of Error Code
optional FORTRAN ADDRESS of no-stack flag
```

(Control returns to the FORTRAN Task Scheduler.)

Notes:

(1) Accumulators and carry are stored in the caller's TCB.

(2) ITASK must be referenced in an .EXTN statement.

(3) A non-zero argument for the optional no-stack flag indicates that no FORTRAN stack is required.

(4) 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
```

--- ROUTINE: RELSE

Supporting Subroutines: FRET, RTID; .CPYL

Subroutine Size: 15_8 locations of normally relocatable memory.

Entry: RELSE

Function: To ready a task with a specified i.d. number.

Calling Sequence:

```
FCALL
RELSE
Integer 2
FORTRAN ADDRESS of i.d. number
FORTRAN ADDRESS of error code
```

ROUTINE: RELSE (Continued)

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) RELSE must be referenced in an .EXTN statement.

(3) This call resets bit 1 of the task's priority and status field word, TPRST. If the task has bits 0 and/or 12 set as the result of an outstanding .SYSTM call or a REC/XMTW, these bits would also have to be reset before the task could be readied.

(4) 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

--- ROUTINE: START

Supporting Subroutines: FRET, DVD, MPY; .CPYL

Subroutine Size: No page zero locations and 114_8 locations of normally relocatable memory.

Entry: START

Function: Starts a task after a specified delay.

Calling Sequence:

```
FCALL
START
Integer 4
FORTRAN ADDRESS of task i.d.
FORTRAN ADDRESS of delay
FORTRAN ADDRESS of unit code
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) START must be referenced by an .EXTN statement.

(3) The delay is an integer variable or constant which specifies the length of time of the delay in units.

(4) The unit code is one of the following:

0 - pulse of RTC
1 - milliseconds
2 - seconds
3 - minutes

(5) The error code is set to zero on input of an illegal unit code or if there is no clock in the system.

--- ROUTINE: STTSK

Supporting Subroutines: FRET, TIDST; .CPYL

Subroutine Size: 11_8 locations of normally relocatable memory.

Entry: STTSK

Function: To obtain the status of a task with a specified 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
```

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) SSTSK must be referenced in an .EXTN statement.

(3) 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 - Codes 1 and 2 both apply
6 - Code 2 and also suspended by XMTW or REC
7 - Codes 2 and 4 both apply
8 - No task exists with this i.d. number

(4) 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

--- ROUTINE: TRNON

Supporting Subroutines: DVD, MPY, FRET, RTID, TIDSR; .CPYL

Subroutine Size: No page zero locations and 141_8 locations of normally relocatable memory.

Entry: TRNON

Function: Executes a task at a specified time of day.

Calling Sequence:

```
FCALL
TRNON
Integer 3
FORTRAN ADDRESS of task i.d.
FORTRAN ADDRESS of time array
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) TRNON must be referenced by an .EXTN statement.

(3) The time array consists of three words. The first word specifies hours, the second specifies minutes, and the third specifies seconds of the time of day for task execution.

(4) If the specified time is negative, the task is started immediately.

(5) The error code is set to zero if an illegal time is specified or there is no clock in the system.

FOREGROUND/BACKGROUND ROUTINES

Foreground/
Background

---ROUTINE: EXBG

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 22_8 locations of normally relocatable memory.

Entry: EXBG

Function: Loads and executes a user program in the background.

Calling Sequence:

```
FCALL
EXBG
Integer 3
FORTRAN ADDRESS of file name to be executed
FORTRAN ADDRESS of priority
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) EXBG must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .EXBG .

(4) The priority is 0 if the background has a lower priority than the foreground and 1 if they have the same priority.

(5) The error code will be set to one of the following states:

```
  0 - Indeterminate error
  1 - No error occured
3...n - RDOS system error code + 3
```

--- ROUTINE: EXFG

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 21_8 locations of normally relocatable memory.

Entry: EXFG

Function: Loads and executes a user program in the foreground.

Calling Sequence:

```
FCALL
EXFG
Integer 3
FORTRAN ADDRESS of file name to be executed
FORTRAN ADDRESS of priority
FORTRAN ADDRESS of error code
```

ROUTINE: EXFG (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) EXFG must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .EXFG .

(4) The priority is 0 if the foreground has a higher priority than the background and 1 if they have the same priority.

(5) This call can only be made from a background program.

(6) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3...n - RDOS system error code + 3

--- ROUTINE: FGND

Supporting Subroutines: FRET; CPYL

Subroutine Size: No page zero locations and 7 locations of normally relocatable memory.

Entry: FGND

Function: Determines whether or not a foreground program is running in the system.

Calling Sequence:

```
FCALL
FGND
Integer 1
FORTRAN ADDRESS of result
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) FGND must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .FGND .

(4) The result is 1 if a foreground program is executing and 0 if a foreground program is not executing.

(5) No error messages are generated.

COMMUNICATION ROUTINES

ROUTINE: FXMT

Supporting Subroutines: FRET, RECC, XMTT, XMTTW; .CPYL

Subroutine Size: 33_8 locations of normally relocatable memory

Entry: XMT

Function: Transmits a one word message to a receiving task, then remains ready to resume other task activity.

Calling Sequence:

```
FCALL
XMT
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)
```

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) XMT must be referenced by an .EXTN statement.

(3) A one word message is placed 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.

(4) The error return is taken if the message address is already in use, which is indicated by non-zero contents.

Entry: XMTW

Function: Transmits a one word message and waits, staying suspended until the message is received.

Calling Sequence:

```
FCALL
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.)
```

ROUTINE: FXMT (Continued)

Notes:

(1) XMTW must be referenced by an .EXTN statement.

(2) Notes (1), (3), and (4) for XMT also apply to XMTW.

Entry: REC

Function: 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.)
```

Notes:

(1) Accumulators and carry are saved in the caller's TCB.

(2) REC must be referenced by an .EXTN statement.

(3) 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 is 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 ready.

---ROUTINE: ICMN

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: ICMN

Function: Defines an array area for use in sending or receiving messages between user programs.

Calling Sequence:

```
FCALL
ICMN
Integer 3
FORTRAN ADDRESS of array pointer
FORTRAN ADDRESS of length specifier (# of words)
FORTRAN ADDRESS of error code
```

ROUTINE: ICMN (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) ICMN must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .ICMN .

(4) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3...n - RDOS system error code + 3

--- ROUTINE: RDOPR

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 22_8 locations of normally relocatable memory.

Entry: RDOPR

Function: Causes transmission of an operator message to either the foreground or background program.

Calling Sequence:

```
FCALL
RDOPR
Integer 3
FORTRAN ADDRESS of array
FORTRAN ADDRESS of byte specifier
FORTRAN ADDRESS of error code.
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) RDOPR must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .RDOP .

(4) The array in the calling sequence is an array large enough to handle 132 characters, including a carriage return terminator.

(5) The byte specifier is set to the number of bytes transferred, including the terminating character. On an error, it is set to 0.

(6) The first character in the message transmitted from the system console must be a CTRL E (which is echoed as an exclamation point) followed by either F or B. F indicates that a foreground program is to receive the message, and B indicates that the background is the receiver. If some character other than

ROUTINE: RDOPR (Continued)

Notes: (Continued)

(6) (Continued)

F or B is typed, no further text string will be accepted until an F or B is typed. If the user tries to transmit a message for which there is no outstanding read operator message call, the bell sounds when CTRL E is depressed. The last character must be the carriage return and the entire message string can be up to 132 characters.

(7) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3...n - RDOS system error code + 3

--- ROUTINE: RWCMN

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 30_8 locations of normally relocatable memory.

Entry: RDCMN

Function: Causes the calling program to read a message from another program's communications area.

Calling Sequence:

```
FCALL
RDCMN
Integer 4
FORTRAN ADDRESS of area to receive message
FORTRAN ADDRESS of word offset
FORTRAN ADDRESS of number of words to be read
FORTRAN ADDRESS of error code
```

Entry: WRCMN

Function: Causes the calling program to write a message in another program's communications area.

Calling Sequence:

```
FCALL
WRCMN
Integer 4
FORTRAN ADDRESS of start of message
FORTRAN ADDRESS of word offset
FORTRAN ADDRESS of number of words to write
FORTRAN ADDRESS of error code
```

ROUTINE: RWCMN (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) RDCMN and WRCMN must be referenced by .EXTN statements.

(3) The program issuing a call to RDCMN or WRCMN may be executing in either the foreground or background.

(4) The word offset in the calling sequence specifies the offset of the message to be read or written within the communication area.

(5) The message that is to be sent or received may originate or be written anywhere within the calling program's address space.

(6) This routine issues the RDOS system call .RDCMN .

(7) 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

--- ROUTINE: WROPR

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 20_8 locations of normally relocatable memory.

Entry: WROPR

Function: Writes an operator message.

Calling Sequence:

```
FCALL
WROPR
Integer 2
FORTRAN ADDRESS of message array
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) WROPR must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .WROP .

(4) An ouput string is written from either the foreground or background areas to the system console. The message consists of an ASCII string less than or equal to 129 characters in length, including the required carriage return

ROUTINE: WROPR (Continued)

Notes: (Continued)

(4) (Continued)

terminator. The system will prefix two exclamation characters and either an F for foreground or B for background to indicate where the message originated.

(5) 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

INTERRUPT ROUTINES

--- ROUTINE: FCNS

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 14_8 locations of normally relocatable memory.

Entry: OEBL

Function: Enables console interrupts CTRL A, CTRL C, and CTRL F.

Calling Sequence:

```
FCALL
OEBL
```

Entry: ODIS

Function: Disables console interrupts CTRL A, CTRL C, and CTRL F.

Calling Sequence:

```
FCALL
ODIS
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) OEBL and ODIS must both be referenced by .EXTN statements.

(3) This routine issues the RDOS system calls .ODIS and .OEBL..

(4) No error messages are generated.

(5) Console interrupts CTRL A, CTRL C, and CTRL F are enabled when a system is first bootstrapped. A call to ODIS is used to disable the interrupts, a call to OEBL reenables them within its program environment.

--- ROUTINE: FINTD

Supporting Subroutines: FRET; .CPYL, .RTER

Subroutine Size: 2 page zero locations and 21_8 locations of normally relocatable memory.

Entry: FINTD

Function: Introduces to the system a non-SYSGENed device capable of generating interrupt requests.

ROUTINE: FINTD (Continued)

Entry: FINTD (Continued)

Calling Sequence:

```
FCALL
FINTD
Integer 2
FORTRAN ADDRESS of the device code
FORTRAN ADDRESS of the three word DCT
```

Notes:

(1) Accumulators and carry are saved.

(2) FINTD must be referenced in an .EXTN statement.

(3) This call causes an entry for this device to be placed in the system interrupt vector table.

(4) A system error code ERDNM is output if an illegal device code is given, and return is made to the CLI.

Entry: FINRV

Function: Removes 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
```

Notes:

(1) Accumulators and carry are saved.

(2) FINRV must be referenced by an .EXTN statement.

(3) This call removes the device entry from the system interrupt vector table.

(4) A system error code ERDNM is output if an illegal device code is given, and return is made to the CLI.

--- ROUTINE: IXMT

Supporting Subroutines: FRET, IXMTT; .CPYL

Subroutine Size: No page zero locations and 15_8 locations of normally relocatable memory.

Entry: IXMT

Function: Transmits 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
```

Notes:

(1) Accumulators and carry are restored upon exit.

(2) Return is to the caller, not to the task scheduler.

(3) This routine is issued only in a user interrupt service routine, outside the multitasking environment.

(4) IXMT must be referenced by an .EXTN statement.

(5) 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
```

FILE MAINTENANCE ROUTINES

--- ROUTINE: CFILW

Supporting Subroutines: FRET; .CPYL, .RTER

Subroutine Size: No page zero locations and 46_8 locations of normally relocatable memory.

Entry: CFILW

Function: Creates an RDOS disk file.

Calling Sequence:

```
FCALL
CFILW
Integer number of arguments, 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
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) The file name is an ASCII byte string.

(3) The file type indicator is 1 for a sequentially organized file, 2 for a randomly organized file, and 3 for a contiguously organized file.

(4) 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.

(5) 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
```

(6) A call to DELETE is equivalent to a call to DFILW.

--- ROUTINE: CHLAT

Supporting Subroutines: FRET; .CPYL, .IOCAT

Subroutine Size: No page zero locations and 26_8 locations of normally relocatable memory.

Entry: CHLAT

Function: Changes, adds or deletes link file access attributes of a file on a specified channel.

Calling Sequence:

```
FCALL
CHLAT
Integer 3
FORTRAN ADDRESS of channel number
FORTRAN ADDRESS of attributes specifier
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) CHLAT must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .CHLAT.

(4) The attributes specifier is a word with the following bit representations:

Bit	Meaning
0	Read-protected file
1	Attribute-protected file
2	Save file
3	Link entry
4	Partition
5	Directory file
6	Link resolution (temporary). Some or all of the other attributes persist for the duration of the open.
7	No link resolution allowed
8	Accessible by direct block I/O only
12	Contiguous file
13	Random file
14	Permanent file
15	Write-protected file

(5) 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
```

--- ROUTINE: CLOSE

Supporting Subroutines: FRET, FSAV; .IOCAT, .RTER, .CPYL, .SOSW

Subroutine Size: No page zero locations and 47_8 locations of normally relocatable memory.

Entry: CLOSE

Function: Frees a FORTRAN logical channel and closes the file associated with that channel. (under RDOS)

Calling Sequence:

```
FCALL
CLOSE
Integer 2
FORTRAN ADDRESS of logical channel number
FORTRAN ADDRESS of error code
```

Entry: FCLOS

Function: Frees a FORTRAN logical channel and closes the file associated with that channel.

Calling Sequence:

```
FCALL
FCLOS
Integer 1
FORTRAN ADDRESS of logical channel number
```

Notes:

(1) Contents of accumulators and carry are restored on exit.

(2) CLOSE and FCLOS must be referenced by .EXTN statements.

(3) The logical channel number is an integer constant with a value between 0 and 63_{10}.

(4) CLOSE issues the RDOS system call .CLOSE .

(5) 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
```

--- ROUTINE: DFILW

Supporting Subroutines: FRET; .CPYL, .RTER

Subroutine Size: No page zero locations and 27_8 locations of normally relocatable memory.

Entry: DFILW

Function: Deletes a disk file.

Calling Sequence:

```
FCALL
DFILW
Integer number of arguments, 1 or 2
FORTRAN ADDRESS of file name
optional FORTRAN ADDRESS of error code
```

Notes:

(1) Contents of accumulators and carry are restored on exit.

(2) DFILW must be referenced by an .EXTN statement.

(3) DELET is equivalent to DFILW.

(4) The file name is an ASCII byte string.

(5) This routine issues the RDOS system call .DELET .

(6) 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
```

--- ROUTINE: DLINK

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 20_8 locations of normally relocatable memory.

Entry: DLINK

Function: Creates a link entry in the current directory to a file in another directory.

Calling Sequence:

```
FCALL
DLINK
Integer 3
FORTRAN ADDRESS of name of link entry
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) DLINK must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .LINK .

(4) The error code is set to one of the following states:

```
  0 - Indeterminate error
  1 - No error occurred
3...n - RDOS system error code + 3
```

--- ROUTINE: FFILE

Supporting Subroutines: FSAV, FRET, FCLOS, FSEEK, FCALL, IOPTR; .CPYARG, .RTER, .IOCAT, .FCALL

Subroutine Size: 1 page zero location and 64_8 locations of normally relocatable memory.

Entry: .FFILE

Function: Positions a sequential file which has been assigned a FORTRAN channel number.

Calling Sequence:

```
JSR @.FFIL
File positioning code
FOFTRAN ADDRESS of FORTRAN channel number
```

(File positioning codes are 1 for position the file at its initial record and 2 for close the file associated with this channel.)

ROUTINE: FFILE (Continued)

Notes:

(1) Accumulators and carry are restored upon exit.

(2) .FFIL must be referenced by an .EXTD statement. The FCALL entry FFIL must be referenced by an .EXTN statement.

(3) I/O error conditions and unopened files will cause error messages to be generated.

--- ROUTINE: FSEEK

Supporting Subroutines: FRET, MPY; .CPYL, .IOCAT, .RTER

Subroutine Size: No page zero locations and 51_8 locations of normally relocatable memory.

Entry: FSEEK

Function. Accesses a particular record on a random access file.

Calling Sequence:

```
JSR @.FCALL    (or FCALL)
FSEEK
Integer 2
FORTRAN ADDRESS of FORTRAN logical channel number
FORTRAN ADDRESS of the record number to be accessed
```

Notes:

(1) Contents of accumulators and carry are restored upon exit.

(2) FSEEK must be referenced by an .EXTN statement.

(3) When more than one record is to be written or read without intervening calls to FSEEK, records will be written or read sequentially.

(4) The file with the given channel number is positioned at the first byte of the first random record whose length was specified by FOPEN.

(5) A run time error is given if the file is not randomly organized or if the file is not open.

--- ROUTINE: FSTAT

Supporting Subroutines: FRET; .CPYL, .IOCA

Subroutine Size: No page zero locations and 26_8 locations of normally relocatable memory.

Entry: FSTAT

Function: Sets 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.
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) This routine issues the RDOS system call .CHRAT . A 1 in the specified bit position of the attributes word sets the given attribute for the 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.
```

(3) 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
```

--- ROUTINE: FSWAP

Supporting Subroutines: FERT0, FERTN; .CPYL, .FRET, .RTER

Subroutine Size: No page zero locations and 43_8 locations of normally relocatable memory.

Entry: FSWAP

Function: Saves the current core image as a disk save file and reads in a new save file at a lower program level.

Calling Sequence:

```
FCALL
FSWAP
Integer 1
FORTRAN ADDRESS of the save file name
```

ROUTINE: FSWAP (Continued)

Entry: FCHAN

Function: Performs a program chain; a new save file is read from disk, overwriting the current core image without changing program levels.

Calling Sequence:

```
FCALL
FCHAN
Integer 1
FORTRAN ADDRESS of the save file name
```

Entry: FBACK

Function: Reads in from disk the next higher program level swap.

Calling Sequence:

```
FCALL
FBACK
```

Notes:

(1) In a call to FSWAP 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 a call to FBACK.

(2) The calling program is overwritten and accumulators and carry are lost in calls to FBACK and FCHAN. Data can be passed via blank common, since blank common is not overwritten during program swapping or chaining.

(3) When the new save file is read into core, control goes to the highest priority ready task within it.

--- ROUTINE: GTATR

Supporting Subroutines: FRET; .CPYL, .IOCAT

Subroutine Size: No page zero locations and 26_8 locations of normally relocatable memory.

Entry: GTATR

Function: Gets 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
```

ROUTINE: GTATR (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) This routine issues the RDOS system call .GTATR . A 1 in the specified bit position of the attributes word indicates that 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.

(3) 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

--- ROUTINE: OPEN

Supporting Subroutines: FRET, IOPTR; .CPYL, .IOCAT , .SOSW

Subroutine Size: No page zero locations and 147_8 locations of normally relocatable memory.

Entry: OPEN

Function: Opens 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 mode or open array
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of blocking factor

Entry: OVOPN

Function: Opens 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

ROUTINE: OPEN (Continued)

Entry: APPEND

Function: Opens 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 mode or open array
FORTRAN ADDRESS of error code
optional FORTRAN ADDRESS of blocking factor
```

Entry: MTOPD

Function: Opens a magnetic or cassette tape device for direct access with the given device mask.

Calling Sequence:

```
FCALL
MTOPD
Integer number of arguments, 4
FORTRAN ADDRESS of FORTRAN channel number
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of device characteristic mask
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored upon exit.

(2) OPEN, OVOPN, MTOPD, APPEND must be referenced by .EXTN statements

(3) The file name is an ASCII byte string, including the file .OL extension for a call to OVOPN.

(4) OVOPN issues the RDOS system call .OVOPN and 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.

(5) The open mode or array is used to indicate the type of open preferred on the given file and optionally the device characteristic mask. A call to MTOPD must give the device characteristic mask. The alternate settings of the argument are given below:

open mode: an integer constant or variable

```
1                   - open for reading only
3                   - open for writing by one user but reading by one or more users
other than 1 or 3   - open for user-shared reading and writing
```

Notes: (Continued)

open array: 3-element integer array

First element - contains -1 (the array flag)
Second element - contains the open mode given above (1, 3, or other than 1 or 3)
Third element - contains the device characteristic mask

The bit/characteristic correspondence used in setting the device characteristic mask is:

Bit	Meaning
1B0	Spooling enabled (0B0 is spooling disabled)*
1B1	80-column device
1B2	device changing lower case ASCII to upper case
1B3	device requiring form feeds on opening
1B4	full word device (reads or writes more than a byte)
1B5	Spoolable device
1B6	output device requiring line feeds after carriage returns
1B7	input device requiring a parity check; output device requiring parity to be computed
1B8	output device requiring a rubout after every tab
1B9	output device requiring nulls after every form feed
1B10	a keyboard input device
1B11	a teletype output device
1B12	output device without form feed hardware
1B13	device requiring operator intervention
1B14	output device requiring tabbing hardware
1B15	output device requiring leader/trailer

(6) The error code will be set to one of the following:

0 - Indeterminate error
1 - No error occurred
3... n - RDOS system error code +3

(7) The blocking factor constant is an integer indicating the number of bytes/record. For random record I/O, the blocking factor should be 128.

(8) Up to 64 FORTRAN channel numbers are allowed, 0 through 63.

(9) If the file does not currently exist on an OPEN or APPEND call, a random disk file will be created or opened.

--- ROUTINE: RESET

Supporting Subroutines: FRET; .IOCAT

Subroutine Size: No page zero locations and 21_8 locations of normally relocatable memory.

Entry: RESET

Function: Closes all currently open files and all FORTRAN channels.

Entry: RESET(Continued)

Calling Sequence:

```
FCALL
RESET
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) 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.

--- ROUTINE: STAT

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 17_8 locations of normally relocatable memory.

Entry: STAT

Function: Obtains status information about the current file directory.

Calling Sequence:

```
FCALL
STAT
Integer 3
FORTRAN ADDRESS of file name
FORTRAN ADDRESS of array to receive status
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) STAT must be referenced by an .EXTN statement.

ROUTINE: STAT (Continued)

Notes: (Continued)

(3) The array to receive the file status must be 22_8 words in length and is supplied with the following data:

Word	Contents
1 - 5	File name
6	Extension
7	File Attributes
10	Link access attributes
11	Number of the last block in the file
12	Number of bytes in the last block
13	Starting logical block address of the file (the random file index for random files)
14	Year/day last accessed
15	Year/day created
16	Hour/minute created
17	UFD temporary
20	UFD temporary
21	User count
22	DCT link

(4) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error code occurred
3... n - RDOS system error +3

(5) This routine issues the RDOS system call .STAT.

--- ROUTINE: UPDATE

Supporting Subroutines: FRET; .CPYL, .IOCAT

Subroutine Size: No page zero locations and 25_8 locations of normally relocatable memory.

Entry: UPDATE

Function: Updates the file size of the named file.

Calling Sequence:

```
FCALL
UPDATE
Integer 2
FORTRAN ADDRESS of channel number
FORTRAN ADDRESS of error code
```

ROUTINE: UPDATE (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) UPDATE must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .UPDAT .

(4) The error code is set to one of the following states:

0 - Indeterminate error
1 - No error occurred
2 - Channel not open
3... n - RDOS system error code +3

CLOCK ROUTINES

--- ROUTINE: DATE

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: DATE

Function: Gets the current day of the year.

Calling Sequence:

```
FCALL
DATE
Integer 2
FORTRAN ADDRESS of date array
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and Carry are restored upon exit.

(2) This routine issues the RDOS system call .GDAY. The date is stored in an integer array of 3 words as follows: Month - element 1, Day - element 2, Year - element 3; and is returned on execution of the call. The year is returned as a two-digit number, i.e., 74, 75, etc.

(3) The error code word will be set to one of the following states:
0 - Indeterminate error
1 - No error occurred

--- ROUTINE: DUCLK

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: DUCLK

Function: Defines a user clock.

Calling Sequence:

```
FCALL
DUCLK
Integer 3
FORTRAN ADDRESS of number of ticks between interrupts
FORTRAN ADDRESS of interrupt response code
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) DUCLK must be referenced by an .EXTN statement.

ROUTINE: DUCLK (Continued)

Notes: (Continued)

(3) This routine issues the RDOS system call .DUCLK .

(4) The number of ticks is an integer variable or constant specifying the integer number of RTC cycles which are to elapse between user clock intervals.

(5) The interrupt response code is a pointer to the user routine which will receive control when an interrupt occurs from the user clock.

(6) The current environment is frozen when the user clock generates an interrupt, and it remains frozen while control goes to the interrupt service routine.

(7) The error code will be set to one of the following states:

0 - Indeterminate error
1 - No error occurred
3 ... n - RDOS system error + 3

--- ROUTINE: FTIME

Supporting Subroutines: FRET; .CPYL, .RTER

Subroutine Size: 2 page zero locations and 41_8 locations of normally relocatable memory.

Entry: FGTIM

Function: Gets the time of day

Calling Sequence:

```
FCALL
FGTIM
Integer 3
FORTRAN ADDRESS to receive the hour
FORTRAN ADDRESS to receive the minute
FORTRAN ADDRESS to receive the second
```

Notes:

(1) Accumulators and carry are saved.

(2) FGTIM must be referenced in an .EXTN statement.

(3) No error message is possible.

(4) The time of day is given by a 24 hour clock.

ROUTINE: FTIME (Continued)

Entry: FSTIM

Function: Sets the system clock.

Calling Sequence:

```
FCALL
FSTIM
Integer 3
FORTRAN ADDRESS of the current hour
FORTRAN ADDRESS of the current minute
FORTRAN ADDRESS of the current second
```

Notes:

(1) Accumulators and carry are saved.

(2) FSTIM must be referenced in an .EXTN statement.

(3) A fatal run time error message, ERTIM, is issued if an attempt is made to set an illegal time. Upon issuance of an error message, control returns either to the Debugger or to the CLI.

--- ROUTINE: GFREQ

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 7 locations of normally relocatable memory.

Entry: GFREQ

Function: Examines the system real time clock and returns the clock frequency.

Calling Sequence:

```
FCALL
GFREQ
Integer 1
FORTRAN ADDRESS of word to receive frequency
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) GFREQ must be referenced by an .EXTN statement.

(3) No error messages are generated.

(4) This routine issues the RDOS system command .GHRZ .

ROUTINE: GFREQ (Continued)

Notes: (Continued)

(5) The word receiving the frequency will be set to one of the following values:

0 - No real time clock in system
1 - 10 HZ
2 - 100 HZ
3 - 1000 HZ
4 - 60 HZ
5 - 50 HZ

--- ROUTINE: RUCLK

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 6 locations of normally relocatable memory.

Entry: RUCLK

Function: Removes a previously defined user clock.

Calling Sequence:

```
FCALL
RUCLK
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) RUCLK must be referenced by an .EXTN statement.

(3) No error messages are generated.

(4) This routine issues the RDOS system call .RUCLK .

--- ROUTINE: TIME

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 16_8 locations of normally relocatable memory.

Entry: TIME

Function: Fetches the current time of day.

Calling Sequence:

```
FCALL
TIME
Integer 2
FORTRAN ADDRESS of time array
FORTRAN ADDRESS of error code
```

ROUTINE: TIME (Continued)

Notes:

(1) Accumulators and Carry are restored upon exit.

(2) 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.

(3) The error code word will be set to one of the following states:

0 - indeterminate error
1 - No error occurred

O/S ROUTINES

--- ROUTINE: BOOT

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 14_8 locations of normally relocatable memory.

Entry: BOOT

Function: Bootstraps a new operating system.

Calling Sequence:

```
FCALL
BOOT
Integer 2
FORTRAN ADDRESS of name of a primary global device
FORTRAN ADDRESS of error code
```

Notes:

(1) Accumulators and carry are restored on exit.

(2) BOOT must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .BOOT.

(4) A call to this routine causes all open files in a currently executing system (both foreground and background) to be closed, all directories to be released, and all system I/O to be reset. Control is then transferred to HIPBOOT, which will then bootstrap a new operating system.

(5) A call to BOOT should not be issued from the background when the foreground is active.

(6) 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
```

--- ROUTINE: GSYS

Supporting Subroutines: FRET; .CPYL

Subroutine Size: No page zero locations and 7 locations of normally relocatable memory.

Entry: GSYS

Function: Gets the name of the current operating system.

Calling Sequence:

```
FCALL
GSYS
Integer 1
FORTRAN ADDRESS of array to receive name
```

ROUTINE: GSYS (Continued)

Notes:

(1) Accumulators and carry are restored on exit.

(2) GSYS must be referenced by an .EXTN statement.

(3) This routine issues the RDOS system call .GSYS .

(4) The array to receive the system name requires 13_8 words.

(5) The name returned will consist of the name plus its two character extension, terminated by a null terminator.

(6) No error messages are generated.

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 routines' NREL entry points is given. This information can be obtained by the user by running an LPE analysis of library programs. The alternate names listed here are confined to those that represent meaningful entry points. FORT. LB, FMT. LB, and FSYS. LB are listed separately. The entry points are listed under each title.

FORT. LB

ABSLT
ABS
AFRTN
AFRTN
AINT
.AINT
ALG
ALG
.ALG10
AMNX0
AMAX0
AMIN0
AMX0
AMNX1
AMAX1
MAIN1
.AMX1
.AMN1
AMOD
.AMOD
ARCTAN
DATN2
DATN
ARDUM
.FLSP
ARGUM
FARGU
ARYSZ
ARYSZ
ATN
ATN2
ATN
BASC
.BASC
BDASC
.BDASC
BREAK
BRK
CABS
.CABS
CADD
CSUB
CAD

CCEQ
CEQ1
CCOS
CCOS
CDIV
CDV
CEXPO
CEXPO
CFILW
CFILW
CGT
CGT
CHSAV
CHSAV
CHRST
CLIP
CLP
CLOAD
CFLD
CLOG
CLOG
CLOSE
CLOSE
FCLOS
CMPLX
CMPLX
CMUL
CMUL
CNEG
CNEG
COMP
COMP
CONJG
CONJG
COS
CS
SN
COSIN
DCS
DSN
COUT
COUT
CIN

CPWR
CPWI
CPYAR
CPYARG
CPYLS
CRCX1
CRX2
CRCX2
DCRX2
CSIN
CSIN
CSQRT
CSQRT
CSTOR
CFST
CXFL1
CIX
CXFL2
DCIV
DATE
DATE
DBREAK
DBRK
DCABS
.DCAB
DCADD
DCSUB
DCAD
DCCEQ
CEQ2
DCCOS
DCCOS
DCDIV
DCDV
DCEXPO
CEXP
DCLOD
DCFLD
DCMPLX
DCMPLX
DCMUL
DCMUL
DCPWR
CPW2

DCSIN
DCSIN
DCSQR
DCSQR
DCSTR
DCFST
DDCLO
DCLOG
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
.LDG1
DMNMX
DMAX1
DMIN1
.DMN1
.DMX1
DMOD
.DMOD
DPOLY
DPLY2

DPWER
DPW
DREAL
DREAL
DAIMA
DSIGN
DSIGN
DSYGN
DSINH
DSIHN
DSNH
DSQRT
DSQR
DTANH
DTNH
EXP
EXPO
EXPC
XPC
FALOC
FALOC
FARG0
FRG1
FRG0
FDELY
FDELY
FFILE
FFIL
FINIT
FINIT
FINTD
FINRV
FINTD
FL
FL
FS
FA
FB
FM
FD
FEQ
FGE
FXL
FLX

FSG
FNG
FLE
FGT
FLT

RUN TIME ROUTINE TITLES AND NREL ENTRIES (Continued)

FLINK
SAV0
SAV2
SAV3
RSTR
QRSTR

FLIP
FLP
FLP0

FOPEN
FOPEN

FOVLD
OVLOD

FPWER
FPW

FREAD
FREAD
FRWRIT
BRD
BWR

FREDI
FREDI

FRGLD
FRGLD

FSBR
FSBR
FSUBA

FSEEK
FSEEK

FSTAT
FSTAT

FSWAP
FBACK
FCHAN
FSWAP

FTIME
FGTIM
FSTIM

I
.I

IABS
.IABS

IDIM
.IDIM

IDINT
.IDIN

IFIX
.IFIX

INT
.INT

IPWER
IPWR

ISIGN
.ISIG

ITEST
ICLR
ISET
ITEST

LD0
LD0
LD1
LD2
ST0
ST1
ST2

LDREG
LDR1
LDR2

LDSTB
LDB
STB

LE
LE
LT
GE
GT

MAD
MAD
MADO

MNMX0
MAX0
MIN0

MNMX1
MAX1
MIN1
.MX1
.MN1

MOD
.MOD

MOVE
MOVE
CMOVE

MOVEF
MOVEF

MULT
MPY
MPY0
DVD

MVBT
MVBT
MVBC

MVF
MVF

MVZ
MVZ

NFRTN
NFRTN

NPTR1
NR

NPTR3
NR3

NPRTR
NR2

OPEN
APPEND
OPEN
MTOPD

OVFLO
OVERF

PLY1
PLY1

RATN1
RATN

RATN2
RTN2

RCABS
CABS

RDBLK
RDBLK

RDCABS
RCAB

RDFLD
RDFLD
RDFCH

READL
READL
WRITL
REDS
WRITS

READR
READR

REAL
.REAL
.AIMA

RESET
RESET

RIPWR
RIPWR

RTER
RTER
RTE0
RTES

SDVD
SDVD

SIGN
SIGN
SYGN

SINH
SHIN
SNH

SMPY
SMPY

SQRT
SQR

STOP
STOP
PAUSE

STREG
ST1
ST2

TAN
TN

TANGE
DTN

TANH
TNH

THREA
ALLOC
THREAD

TIME
TIME

WRCH
WRCH

FMT.LB

ABORT
  ABORT

ASSOC
  ASSOC

CHNGE
  CHNGE

FACAL
  ASUSP
  AKILL
  ARDY

FOVLY
  FOVLD
  FOVRL

FPEND
  SUSP
  PEND

FPRI
  PRI

FQTASK
  FQTAS

FTASK
  FTASK

FTMAX
  FTMAX
  TMAX2
  SYST2
  TSAVE
  TKILL
  KILL
  CTASK
  SVVAR
  TNXT1
  TUNLK
  TLINK
  TREL
  LNKPR
  TERCM
  LQTSC
  QTCNT
  TIDSR

FXMT
  XMT
  XMTW
  REC

HOLD
  HOLD

IOPC
  IOPC
  IOPROG

ITASK
  ITASK

IXMT
  IXMT

MTI
  .I
  ITCB
  FERT0
  FERT1
  FERTN
  IOPTR

OVEXT
  OVEXT
  OVEXX

OVKIL
  OVKIL
  OVKIX

RELSE
  RELSE

START
  START

TACAL
  TAKIL
  TAPEN
  TAUNP

TIDC
  TIDST
  STID
  RTID
  KTID

TPEND
  TPEND

TPRI
  TPRI

TQTAS
  TQTSK
  QTCK

TOVLY
  TOVRL
  TOVLD
  OVWRT

TRNON
  TRNON

TUMOD
  SMSK

TXMT
  XMTT
  IXMTT
  XMTTW
  RECC
  XSRH

FSYS.LB

BOOT
  BOOT

CDIR
  CDIR

CHLAT
  CHLAT

CHSTS
  CHSTS

CPART
  CPART

DIR
  DIR

DLINK
  DLINK

DUCLK
  DUCLK

DULNK
  DULNK

EQUIV
  EQUIV

EXBG
  EXBG

EXFG
  EXFG

FCNS
  ODIS
  OEBL

FGND
  FGND

FSPOL
  SPEBL
  SPDIS
  SPKIL

GCIN
  GCIN

GCOUT
  GCOUT

GDIR
  GDIR

GFREQ
  GFREQ

GSYS
  GSYS

GTATR
  GTATR

ICMN
  ICMN

INIT
  INIT

MDIR
  MDIR

MTDIO
  MTDIO

RDOPR
  RDOPR

RENAM
  RENAM

RLSE
  RLSE

RUCLK
  RUCLK

RWCMN
  RDCMN
  WRCMN

STAT

UPDATE
  UPDATE

WROPR
  WROPR

INDEX

DataGeneral

SOFTWARE DOCUMENTATION
REMARKS FORM

Document Title	Document No.	Tape No.

SPECIFIC COMMENTS: List specific comments. Reference page numbers when applicable. Label each comment as an addition, deletion, change or error if applicable.

GENERAL COMMENTS: Also, suggestions for improvement of the Publication.

FROM:

Name	Title	Date

Company Name

Address (No. & Street)	City	State	Zip Code

Form No. 10-24-004 Rev. 7-75

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: Software Documentation

FOLD UP SECOND FOLD UP

STAPLE