ComputerAutomation

NAKED MINI® Division

18651 Von Karman, Irvine, California 92715
Telephone: (714) 833-8830 TWX: 910-595-1767

FORTRAN IV OPERATIONS MANUAL

96510-01B0 APRIL 1976

PRINTED IN THE U.S.A.

TABLE OF CONTENTS

TABLE OF CONTENTS (Con't)

TABLE OF CONTENTS (Con't)

Paragraph Page

Section 5. SYSTEM GENERATION

Appendix A. DEBUGGING AIDS

Appendix B. SAMPLE JOB SEQUENCES

Appendix C. FORTRAN SUBPROGRAM LIST

TABLE OF CONTENTS (Con't)

Appendix D. ERROR MESSAGES/HALTS

LIST OF FIGURES

TABLE OF CONTENTS (Con't)

Section 1

INTRODUCTION

SCOPE

This manual is intended to aid the Computer Automation FORTRAN IV programmer in compiling and executing his programs on the ALPHA-LSI series computer. It assumes that the reader knows how to write a FORTRAN program and is familiar with the FORTRAN IV Reference Manual, as well as the Computer Automation Operating System (OS) User's Manual, since compilation and linking must be, and execution may be, performed under control of the Operating System. Also, since FORTRAN programs may be executed under the Real Time Executive (RTX), the reader should be familiar with the RTX User's Manual as well if he intends to use the RTX or LSI-3/05 options.

The discussions are organized in a generally chronological order, according to the normal sequence of operations; that is, the FORTRAN operating environment and the Compiler are described first, followed by library structure and linking, and then run-time (execution). Thus the manual is structured similarly to the normal FORTRAN operation sequence (see figure 1-1).

System generation procedures are described at the end of the manual, as they are issued less frequently.

OPERATING ENVIRONMENT

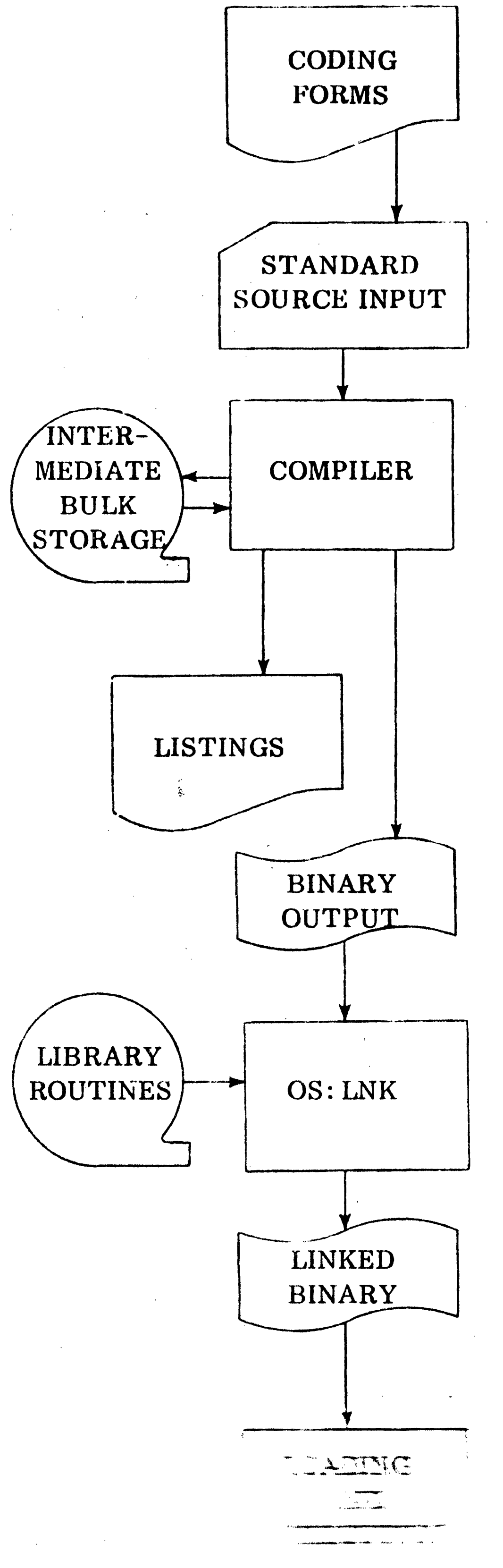
Configuration for Compilation

The FORTRAN IV compiler requires an ALPHA LSI-2 processor with at least 16K words of memory. A Computer Automation Operating System (DOS, MTOS or COS) must be present as well as an OS-labeled bulk device for intermediate storage of the source information.

The typical system, assumed for the examples in this manual, is a Disk Operating System (DOS) operating in an LSI processor with card reader, ASR-33 teletype, high speed paper tape reader and punch, and line printer.

Configurations for Linking and Execution

Once compiled, the output (object) program is then linked to the library routines it needs by means of the OS:LNK utility before it is executed. (The library routines are not included in the object output during compilation, so as to conserve space at execution time.) If the user intends to execute his program under OS (and not RTX) OS:LNK will assume that execution will take place under the same version of OS as the one which controls OS:LNK itself. This means that the linked program may not then be executed under an OS which has a different Root configuration or a different working core address. However, linking a program for execution under RTX causes the entire RTX/IOX monitor to be included within the linked program. Thus such a program may be loaded into any ALPHA-LSI processor and executed, provided that the processor contains sufficient memory to hold the linked object program.



FORTRAN program is coded, then stored onto suitable input medium (cards, paper tape or magnetic file)

Source is input to compiler, which manipulates and converts it to object format, using an intermediate bulk storage file.

Once converted, the compiler outputs source and object listings, allocation and subroutine usage maps to the list device. It outputs the compiled binary code in the requested form (magnetic file or paper tape).

The compiler-generated program is input to the link editor (OS:LNK), which links it to the required library routines.

The linked binary code is then output in standard loadable format.

Compilation and linking of a program to be executed on an LSI-3/05 processor must be done with the type 3/05 option specified. Execution can only be done under RTX, since OS itself is not supported on the LSI-3/05.

Section 2

FORTRAN IV COMPILER

PURPOSE

The purpose of the FORTRAN compiler is to input each source record (FORTRAN statement) through the source input (SI) device, convert the program statements into their component machine-language instructions utilizing the assigned Source Save (SS) device to assist with intermediate storage requirements, and then to output the linkable (but not loadable) binary code to the assigned binary output (BO) device, and the source listing and allocation map to the assigned list output (LO) device. (Note from figure 1-1 that the compiler does not produce a program which is directly executable; the program is linked to the needed library routines and converted into standard loadable format, and then loaded by one of the standard loaders.)

COMPILER ORGANIZATION

Compiler Modules

The FORTRAN compiler is a three-phase, two-pass compiler which processes FORTRAN source programs one at a time, in a batch mode. It is configured as a control program and three overlays resident on the system file (SF) device (see figure 2-1). Note that an alternative configuration is used when only 16K of memory is present. This involves the Scan and Gen overlays being further segmented into 3 overlays each. A complete description of this organization may be found in the System Generation section – Generating the FORTRAN Compiler.

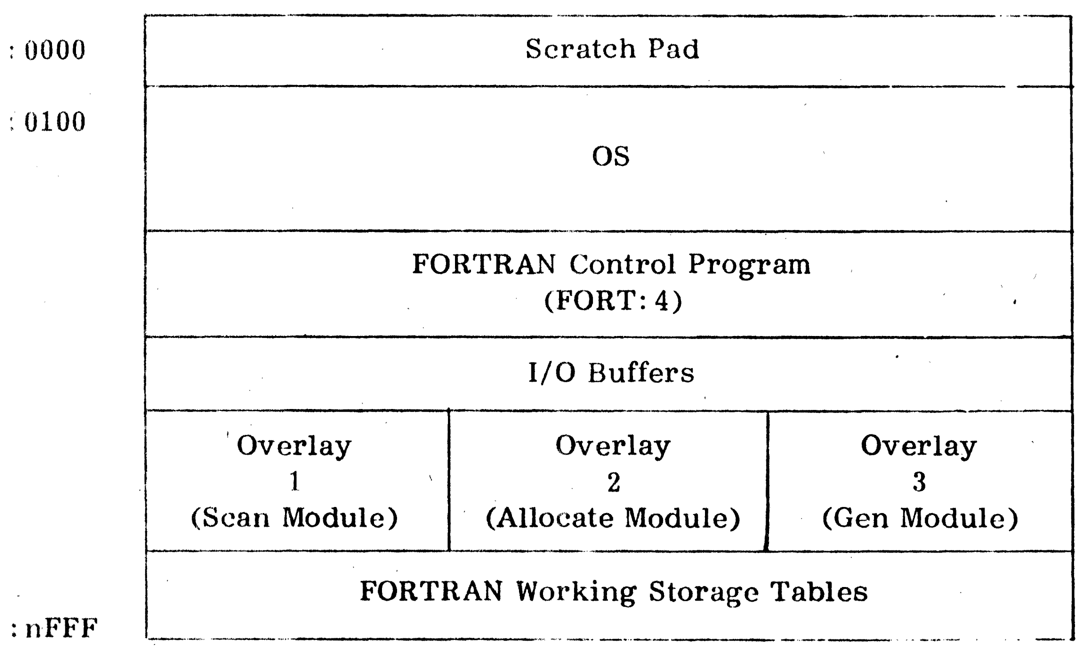


Figure 2-1. FORTRAN Compile-Time Memory Layout

Control Program

The control program, utilizing the FORTRAN/OS I/O Interface routine, causes each overlay to be loaded, then passes control to it. When each overlay has completed its processing it returns to the control program, which then calls the next overlay. The control program also handles all input/output and other communication to the operating system.

Overlay 1 - Scan Phase

The Scan phase inputs each record of the FORTRAN source program, builds symbol tables in its working storage area and outputs the source program listing and syntax-type error messages to the LO device, and the intermediate program code to the SS bulk device. The Scan phase is completed when a FORTRAN END statement is encountered.

Overlay 2 - Allocate Phase

The Allocate phase uses the symbol tables created during the Scan phase to allocate storage for program variables. It then outputs, to the LO device, the allocation map and error messages for any COMMON, EQUIVALENCE or undefined label errors.

Overlay 3 - Object Generation Phase

The Object Generation (or "Gen") phase operates on the intermediate program code stored onto SS during the Scan phase, together with the storage allocation information produced during the Allocation phase, and from these it outputs object code to the BO device, and symbolic object text to the LO device (if requested). It then outputs the subroutine usage map, statement label location list and program size information to the LO device.

Batch Mode

The "batch" mode organization of the compiler means that completion of the Gen phase (overlay 3) causes control to return to the control program; this in turn calls overlay 1 again, etc., until an end-of-file condition is sensed from the Source input device. Each compilation is a complete sequence of procedures. (Various compiler options exist to permit the operator to tailor the compiler output to his specific needs - see Compiler Options.)

Working Storage

To make maximum use of available memory, the compiler dynamically allocates its working storage tables, thus each table is variable in length so that no table can be completely filled if any unused memory is available.

I/O CONSIDERATIONS

The compiler Control program, which coordinates overlay calls, also handles I/O requests to the Operating System. Since the standard OS I/O drivers are used, all I/O is interrupt driven, rather than sense-driven. These requests are made to and from the following logical units, which must be assigned to physical devices prior to beginning the compilation:

System File (SF)

This is the file containing the compiler itself (control program and overlays). It should reside on a file-type device (see System Generation).

Source Input (SI)

This is the file containing the source records (FORTRAN statements) to be compiled. It may be assigned to any OS-supported input device (card reader, teletype keyboard, paper tape reader, or magnetic device file). The standard length for OS source input records is 80 characters. However, less than this number may be input if a record is terminated by a carriage return character. In addition, even though OS will input 80 characters, the compiler processes only the first 72 as a valid statement. Characters in excess of 72 are treated as comment characters and ignored in the compilation.

A complete source input file is comprised of one or more FORTRAN programs, each of which must contain an END statement as its last record. The file itself must be terminated with an end-of-file mark. If the file contains two or more programs, each program is compiled before the next is input, in a "batch" mode. Processing of a batch file will result in binary output of a single file, however, and is to be used only for a main program followed by subprograms (subroutines or tasks). It is illegal to input two or more main programs (which do not reference each other) in the batch mode.

Source Save (SS)

This is the file created by the Scan phase of the compiler, and must be on a file-type device. The data written to this file is the source information, in abbreviated form. The data is later read back into memory during the Gen phase of compilation. It is normally not necessary for the user to assign this file before compilation, since its normal default assignment is to the system file device under the file name "S:::S". However, it may be assigned to any file-type device, if desired, and a different file name may or may not be included in the assignment. In any case, the SS file will be set up by the compiler under the "close/delete" format, which means that the file will automatically be deleted upon completion of the Gen phase.

Binary Output (BO)

This is the file to which the compiled binary code will be output during the compiler's Gen phase, and which must be subsequently linked to the FORTRAN library file by the OS:LNK utility. It is normally assigned to a magnetic file or to the paper tape punch.

Format of the binary output is in standard Computer Automation object code format, including several type codes designed specifically for the FORTRAN compiler. This output must be subsequently linked with the applicable library routines using OS:LNK rather than LAMBDA or OS:LDR. OS:LNK recognizes all of the specialized type codes used by FORTRAN, while LAMBDA and OS:LDR do not.

List Output (LO)

This file should be assigned to the list device for output of the compiler-generated listings which include source listing, diagnostics, allocation map, object listing (if specifically requested), subroutine usage map, statement label locations, and program size information.

Assignment of the SI, SS and BO devices should be made with a thought to optimizing I/O throughput. For example, since the four compiler modules must be input from the SF device at different times during a compilation, compiling under MTOS with the SI, SS or BO file also assigned to the System file device will cause markedly slower operation due to excessive tape repositioning. While this is not a problem under DOS because of the disk's random access capability, assigning several logical units to the SF device will require partitioning of the disk into 4 or 8 partitions.

COMPILER LISTINGS

Figure 2-2 is a sample FORTRAN output listing:

```
PAGE 0001  09/24/74  15:13:46  FORT:4 (A1)
 BO FILE:  FOUT     OPTIONS:    LO

      C           DEMONSTRATE OBJECT LISTING
            INTEGER NN(25), LL(10)
            DOUBLE PRECISION DX, DY
            COMMON MM(100), M /BLK/ Y
            EQUIVALENCE (L,LL)
            ISF(KD) = KD*8
         10 K = (I +300)*M - 74
            MM(I) = K
            X = ABS(Y+4)
            DX = DABS(DY/4.3)
            IF (DX .LT. 0) GO TO 70
            CALL SUB(L+300,7HABCDE  ,Y+4)
         20 WRITE(6,30) Y
         30 FORMAT( 5X , I5 , ' VALUES.' )
            IF (K .EQ. M) GO TO 10
               ASSEMBLER
               LAP     :2A
               ADD     K                      (LOCAL VARIABLE IN RANGE)
               STA     *BP(MY:NAM)            (SPECIAL SYSTEM NAME)
               JMP     #60                    (FORWARD REFERENCE IN RANGE)
      #40      RES     32,'  '                (TO FORCE LITERAL POOL)
               FORTRAN
         50 DO 60 I = 1,10
         60 MM(I) = -1
            ASSIGN 40 TO K
            MN(3) = 0
                $
 01) UNDIMENSIONED E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E


         70 STOP
            END
```

Figure 2-2. Sample FORTRAN Output Listing

```
PAGE 0002   09/24/74   15:13:46   FORT:4 (A1)
 BO FILE:   FOUT      OPTIONS:     LO

COMMON BLOCK/F:BCMN/ ALLOCATION  :0065 WORDS

LOCN   NAME    TYPE        WORDS    LOCN   NAME    TYPE        WORDS

:0000  MM      INTEGER       100    :0064  M       INTEGER        1

COMMON BLOCK/BLK    / ALLOCATION  :0002 WORDS

LOCN   NAME    TYPE        WORDS    LOCN   NAME    TYPE        WORDS

:0000  Y       REAL            2

ARRAY ALLOCATION

LOCN   NAME    TYPE        WORDS    LOCN   NAME    TYPE        WORDS

:0000  NN      INTEGER        25

EQUIVALENCE ALLOCATION

LOCN   NAME    TYPE        WORDS    LOCN   NAME    TYPE        WORDS

:0022  L       INTEGER         1    :0022  LL      INTEGER       10

SCALAR ALLOCATION

LOCN   NAME    TYPE        WORDS    LOCN   NAME    TYPE        WORDS

:002C  K       INTEGER         1    :002D  I       INTEGER        1
:002E  X       REAL            2    :0030  DX      DOUBLE         4
:0034  DY      DOUBLE          4
```

Figure 2-2. Sample FORTRAN Output Listing (Cont'd)

BO FILE: FOUT   OPTIONS: LO

```
0001 C               DEMONSTRATE OBJECT LISTING
0002           INTEGER NN(25), LL(10)
0003           DOUBLE PRECISION DX, DY
0004           COMMON MM(100), M /BLK/ Y
0005           EQUIVALENCE (L,LL)
0006           ISF(KD) = KD*8
                    :0038 :F200 F           JMP    #M7
                    :0039 :0800    #M8      ENT
                    :003A :F900 B           JST    *BP(F:RDMY)
                    :003B :0001             DATA   1
                    :003C :0000    KD       DATA   0
                    :003D :B701             LDA    *KD
                    :003E :1052             ALA    3
                    :003F :F706             JMP    *#M8
0007      10 K = (L+300)*M - 74
                    :0040          #M7      EQU    :0040
                    :0040 :B200 F  #10      LDA    #IC1                 :012C
                    :0041 :8E1F             ADD    L
                    :0042 :9A00 F           STA    #T0
                    :0043 :F900 B           JST    *BP(F:RMPY)
                    :0044 :0064 C           DATA   M
                    :0045 :0D4A             SAI    74
                    :0046 :9E1A             STA    K
0008           MM(I) = K
                    :0047 :E61A             LDX    I
                    :0048 :9D00 B           STA    **BP(MM    -1)
0009           X = ABS(Y+4)
                    :0049 :F900 B           JST    *BP(F:RREL)
                    :004A :AA00 F           LDR    #RC1                 :4180:0000
                    :004B :8900 B           ADD    *BP(Y      )
                    :004C :9A00 F           STA    #T1
                    :004D :0005             ABS
                    :004E :9E20             STA    X
0010           DX = DABS(DY/4.3)
                    :004F :B618             LDD    DY
                    :0050 :A200 F           DVM    #RC2                 :4189:9999
                    :0051 :0005             ABS
                    :0052 :9E22             STA    DX
0011           IF (DX .LT. 0) GO TO 70
                    :0053 :0000             XIT
                    :0054 :2080 F           JAM    #M9
0012           CALL SUB(L+300,7HABCDE  ,Y+4)
                    :0055 :F900 B           JST    *BP(SUB    )
                    :0056 :0003             DATA   3
                    :0057 :0000 F           DATA   #T0
                    :0058 :0000 F           DATA   #HC0
                    :0059 :0000 F           DATA   #T1
0013      20 WRITE(6,30) Y
                    :005A :F900 B  #20      JST    *BP(F:RWF )
                    :005B :0000 F           DATA   #IC5                 :0006
                    :005C :0000             DATA   #30
```

```
                :005D :F900 B          JST    *BP(F:RROL)
                :005E :0000 C          DATA   Y
                :005F :F900 B          JST    *BP(F:RSIO)
0014      30 FORMAT( 5X , I5 , ' VALUES.' )
                :0000 :A8B5    #30     TEXT   '(5X,I5,' VALUES.')'
0015         IF (K .EQ. M) GO TO 10
                :0060 :B634            LDA    K
                :0061 :9100 B          SUB    *BP(M      )
                :0062 :2100 F          JAZ    #M10
                :0040          #M10    EQU    :0040
0016            ASSEMBLER
0017            LAP       :2A
                :0063 :C62A            LAP    :002A
0018            ADD       K                   (LOCAL VARIABLE IN RANGE)
                :0064 :8E38            ADD    K
0019            STA    *BP(MY:NAM)            (SPECIAL SYSTEM NAME)
                :0065 :9900 B          STA    *BP(MY:NAM)
0020            JMP       #50                 (FORWARD REFERENCE IN RANGE)
                :0066 :F200 F          JMP    #50
0021 #40        RES       32,' '              (TO FORCE LITERAL POOL)
                :0067 :A0A0    #40     RES    32,' '
0022            FORTRAN
0023      50 DO 60 I = 1,10
                :0087 :C401    #50     LXP    1
                :0088 :EE5B    #M11    STX    I
0024      60 MM(I) = -1
                :0089 :C701    #60     LAM    1
                :008A :9000 B          STA    **BP(MM     -1)
0025         ASSIGN 40 TO K
                :008B :C201            AXI    1
                :008C :0030            TXA
                :008D :000A            SAI    10
                :008E :21C6            JAL    #M11
                :008F :B200 F          LDA    #40
                :0090 :F200 F  #L      JMP    #M12               LITERAL POOL
                :0091 :F200 F          JMP    #M9
                :0092 :9E66    #M12    STA    K
0026         MN(3) = 0
              $
01) UNDIMENSIONED E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E
                :0093 :F900 B          JST    *BP(F:RERR)
                :0094 :001A            DATA   26
                :0095 :0000 F          DATA   #RC4                 163C6:4:57
00[illegible]      STOP
                :[illegible]   #7[illegible]  EQU   :[illegible]
                :0096 :F400 B #M5      JST    *BP(F:RSTO)
                [illegible]            DATA   I
```

```
PAGE 0005   09/24/74   15:13:46   FORT:4 (A1)
 BO FILE:   FOUT     OPTIONS:     LO

                        :009B :4189   #RC2    DATA   16777
                        :009C :9999           DATA   -26215
                        :009D :9999           DATA   -26215
                        :009E :999A           DATA   -26214
                        :009F :0000   #T0     DATA   0
                        :00A0 :0000   #T1     DATA   0
                        :00A2 :0067           DATA   #40
                        :00A3 :0007           DATA   7
                        :00A4 :C1C2   #HC0    DATA   'AB'
                        :00A5 :C3C4           DATA   'CD'
                        :00A6 :C5A0           DATA   'E '
                        :00A7 :A0A0           DATA   '  '
                        :00A8 :0006   #IC5    DATA   6
                        :00A9 :63C6   #RC4    DATA   25542
                        :00AA :4167           DATA   16743

SUBPROGRAMS CALLED

NAME      TYPE      ARGS    NAME     TYPE      ARGS    NAME     TYPE      ARGS

ABS       REAL      1       DABS     DOUBLE    1       SUB      REAL      3
F:RWF     RUNTIME           F:RROL   RUNTIME           F:RSIO   RUNTIME
MY:NAM    RUNTIME           F:RERR   RUNTIME           F:RSTO   RUNTIME
F:RU06    RUNTIME           F:RREL   RUNTIME           F:RDBL   RUNTIME
F:RFZ     RUNTIME           F:RFF    RUNTIME           F:RDMY   RUNTIME
F:RMPY    RUNTIME

STATEMENT LABELS

LOCN   LABEL   USE          LOCN   LABEL   USE          LOCN   LABEL   USE

:0040  #10                  :0096  #70                  :005A  #20     UNUSED
:0000  #30     FORMAT       :0087  #50                  :0067  #40
:0069  #60     DO END       :0040  #M7                  :0039  #M8
:0096  #M9                  :0040  #M10                 :0088  #M11
:0092  #M12

ENTRY=:0038
PROGRAM SIZE=:00AB WORDS
BASE PAGE USED=:000D WORDS
COMPILATION COMPLETE  1 ERRORS
```

Figure 2-2. Sample FORTRAN Output Listing (Cont'd)

The full listing of a compiled program consists of four parts:

1. Source listing
2. Variable storage allocation
3. Object listing
4. Summary

When no special options are requested, the object listing is not produced, but the other three are. The LO (List Object) option causes the object listing to be produced. If the EL (Error List only) option is specified, the source listing is suppressed, except for the first line and any lines that have errors. This can be used to save time and paper, while still being informed of any errors. Figure 2-2 shows a complete program listing. Following is a description of the four parts.

Source Listing (Page 0001)

The source listing shows each source line, preceded by a decimal line number beginning with 0001. One space separates the line number and the first column of the source line. Every line is numbered, including continuation lines and comments. If EL (Error List only) is requested, the first source line is automatically output, and the correct line number will be shown for any error source lines. Error messages may be interspersed, as shown after line 0026 of the sample program in figure 2-2. Note that each such message is followed by a string of E's (or W's) and asterisks, so that it will stand out. See "Compiler Diagnostics" for more information.

Variable Storage Allocation (Page 0002)

Several kinds of tables can appear here, depending on the variables used in the program, and their allocation. If any variables have been allocated in COMMON, a storage map will appear for each COMMON block, including blank COMMON which is known as F:BCMN. Each map gives the name of the block and its size in hexadecimal. Then each variable is listed, showing its location (in hexadecimal), name, type, and size (in decimal). The size is the total number of words occupied. Remember that floating point quantities occupy more than one word per element. (Others may too in ANSI mode.)

If there are local (non-COMMON) arrays that have not appeared in EQUIVALENCE, these are shown next, with the same information as for the variables in COMMON. Next comes the map for any local variables (arrays or scalars) that have appeared in EQUIVALENCE. And finally, a table of all the local scalar variables (not in COMMON, not EQUIVALENCEd).

A table heading appears only if there are any items to appear in it. The variables in each table are listed in storage order, except for the effects of EQUIVALENCE, which [illegible] variables to appear out of order. Thus the local arrays and scalars will appear [illegible]

Object Listing (Pages 0003-0005)

Figure 2-2 shows a sample object listing. Some descriptions below refer to it, either by source line number or by hexadecimal location.

An object listing always includes all source lines, even if suppressed in the source listing by the EL (Error List only) option. The source lines are interspersed so that in most cases they are followed by the instructions that were generated for them. When examining the object code produced for one individual statement note the following:

1. The compiler does not generate object code one statement at a time. It remembers computations and the contents of the registers from previous statements within a block. (A block is ended by a label that is jumped to or in other ways.) Therefore, the code for one statement may look incomplete, since it is making use of results from previous statements. See, for example, source line 0012, which uses two values computed earlier and stored in temps, line 0011, which uses the contents of the floating point accumulator, and line 0024, which uses the contents of the index register.

2. Literal pools may be generated at almost any point in the program, making the code for that statement look longer.

3. The code to terminate a DO loop is not listed after the terminal statement, but after the following statement. This is illustrated by source line 0025, which also contains a literal pool, thus making its two instructions look like eight.

The layout of an object program is shown in figure 2-3.

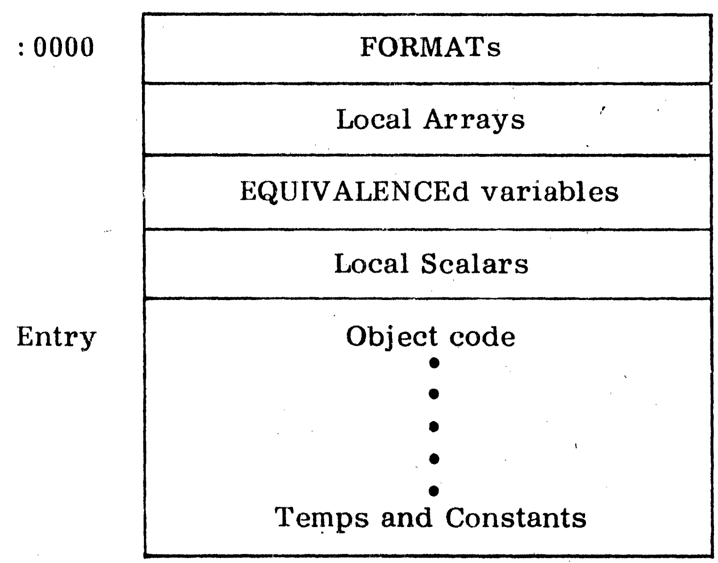


Figure 2-3. Layout of Object Program

The allocation of variables was shown in the allocation maps, so is not reproduced in the object listing. The FORMATs, although generated apart at the head of the program, are listed where they appear in the source program. Here the program is not listed in strict forward order (i.e. the memory locations are not listed sequentially). Another place is the temps at the end of the program. For the most part, however, the program is listed in forward order, beginning at the entry point and ending at the last temp or constant.

Each line of object code listed consists of seven parts (four of which are optional) and from left to right these are:

1. The hexadecimal location counter. See below for a complete list of the situations in which the location counter does not increase by one at each line.

2. The hexadecimal representation of the generated word, which may be an instruction or a data value. In many cases, this is only a skeleton word, since the actual address is not known at the time it is listed. This includes references to COMMON, externals, base page, and most forward locations. Also, an instruction may turn out to be indirect through a literal pool pointer, even though it is not listed that way.

3. An optional alphabetic tag letter, which indicates for some operands the kind of addressing that the generated word is actually using. These are:

   B Base page
   C COMMON (blank or labeled)
   F Forward reference
   S Scratchpad Relocatable data

4. These next four items in the line are parts of a simulated assembly language listing of the instruction. It is not always possible to list the instruction exactly as it would appear in assembly language, but in most cases the representation is very close and makes it clear what the compiler is doing. See below for a list of differences. The first field is the label field, beginning in column 1 of the simulated assembly listing. For normal instructions (i.e. not temps, constants, or literal pools), there are three kinds of labels that can appear:

   #n Statement number from the source program. (For example, see location :005A).

   #Mn "Made" label, an internal transfer point generated by the compiler. (E.g. location :0040, which is the target of the jump around the statement function above.) Note that in this case there are two labels attached to the same location.

   name This occurs only on the dummies of statement functions (e.g. location :003C). The dummies of FUNCTIONs and SUBROUTINEs are not labeled, nor is the entry point.

   Several other kinds of labels can appear in special places:

   #Tn Temp. Appear at the end of the program (e.g. location :009F).

   #ICn Integer constant. Usually appear at the end (e.g. location :00A8), but can also appear in literal pools.

   #RCn Real (or double precision or complex) constant. Appear only at the end of the program (e.g. location :0099).

#HCn Hollerith constant. Appear only at the end (e.g. location :00A2), and are always preceded by the character count.

#L Literal pool. This label serves only to signal the beginning of a literal pool (location :008F). It is never referenced, and can appear more than once without constituting a duplicate definition. It always appears on the jump around the literal pool, and therefore does not appear on pools generated by the LPOOL directive.

5. Op-code field. All of the possible op-codes are shown in the section on in-line assembly language in the FORTRAN Reference Manual. They are all either standard assembler mnemonics or floating point interpretive op-codes.

6. Operand field. Where appropriate, it may begin with * (indirect) and/or @ (indexed). A large variety of operands can appear, some only as the result of having been used on an in-line assembly instruction.

   a. Blank. For op-codes like TXA or ABS that have no operand (e.g. location :0039).

   b. Decimal value, optionally preceded by minus sign (location :0045).

   c. Hexadecimal value, always preceded by a colon (location :0063).

   d. Alphanumeric string, enclosed in quotes (location :00A2).

   e. #n (statement label) (location :005C). Can be followed by decimal addend only from in-line assembly.

   f. #Tn (Temp, e.g. location :0042), #Mn ("made" label, location :0038), #ICn (Integer Constant, location :0040), #RCn (Real Constant, location :004A), or #HCn (Hollerith Constant, location :0058).

   g. $ (current location), optionally followed by a decimal addend. This can occur only from in-line assembly. Otherwise the compiler always generates a "made" label.

   h. FORTRAN name (variable or subprogram), optionally followed by decimal addend (location :003D or :0047).

   i. Special system (or run time) name, which always contains a colon (location :005F or :0065). As shown, these are usually in combination with a BP (Base Page) reference, since most instructions cannot address external references directly.

   j. BP(x), base page reference, where x is a FORTRAN name or system name, possibly with an addend (location :0048). BP of other operands can result only from in-line assembly language.

NOTE

In certain cases (notably e, f, and h above), operands may be listed as direct when, in fact, they turn out to be indirect through a literal pool pointer word, because they are out of range. The only way to determine this is to look at the actual word in memory after the program is loaded.

7. Comment field. When numeric constants are referenced, their hexadecimal value is shown in the comment field (location :004A). This value may differ by one bit from the actual value printed at the end of the program, because the rounding is not applied until then. Note that on location :0050, only the first three words of a four word constant are shown, because the printer line width was not large enough to fit them all in.

Summary (Page 0005)

The summary is printed immediately following the object listing, if there is one, otherwise following the allocation tables. First the subprograms called by the program are listed. This includes functions and subroutines referenced explicitly by the program, as well as run-time routines referenced by the generated object code (e.g. for floating point, input/output, etc.). Names referenced by the program are FORTRAN names, i.e. beginning with a letter and containing only letters and digits. Run-time routines are non-FORTRAN names, because they always contain a colon (e.g. F:RWF, F:RREL). This may include special system names referenced by in-line assembly language (e.g. MY:NAM in the sample program).

The table shows first the name of the subprogram. Next is the type (e.g. REAL, INTEGER) if it is a FORTRAN referenced name, or the word RUNTIME otherwise. Then, again for FORTRAN referenced subprograms only, appears the number of arguments it has been called with. If the number of arguments is variable (e.g. to AMAX1) or unknown (name declared external but not directly called), the number of arguments is shown as zero.

With the exception of intrinsic functions, this list of subprograms called represents the names that must be found during loading, either from the library or from other programs compiled or assembled by you. Intrinsic functions (e.g. ABS) are listed here but are not actually referenced externally. They are generated in-line.

Second in the summary is a map of the statement labels. This includes the statement numbers used in the source program and also the "made" labels generated by the compiler (#Mn). They appear in the order defined or referenced in the object program, which is not necessarily storage order. Each entry contains the hexadecimal location, the label, and in certain cases an indication of the use. There are three such indications:

FORMAT	This is the label of a FORMAT statement.
DO END	This has been used only as the terminus of a DO.
UNUSED	This label was defined on a statement, but never referenced.

Finally, four pieces of information are given about the program:

1. Location (in hexadecimal) of the entry point.
2. Total size (in hexadecimal) of the program, including local variables but not COMMON.
3. Number of base page words used (in hexadecimal).
4. Message COMPILATION COMPLETE followed by the number of errors (even if zero).

DIFFERENCES FROM ASSEMBLY LANGUAGE

As noted above, the simulated assembly language listing of the object program is an approximation of how the program would appear in assembly language. In most cases it is exactly the same, but there are some differences you should be aware of, both to aid your understanding of the generated code, and also in case you should try to extract code from a compiled program and use it in an assembled program. These differences are listed below.

1. Operands that are out of range are not always shown as referenced indirectly through a literal pool pointer, even though that happens. This can happen on statement numbers, "made" labels, temps, and floating and Hollerith constants. For example, location :0054 shows a direct reference to #M9, but actually ends up being indirect through the literal pool address in location :0091.

2. Similarly, references to array offsets that have to be stored in temps (in No Scratchpad mode) may show just the name of the array, when they actually address a constant containing the array base minus an offset.

3. Also in the same vein, the ASSIGN statement lists a load of a statement label instead of a constant containing the address of the label (e.g. location :008F).

4. Instead of increasing by one each time, the location counter may jump suddenly without indication in the assembly language. This can happen in the following places:

   a. FORMATs are generated starting in location :0000 (program relative), regardless of where they appear in the source program (see source line 0014).

   b. Not all of the generated hexadecimal words are shown for the TEXT command in a FORMAT statement. Only the first word is shown (in order to save paper in the object listing), unless the string is more than 32 characters long, in which case every sixteenth word will have a new TEXT command and one word of hexadecimal. For example, see source line 0014.

   c. The temps listed at the end of the program may not be in order; the location counter may jump around. Also, although all temps are listed as DATA 0, some of them actually occupy two or four words, so the location counter will increment by that amount.

5. Whenever a name (FORTRAN or runtime) is listed as an operand, the full six spaces are always reserved for it. Thus if there is something to follow the name (e.g. an addend), and the name is shorter than six characters, there will be blanks in between, which would not be allowed in assembly language. (For example, location :0048 or :005A).

6. The decimal value -32768 is listed as -0.

7. If a quote mark appears within an alphanumeric string that is enclosed in quotes, it is represented only as a single quote mark, rather than as two quotes (which would be required normally in such a string).

8. #L appears in the label field of all compiler generated literal pools (i.e. those not called forth by the LPOOL directive). It is only a signal and never gets defined, but in assembly language it would constitute a double definition.

9. The double-word op-codes MPY, DVD, and NRM, instead of being listed as, for example,

```
MPY     a,b
```

are listed as:

```
MPY
DATA    a,b
```

but they generate the correct object code, which is:

```
MPY     b
DATA    a
```

10. A number of things are implied in the object listing, without being specifically shown. This includes:

   a. The scalars and arrays are not allocated (i.e. by RES directives). The compiler knows where they are and tabulates this information in the allocation maps preceding the object listing.

   b. External definitions and references and allocation of variables into COMMON are not shown.

   c. The dummies of FUNCTIONs and SUBROUTINEs are not labeled with their names.

   d. The entry point of the program is not labeled (i.e. with the subprogram name or F:MAIN). However, it is identified as such in the summary.

   e. No END line is listed, and therefore no transfer address (to F:MAIN) in a main program.

COMPILER OPTIONS

Compilation may be performed under nine different options. Each is described below, and may be requested by the user by including the option names as parameters in the OS/EXECUTE or /BEGIN command when starting the compilation. The compiler looks at only the first two characters of the option name; thus either the first two characters or the entire option name may be specified. The options requested are output on the listings (in 2-character format) as the second header line on each page, along with the BO file name, if any.

EList (Error-only listing)

Requesting this option will cause the compiler source output listing to be suppressed, except for those statements with Error or Warning diagnostics.

(The first source line of the program is always printed.)

LObj (Object code listing)

This option lists, following the source listing and allocation map, the actual machine language code generated by each FORTRAN statement, and its symbolic representation in FORTRAN assembly format (see Figure 2-2, pages 0003-0005). The code for each FORTRAN statement is preceded by the source statement. This listing can be useful to the programmer who wishes to see how the source statement is expanded into binary code, and thus offers a convenient method for use in debugging, or for comparing memory usage and execution time for the various statements. This listing can be rather long, however, since several lines are generated for every source statement.

NBinary (Suppress binary output)

This option suppresses output to the BO device. This option is requested when it is likely that the source statements contain errors (e.g. in a preliminary compilation), and thus the resultant binary output will not be useable. Output of the normal printer listings is unaffected by this option.

RScratchpad (Reduced scratchpad usage)

This option reduces the amount of scratchpad area used during the execution of the compiled FORTRAN program. An example is where the user compiles a large FORTRAN program, then links it using OS:LNK, only to find a scratchpad overflow condition. At this time, he should re-compile the program using the "RS" option.

Note that this option does not totally preclude scratchpad usage, but rather causes the compiler to minimize its use, by creating address pointers to external subprograms in main memory rather than in scratchpad. Note, however, that references to arrays and COMMON variables remain in scratchpad.

NScratchpad (no scratchpad usage)

This option causes the compiler to avoid the use of scratchpad for address pointers to external subprograms, arrays, and variables in COMMON.

Note: There are 20 words of relocatable scratchpad (SREL) program which are always required in scratchpad, even when the NS option is requested. These are used by FORTRAN at run-time for its floating point accumulator and other special temp cells.

This option should be used when the FORTRAN programmer requires a large amount of scratchpad for his own purposes. This option causes less efficient run-time code to be generated in order to compensate for the avoidance of scratchpad.

XOn (Compile "X" statements)

This option compiles any FORTRAN statement containing an "X" in column 1. If the option is not requested, such statements will be treated as comments. This is a useful option for debugging purposes during program checkout. Once the program has been shown to be correct, it may be compiled without the XOn option, and the "X" statements then serve as historical references. Refer to the XON example in the FORTRAN Reference Manual.

ADp (Automatic Double Precision option)

This option changes all real variables, arrays, constants and non-library subprograms in the FORTRAN source program to double precision. In effect, the compiler proceeds as if all real variables and arrays had been typed as double precision, and all floating point constants are assumed to be double precision. In addition, references to all library functions (intrinsic and basic external) of the real type are changed to reference the double precision equivalents of those functions. These changes do not appear on the source output listing, which is simply a printout of the source record images. The changes do appear on the object listing (if requested).

This option is normally requested when the single precision accuracy of an existing FORTRAN program is found to be insufficient. However, because of some inconsistencies which may arise in the usage of this option (see below), it generally is better to write a double precision program than to convert a floating point program using ADP. The following considerations should be taken into account when using this option:

1. Complex numbers are not converted to double precision.

2. Any programs which interface to the converted program should also be double precision so that arguments will be of the same type, and COMMON will be correctly aligned.

3. If a standard library routine is declared EXTERNAL, the compiler will not recognize it as one of the standard routines, and thus will not automatically substitute the equivalent double precision routine.

4. Operands used under the FORTRAN in-line assembly feature may be converted to double precision, but op-codes will not be changed.

Figures 2-3 and 2-4 demonstrate the function of the ADP option. Figure 2-3 was compiled without the ADP option, Figure 2-4 with the option. The differences are circled on the listings. Note that the variables X, Y and NUM, which would normally be single precision real types, are converted to double precision. Also, the constants 2.3 (real) and 17 (integer) are also converted to double precision. The external function F is assumed to be double precision, and references to the FORTRAN functions SIN and ABS are actually made to DSIN and DABS, as shown in the subprogram usage map. (In the case of DABS, the actual object generation--during the compiler Gen phase--does not require a call to this function, and so none appears in the object listing. The reference to DABS still appears in the subprogram map, because it was made during the Scan phase prior to object generation.)

ANsi (ANSI - compatible allocation)

This option allocates two words of memory instead of one to all integer and logical quantities. This is used where a program requires storage allocation to be ANSI compatible, since the ANSI standard specifies that integer, logical and real quantities must be the same size. In most instances this option will have no adverse effect on the program's operation, however, note the following exceptions:

1. Any operation which steps through each word of memory should not be used on an integer or logical buffer or array (e.g. ENCODE or DECODE statements) where the ANSI option is used.

2. Any programs interfacing to an ANSI program should also be ANSI to avoid any conflicting COMMON variables which are integers or logicals.

Figures 2-5 and 2-6 are examples of ANSI option usage. Figure 2-5 was created without the ANSI option, Figure 2-6 with the option. The differences are circled, and demonstrate the doubling in size of the integer and logical variables:

```
AGE 0001   09/04/74   17:18:45   FORTRAN (X3) COMPILATION
           OPTIONS:   LO

     C          DEMONSTRATE ADP OPTION
                REAL NUM
                NUM = 2.3
                X = F(NUM) + 17
                Y = ABS(SIN(X))
                END
```

```
AGE 0002   09/04/74   17:18:45   FORTRAN (X3) COMPILATION
           OPTIONS:   LO

CALAR ALLOCATION

OCN    NAME    TYPE       WORDS    LOCN    NAME    TYPE       WORDS

:0000  NUM     REAL           2    :0002   X       REAL           2
:0004  Y       REAL           2
```

Figure 2-3. Compilation without ADP Option Example

```
PAGE 0003   09/04/74   17:18:45   FORTRAN (X3) COMPILATION
            OPTIONS:   LO

0001 C           DEMONSTRATE ADP OPTION
0002        REAL NUM
0003        NUM = 2.3
               :0006 :F900 B          JST    *BP(F:RINT)
               :0007 :AA00 F          LDR    #RC0                 :4113:3333
               :0008 :9E08            STA    NUM
0004        X = F(NUM) + 17
               :0009 :0000            XIT
               :000A :F900 B          JST    *BP(F      )
               :000B :0001            DATA   1
               :000C :0000            DATA   NUM
               :000D :F900 B          JST    *BP(F:RREL)
               :000E :8A00 F          ADD    #RC1                 :4288:0000
               :000F :9E0D            STA    X
0005        Y = ABS(SIN(X))
               :0010 :0000            XIT
               :0011 :F900 B          JST    *BP(SIN    )
               :0012 :0001            DATA   1
               :0013 :0002            DATA   X
               :0014 :F900 B          JST    *BP(F:RREL)
               :0015 :0005            ABS
               :0016 :9E12            STA    Y
0006        END
               :0017 :0000            XIT
               :0018 :F900 B          JST    *BP(F:RSTO)
               :0019 :0000            DATA   0
               :001A :4113     #RC0   DATA   16659
               :001B :3333            DATA   13107
               :001C :4288     #RC1   DATA   17032
               :001D :0000            DATA   0

SUBPROGRAMS CALLED

NAME    TYPE     ARGS   NAME    TYPE     ARGS   NAME    TYPE     ARGS

F       REAL     1      ABS     REAL     1      SIN     REAL     1
F:RSTO  RUNTIME         F:RREL  RUNTIME         F:RINT  RUNTIME

ENTRY=:0006
PROGRAM SIZE=:001E WORDS
BASE PAGE USED=:0005 WORDS
COMPILATION COMPLETE  0 ERRORS
```

Figure 2-3. Compilation without ADP Option Example (Cont'd)

```
PAGE 0001   09/04/74   17:20:12   FORTRAN (X3) COMPILATION
            OPTIONS:   LO, AD

      C          DEMONSTRATE ADP OPTION
            REAL NUM
            NUM = 2.3
            X = F(NUM) + 17
            Y = ABS(SIN(X))
            END
```

```
PAGE 0002   09/04/74   17:20:12   FORTRAN (X3) COMPILATION
            OPTIONS:   LO, AD

SCALAR ALLOCATION

LOCN   NAME    TYPE        WORDS   LOCN   NAME   TYPE        WORDS

:0000  NUM     DOUBLE          4   :0004  X      DOUBLE          4
:0008  Y       DOUBLE          4
```

Figure 2-4. Compilation with ADP Option Example

```
PAGE 0003   09/04/74   17:20:12    FORTRAN (X3) COMPILATION
     ^      OPTIONS:   LO, AD

0001 C              DEMONSTRATE ADP OPTION
0002         REAL NUM
0003         NUM = 2.3
                  :000C :F900 B          JST    *BP(F:RINT)
                  :000D :B200 F          LDD    #RC0                     :4113:3333:
                  :000E :9E0E            STA    NUM
0004         X = F(NUM) + 17
                  :000F :0000            XIT
                  :0010 :F900 B          JST    *BP(F      )
                  :0011 :0001            DATA   1
                  :0012 :0000            DATA   NUM
                  :0013 :F900 B          JST    *BP(F:RDBL)
                  :0014 :8A00 F          ADD    #RC1                     :4288:0000:
                  :0015 :9E11            STA    X
0005         Y = ABS(SIN(X))
                  :0016 :0000            XIT
                  :0017 :F900 B          JST    *BP(DSIN )
                  :0018 :0001            DATA   1
                  :0019 :0004            DATA   X
                  :001A :F900 B          JST    *BP(F:RDBL)
                  :001B :0005            ABS
                  :001C :9E14            STA    Y
0006         END
                  :001D :0000            XIT
                  :001E :F900 B          JST    *BP(F:RSTO)
                  :001F :0000            DATA   0
                  :0020 :4113      #RC0  DATA   16659
                  :0021 :3333            DATA   13107
                  :0022 :3333            DATA   13107
                  :0023 :3333            DATA   13107
                  :0024 :4288      #RC1  DATA   17032
                  :0025 :0000            DATA   0
                  :0026 :0000            DATA   0
                  :0027 :0000            DATA   0

SUBPROGRAMS CALLED

NAME     TYPE     ARGS    NAME     TYPE     ARGS    NAME     TYPE     ARGS

F        DOUBLE   1       DABS     DOUBLE   1       DSIN     DOUBLE   1
F:RSTO   RUNTIME          F:RDBL   RUNTIME          F:RINT   RUNTIME

ENTRY=:000C
PROGRAM SIZE=:0028 WORDS
BASE PAGE USED=:0005 WORDS
COMPILATION COMPLETE   0 ERRORS
```

Figure 2-4. Compilation with ADP Option Example (Cont'd)

```
PAGE 0002  09/25/74  09:25:06  FORT:4 (A1)
 DO FILE:  FOUT    OPTIONS:

COMMON BLOCK/F:BCMN/ ALLOCATION  :0006 WORDS

LOCN   NAME    TYPE      WORDS   LOCN   NAME    TYPE      WORDS

:0000 BLNK    REAL         6    :0002 RROOT   REAL         2

COMMON BLOCK/LABLD / ALLOCATION  :0002 WORDS

LOCN   NAME    TYPE      WORDS   LOCN   NAME    TYPE      WORDS

:0000 LAB1    INTEGER      1    :0001 LAB2    INTEGER      1

ARRAY ALLOCATION

LOCN   NAME    TYPE      WORDS   LOCN   NAME    TYPE      WORDS

:0007 N       INTEGER      4

EQUIVALENCE ALLOCATION

LOCN   NAME    TYPE      WORDS   LOCN   NAME    TYPE      WORDS

:0008 JEQUIV  INTEGER      1    :0008 J       INTEGER      1

SCALAR ALLOCATION

LOCN   NAME    TYPE      WORDS   LOCN   NAME    TYPE      WORDS

:000C R       REAL         2    :000E S       REAL         2
:0010 L       LOGICAL      1    :0011 D       DOUBLE       4
:0015 C       COMPLEX      4    :0019 I       INTEGER      1
:001A K       INTEGER      1    :001B Q       INTEGER      1
:001C DROOT   DOUBLE       4    :0020 CROOT   COMPLEX      4
```

Figure 2-5. Listing without ANSI Option Example

```
PAGE 0002  09/25/74   09:27:34   FORT:4 (A1)
 30 FILE:  FOUT     OPTIONS:     AN

COMMON BLOCK/F:BCM / ALLOCATION   :0006 WORDS

LOCN   NAME    TYPE        WORDS   LOCN   NAME    TYPE        WORDS

:0000 BLNK     REAL            6   :0002 RROOT   REAL            2

COMMON BLOCK/LABLD / ALLOCATION   :0004 WORDS

LOCN   NAME    TYPE        WORDS   LOCN   NAME    TYPE        WORDS

:0000 LAB1    INTEGER          2   :0002 LAB2    INTEGER         2

ARRAY ALLOCATION

LOCN   NAME    TYPE        WORDS   LOCN   NAME    TYPE        WORDS

:0007 N       INTEGER          8

EQUIVALENCE ALLOCATION

LOCN   NAME    TYPE        WORDS   LOCN   NAME    TYPE        WORDS

:000F JEQUIV  INTEGER          2   :000F J       INTEGER         2

SCALAR ALLOCATION

LOCN   NAME    TYPE        WORDS   LOCN   NAME    TYPE        WORDS

:0011 R       REAL             2   :0013 S       REAL            2
:0015 L       LOGICAL          2   :0017 D       DOUBLE          4
:0019 C       COMPLEX          4   :001F I       INTEGER         2
:0021 K       INTEGER          2   :0023 Q       INTEGER         2
:0025 DROOT   DOUBLE           4   :0029 CROOT   COMPLEX         4
```

Figure 2-6. Listing with ANSI Option Example

TRace (Compile for execution with Trace function)

When the TRace option is specified, the compiler generates extra run time calls in the compiled program that cause it to print out trace information (on unit 6) in three places:

1. Whenever a labeled statement is reached, the message:

   xxxxxx LINE ddddd

   is printed before the statement is executed, where

   xxxxxx is the name of the program (F:MAIN if main program). If the name is the same as that on the previous trace line, it is not printed. In other words, the name will be printed once when the program is entered, and not again until a new program is entered (or returned to).

   ddddd is the source line number of the statement about to be executed.

2. When a SUBROUTINE or FUNCTION is entered, the message:

   xxxxxx ENTRY

   is printed immediately after entry. Again xxxxxx is the subprogram name, which will always be printed. Note that the tracing is done upon entry, not upon call. Therefore only subprograms that are compiled in TRACE mode will be traced.

3. When a RETURN statement is reached (whether or not labeled), the message:

   xxxxxx RETURN LINE ddddd

   is printed before executing the RETURN.

This information is sufficient to follow the flow of the program, since it will trace all jumps (the transfer point will be labeled) and all calls, except to library routines (which are assumed to operate correctly) and to subprograms not compiled in TRACE mode (which are also assumed to operate correctly). It is not necessary that all of the programs loaded be compiled in TRACE mode. As soon as certain parts are checked out, they can be compiled normally, so only the remaining parts are traced. Note that assembly language subprograms are not traced, nor are sections of in-line assembly language.

The following example demonstrates the use of the TRace option:

```
PAGE 0001   07/17/74   14:10:42   FURTRAN (X1) COMPILATION

0001             I = 5
0002 10          CALL MYSUB
0003 20          WRITE (6,30)
0004 30          FORMAT (' WRITE MESSAGE')
0005 40          I = I -1
0006             IF (I .EQ. 0) GO TO 50
0007             GO TO 10
0008 50          STOP 1
0009             END
```

```
PAGE 0001   07/17/74   14:10:42   FURTRAN (X1) COMPILATION

0010             SUBROUTINE MYSUB
0011             RETURN
0012             END
```

Note that the main program contains four labeled statements (line 2, 3, 5 and 8). Line 4, the format statement, is not traced since it is not executed. Also, line 2 contains a CALL to the subroutine, MYSUB.

The following lines were executed by this program when compiled without the TRace option:

```
WRITE MESSAGE
WRITE MESSAGE
WRITE MESSAGE
WRITE MESSAGE
WRITE MESSAGE
```

The following lines were output during execution of the same program, after being compiled with the TRace option:

```
  F:MAIN   LINE      2
  MYSUB  ENTRY
         RETURN LINE      11
  F:MAIN   LINE      3
WRITE MESSAGE
           LINE      5
           LINE      2
  MYSUB  ENTRY
         RETURN LINE      11
  F:MAIN   LINE      3
WRITE MESSAGE
           LINE      5
           LINE      2
  MYSUB  ENTRY
         RETURN LINE      11
```

(Continued on next page)

```
   F:MAIN   LINE      3
  WRITE MESSAGE
            LINE      5
            LINE      2
   MYSUB   ENTRY
           RETURN LINE      11
   F:MAIN   LINE      3
  WRITE MESSAGE
            LINE      5
            LINE      2
   MYSUB   ENTRY
           RETURN LINE      11
   F:MAIN   LINE      3
  WRITE MESSAGE
            LINE      5
            LINE      8
```

RTX (Compile for execution under the Real-Time Executive RTX/IOX)

This option must be specified when a FORTRAN program is to be compiled for execution as a task under RTX. The option causes references to common FORTRAN library subprograms to be made via the RTX SUBR: function; also, no execution address is output at the end of the compilation, since it is assumed that the task(s) will ultimately be linked to an assembled Mainline sequence (called F:MAIN).

A program run under RTX normally consists of a Mainline sequence and one or more tasks to be run simultaneously. Refer to the RTX User's Manual for a complete description of an RTX program. The following discussion encompasses only the differences between the standard RTX program and a FORTRAN program run under RTX.

A FORTRAN program is considered a "task" to RTX. Several FORTRAN (or non-FORTRAN, or intermixed) tasks may be linked together with a Mainline sequence, to be run simultaneously.

RTX Mainline Sequence

The Mainline sequence is simply a calling routine to initialize and begin each task using the RTX BEGIN: subroutine. Normally the Mainline is assembled using OS:ASM, while a FORTRAN task is compiled by the FORTRAN compiler using the RTX option, and having a TASK statement as its first source statement. The organization of the Mainline sequence is described in the RTX User's Manual. Additional considerations for a Mainline sequence which is to initiate FORTRAN tasks are described below. (See figure 2-7 for an example of a Mainline and two tasks.)

Mainline Entry Point (F:MAIN)

For proper linking under OS:LNK, the mainline sequence must contain as its entry point the label "F:MAIN". This label must also appear in a NAM directive at the start of the mainline sequence.

Input/Output Block (IOB)

A non-FORTRAN RTX program requires that each task contain an IOB (Input/Output Block) which contains pertinent information for I/O operations. Under FORTRAN, however, I/O information is expressed in FORTRAN I/O statements. This information is then converted by the FORTRAN/RTX I/O Interface module into the IOB format required by RTX. Thus the FORTRAN user does not supply the IOB.

Unit Assignment Table (UAT)

Executing a program (Fortran or otherwise) under OS control differs greatly from execution under RTX control. One important difference is the manner in which logical units are assigned to physical I/O devices. Under OS, this is accomplished by the /ASSIGN command. Under RTX, however, a Unit Assignment Table (UAT) must exist, which is a table of two-word entries, each providing a connection between a logical unit number and a physical I/O device. Thus RTX requires that device assignment be made at assembly time, rather than allowing dynamic assignment at execution time, as does OS.

In FORTRAN, the most convenient location for the UAT is within the assembled mainline program, and it is suggested that the user follow this practice to provide the greatest ease in changing the UAT when necessary. (It is because of the great variability in UAT construction, and the dependence of its organization on the FORTRAN unit numbers used as well as the physical devices configured on the user's system, that no standard UAT is included in the FORTRAN library modules.)

The UAT is simply a table of two-word entries for each logical unit which can be referenced within the IOX section of RTX, plus a terminating word containing the UAT word length. (Refer to the RTX User's Manual for a complete description, and see the RTX mainline example below, which contains a UAT.) The first word of each entry is the FORTRAN unit number. The second word of each entry is the address of the corresponding DIB (Device Information Block) table within RTX. A NAM directive to the label I:UAT must be included at the start of the Mainline program, as this is the name used by RTX/IOX when referencing the UAT. (I:UAT is defined as the last, rather than the first, word of the UAT.)

As mentioned in the RTX User's Manual, certain DIB's exist within RTX/IOX (for disk, line printer and teletype) which reference special FORTRAN drivers within RTX/IOX. This is because FORTRAN requires more capability within the driver than IOX normally supplies. The special teletype and printer drivers are needed to recognize carriage control characters. The special disk drivers handle record numbers internally, and can recognize and create end-of-file marks. Since an RTX mainline sequence may reference both FORTRAN and non-FORTRAN tasks, both types of DIB may be required. Fortran unit numbers in UAT entries should reference FORTRAN type DIB's, if they exist.

Note also that the standard disk DIB's in RTX/IOX each refer to a single file, or "extent" on the disk. Since there is no way for RTX to know before-hand how much of the disk or how many separate disk files the user may require, the disk DIB's have been established for the general case; each DIB refers to an entire disk platter and considers it a single file. Since in many cases an entire platter is an excessive amount of disk space to reserve for a single file, the user may wish to specify his own DIB, describing a different "extent" on the disk. The procedure for doing this is in the System Generation section of this manual.

Parameter Blocks

When the Mainline is to be used to call FORTRAN (as opposed to non-FORTRAN) tasks, a parameter block area and I/O buffer must be included in the mainline for each FORTRAN I/O call to be run simultaneously. (Since RTX does not know in advance how many tasks are to be run simultaneously, it is up to the user to reserve these areas.)

This implies that the user must determine the size required of the I/O buffer; in general, for binary (unformatted) I/O, 255 words should be reserved. For ASCII I/O, the size to be reserved is dependent on the type of device and the data to be output.

The user must reserve at least one parameter block. It may be useful to reserve more than one block in some cases; for example, when both ASCII and binary I/O are called for in a task, two blocks should be reserved, one containing a 66-word (for example) buffer for ASCII and the other containing a 255-word buffer, for binary I/O. In addition, certain error messages which are output by FORTRAN may require a parameter block while executing a task whose ASCII buffer is already in use. In any case, if a parameter block is needed, and none are currently available, the particular task will "hang-up" (within the interface) until one becomes available.

In general, the user should reserve an I/O length which is large enough to accommodate an I/O operation to a particular device, up to 255 words.

A parameter block is reserved as follows:

CHAN	F:PRAM	Chain to other parameter blocks
DATA	xx + xx	Length of I/O buffer (in bytes, where xx is the word length)
RES	85	Space for FORTRAN temp cells, parameters and IOB
RES	xx	I/O buffer. xx (word length) is determined by the user depending on the capabilities of the particular I/O device, as well as the needs of his FORTRAN tasks.
CHAN	F:PRAM	Next parameter block

Note that the chain reference must be to

"F:PRAM"

for each chain node. Note also that no parameter block is dedicated to any particular task; rather, the chain is used when a block is needed, to find an unused block for whatever task is about to perform I/O. This procedure occurs as follows:

When a FORTRAN task performs an I/O operation, the I/O interface is alerted. The interface then uses its own chain node to F:PRAM to find an unused parameter block, whose I/O buffer is of sufficient length, according to the length specified in the DIB of the applicable unit. Thus, once the buffer requirements are known to the interface (by means of the maximum record size within the unit's DIB) the lengths of the available I/O buffers are scanned in order to locate the smallest buffer which will be capable of holding the I/O data.

RTX task

A task is merely a FORTRAN program which has been compiled under the RTX option, and which contains a TASK statement as its first statement. The TASK statement defines the task name, which is referenced in the Mainline sequence during the call to the RTX BEGIN: routine.

Sample FORTRAN/RTX Listing

Figure 2-7 is an example of a FORTRAN Mainline and two tasks. The first task (TASK1) calculates and prints the square root of each integer from 1 to 50. The second does the same thing for numbers from 51 to 100. This causes both tasks to make calls to the SQRT external function routine, and to share the line printer for their output.

Mainline Example Description

Note that it is generally more convenient to assemble the Mainline sequence using OS:ASM, rather than to compile it in FORTRAN.

NAM directives must be included for the mainline sequence itself (F:MAIN) and for the Unit Assignment Table (I:UAT).

External references are required for the RTX routines used by F:MAIN:

RTX:
BEGIN:
END:

and for the DIB's referenced in the Unit Assignment Table:

D:TY00	(system teletype DIB)
D:LPF0	(FORTRAN line printer DIB)

```
PAGE  0001  10/13/75  16:45:19  FORTRAN / RTX MAINLINE ASSEMBLY
MALRD2 (A2) SI=        BO=

                               *THIS IS THE MAINLINE SEQUENCE FOR THE
                               *TWO-TASK EXAMPLE.
                               *
        0000                           NAM    F:MAIN,I:UAT
        0075
                                       EXTR   RTX:,BEGIN:,END:,D:TY00,D:LPF2
                                       EXTR   TASK1,TASK2
             0014              NN      EQU    20         NUMBER OF RTX WORKING TABLES
        0000                           REL    0
             0000              F:MAIN  EQU    $          EXECUTION ENTRY POINT
        0000 F900 0000                 JST    RTX:       INITIALIZE THE TASKS
        0001 0014                      DATA   NN         NUMBER OF WORKING TABLES
        0002 0006                      DATA   WKAREA     ADDRESS OF WKG TABLES
        0003 0800                      HLT               STOP ON UNSUCCESSFUL
                               *                       INITIATION
        0004 F266 006A                 JMP    START      GO EXECUTE THE TASKS
        0005                   ZBG     REF               TO PULL IN ZEBUG
        0006 0000              WKAREA  RES    NN+NN+NN+NN+NN,0   RTX WORKING TBLS
        006A F900 0000         START   JST    BEGIN:     BEGIN TASK 1
        006B 0000                      DATA   TASK1
        006C 0064                      DATA   100        AT PRIORITY 100
        006D F900 0000                 JST    BEGIN:     BEGIN TASK 2
        006E 0000                      DATA   TASK2
        006F 0064                      DATA   100
        0070 F900 0000                 JST    END:       END INITIALIZATION SEQUENCE
                               *
                               *       UNIT ASSIGNMENT TABLE
                               *
        0071 C3CF              UATTOP  DATA   'CO',D:TY00 CO DEVICE FOR ERROR MSGS
        0072 0000
        0073 0006                      DATA   6,D:LPF2 FORTRAN UNIT 6=PRINTER
        0074 0000
        0075 FFFA              I:UAT   DATA   UATTOP-$-2  UAT LENGTH
                               *
                               *       PARAMETER BLOCKS, I/O BUFFERS
                               *
        0076                           CHAN   F:PRAM     CHAIN NODE
        0077 0084                      DATA   132        BUFFER BYTE LENGTH
        0078                           RES    85         FORTRAN TEMP CELLS
                               *                       AND IOB
        00CD                           RES    66         I/O BUFFER (132 BYTES)
                               *
        010F                           CHAN   F:PRAM     CHAIN NODE
        0110 0084                      DATA   132        I/O BUFFER BYTE LENGTH
        0111                           RES    85         FORTRAN TEMP CELLS
                               *                       AND IOB
        0166                           RES    66         I/O BUFFER (132 BYTES)
                               *
        01A8                           CHAN   F:PRAM     CHAIN NODE
        01A9 0084                      DATA   132        I/O BUFFER BYTE LENGTH
        01AA                           RES    85         FORTRAN TEMP CELLS
```

Figure 2-7. FORTRAN/RTX Example

```
PAGE  0002  08/12/74  10:39:35  FORTRAN / RTX MAINLINE ASSEMBLY

LINE  LOC   INST ADDR  LABEL  MNEM OPERAND   COMMENT
0050                   *                     AND IUB
0051  01FF                    RES  66        I/O BUFFER (132 BYTES)
0052                   *
0053        0000              END  F:MAIN

0000  ERRORS
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
PAGE 0001  08/12/74  09:12:55   FORTRAN (X3) COMPILATION
           OPTIONS:  LO, RT

              TASK TASK1
        C     THIS TASK CALCULATES AND PRINTS NUMBERS
        C     FROM 1 TO 50, AND THEIR SQUARE ROOTS.
        C
        C     LOOP FROM 1 TO 50
              DO 10 JNUM = 1,50
        C
        C     CONVERT NUMBER TO FLOATING POINT FOR SQRT
              RNUM=JNUM
        C
        C     CALCULATE SQUARE ROOT
              SQROOT = SQRT (RNUM)
        C
        C     PRINT TASK NAME, NUMBER, SQUARE ROOT
              WRITE (6,20) JNUM, SQROOT
        20    FORMAT (' TASK1    N=',I3,',   SQRT=',F7.3)
        C
        C     DO NEXT NUMBER
        10    CONTINUE
        C
        C     AT END, DISPLAY TASK NO. AND TERMINATE
              STOP 1
              END
```

```
PAGE 0002  08/12/74  09:12:55   FORTRAN (X3) COMPILATION
           OPTIONS:  LO, RT

SCALAR ALLOCATION

LOCN   NAME    TYPE       WORDS    LOCN   NAME   TYPE     WORDS

:0011  JNUM    INTEGER      1      :0012  RNUM   REAL       2
:0014  SQROOT  REAL         2
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
PAGE 0003   08/12/74  09:12:55   FORTRAN (X3) COMPILATION
            OPTIONS:  LO, RT

              TASK TASK1
     C        THIS TASK CALCULATES AND PRINTS NUMBERS
     C        FROM 1 TO 50, AND THEIR SQUARE ROOTS.
     C
     C        LOOP FROM 1 TO 50
              DO 10 JNUM = 1,50
                   :0016 :C401            LXP    1
                   :0017 :EE06     #M2    STX    JNUM
     C
     C        CONVERT NUMBER TO FLOATING POINT FOR SQRT
              RNUM=JNUM
                   :0018 :B607            LDA    JNUM
                   :0019 :F900 B          JST    *BP(F:RINT)
                   :001A :0002            REL
                   :001B :9E09            STA    RNUM
     C
     C        CALCULATE SQUARE ROOT
              SQROOT = SQRT (RNUM)
                   :001C :0000            XIT
                   :001D :F900 B          JST    *BP(SUBR: )
                   :001E :0000            DATA   SQRT
                   :001F :0001            DATA   1
                   :0020 :0012            DATA   RNUM
                   :0021 :F900 B          JST    *BP(F:RREL)
                   :0022 :9E0E            STA    SQROOT
     C
     C        PRINT TASK NAME, NUMBER, SQUARE ROOT
              WRITE (6,20) JNUM, SQROOT
                   :0023 :0000            XIT
                   :0024 :F900 B          JST    *BP(F:RWF )
                   :0025 :0000 F          DATA   #IC2                        :0006
                   :0026 :0000            DATA   #20
                   :0027 :F900 B          JST    *BP(F:RIOL)
                   :0028 :0011            DATA   JNUM
                   :0029 :F900 B          JST    *BP(F:RROL)
                   :002A :0014            DATA   SQROOT
                   :002B :F900 B          JST    *BP(F:RSIO)
     20       FORMAT ('  TASK1    N=',I3,',   SQRT=',F7.3)
                   :0000 :A8A7     #20    TEXT   '('  TASK1    N=',I3,',   SQRT=',F7.
                   :0010 :B3A9            TEXT   '3)'
     C
     C        DO NEXT NUMBER
     10       CONTINUE
     C
                   :002C :E61B     #10    LDX    JNUM
                   :002D :C201            AXI    1
                   :002E :0030            TXA
                   :002F :0D32            SAI    50
                   :0030 :2109            JAL    #M2
     C        AT END, DISPLAY TASK NO. AND TERMINATE
              STOP 1
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
AGE 0004   08/12/74   09:12:55    FORTRAN (X3) COMPILATION
           OPTIONS:   LO, RT

               :0031 :F904 B           JST     *BP(F:RSTO)
               :0032 :0001             DATA    1
0025          END
               :0033 :0006     *IC2    DATA    6

UBPROGRAMS CALLED

AME      TYPE      ARGS     NAME      TYPE      ARGS     NAME      TYPE      ARGS

QRT      REAL      1        F:RUF     RUNTIME            F:RIOL    RUNTIME
:RROL    RUNTIME            F:RSIO    RUNTIME            F:RSTO    RUNTIME
:RREL    RUNTIME            F:RFZ     RUNTIME            F:RFF     RUNTIME
RIT      RUNTIME            SUPP:     RUNTIME

TATEMENT LABELS

OCN    LABEL   USE           LOCN   LABEL   USE           LOCN   LABEL   USE

0020  #10     DO END        :0100  #20     FORMAT        :0017  #M2

  Y=:0116
  RAM SIZE=:0034 WORDS
  SE PAGE USED=:0006 WORDS
OMPILATION COMPLETE    0 ERRORS
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
PAGE 0001   08/12/74   09:12:55   FORTRAN (X3) COMPILATION
            OPTIONS:   LO, RT

          TASK TASK2
C         THIS TASK CALCULATES AND PRINTS NUMBERS
C         FROM 51 TO 100, AND THEIR SQUARE ROOTS.
C
C         LOOP FROM 51 TO 100
          DO 10 JNUM = 51,100
C
C         CONVERT NUMBER TO FLOATING POINT FOR SQRT
          RNUM=JNUM
C
C         CALCULATE SQUARE ROOT
          SQROOT = SQRT (RNUM)
C
C         PRINT TASK NAME, NUMBER, SQUARE ROOT
          WRITE (6,20) JNUM, SQROOT
   20     FORMAT (' TASK2   N=',I3,',   SQRT=',F7.3)
C
C         DO NEXT NUMBER
   10     CONTINUE
C
C         AT END, DISPLAY TASK NO. AND TERMINATE
          STOP 2
          END
```

```
PAGE 0002   08/12/74   09:12:55   FORTRAN (X3) COMPILATION
            OPTIONS:   LO, RT

SCALAR ALLOCATION

LOCN    NAME     TYPE        WORDS    LOCN    NAME    TYPE       WORDS

'00011  JNUM     INTEGER         1    '00012  RNUM    REAL           2
'00014  SQROOT   REAL            2
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
PAGE 0003  08/12/74  09:12:55   FORTRAN (X3) COMPILATION
           OPTIONS:  LO, RI

             TASK TASK2
      C      THIS TASK CALCULATES AND PRINTS NUMBERS
      C      FROM 51 TO 100, AND THEIR SQUARE ROOTS.
      C
      C      LOOP FROM 51 TO 100
             DO 10 JNUM = 51,100
                  :0016 :C433                LXP     51
                  :0017 :EE06     #M2        STX     JNUM
      C
      C      CONVERT NUMBER TO FLOATING POINT FOR SQRT
             RNUM=JNUM
                  :0018 :B607                LDA     JNUM
                  :0019 :F910 B              JST     *BP(F:RINT)
                  :001A :0002                REL
                  :001B :9E09                STA     RNUM
      C
      C      CALCULATE SQUARE ROOT
             SQROOT = SQRT (RNUM)
                  :001C :0030                XIT
                  :001D :F900 B              JST     *BP(SUBR: )
                  :001E :0030                DATA    SQRT
                  :001F :0001                DATA    1
                  :0020 :0012                DATA    RNUM
                  :0021 :F900 B              JST     *BP(F:RREL)
                  :0022 :9E0E                STA     SQROOT
      C
      C      PRINT TASK NAME, NUMBER, SQUARE ROOT
             WRITE (6,20) JNUM, SQROOT
                  :0023 :0000                XIT
                  :0024 :F900 B              JST     *BP(F:RWF )
                  :0025 :0000 F              DATA    #IC3                    :0006
                  :0026 :0000                DATA    #20
                  :0027 :F900 B              JST     *BP(F:RIOL)
                  :0028 :0011                DATA    JNUM
                  :0029 :F900 B              JST     *BP(F:RROL)
                  :002A :0014                DATA    SQROOT
                  :002B :F900 B              JST     *BP(F:RSIO)
      20     FORMAT ('  TASK2    N=',I3,',   SQRT=',F7.3)
                  :0000 :A8A7     #20        TEXT    '('  TASK2    N=',I3,',   SQRT=',F7.'
                  :0010 :B3A9                TEXT    '3)'
      C
      C      DO NEXT NUMBER
      10     CONTINUE
      C
                  :002C :E61B     #10        LDX     JNUM
                  :002D :C201                AXI     1
                  :002E :0030                TXA
                  :002F :0D64                SAI     100
                  :0030 :2109                JAL     #M2
      C      AT END, DISPLAY TASK NO. AND TERMINATE
             STOP 2
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
PAGE 0004   08/12/74   09:12:55   FORTRAN (X3) COMPILATION
            OPTIONS:   LO, RT

                :0031 :F900 B          JST    *BP(F:RSTO)
                :0032 :0002            DATA   2
0046        END
                :0033 :0006   #IC3     DATA   6

SUBPROGRAMS CALLED

NAME     TYPE     ARGS    NAME     TYPE     ARGS    NAME     TYPE     ARGS

SQRT     REAL     1       F:RWF    RUNTIME          F:RIOL   RUNTIME
F:RROL   RUNTIME          F:RSIO   RUNTIME          F:RSTO   RUNTIME
F:RREL   RUNTIME          F:RFZ    RUNTIME          F:RFF    RUNTIME
F:RINT   RUNTIME          SUBR:    RUNTIME

STATEMENT LABELS

LOCN   LABEL   USE         LOCN   LABEL   USE          LOCN   LABEL   USE

:002C #10      DO END      :0000 #20      FORMAT       :0017 #M2

ENTRY=:0016
PROGRAM SIZE=:0034 WORDS
BASE PAGE USED=:0008 WORDS
COMPILATION COMPLETE  0 ERRORS
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
TASK1   N=  1,   SQRT=  1.000
TASK2   N= 51,   SQRT=  7.141
TASK1   N=  2,   SQRT=  1.414
TASK2   N= 52,   SQRT=  7.211
TASK1   N=  3,   SQRT=  1.732
TASK2   N= 53,   SQRT=  7.280
TASK1   N=  4,   SQRT=  2.000
TASK2   N= 54,   SQRT=  7.348
TASK1   N=  5,   SQRT=  2.236
TASK2   N= 55,   SQRT=  7.416
TASK1   N=  6,   SQRT=  2.449
TASK2   N= 56,   SQRT=  7.483
TASK1   N=  7,   SQRT=  2.646
TASK2   N= 57,   SQRT=  7.550
TASK1   N=  8,   SQRT=  2.828
TASK2   N= 58,   SQRT=  7.616
TASK1   N=  9,   SQRT=  3.000
TASK2   N= 59,   SQRT=  7.681
TASK1   N= 10,   SQRT=  3.162
TASK2   N= 60,   SQRT=  7.746
TASK1   N= 11,   SQRT=  3.317
TASK2   N= 61,   SQRT=  7.810
TASK1   N= 12,   SQRT=  3.464
TASK2   N= 62,   SQRT=  7.874
TASK1   N= 13,   SQRT=  3.606
TASK2   N= 63,   SQRT=  7.937
TASK1   N= 14,   SQRT=  3.742
TASK2   N= 64,   SQRT=  8.000
TASK1   N= 15,   SQRT=  3.873
TASK2   N= 65,   SQRT=  8.062
TASK1   N= 16,   SQRT=  4.000
TASK2   N= 66,   SQRT=  8.124
TASK1   N= 17,   SQRT=  4.123
TASK2   N= 67,   SQRT=  8.185
TASK1   N= 18,   SQRT=  4.243
TASK2   N= 68,   SQRT=  8.246
TASK1   N= 19,   SQRT=  4.359
TASK2   N= 69,   SQRT=  8.307
TASK1   N= 20,   SQRT=  4.472
TASK2   N= 70,   SQRT=  8.367
TASK1   N= 21,   SQRT=  4.583
TASK2   N= 71,   SQRT=  8.426
TASK1   N= 22,   SQRT=  4.690
TASK2   N= 72,   SQRT=  8.485
TASK1   N= 23,   SQRT=  4.796
TASK2   N= 73,   SQRT=  8.544
TASK1   N= 24,   SQRT=  4.899
TASK2   N= 74,   SQRT=  8.602
TASK1   N= 25,   SQRT=  5.000
TASK2   N= 75,   SQRT=  8.660
TASK1   N= 26,   SQRT=  5.099
TASK2   N= 76,   SQRT=  8.718
TASK1   N= 27,   SQRT=  5.196
TASK2   N= 77,   SQRT=  8.775
TASK1   N= 28,   SQRT=  5.292
TASK2   N= 78,   SQRT=  8.832
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

```
TASK1    N= 29,    SQRT=   5.385
TASK2    N= 79,    SQRT=   8.888
TASK1    N= 30,    SQRT=   5.477
TASK2    N= 80,    SQRT=   8.944
TASK1    N= 31,    SQRT=   5.568
TASK2    N= 81,    SQRT=   9.000
TASK1    N= 32,    SQRT=   5.657
TASK2    N= 82,    SQRT=   9.055
TASK1    N= 33,    SQRT=   5.745
TASK2    N= 83,    SQRT=   9.110
TASK1    N= 34,    SQRT=   5.831
TASK2    N= 84,    SQRT=   9.165
TASK1    N= 35,    SQRT=   5.916
TASK2    N= 85,    SQRT=   9.220
TASK1    N= 36,    SQRT=   6.000
TASK2    N= 86,    SQRT=   9.274
TASK1    N= 37,    SQRT=   6.083
TASK2    N= 37,    SQRT=   9.327
TASK1    N= 38,    SQRT=   6.164
TASK2    N= 88,    SQRT=   9.381
TASK1    N= 39,    SQRT=   6.245
TASK2    N= 89,    SQRT=   9.434
TASK1    N= 40,    SQRT=   6.325
TASK2    N= 90,    SQRT=   9.487
TASK1    N= 41,    SQRT=   6.403
TASK2    N= 91,    SQRT=   9.539
TASK1    N= 42,    SQRT=   6.481
TASK2    N= 92,    SQRT=   9.592
TASK1    N= 43,    SQRT=   6.557
TASK2    N= 93,    SQRT=   9.644
TASK1    N= 44,    SQRT=   6.633
TASK2    N= 94,    SQRT=   9.695
TASK1    N= 45,    SQRT=   6.708
TASK2    N= 95,    SQRT=   9.747
TASK1    N= 46,    SQRT=   6.782
TASK2    N= 96,    SQRT=   9.798
TASK1    N= 47,    SQRT=   6.856
TASK2    N= 97,    SQRT=   9.849
TASK1    N= 48,    SQRT=   6.928
TASK2    N= 98,    SQRT=   9.899
TASK1    N= 49,    SQRT=   7.000
TASK2    N= 99,    SQRT=   9.950
TASK1    N= 50,    SQRT=   7.071
TASK2    N=100,    SQRT=  10.000
```

Figure 2-7. FORTRAN/RTX Example (Cont'd)

and for the tasks to be run simultaneously:

TASK1
TASK2

The equated value "NN" specifies the number of RTX work area blocks needed for the two tasks (refer to the RTX User's Manual for a discussion of how to determine the number of blocks required).

F:MAIN is the Mainline entry point where the tables and tasks are initialized by the "RTX:" routine.

"WKAREA" is the actual work area reserved for RTX usage; its size is the number of blocks (NN) times 5.

"START" is the point at which the tasks are initiated, by calls to the RTX BEGIN: routine. Note that in this example both tasks are begun at the same priority (100). Thus the tasks will vie with each other for the use of the printer and the library functions they both require. (Refer to the RTX User's Manual for a discussion of task priorities.)

After the tasks have been initiated, a call is made to the RTX END: routine to terminate the mainline sequence.

The Unit Assignment Table (UAT) begins at "UATTOP" and ends at I:UAT. Note that the NAM directive must point to the end of the UAT, not the start, and it must be called "I:UAT:"; this is the name RTX references externally to access the table. Each table entry is two words in length, the first being the FORTRAN unit number referenced in the tasks' I/O statements, and the second being the DIB label corresponding to the physical device. Refer to the RTX User's Manual for a complete list of DIB labels. In addition to the unit numbers, an entry must exist for a 'CO' device, for use by the FORTRAN PAUSE and STOP calls.

The last word in the table represents the negative length of the table (including the length word itself) plus one, that is, -(L + 1).

There are three parameter blocks in the example. Two of them are used for the printer [illegible]

T3 (Compile for Execution on an LSI-3/05 Processor

This option must be specified when a FORTRAN program is to be compiled for execution in an LSI-3/05 processor. Since OS is not supported on the 3/05, FORTRAN will assume that the RTX option is required, even if you do not specify RT as a parameter. Therefore, everthing described in the RTX option (above) automatically applies to the T3 option as well.

The sample listings shown above in the RTX option discussion are reproduced below in LSI-3/05 object code (see Figure 2-8). The only real differences in the two sets of examples are the actual machine language code, of course, and the fact that the 3/05 Mainline sequence assumes the use of an I/O Distributor system for input and output. (The RTX User's Manual contains the DIB names for these devices.) Also, certain in-line assembly language instructions do not exist on the LSI-3/05, and are performed by an emulator routine which is part of the FORTRAN library for the 3/05 (F3RXLB). These instructions are marked with an asterisk in Section 8 of the FORTRAN Reference Manual. Note also that three of these instructions (SCM, SCMB and IPX) are not allowed under the T3 option.

```
PAGE  0001   04/27/76   18:57:35  FORTRAN / RTX MAINLINE ASSEMBLY
MACRO3 (A2) SI=          BO= OMAIN

                              *THIS IS THE MAINLINE SEQUENCE FOR THE
                              *TWO-TASK EXAMPLE.
                              *
        0000                           NAM    F:MAIN,T:UAT
        0075
                                       LOAD   IONIT:
                                       EXTR   RTX:,BEGIN:,END:,D:TY00,D:LPED
                                       EXTR   TASK1,TASK2
                  0014        NN       EQU    20        NUMBER OF RTX WORKING TABLES
        0000                           REL    0
                  0000        F:MAIN   EQU    $         EXECUTION ENTRY POINT
        0000 B000 0000                 JST    RTX:      INITIALIZE THE TASKS
        0001 0014                      DATA   NN        NUMBER OF WORKING TABLES
        0002 0006                      DATA   WKAREA    ADDRESS OF WKG TABLES
        0003 0E0D                      HLT              STOP ON UNSUCCESSFUL
                              *                       INITIATION
        0004 9EF5 006A                 JMP    START     GO EXECUTE THE TASKS
        0005                  ZBG      REF              TO PULL IN ZEBUG
        0006 0000             WKAREA   RES    NN+NN+NN+NN+NN,0   RTX WORKING TBLS
        006A B000 0000        START    JST    BEGIN:    BEGIN TASK 1
        006B 0000                      DATA   TASK1
        006C 0064                      DATA   100       AT PRIORITY 100
        006D B000 0000                 JST    BEGIN:    BEGIN TASK 2
        006E 0000                      DATA   TASK2
        006F 0064                      DATA   100
        0070 B000 0000                 JST    END:      END INITIALIZATION SEQUENCE
                              *
                              *        UNIT ASSIGNMENT TABLE
                              *
        0071 C3CF             UATTOP   DATA   'CO',D:TY00 CO DEVICE FOR ERROR MSGS
        0072 0000
        0073 0006                      DATA   6,D:LPED FORTRAN UNIT 6=PRINTER
        0074 0000
        0075 FFFA             I:UAT    DATA   UATTOP-$-2  UAT LENGTH
                              *
                              *        PARAMETER BLOCKS, I/O BUFFERS
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05

```
PAGE 0001  04/27/76  18:59:41  FORT:4 (B0)
 BO FILE:  TASKS   OPTIONS:    I3  LO

            TASK TASK1
      C     THIS TASK CALCULATES AND PRINTS NUMBERS
      C     FROM 1 TO 50, AND THEIR SQUARE ROOTS.
      C
      C     LOOP FROM 1 TO 50
            DO 10 JNUM = 1,50
      C
      C     CONVERT NUMBER TO FLOATING POINT FOR SQRT
            RNUM=JNUM
      C
      C     CALCULATE SQUARE ROOT
            SQROOT = SQRT (RNUM)
      C
      C     PRINT TASK NAME, NUMBER, SQUARE ROOT
            WRITE (6,20) JNUM, SQROOT
      20    FORMAT (' TASK1   N=',I3,',   SQRT=',F7.3)
      C
      C     DO NEXT NUMBER
      10    CONTINUE
      C
      C     AT END, DISPLAY TASK NO. AND TERMINATE
            STOP 1
            END
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0002   04/27/76   18:57:35  FORTRAN / RTX MAINLINE ASSEMBLY
MACRO3 (A2) SI=              BO= OMAIN

                        *
      0076                         CHAN     F:PRAM      CHAIN NODE
      0077  0084                   DATA     132         BUFFER BYTE LENGTH
      0078                         RES      85          FORTRAN TEMP CELLS
                        *                            AND IOB
      00CD                         RES      66          I/O BUFFER (132 BYTES)
                        *
      010F                         CHAN     F:PRAM      CHAIN NODE
      0110  0084                   DATA     132         I/O BUFFER BYTE LENGTH
      0111                         RES      85          FORTRAN TEMP CELLS
                        *                            AND IOB
      0166                         RES      66          I/O BUFFER (132 BYTES)
                        *
      01A8                         CHAN     F:PRAM      CHAIN NODE
      01A9  0084                   DATA     132         I/O BUFFER BYTE LENGTH
      01AA                         RES      85          FORTRAN TEMP CELLS
                        *                            AND IOB
      01FF                         RES      66          I/O BUFFER (132 BYTES)
                        *
            0000                   END      F:MAIN

0000  ERRORS
0000  WARNING
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0002  04/27/76  18:59:41  FORT:4 (BO)
 BO FILE:  TASKS   OPTIONS:    13  LO

SCALAR ALLOCATION

LOCN  NAME    TYPE       WORDS  LOCN  NAME    TYPE       WORDS  LOCN  NAME    TYPE       WORDS

:0011 JNUM    INTEGER        1  :0012 RNUM    REAL           2  :0014 SQROOT  REAL           2
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0003  04/27/76  18:59:41  FORT:4 (B0)
 B0 FILE:  TASKS   OPTIONS:     I3  L0

0001          TASK TASK1
0002 C        THIS TASK CALCULATES AND PRINTS NUMBERS
0003 C        FROM 1 TO 50, AND THEIR SQUARE ROOTS.
0004 C
0005 C        LOOP FROM 1 TO 50
0006          DO 10 JNUM = 1,50
                 :0016 :2901             LXP    1
                 :0017 :A579    #M2      STX    JNUM
0007 C
0008 C        CONVERT NUMBER TO FLOATING POINT FOR SQRT
0009          RNUM=JNUM
                 :0018 :8278             LDA    JNUM
                 :0019 :B000 B           JST    *BP(F:PINT)
                 :001A :0002             REL
                 :001B :8676             STA    RNUM
0010 C
0011 C        CALCULATE SQUARE ROOT
0012          SQROOT = SQRT (RNUM)
                 :001C :0000             XIT
                 :001D :B000 B           JST    *BP(SUBP: )
                 :001E :0000             DATA   SQRT
                 :001F :0001             DATA   1
                 :0020 :0012             DATA   RNUM
                 :0021 :B000 B           JST    *BP(F:RRFL)
                 :0022 :8671             STA    SQROOT
0013 C
0014 C        PRINT TASK NAME, NUMBER, SQUARE ROOT
0015          WRITE (6,20) JNUM, SQROOT
                 :0023 :0000             XIT
                 :0024 :B000 B           JST    *BP(F:RWF )
                 :0025 :0000 F           DATA   #IC2                 :0006
                 :0026 :0000             DATA   #20
                 :0027 :B000 B           JST    *BP(F:RIOL)
                 :0028 :0011             DATA   JNUM
                 :0029 :B000 B           JST    *BP(F:RPOL)
                 :002A :0014             DATA   SQROOT
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0004  04/27/76  19:59:41  FORT:4 (80)
 RP FILE:  TASKS   OPTIONS:     I3  L0

                :0028 :B000 B              JST    *RP(F:RSTO)
 0016 20       FORMAT (' TASK1    N=',I3,'.   SQRT=',F7.3)
                :000F :A647    #20         TEXT   '(' TASK1    N=',I3,',   SQRT=',F7.'
                :0010 :B3A9                TEXT   '3)'
 0017 C
 0018 C        NEXT NUMBER
 0019 10       CONTINUE
 0020 C
                :002C :A264    #10         LDX    INUM
                :002D :2801                AXI    1
                :002E :0020                TXA
                :002F :0ACE                SAT    50
                :0030 :12A6                JAL    #M2
 0021 C        AT END, DISPLAY TASK NO. AND TERMINATE
 0022          STOP 1
                :0031 :B000 B              JST    *RP(F:RSTO)
                :0032 :0001                DATA   1
 0023          END
                :0033 :0006    #IC2        DATA   6

 SUBPROGRAMS CALLED

 NAME    TYPE    ARGS   NAME    TYPE    ARGS   NAME    TYPE    ARGS   NAME    TYPE    ARGS

 SQRT    REAL     1     F:RNF   RUNTIME        F:RIOL  RUNTIME        F:RROL  RUNTIME
 F:RSIO  RUNTIME        F:RSIO  RUNTIME        F:RU06  RUNTIME        F:RREL  RUNTIME
 F:RF7   RUNTIME        F:RFF   RUNTIME        F:RLS3  RUNTIME        F:RINT  RUNTIME
 SURF:   RUNTIME

 STATEMENT LABELS

 LOCN  LABEL  USE         LOCN  LABEL  USE         LOCN  LABEL  USE         LOCN  LABEL  USE

 :002C #10    DO END      :000F #20    FORMAT      :0017 #M2

 ENTRY=:0014
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0005  04/27/76  18:59:41  FORT:4 (80)
 BO FILE:  TASKS   OPTIONS:    I3  LO

PROGRAM SIZE=:0034 WORDS
BASE PAGE USED=:0008 WORDS
COMPILATION COMPLETE  0 ERRORS
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0001   04/27/76   18:59:41   FORT:4 (B0)
 B0 FILE:  TASKS   OPTIONS:    T3  LO

             TASK TASK2
C            THIS TASK CALCULATES AND PRINTS NUMBERS
C            FROM 51 TO 100, AND THEIR SQUARE ROOTS.
C
C            LOOP FROM 51 TO 100
             DO 10 JNUM = 51,100
C
C            CONVERT NUMBER TO FLOATING POINT FOR SQRT
             RNUM=JNUM
C
C            CALCULATE SQUARE ROOT
             SQROOT = SQRT (RNUM)
C
C            PRINT TASK NAME, NUMBER, SQUARE ROOT
             WRITE (6,20) JNUM, SQROOT
  20         FORMAT (' TASK2   N=',I3,',   SQRT=',F7.3)
C
C            DO NEXT NUMBER
  10         CONTINUE
C
C            AT END, DISPLAY TASK NO. AND TERMINATE
             STOP 2
             END
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0002  04/27/76  18:59:41  FORT:4 (30)
 RO FILE:  TASKS  'OPTIONS:    I3  LO

SCALAR ALLOCATION

LOCN  NAME   TYPE      WORDS   LOCN  NAME   TYPE      WORDS   LOCN   NAME   TYPE     WORDS

:0011 JNUM   INTEGER      1    :0012 RNUM   REAL         2    :0014 SQROOT REAL         2
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0003   04/27/76   18:59:41   FORT:4 (BO)
 BO FILE:   TASKS    OPTIONS:     T3  LO

0024            TASK TASK2
0025 C          THIS TASK CALCULATES AND PRINTS NUMBERS
0026 C          FROM 51 TO 100, AND THEIR SQUARE ROOTS.
0027 C
0028 C          LOOP FROM 51 TO 100
0029            DO 10 JNUM = 51,100
                   :0016 :2933              LXP    51
                   :0017 :A679    #M2       STX    JNUM
0030 C
0031 C          CONVERT NUMBER TO FLOATING POINT FOR SQRT
0032            RNUM=JNUM
                   :0018 :8278              LDA    JNUM
                   :0019 :BD00 B            JST    *BP(F:PINT)
                   :001A :0002              PFL
                   :001B :8676              STA    RNUM
0033 C
0034 C          CALCULATE SQUARE ROOT
0035            SQROOT = SQRT (RNUM)
                   :001C :0000              XIT
                   :001D :BD00 B            JST    *BP(SUBR: )
                   :001E :0000              DATA   SQRT
                   :001F :0001              DATA   1
                   :0020 :0012              DATA   RNUM
                   :0021 :BD00 B            JST    *BP(F:RBFL)
                   :0022 :8671              STA    SQROOT
0036 C
0037 C          PRINT TASK NAME, NUMBER, SQUARE ROOT
0038            WRITE (6,20) JNUM, SQROOT
                   :0023 :0000              XIT
                   :0024 :BD00 B            JST    *BP(F:RWF )
                   :0025 :0000 F            DATA   #IC3                     :0006
                   :0026 :0000              DATA   #20
                   :0027 :BD00 B            JST    *BP(F:RIOL)
                   :0028 :0011              DATA   JNUM
                   :0029 :BD00 B            JST    *BP(F:RBOL)
                   :002A :0014              DATA   SQROOT
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0004  04/27/76  18:59:41  FORT:4 (B0)
 BC FILE:  TASKS   OPTIONS:    I3  LO

                 :002B :BD00 B          JST    *BP(F:RSTO)
0039 20         FORMAT (' TASK2   N=',I3,',   SQRT=',F7.3)
                 :0000 :A8A7    #20     TEXT   '(' TASK2   N=',I3,',   SQRT=',F7.'
                 :0010 :B3A9            TEXT   '3)'
0040 C
0041 C          DO NEXT NUMBER
0042 10         CONTINUE
0043 C
                 :002C :A264    #10     LDX    JNUM
                 :002D :2801            AXI    1
                 :002E :0020            TXA
                 :002F :0A9C            SAT    100
                 :0030 :1246            JAL    #M2
0044 C          AT END, DISPLAY TASK NO. AND TERMINATE
0045            STOP 2
                 :0031 :BD00 B          JST    *BP(F:RSTO)
                 :0032 :0002            DATA   2
0046            END
                 :0033 :0006    #IC3    DATA   6

SUBPROGRAMS CALLED

NAME    TYPE     ARGS  NAME    TYPE     ARGS  NAME    TYPE     ARGS  NAME    TYPE     ARGS

SQRT    REAL     1     F:RWF   RUNTIME        F:RIOL  RUNTIME        F:RROL  RUNTIME
F:RSIO  RUNTIME        F:RSTO  RUNTIME        F:RU06  RUNTIME        F:RREL  RUNTIME
F:RFZ   RUNTIME        F:RFF   RUNTIME        F:RLS3  RUNTIME        F:RINT  RUNTIME
SUBR:   RUNTIME

STATEMENT LABELS

LOCN  LABEL  USE         LOCN  LABEL  USE         LOCN  LABEL  USE         LOCN  LABEL  USE

:002C #10    DO END      :0000 #20    FORMAT      :0017 #M2

ENTRY=:0016
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

```
PAGE 0005  04/27/76  18:59:41  FORT:4 (B0)
 B0 FILE:  TASKS  OPTIONS:   I3  LO

PROGRAM SIZE=:0034 WORDS
BASE PAGE USED=:0008 WORDS
COMPILATION COMPLETE  0 ERRORS
```

Figure 2-8. FORTRAN/RTX Example for LSI-3/05 (Con't)

COMPILER DIAGNOSTICS

The compiler can produce several different kinds of diagnostics (see figure 2-9 for examples). Most are detected during the Scan phase and are printed on the source listing immediately following the statement in error. A dollar sign is printed underneath the position at which the error was detected, followed by a brief message. For example:

```
               DIMENSION BETA (0,10)
                                  $
     01) DIMENSION OUT OF BOUNDS E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*
```

The E's serve as a marker to make the message stand out and also signify "Error". This indicates that the statement could not be processed. Instead, a call to a run-time error routine is generated. Thus if any statement with an "E" type error is executed, a run-time diagnostic will occur.

Other errors are not so severe and can be recovered from. These are called Warnings, and they have the same format, except that the E's are replaced by W's For example:

```
               FORMAT(3X,F10.3,         ,I6)
        $                                    $
     01) LABEL MISSING W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*
     02) EXTRA COMMA  W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*
```

As shown above, it is possible to get more than one warning (and/or more than one error) on the same statement. In this case the numbers at the left of each diagnostic message indicate which dollar sign is referred to, counting from left to right.

Most of the messages are self-explanatory; however, Appendix D lists them with explanations of their cause.

The second group of diagnostics is produced during the Allocate phase. These are listed in the appendix, and include undefined labels, storage allocation conflicts (caused by COMMON or EQUIVALENCE), and storage overflow. These are all listed as "E" type errors, since there is no reasonable recovery, but most do not generate any run-time error call since they are not attached to any specific statement. Some of them are followed by a list of labels or variable names that are in error. For example:

```
     UNDEFINED LABELS E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*

             7 FIRST REF AT LINE     26
           296 FIRST REF AT LINE   132
```

The third group consists of the diagnostics produced during the Gen phase. There are only two such errors, and they both pertain to in-line assembly language. They are printed in the <u>object</u> listing, out to the right of a simulated assembly language instruction that has been generated. If no object listing is being printed, the line with the error will be printed anyway, to make sure of signalling the error. These diagnostics are listed in Appendix D.

The fourth group includes diagnostics that are not caused by source program error, but by compiler inability to continue. These errors always cause the compilation to be aborted. They have the following format:

FORT ER ptt

where p identifies the phase of the compiler that was operating:

p =	1	Scan
	2	Allocation
	3	Gen

and tt identifies the type of error:

tt =	11	Pointer overflow
	18	I/O error during overlay loading
	21	Working storage overflow
	28	Memory overflow during overlay loading
	31	Compiler error
	38	Illegal type code during overlay loading
	41	Compiler error
	51	Compiler error during collapse

Except for 21 and 28, all of these result from hardware or software errors. If they occur in a reproducible way, they are probably software errors, which should be reported. 28 indicates that the compiler will not fit in memory. 21 indicates that the program cannot be compiled in the given amount of memory.

```
PAGE 0001  09/24/74  15:15:08  FORT:4 (A1)
 DO FILL:  FOUT     OPTIONS:

0001 C          DEMONSTRATE COMPILER DIAGNOSTICS
0002            DIMENSION MM(10,10)
0003            COMMON X, Y, X
                             $
01) DECLARATION CONFLICT E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*


0004            EQUIVALENCE (X,Y)
0005            LOGICAL LGL, N
0006            INTEGER A,      , C
                               $
01) EXTRA COMMA W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W


0007            SF(P,Q) = P+Q/2
0008            X = 1E50 + LGL
                       $     $
01) CONSTANT SIZE E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*

02) TYPE CONFLICT E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*


0009            IF (A) 2,3,2
0010          2 X = SQRT(A) + SF(Y)
                         $       $
01) ARGUMENT CONVERTED W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W*W

02) ARGUMENT COUNT E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*


0011            B(10) = 0
                $
01) UNDIMENSIONED E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*


0012            X = (RX+ABS(SX))/(VAL+3))
                                        $
01) SYNTAX E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E


0013            J = MM(N)
                      $
01) NOT INTEGER E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*

02) NUMBER OF SUBSCRIPTS E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*


0014            END
```

Figure 2-9. Compiler Diagnostics Example

```
PAGE 0002  09/24/74  15:15:08  FORT:4 (A1)
JOB FILE:  FOUT      OPTIONS:

UNDEFINED LABELS E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*

     3 FIRST REF AT LINE        9

COMMON BLOCK/F:BCOM/ ALLOCATION  :0004 WORDS

LOCN   NAME    TYPE       WORDS   LOCN   NAME   TYPE      WORDS

:0000  X       REAL           2   :0002  Y      REAL          2

ARRAY ALLOCATION

LOCN   NAME    TYPE       WORDS   LOCN   NAME   TYPE      WORDS

:0003  MM      INTEGER      100

SCALAR ALLOCATION

LOCN   NAME    TYPE       WORDS   LOCN   NAME   TYPE      WORDS

:0061  A       INTEGER        1   :0065  J      INTEGER       1
:0066  K       LOGICAL        1

ALLOCATION ERRORS E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*E*
```

Figure 2-9. Compiler Diagnostics Example (Cont'd)

```
PAGE 0003  09/21/74  15:15:08  FORT.4 (A1)
 DO FILE:  FOUT     OPTIONS:

SUBPROGRAMS CALLED

NAME     TYPE     ARGS    NAME     TYPE     ARGS    NAME     TYPE     ARGS

F:RERR   RUNTIME          SQRT     REAL     1       F:RSTO   RUNTIME
F:RREL   RUNTIME          F:RDMY   RUNTIME

STATEMENT LABELS

LOCN  LABEL  USE         LOCN  LABEL  USE         LOCN  LABEL  USE

:0078 #2                 :FFFF #3                 :0076 #M2
:006B #M3

ENTRY=:0067
PROGRAM SIZE=:0094 WORDS
BASE PAGE USED=:0004 WORDS
COMPILATION COMPLETE  12 ERRORS
```

Figure 2-9. Compiler Diagnostics Example (Cont'd)

Section 3

LIBRARY STRUCTURE AND LINKING

GENERAL

A compiled FORTRAN program contains references to external subprograms. These references may be generated by:

1. EXTERNAL and CALL statements to a specific SUBROUTINE subprogram.
2. An external function, either library or user-defined.
3. A compiler generated reference to the mathematical or I/O routines.

References to these routines will appear within the object code listing and the subprogram usage map. In turn, these routines (the I/O routines in particular) may reference the OS or RTX I/O Interface routines which make the actual I/O calls to the OS or RTX drivers.

Each of these subroutines must therefore be available for linking, either by being compiled behind the main FORTRAN program in batch mode (if a FORTRAN subprogram), or by being resident on the FORTRAN library file, which is normally found on the System File (SF) device. (See System Generation, section 5 for a discussion of generation and ordering of the FORTRAN library file).

The following types of routines are resident on the library file:

1. The Basic External Functions, which are referenced by name within the FORTRAN statement. The function names are generally indicative of the functions, e.g. "SIN", "SQRT", etc. (A complete list of the functions and their descriptions is in the appendix).

2. The Mathematical and I/O Routines are references created by the compiler during the generation of the object code. A naming convention has been established for these routines whereby the routine name is of the form F:Exxx, F:Ixxx or F:Rxxx. A complete list of these routines and their descriptions is in the appendix.

3. The System I/O Interface routines are not referenced directly by the compiler: rather they are called by the I/O routines mentioned in item 2 above. The names of each of these routines are of the form "F:Xxxx"; a complete list is shown and described in the appendix.

4. The RTX/IOX routines are the standard RTX and IOX FUNCTIONS which may be referenced by in-line assembly language within a FORTRAN program executed under RTX control. The name of each routine is descriptive of its function (e.g. BEGIN:, END:, DELAY:), and contains a colon as its terminating character. These routines are described in detail in the RTX User's Manual.

5. The LSI-2 Instruction Emulator contains sequences used by FORTRAN when executing on an LSI-3/05 processor. These sequences emulate various LSI-2 instructions which do not exist in the LSI-3/05 computer. The emulator also includes within it a version of the LSI-3/05 software console routine.

The System Generation section of this manual describes the generation of the library. Specifically, three separate library files must be created, one to be linked for execution of the FORTRAN program under OS control (F:OSLB), and the other two for execution under RTX (F:RXLB for LSI-2 execution, and F3RXLB for LSI-3/05 execution). This allows the correct I/O Interface routines (OS or RTX) to be linked.

LINKING (OS:LNK)

Once a program has been compiled, it must be linked to various referenced library subprograms before it can be loaded and executed. OS:LNK, the standard OS link editor, performs this function. Its output is a self-contained module in absolute or relocatable binary format, including the FORTRAN program and all referenced library subroutines, which is suitable for loading by OS:LDR or the /EXECUTE or /LOAD commands (if it is to be run under OS control) or LAMBDA, BLD, or AUTOLOAD (if it is to be run under RTX). Note that OS:LDR and LAMBDA, which are "linking" loaders, cannot be used to link a FORTRAN program, because they do not recognize many of the special loader type codes generated by the compiler.

The reader should refer to the OS:LNK description in the OS User's Manual for detailed information regarding link editing. The following discussion encompasses those aspects of OS:LNK most pertinent to the linking of FORTRAN programs. Note: OS:LNK version B 2 or higher should be used to link FORTRAN programs.

I/O Device Assignments

The following logical devices must be assigned to specific physical devices prior to execution of OS:LNK:

1. System File Device (SF). Assigned to the device containing OS:LNK itself.

2. Binary Input Device (BI). Assigned to the file containing the binary output from the FORTRAN compiler (normally a magnetic file or the paper tape reader).

3. Library Input Device (LI). Assigned to the file containing the FORTRAN library module to be linked to the compiled binary code. As described in the System Generation section, three separate library files are normally constructed during generation; one for the OS Run-time library (F:OSLB), and two for the RTX Run-time library (F:RXLB for LSI-2 execution, or F3RXLB for LSI-3/05 execution).

4. Binary Output Device (BO). Assigned to the file which is to contain the linked binary output from OS:LNK. (Normally assigned to a magnetic file or the paper tape punch). This file is loaded and executed at FORTRAN run-time. Note that if the FORTRAN program is to be run under control of RTX, then the BO device must be assigned to the paper tape punch, since paper tape is the medium required by LAMBDA, BLD or AUTOLOAD, at execution time.

5. List Output File (LO). Assigned to the list output device (line printer) for output of the link map.

OS:LNK parameters

OS:LNK permits several options to be input as parameters. These are described in the OS:LNK User's Manual and familiarity with them is assumed here. The standard sequences of options normally used for linking FORTRAN programs are discussed here.

For Execution Under OS

When linking for OS execution, the link process must take place within the same OS System as that to be used for execution, since various OS routines, (e.g., the I/O driver entry points) have fixed addresses which must be referenced in the linked output. Thus the NH, SP, AB, RL and SR options need not be requested, because the default addresses for these options are available to OS:LNK from within OS itself. Also the XA, XR and XS options are not required, since the FORTRAN object module will contain the execution address (this is the memory address of the first executable FORTRAN statement in the main program; i.e., the location defined as F:MAIN). A typical calling sequence might be:

```
/AS BI=D0.FPROG (name of compiled FORTRAN program)
/AS  LI=D0.F:OSLB
/AS BO=D1.EXPROG (executable output)
/EX OS:LNK,LL,TE
```

In addition, the user may wish to utilize one or more of the following options:

NB	(Suppress binary output)
NL	(Suppress listing)
LI	(Re-enable listing)
MA	(Output link map at end)

(Refer to the OS:LNK description in the OS User's Manual for a discussion of the usage of these options.)

For Execution under RTX

When linking for RTX execution, the NH (or T3 if LSI-3/05), AB (or RL) and SR options are normally required, since the default addresses associated with these parameters are in relation to OS, and do not apply to the RTX system. Also, linking for the LSI-3/05 requires the SX option.

NH or T3 — This option specifies that the linked program is not intended to run under the host OS system. T3 should be used for LSI-3/05 execution.

AB (or RL) — This option specifies the starting absolute or relative memory address for loading the executable program. This may be any address or bias; however, it is a good idea to avoid loading in the base page area, which is needed for scratchpad literals and address pointers. Normally an input of AB (or RL) = 100 is optimal for FORTRAN loading under RTX.

NOTE

> Using an absolute load location (AB=) insures that the linked output is loadable by BLD, AUTOLOAD, or LAMBDA. If relative linking (RL=) is used, only LAMBDA should be used for loading, since BLD and AUTOLOAD do not recognize all the possible type codes which may be generated by OS:LNK in Rel mode.

SR This option specifies the starting address for any SREL (Relocatable Scratchpad) data encountered. RTX itself does not contain any SREL data; however, the FORTRAN compiler does output some in various object programs, and it always needs 20 SREL cells for its own subroutines, and they must be contiguous; these are used as temp cells, floating point accumulators, etc. When linking for LSI-2 execution, a usually safe location for SR is :60, since it is higher in memory than any of the standard interrupt locations. For LSI-3/05 execution, SR = 20 is recommended, because the addresses of some of the 20 SREL cells needed by the compiler are used as indexing offsets; if these cells are defined above location :3F, indirect index pointers will be created as needed, at the SX locations.

SX This option is meaningful only for T3 linking, and specifies the starting address for indirect indexing pointers. On the LSI-2, indirect indexing pointers are lumped together with the SP pointers; however, on the LSI-3/05, all indirect index pointers must reside below location :40, and so the SX option is required. These pointers are allocated beginning at the SX address, and continue upward, toward high memory. LSI-3/05 RTX needs location zero, so the SX address should be at least :0001.

The SP option is not required unless the user wishes to avoid using the default area for some specific reason.

The XA, XR, and XS options are not generally required if the RTX main program contains the entry point "F:MAIN", as described in the RTX example in the compiler options section of this manual.

An RTX program, since it contains tasks as well as library routines, requires the OS:LNK user to assign the BI device to the Mainline file and the LI device to the file containing the tasks, and then to re-assign LI to the library routines file. Also, since the resultant executable program must be loadable by LAMBDA, BLD, or AUTOLOAD, BO must be assigned to paper tape. Thus, a typical calling sequence might be:

(for LSI-2 execution)

```
/AS BI=DO.F:MAIN
/AS LI=DO.TASKS
/AS BO=PP
/EX OS:LNK,NH,AB=100,SR=60,LL
/AS LI=DO.F:RXLB
LL,TE
```

(for LSI-3/05 execution)

```
/AS BI=DO F:MAIN
/AS LI=DO.TASKS
/AS BO=PP
/EX OS:LNK,T3,AB=100,SR=20,SX=1,LL
/AS LI=DO.F3RXLB
LL,TE
```

In addition, the user may wish to utilize one or more of the following options:

NB (suppress binary output)
NL (suppress listing)
LI (re-enable listing)
MA (output link map at end)

(Refer to the OS:LNK description in the OS User's Manual for a discussion of these options.)

Memory Usage

During the link process, memory is allocated as shown by the arrows in figures 3-1 and 3-2. Note that this allocation information is being transferred to the BO device during OS:LNK; the actual data is not stored in memory until load time.

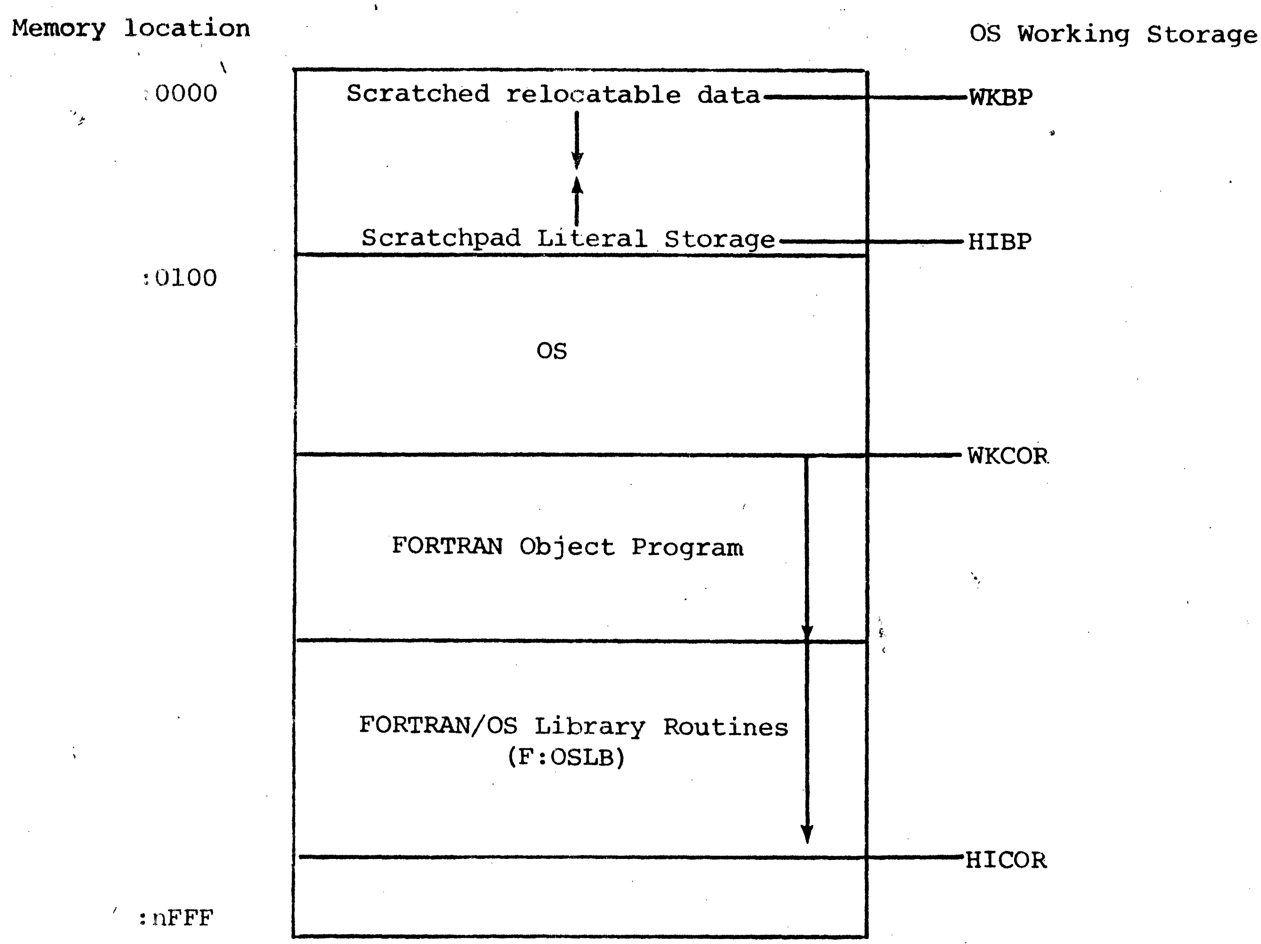


Figure 3-1. OS:LNK Memory Allocation for OS Execution

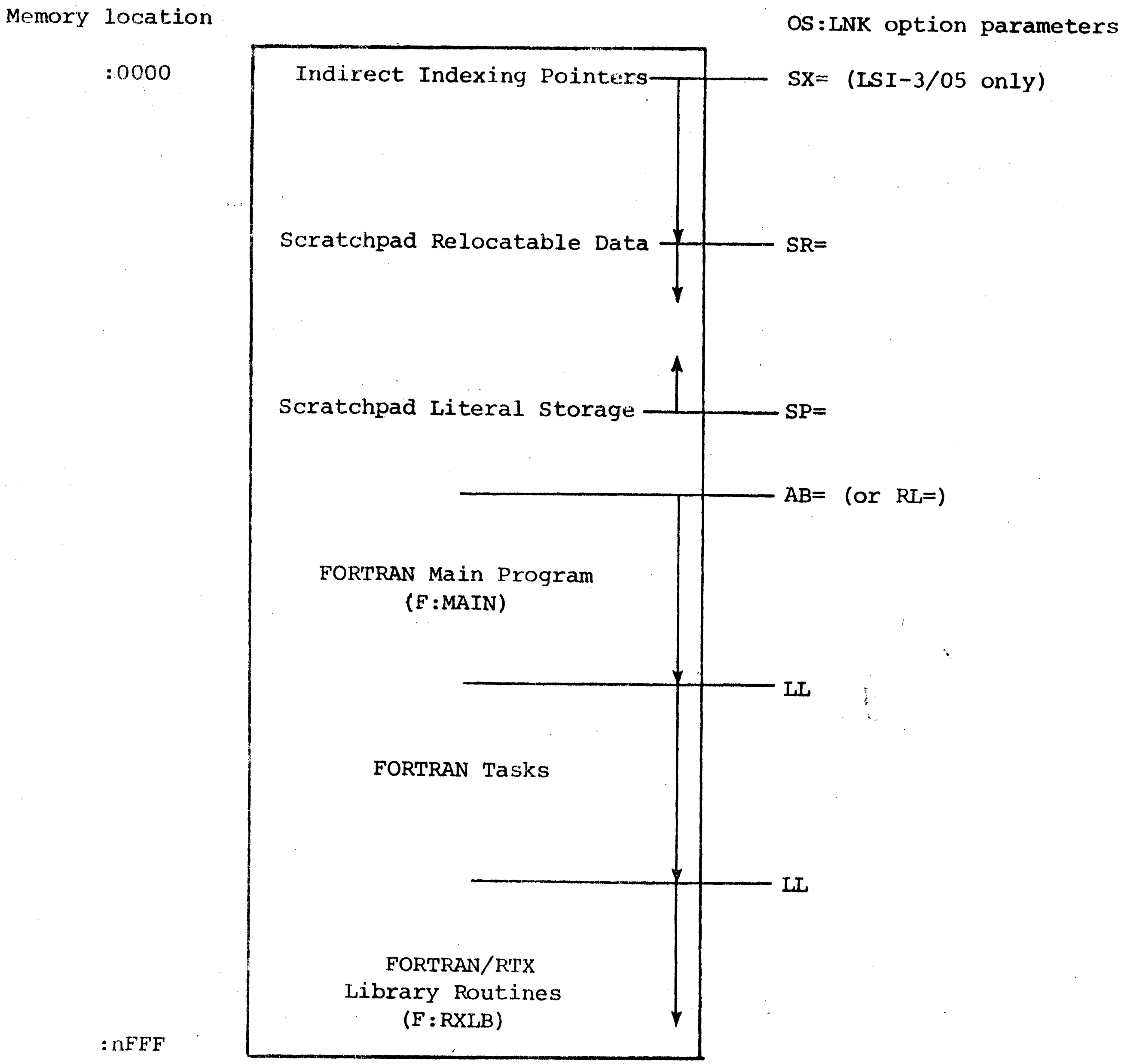


Figure 3-2. OS:LNK Memory Allocation Map for RTX Execution

OS:LNK Memory Map

As each input file is processed by OS:LNK, a list of undefined references (if any) is output to the list device. This listing may be suppressed by the NL option. Upon input to OS:LNK of a Terminate (TE) parameter, a memory map is output, which lists each external definition and COMMON allocation, with its associated memory location (which may be absolute or relocatable, depending on the "AB" or "RL" option input to OS:LNK). Figure 3-3 is the memory map generated by linking the LSI-3/05 RTX sample program from Figure 2-8.

```
PAGE   2    03/17/76   10:06:59   OS:LNK (B1) MEMORY MAP

CREATED FILE  EXAMPL

MISSING
   R:0      R:1      R:2      R:3      R:4      R:5      R:6      R:7
   R:8      R:9      R:A      R:B      R:C      R:D      R:F      R:G
   R:H      R:I      R:J      R:K      R:L      R:M      R:N      R:O
   R:P      R:Q      R:R      R:S      R:T      R:U      R:V      R:W
   R:X      R:Y      R:Z

PROGRAM
   F:RURF  0026    F:RBPG  0027    F:RRPP  0027    F:RACS  0028
   F:RACE  0029    F:RAC1  002A    F:RAC2  002B    F:RAC3  002C
   F:RAC4  002D    F:ROPS  002E    F:ROPF  002F    F:ROP1  0030
   F:ROP2  0031    F:ROP3  0032    F:ROP4  0033    F:RARG  0034
   F:RXRG  0035    F:FLOC  0036    F:FQL1  0037    F:FQL2  0038
   F:RPAB  003A    F:MAIN  0100    I:UAT   0175    TASK1   0357
   TASK2   038B    D:LPFD  03A9    C:LPFD  03B5    D:TY00  03DB
   C:TY0   03E6    I:RITF  040C    I:READ  045F    I:RITE  04E1
   RITE2   04F0    I:FUN   0522    SCH1    055F    LOKIN:  056F
   I:DIR   0576    SCH:    057C    RTX:    05FC    RTOS:   05FC
   BEGIN:  0623    END:    063C    PAUSE:  0641    NEWPR:  0644
   DOIT:   0645    GETFR:  0647    PUTFR:  0652    SVST1:  0659
   R:F     0673    READY:  0673    FIFOG:  0674    DLYCH:  0675
   COMN:   0676    IOCH:   0677    GETCH:  0678    PUTCH:  0679
   GETRF:  0686    PUTRF:  068C    PUTPR:  0691    SCHED:  06A3
   DEBUG3  06BD    ZBG     06C1    ZEBUG   06C1    SORT    0CAC
   F:ISQR  0CB7    F:TRAD  0CFF    F:TRSB  0D04    F:TRML  0D0A
   F:IRDV  0D10    F:TRLD  0D20    F:IRST  0D2C    F:TRMV  0D38
   F:IFR1  0D51    F:IFR2  0D66    F:IRUN  0D7D    F:IFC1  0D86
   F:RRFI  0DA6    F:RRAB  0DAF    F:RRAD  0DB3    F:RRAU  0DB7
   F:RRDM  0E0B    F:RRDV  0E14    F:RRDU  0E1C    F:RIIP  0E4F
   F:RRLD  0E65    F:RRML  0E70    F:RRMU  0E74    F:RRUN  0EA4
   F:RRNG  0EBC    F:RRTI  0ECF    F:RRSG  0EF0    F:RRST  0EF5
   F:RRSB  0F17    F:RRSU  0F1B    F:RINT  0F21    F:RIIP  0F29
   F:RSIO  103C    F:RINP  1047    F:ROUT  104D    F:REFN  1056
   F:RDFN  105D    F:RWFR  1063    F:RWFN  1069    F:RWFB  1070
   F:RWF   1078    F:RRFR  107F    F:RRFN  1085    F:RRFB  108C
   F:RRF   1094    F:RIUS  1125    F:RRUS  112B    F:RDUS  1130
   F:RLUS  1135    F:RCUS  113B    F:RHUS  1143    F:RIOL  1157
   F:RROL  115C    F:RDOL  1161    F:RLOL  1166    F:RCOL  116B
   F:RFAA  11E7    F:RFAF  11E9    F:RFDE  11EE    F:RFDF  11EF
   F:RFFS  11F0    F:RFPF  11FD    F:RFRA  1202    F:RFSF  1203
   F:RFW1  1207    F:RFWE  1208    F:RFWS  1209    F:RFWF  120A
   F:RSIO  1222    F:RFSI  1229    F:RFRW  13F0    F:RFRN  13EA
   F:RFFD  1431    F:RFSW  144D    F:RFSO  145B    F:RFWD  1469
   F:RFI   1501    F:RFI   1559    F:RFZ   15A8    F:RHFO  163D
   F:RFG   1652    F:RFIR  1700    F:RFF   1711    F:RFF   1732
   F:RFD   1738    F:RFTS  1A61    F:RFAD  1A6C    F:RFDA  1A88
   F:RFFO  1AA0    F:RFWB  1AA9    F:RUGN  1AB8    F:RUIR  1AC4
   F:RUIS  1ACC    F:RUST  1ADB    F:RURT  1AF7    F:RUAV  1B14
   F:RUAA  1B1A    F:EATL  1B4C    F:EBAZ  1B51    F:EDV0  1B56
   F:EINA  1B5B    F:ENGA  1B5F    F:EOVR  1B63    F:ESGL  1B63
```

Figure 3-3. Link Map Example

```
PAGE   3     03/17/76  10:06:59  OS:LNK (B1) MEMORY MAP

F:RCII  1B68    F:RDTI  1B68    F:RDML  1B68    F:RCML  1B68
F:RDST  1B68    F:RCST  1B68    F:RDAD  1B68    F:RCAD  1B68
F:RDSB  1B68    F:RCSB  1B68    F:RDDV  1B68    F:RCDV  1B68
F:RDID  1B68    F:RCID  1B68    F:RDIR  1B68    F:RCIR  1B68
F:RTID  1B68    F:RRTD  1B68    F:RCTD  1B68    F:RTIC  1B68
F:RRTC  1B68    F:RDTC  1B68    F:RDAB  1B68    F:RDDM  1B68
F:RDSG  1B68    F:RDNG  1B68    F:RCNG  1B68    SRF:FR  1B68
F:ERRS  1BBF    F:ERRC  1BC3    F:RU06  1C48    F:XRDS  1C48
F:RU0T  1C48    F:RUIN  1C48    F:RUNN  1C48    F:RU01  1C48
F:RU02  1C48    F:RU03  1C48    F:RU04  1C48    F:RU05  1C48
F:XINP  1C50    F:XWTS  1C64    F:XOUT  1C6C    F:XRWD  1C73
F:XBSP  1C7A    F:XEOF  1C81    F:XCLS  1C86    F:XRCS  1C8F
F:XDEL  1D12    F:XERR  1D1C    F:XPSE  1D51    F:XSTP  1D64
F:RLS3  1F3F    MD1A:   1F3F    ZAX:    1E48    AXP:    1E4B
AXM:    1E4F    NRA:    1F51    NRX:    1E55    LAX:    1E5B
IXA:    1E5F    DAX:    1F61    DXA:    1E64    MDRPC:  1E7F
ANX:    1E9A    CAR:    1F9F    CXR:    1EA3    EAX:    1EA7
LAX:    1EAA    CXA:    1FAF    ANA:    1EB3    MDMDN:  1EC2
NRM:    1ECB    MPY:    1FF8    DVD:    1F05    MDLSH:  1F22
LRR:    1F30    LBL:    1F38    LLR:    1F40    LLL:    1F46
MDEOV:  1F51    BAO:    1F51    BXO:    1F51    LAO:    1F51
EXO:    1F51    SAO:    1F51    SXO:    1F51    MDASH:  1F62
AEX:    1F60    AEA:    1F76    ARA:    1F81    APX:    1F84
FMUL:   1FA8    CNSOI:  1FA8    COV:    1FF1    XEMOV:  1FF6
XFM:    1FFC    UIN:    2012    LOC:    20B8    INST:   20B9
AP:     20BA    XR:     20BB    STAT:   20BC    IOHTI:  20CD
IORSI:  20CD    IO:     210B    FNDIB:  2182    FOR:    2196
FORST:  2197    FOFQ:   21CF    EOF:    21D5    SIO:    21DB
INTP:   21FF    WATCH:  2249    UNRES:  2251    SINT:   225A
CKSUM:  2276    FETCH:  2280    BUFFQ:  2293    WAIT:   22A5
FOFCK:  22BF    GETPR:  22D5    SETPR:  22DD    INCPR:  22E9
DELPR:  22FE    DELAY:  22F7    LOCK:   235F    UNLK:   236F
UNPR:   236F    ABORT:  2377    TERM:   2378    SUBR:   2392
SBBLK:  2397    SUBX:   23D3    SBXNK:  23D8    UNDO:   23DF
INTQ:   23FD    SCAN:   2440    DELET:  2448    INSRT:  244C
SENDI:  2452    RTOS7:  2457
```

Figure 3-3. Link Map Example (Con't)

```
PAGE   4    03/17/76  10:06:59  OS:LNK (B1) MEMORY MAP

MEMORY USAGE
  SCRATCH-PAD LITERAL      0008-007E
  SCRATCH-PAD PROGRAM      0002-003A
  MAIN MEMORY PROGRAM      0082-2458
  EXEC ADDRESS             06C1
SCRATCHPAD USAGE TABLE:
ADDR 0 1 2 3 4 5 6 7 8 9 A B C D E F     LEGEND:
0000 . . P P P P P P X . . . . . . .       A=ABSOLUTE LITERAL
0010 . . . . . . . . . . . . . . . .       B=BYTE RELOCATABLE LITERAL
0020 . . . . . . S S S S S S S S S S       P=ABSOLUTE PROGRAM
0030 S S S S S S S S S S S . . . . .       R=WORD RELOCATABLE LITERAL
0040 . . . . . . . . . . . . . . . .       S=SREL PROGRAM
0050 . . . . . . . . . . . . . . . .       X=ABSOLUTE INDEX POINTER
0060 . . . . . . . . . . . . . . . .       W=WORD RELOCATABLE INDEX POINTER
0070 . A A A A A A A A A A A A A A .       Y=BYTE RELOCATABLE INDEX POINTER

PROCESSED 1ST 3 OBJECT

NO ERRORS
```

Figure 3-3. Link Map Example (Con't.)

The created file name is listed first, followed by a list of missing names (undefined references), if any. This is followed by a listing of defined references and their addresses. This listing is in order of occurrence, reading from left to right across each line.

Following the list of definitions, the COMMON areas are described with their lengths and starting addresses. Blank COMMON is not allocated to a particular memory location by OS:LNK until input of the "TE" parameter, and so it generally has the highest address of all the linked modules. Labeled COMMON, however, is allocated upon its first occurrence when passing through OS:LNK. The OS:LNK memory map concludes with a list of address ranges required for scratchpad (literals and input data) and main memory usage, a map of scratchpad usage, and the execution address (normally the location of F:MAIN or DEBUG).

OS:LNK Error Reporting

During the link process, various error conditions may occur. These errors may be grouped into three types of messages:

1. Diagnostics. Output to the LO device as they are encountered. They indicate memory usage conflict of various forms, and are usually caused by scratchpad or main memory overflow, or an attempt to store data into a scratchpad location already occupied. These errors do not terminate OS:LNK, but may produce erroneous results during program execution.

2. Termination errors. Output to the CO and LO devices, indicating an error which prevents OS:LNK from completing the link operation. A memory map is printed at this time, and OS:LNK terminates.

3. I/O errors. Output to the CO device, and reflect an error status returned from OS following an I/O operation.

A complete list of OS:LNK error messages may be found in Appendix D.

Section 4

RUN-TIME

INTRODUCTION

Once the FORTRAN program has been successfully compiled and link edited, it is ready to be loaded and executed. Prior to this time, however, consideration should be given to the I/O operations which will be performed during execution.

I/O DEVICE ASSIGNMENT

All input/output operations specified in the FORTRAN source program (READ, WRITE, INPUT, OUTPUT, BACKSPACE, REWIND, and END FILE) make use of FORTRAN unit numbers (1 through 99) to specify the particular device on which the I/O operation is to be performed. INPUT and OUTPUT statements do not include specific unit numbers, but imply input from logical unit 5 and output to logical unit 6. The other I/O statements must include a logical unit number, expressed either as an integer constant or a simple integer variable. Prior to execution of the program, any FORTRAN unit numbers used in the program must be assigned to specific I/O devices. In addition, the Command Output (CO) unit must be assigned to a device (normally the teletype) for output of PAUSE, STOP and run-time error messages; also, for OS execution, a CI assignment is required to enable the operator to resume a program following PAUSE suspension.

Device Assignment for Execution under OS

For execution under OS, device assignment is accomplished by the /ASSIGN command. Usage of the /ASSIGN command, however, implies in turn that entries exist within the OS Logical Unit Table (LUT) for the FORTRAN unit numbers used in the FORTRAN source program. Thus, although the FORTRAN compiler will accept any logical unit number from 1 to 99, the FORTRAN programmer is limited to the unit numbers in the LUT. The standard OS systems distributed by Computer Automation, Inc., contain LUT entries for FORTRAN units 1 through 6 only, with the following default assignments:

Unit 1	Unassigned
Unit 2	Unassigned
Unit 3	Unassigned
Unit 4	Unassigned
Unit 5	Card Reader
Unit 6	Centronics Line Printer

To add additional FORTRAN units to the table, or add default assignments to unassigned units, re-assemble the OS ROOT program with the desired changes and re-generate your OS system; it is also necessary to as add a File Control Block (FCB) entry to the FCB tables within the FORTRAN/OS library package, for each additional unit number. These procedures are fully described in Section 5, System Generation.

The actual unit assignment is in the standard format, where the logical unit number is specified as a one or two digit number, e.g.:

/Assign 2=PR	(assign FORTRAN unit 2 to the paper tape reader)

or

/ASSIGN 03=DO.FILNAM	(assign FORTRAN unit 3 to a file on disk unit 0)

Note that usage of a bulk storage device requires that the device be previously labeled for OS (by using the OS:LBL utility).

Device Assignment for Execution Under RTX

When preparing a FORTRAN program for execution under the Real Time Executive, device assignment is made by creation of a Unit Assignment Table, which should be assembled to the RTX mainline program. Refer to the RTX option description in the Compiler Options section for a discussion of the Unit Assignment Table.

FORMS CONTROL FOR LIST DEVICES

Forms control for printed output to the line printer or teletype is accomplished by use of a carriage control character. This character must occupy the first position of any print line, and is never printed. (Exception: when using the free-form OUTPUT statement, output always begins in column 2 of the printer; thus allowance for a carriage control character is not necessary.)

The carriage control characters and their functions are as follows:

Character	Function
1	Causes page eject (top of form) before printing
0	Causes double up space before printing
Any other	Causes single up space before printing

(Note that Overprint capability is not supported.)

The carriage control capability is useful for printing data in a user-defined format, such as report generation. Judicious use of these control characters will enable various formatting arrangements of the printer output. (There are 54 lines to a printer page.) Note that the user who does not wish to use carriage control and merely wants single spaced output must insure that the print line does not contain a "1" or "0" in column 1. This is most easily done by using the OUTPUT statement, or by beginning the FORMAT statement with a 1Hb format.

POSITIONING CONTROL FOR MAGNETIC DEVICES

The REWIND, BACKSPACE, and END FILE statements are for magnetic devices only and are described in the FORTRAN Reference Manual in relation to magnetic tape or cassette usage. For operation to a disk file, the internal operation is slightly different (for example, an end-of-file mark is a normal record with a special character in the first word rather than a hardware function as on magnetic tape), however, the user may use these functions just as he would for magnetic tape or cassette. A BACKSPACE statement will cause the disk to reposition itself to the previous record to be re-read or re-written, a REWIND statement will reposition the disk to the start of the file, etc. (This is not done by actual physical repositioning, but rather by re-setting the current relative record number internally by the OS File Manager or RTX disk handler.)

PROGRAM LOADING PRIOR TO EXECUTION

The procedure used for loading a linked FORTRAN program basically depends on whether the program is to execute under OS or RTX control.

Loading for OS Execution

OS is executed under the same OS system used to link the program. The following sequence may be used:

a. Issue a /JOB command to initialize the unit assignments.
b. Assign all pertinent FORTRAN unit numbers to the required physical devices.
c. Assign the SF (System File) unit to the device containing the linked FORTRAN program.
d. Issue an /EXECUTE command to load and execute the program.

Loading for RTX Execution

For execution under RTX, the linked FORTRAN program, may be loaded by one of the following loader programs:

1. LAMBDA linking loader
2. OS:ILD
3. BLD binary loader
4. AUTOLOAD
5. DLD (LSI-3/05 only)

Note that if relative linking was used during the OS:LNK procedure (RL=), certain type codes may have been output which are not recognized by BLD or AUTOLOAD. IF linked in absolute mode (AB=), any binary loader may be used.

Refer to the documentation of the desired loader for specific operating instructions.

Errors During the Load Procedure

If a load error occurs during the loading procedure, consult the documentation for the applicable loader. A memory overflow error indicates that the linked FORTRAN program is too large, and may require re-compilation using some form of coding optimization. Output of an object code listing during compilation can aid the programmer in this respect.

PROGRAM EXECUTION

Once the linked FORTRAN program has been loaded and execution has begun, various conditions can occur to which the user (or the operator) must respond.

PAUSE Messages

The PAUSE statement causes the message

"PAUSE xxxxx"

to be output to the Console Output (CO) device (which must have been previously assigned). "xxxxx" represents a decimal number from 0 to 32767, and may assume any meaning the programmer wishes it to have, to the operator (e.g., a certain number may indicate that the operator is to load data records into an input device).

When a PAUSE message occurs during execution under OS, it is automatically followed by a "suspended" condition, during which the operator may perform some required function. The program may then be resumed by inputting a "/RESUME" command. (The /RESUME command must be input through the default assigned CI device, normally the teletype keyboard, no matter which device is currently assigned as CI).

Run-Time Error Handling

Diagnostics at run-time can originate in either the FORTRAN library or the OS system. (Under RTX there are no system error messages.) The FORTRAN diagnostics are output to the list device and the console, and have the form:

```
     'routine name', 'message' ERROR AT :xxxx
```

where :xxxx is the location of the call in the user program. In addition, under RTX this information will be followed by:

```
     PRI: ddddd
```

where ddddd is the decimal value of the priority assigned to the task that was active. This helps in identifying the task.

The FORTRAN run-time diagnostics are listed in the appendix, with the messages in alphabetic order (since the same message can often be produced by several routines). Note that occasionally there is no routine name given, e.g. NUMBER OF ARGUMENTS, since the name is not known at run time. The "comments" column explains the error and indicates whether it causes an abort or whether some recovery is made.

When running under OS, some error conditions will be detected by the system rather than the FORTRAN library. You should be familiar with the OS User's Manual; however, the appendix shows the OS diagnostics that are relevant to FORTRAN jobs. In many cases, errors in the use of input/output files are detected at the time the file is opened. In FORTRAN this happens automatically the first time the file is used. Therefore some OS messages will appear only if the error is made on the first use of a unit number. For example, if you write on the line printer, then try to read from it, you will get a FORTRAN message, whereas if you tried to read from it first you would get an OS message.

Note that OS messages are written on the console device, not on the listing device. In addition, some of them cause the program to be suspended, in which case recovery must be made at the console before resuming (for example, by reassigning a unit number or readying a device.) If OS returns, instead of suspending, there will typically be a FORTRAN error message that follows. The OS message, then, will identify the device or unit number, while the FORTRAN message will identify the operation that was being performed (e.g. FORMATTED, BACKSPACE) and the location of the call. In addition, some of these will cause the ERR= exit to be taken, if this option was specified in the READ or WRITE statement. In the appendix, the second column of these messages shows whether OS returns or suspends. The last column explains the error.

Console Interrupt

Console interrupt is not enabled when executing FORTRAN under RTX. Under OS, however console interrupt is enabled at all times, and may be used to pass control back to the OS Executive. The FORTRAN program is normally resumable once it has been interrupted

Section 5

SYSTEM GENERATION

INTRODUCTION

The ALPHA LSI FORTRAN IV System is delivered as several separate files, from which the user may configure his system to meet his individual requirements. These files are available on various types of media (paper tape, disk cartridges, etc.). The examples in this section assume floppy disk. If the user's files are on another medium, he should alter the generation procedure in accordance with his requirements.

GENERATING THE FORTRAN COMPILER

When delivered, the FORTRAN compiler resides on the following files:

Compiler Root	F:CROT	(96510-30)	
Compiler Interface	F:CFAC	(96510-31)	
Compiler Scan (Complete)	F:CSCN	(96511-30)	
Compiler Scan Overlay 1	F:CSCO	(96511-31)	
Compiler Scan Overlay 2	F:COS1	(96511-32)	
Compiler Scan Overlay 3	F:COS2	(96511-33)	
Compiler Allocate Module	F:CALL	(96512-30)	
Compiler Gen (Complete)	F:CGEN	(96513-30)	For LSI-2 Run-time
Compiler Gen Overlay 1	F:CGEO	(96513-31)	For LSI-2 Run-time
Compiler Gen Overlay 2	F:COG1	(96513-32)	For LSI-2 Run-time
Compiler Gen Overlay 3	F:COG2	(96511-33)	For LSI-2 Run-time
Compiler Gen (Complete)	F:CGE3	(96513-34)	For LSI-3/05 Run-time
Compiler Gen Overlay 1	F:CGE4	(96513-35)	For LSI-3/05 Run-time
Compiler Gen Overlay 2	F:COG5	(96513-36)	For LSI-3/05 Run-time
Compiler Gen Overlay 3	F:COG6	(96513-37)	For LSI-3/05 Run-time
Compiler Root LSI-3/05 Overlay	F:CRT3	(96510-33)	

The above listed files comprise the several parts of the compiler:

1. The Compiler "Control" program consists of the Compiler Root (F:CROT), and the Compiler I/O Interface (F:CFAC), which must be linked together by the user into a single file, called "FORT:4". This is the file that is actually called by the operator to begin a compilation.

2. The Scan phase is provided in two forms, one or the other of which is called by FORT:4 depending on the amount of available memory the user's system contains. If more than 16K words of memory, FORT:4 will automatically call in the "complete" Scan module (F:CSCN) at Scan time. If the system has only 16K, FORT:4 will automatically call in the three Scan overlays (F:CSCO, F:COS1 and F:COS2) as needed.

3. The Allocate phase is provided in non-overlayed ("complete") format only (F:CALL), as it is small enough to fit, with FORT:4, into 16K of memory.

4. The Gen phase for LSI-2 programs is, like Scan, provided in two forms; F:CGEN (the complete Gen module) is called if more than 16K of memory exists; otherwise, the three Gen overlays (F:CGEO, F:COG1 and F:COG2) are called in as needed.

5. The Gen phase for LSI-3/05 programs has an exact correspondence to the LSI-2 Gen, except that LSI-3/05 versions are used when the T3 option is specified. F:CGE3 is called when more than 16K of memory is present; otherwise the overlays F:CGE4, F:COG5 and F:COG6 are used.

6. Besides determining which Gen to use, the T3 option also causes that part of the Root which contains the LSI-2 instruction skeletons to be overlayed by F:CRT3, which is the equivalent list of LSI-3/05 instruction skeletons.

Figure 5-1 shows the compiler configuration in memory when more than 16K is present. The Scan, Allocate, and Gen phases share memory by overlaying each other, as shown.

Figure 5-2 shows the compiler configuration when only 16K memory is present. Note that F:CSCO, F:CALL, and F:CGEO all share memory by overlaying each other. In addition, F:CSCO contains within it an area which is shared by F:COS1 and F:COS2 in overlay fashion. Likewise, F:CGEO contains F:COG1 and F:COG2 within it, which overlay each other.

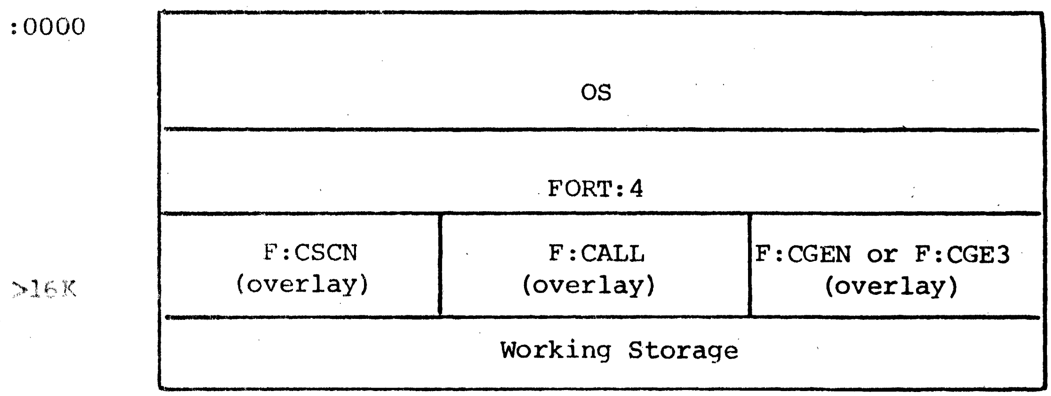


Figure 5-1. Compiler configuration when more than 16K memory

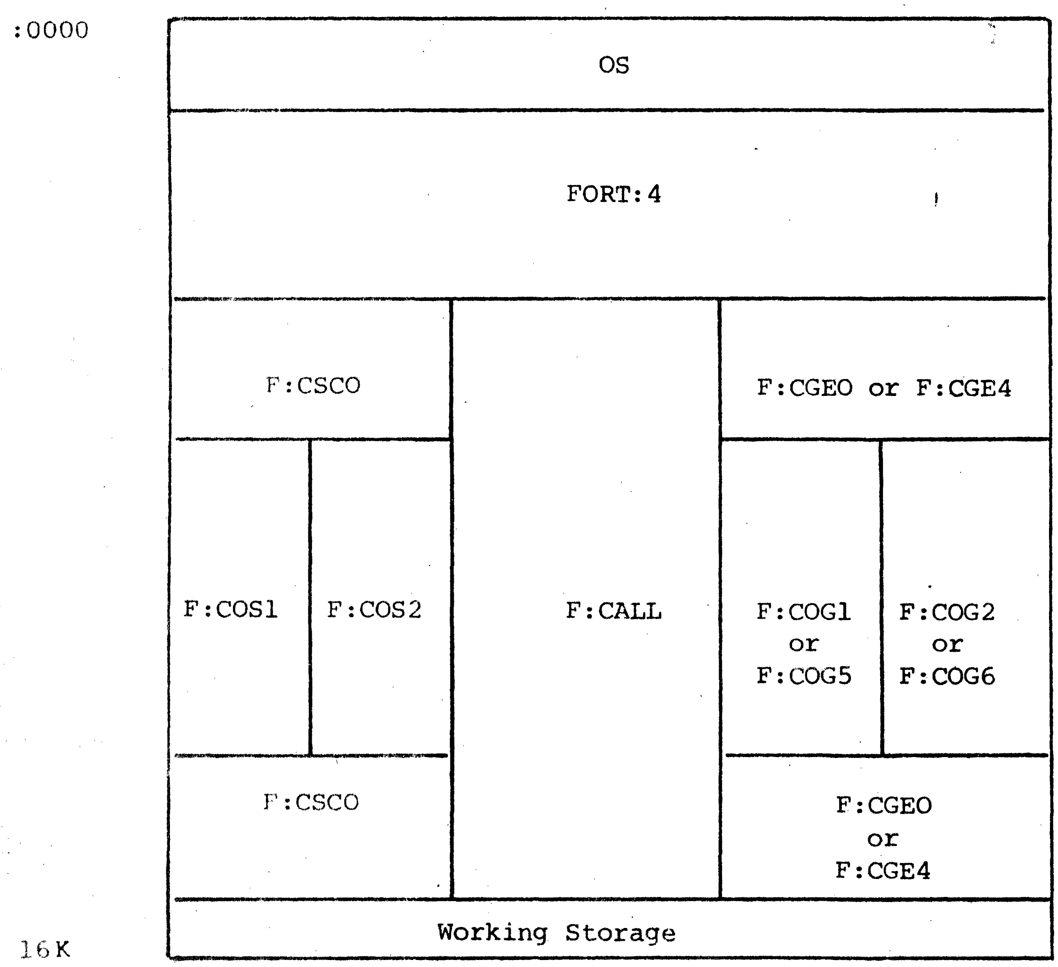


Figure 5-2. Compiler Configuration with 16K memory

The generation procedure consists of two main steps:

STEP 1: Copy the F:CROT and F:CFAC modules to the system file device using the OS:CPY utility, then link them together into FORT:4 using the OS:LNK utility:

1. /JOB
2. /EX OS:CPY
3. CB,FO.F:CROT,DO.F:CROT
4. CB,FO.F:CFAC,DO.F:CFAC,TE
5. /JOB
6. /AS BI=DO.F:CROT,LI=DO.F:CFAC,BO=DO.FORT:4
7. /EX OS:LNK,LL,TE

STEP 2: Copy each of the remaining compiler modules to the system device, using the OS:CPY utility:

1. /JOB
2. /EX OS:CPY
3. CB,FO.F:CRT3,DO,F:CRT3
4. CB,FO.F:CSCN,DO,F:CSCN
5. CB,FO.F:CSCO,DO.F:CSCO
6. CB,FO.F:COS1,DO.F:COS1
7. CB,FO.F:COS2,DO.F:COS2
8. CB,FO.F:CALL,DO.F:CALL
9. CB,FO.F:CGEN,DO.F:CGEN
10. CB,FO.F:CGEO,DO.F:CGEO
11. CB,FO.F:COG1,DO.F:COG1
12. CB,FO,F:COG2,DO.F:COG2
13. CB,FO.F:CGE3,DO.F:CGE3
14. CB,FO.F:CGE4,DO.F:CGE4
15. CB,FO.F:COG5,DO.F:COG5
16. CB,FO.F:COG6,DO.F:COG6

GENERATING THE FORTRAN LIBRARY FILE

The delivered files include several routines which must be merged by the user (using the OS:CPY utility) onto the system file device as one of two library files. Since a FORTRAN program may be compiled to run under either OS or RTX, and since these operating systems require different library routines, a single library file may not be created which will serve the purposes of both the OS and the RTX system. This means that three distinct library files must be generated, one for OS execution and two for RTX execution (LSI-2 and LSI-3/05 versions). The following file names have been established to differentiate the libraries:

F:OSLB	(for execution under OS)
F:RXLB	(for LSI-2 execution under RTX)
F3RXLB	(for LSI-3/05 execution under RTX)

The following sections describe the generation procedures for these files:

OS Run-time Library Generation (F:OSLB)

1. FORTRAN LSI-2 Basic External Functions Library Module (F:EXTR) (96514-30)
2. FORTRAN LSI-2 Math and I/O Routines Library Module (F:MATH) (96514-31)
3. FORTRAN/OS I/O Interface Module (F:OS10) (96515-30)

The modules must be merged into one system device file, named F:OSLB. The order shown above reflects the order in which the modules must reside in the library file, to enable the OS:LNK utility to link edit a FORTRAN program in a single pass.

The following procedure will merge these modules as required for correct linking:

1. (Operator mounts the FORTRAN library modules diskette on unit F0)
2. /JOB
3. /EX OS:CPY
4. MB, F0.F:EXTR,D0.F:OSLB
5. (OS:CPY merges the Basic External module and outputs the "READY NEXT FILE" MESSAGE).
6. F0.F:MATH
7. (OS:CPY merges the Math and I/O Routines module and outputs the "READY NEXT FILE" message)
8. F0.F:OSIO
9. (OS:CPY merges the FORTRAN/OS I/O Interface module, then outputs the "READY NEXT FILE" message)
10. MT,TE

-2 RTX Run-time Library Generation (F:RXLB)

The following five modules comprise the LSI-2 RTX Run-time Library:

1. FORTRAN LSI-2 Basic External Functions Library Module (F:EXTR) (96514-30)
2. LSI-2 RTX/IOX Segment 1 module* (93300-30)
3. FORTRAN LSI-2 Math and I/O Routines Library module (F:MATH) (96514-31)
4. FORTRAN/RTX LSI-2 I/O Interface module (F:RX10) (96516-30)
5. LSI-2 RTX/IOX Segment 2 module* (93300-31)

*included in the RTX Software Package

These modules must be merged into one system device file, named F:RXLB. The order shown above reflects the order in which the modules must reside in the library file, to enable the OS:LNK utility to link edit a FORTRAN program in a single pass.

The following procedure will merge these modules as required for correct linking:

1. (Operator mounts the FORTRAN Library Modules diskette on unit F0)
2. /JOB
3. /EX OS:CPY
4. MB,F0.F:EXTR,D0.F:RXLB
5. (OS:CPY merges the Basic External Functions routine, then outputs the "READY NEXT FILE" message)
6. (Operator mounts the LSI-2 RTX/IOX Segment 1 module tape into the paper tape reader)
7. PR
8. (OS:CPY merges the RTX/IOX Segment 1 module, then outputs the "READY NEXT FILE" message)
9. F0.F:MATH
10. (OS:CPY merges the FORTRAN Math and I/O routines module, then outputs the "READY NEXT FILE" message)
11. F0.F:RXIO
12. (OS:CPY merges the FORTRAN/RTX I/O Interface module, then outputs the "READY NEXT FILE" message)
13. (Operator mounts the LSI-2 RTX/IOX Segment 2 module tape into the paper tape reader

14. PR
15. (OS:CPY merges the RTX/IOX Segment 2 module, then outputs the "READY NEXT FILE" message)

16. MT,TE

LSI-3/05 RTX Run-time Library Generation (F3RXLB)

The following six modules comprise the LSI-3/05 RTX Run-time library:

1. FORTRAN LSI-3/05 Basic External Functions library module (F3EXTR) (96514-32)
2. LSI-3/05 RTX/IOX Segment 1 module* (93301-30)
3. FORTRAN LSI-3/05 Math and I/O Routines library module (F3MATH) (96514-33)
4. FORTRAN/RTX LSI-3/05 I/O Interface module (F3RXIO) (96516-31)
5. FORTRAN LSI-2 to LSI-3/05 Instruction Emulator and Software Console module (F3EMUL) (96516-32)
6. LSI-3/05 RTX/IOX Segment 2 module* (93301-31)

*included in the RTX Software Package

These modules must be merged into one system device file, named F3RXLB. The order shown above reflects the order in which the modules must reside in the library file, to enable OS:LNK to link edit a FORTRAN program in a single pass.

The following procedure will merge these modules as required for correct linking:

1. (Operator mounts the FORTRAN library modules diskette on unit F0)
2. /JOB
3. /EX OS:CPY
4. MB,F0.F3EXTR,D0.F3RXLB
5. (OS:CPY merges the Basic External Functions, then outputs "READY NEXT FILE" message)
6. (Operator mounts the LSI-3/05 RTX/IOX Segment 1 module tape into the paper tape reader)
7. PR
8. (OS:CPY merges RTX Segment 1, then outputs "READY NEXT FILE" message)
9. F0.F3MATH
10. (OS:CPY merges the FORTRAN Math and I/O Routines, then outputs "READY NEXT FILE" message)
11. F0.F3RXIO
12. (OS:CPY merges the FORTRAN/RTX I/O Interface module, then outputs "READY NEXT FILE" message)
13. F0.F3EMUL
14. (OS:CPY merges the FORTRAN Emulator and Software Console Routine module, then outputs "READY NEXT FILE" message)
15. (Operator mounts the LSI-3/05 RTX/IOX Segment 2 module tape into the paper tape reader)
16. PR
17. (OS:CPY merges RTX Segment 2, then outputs "READY NEXT FILE" message)
18. MT,TE

ADDING OR REPLACING LIBRARY PROGRAMS

The ordering of the routines on the FORTRAN library files F:OSLB, F:RXLB and F3RXLB is an important consideration, for two reasons:

1. The standard ordering described in the Library Generation section is such that OS:LNK can link edit the FORTRAN program with the library in a single pass.

2. In the RTX libraries the modules which are loaded between RTX/IOX Segments 1 and 2 are those which are otherwise vulnerable to re-entrance. RTX contains logic which assists in preventing re-entrance to the routines within its boundaries by a subsequent call before the first call has completed.

Thus alteration of a library file to add or replace a program must take these ordering factors into account. Basically, the user must be sure that the first reference to a routine occurs prior to that routine's being passed through the link editor, so as to insure its being loaded.

With these considerations in mind, the user has various methods at his disposal in altering the library, as described below.

To replace a library module with another (as in an update) the user should follow the Library Generation description, substituting the new module for the old one.

To add a new routine to the library, or to replace a single routine on the library (which was originally catalogued from a paper tape module containing other routines which the user wishes to retain), the user may regenerate the library file by following the description in the Library Generation section, and merging in the new routine at the appropriate place, bearing in mind the ordering restrictions mentioned above. If replacing a routine of the same name which already exists on a paper tape module, it is not necessary for the user to delete the old routine, but simply to merge in the new routine immediately preceding the tape module containing the old routine. Alternatively, if a new routine is referenced by the compiled FORTRAN program rather than from within some routine in the library file, the routine need not be included during library generation at all, but simply referenced as the LI file during OS:LNK time. Once the new program has been linked, the LI file may be re-assigned to the FORTRAN library before continuing with OS:LNK.

ADDING FORTRAN LOGICAL UNIT NUMBERS TO OS

The standard OS system contains within its Logical Unit Table (LUT) references to FORTRAN units 1 through 6. The user may add additional entries for any unit number between 7 and 99, and set default assignments for any unit number to a specific physical device (as is currently done for units 5 and 6, which are default-assigned to the card reader and line printer, respectively). Adding FORTRAN unit numbers requires alteration of two areas: the LUT table within OS Root, and the OS File Control Block (FCB) tables within the OS I/O Interface (F:OSIO) in the OS Library File (F:OSLB).

Altering the LUT in OS ROOT

Each delivered OS system includes an OS Root listing (96530-10), and its corresponding source program paper tape. Changes to OS Root are most easily accomplished by addition, deletion, or replacement of source lines using the OS:SFE utility.

The logical unit table begins at the label "LUT:" in OS ROOT. Each entry in the table is six words long, as follows:

Word 1	Logical Unit name, in ASCII, 2 characters (word 1 may be given any label, as it is not referenced and is only for the convenience of the reader).
Word 2	Address of current physical unit (if using default assignment).
Word 3	Address of initial (default) physical unit (if using default assignment).
Word 4-6	Used to hold a file name - should be set to zero at assembly time.

In the standard setup, FORTRAN units 1 through 6 comprise the last six entries in the LUT. It is after these that additional units should be added.

Example: to add a unit (unassigned) to the LUT, the entry should be coded:

```
DATA '07',0,0
RES 3,0
```

The first data word, if the unit number is between 1 and 9, must be of the form '07', not '7' or ̸b7: the leading zero must be supplied.

Example: to add unit 13 to the LUT, default-assigned to the high speed paper tape reader:

```
DATA '13',PR,PR
RES  3,0
```

Note that the second and third words must both contain addresses. The addresses used must be one of the labels which appear in the physical unit table. This table is found directly behind the logical unit table in OS Root, and begins at the label "PUT:".

Once the OS Root source file has been edited with the desired changes, it may be assembled with OS:ASM, and the object output used to re-generate the OS system, following the description in the OS User's Manual.

OS File Control Block (FCB) Tables

The standard OS File Control Block (FCB) Tables, which are part of the OS I/O Interface Module (F:OSIO), contains six File Control Blocks (for FORTRAN units 1 through 6) which are required by the OS I/O drivers during execution of a FORTRAN program under OS control. (Execution under RTX control does not require FCB tables and so F:RXLB and F3RXLB need not be altered when adding unit numbers.)

The listing of the standard FCB tables is reproduced below (see Figure 5-3). Each FCB is referenced by the label F:RUnn, where nn is the FORTRAN unit number.

NOTE

> The FCB tables for FORTRAN units 1-5 are separate programs, each terminated with an END statement, and reside prior to the Interface itself in the FORTRAN/OS I/O Interface Module (F:OSIO). FORTRAN unit 6 is used to output run-time error messages, since it is the default OUTPUT device. Therefore, it is assembled within the interface itself, to insure its being linked unconditionally.

When the compiler encounters a reference to a unit number (e.g., an I/O statement such as "WRITE (3,25)"), it generates an external reference to F:RU03 and causes the corresponding FCB to be linked.

In addition to the FCB's themselves, the FCB tables include three short programs, called F:RUNN, F:RUIN, and F:RUOT. Each is described below:

F:RUNN Program

If, during a FORTRAN compilation, the compiler encounters a statement of the form

```
WRITE (JUNIT,25)
```

where JUNIT is an integer variable, the specific unit number is indeterminate, and the compiler does not know which FCB to reference. It therefore creates an external reference to F:RUNN, which is merely a list of references to all FCB's. Thus linking of the F:RUNN routine causes loading of all FCB's.

F:RUIN and F:RUOT Programs

A FORTRAN INPUT statement does not reference any unit. Thus the compiler will reference F:RUIN, which in turn references F:RU05, the FCB for FORTRAN unit 5. Similarly, a FORTRAN OUTPUT statement causes the compiler to generate an external reference to F:RUOT, which in turn references F:RU06, the FCB for FORTRAN unit 6. (In addition, the FORTRAN Run-time Error output routine outputs to unit 6. For this reason, unit 6 should always be assigned to the list device.)

FCB Format

Each FCB is a block of 21 words in length:

Word 1 - A "CHAN" directive, which allows the I/O Interface to search through each linked FCB and compare Word 3 against the requested unit number. Word 1 must be labeled F:RUxx, where xx is the unit number. (Units 1 through 9 must be labelled F:RU01 - F:RU09.) The chain operand must be F:RFCB.

Word 2 - must contain zero.

Word 3 - must contain the logical unit number, in ASCII, which matches the last two characters of Word 1's label.

Words 4-21 - must contain zero.

Adding FCBs to the Tables

Adding one or more FCB's to the OS Library requires the following:

1. The F:RUNN table, which is referenced when a variable is used for a FORTRAN unit number, must be reassembled to include a reference to each new unit. Refer to the sample listing below, of the F:RUNN table, each entry of which is a LOAD instruction for the individual FCB table to be loaded.

2. A 21-word FCB table must be assembled for each new unit number to be added, as described above.

Once the new F:RUNN module and new FCB(s) have been assembled, re-generate the OS Library (F:OSLB) as described previously, merging the files as follows:

FORTRAN Basic External Functions	(F:EXTR)
FORTRAN Math and I/O Routines	(F:MATH)
New F:RUNN Module	
New FCB tables	
FORTRAN/OS I/O Interface	(F:OSIO)

```
PAGE   0001   09/27/74   10:48:46   CALLER TO FORTRAN/OS FCB'S 1-6
LINE   LOC   INST ADDR   LABEL   MNEM OPERAND    COMMENT
0002                     *                     (F:RUNN)
0003                     *COPYRIGHT 1974 COMPUTER AUTOMATION INC
0004                     *
0005                     *THIS SEGMENT IS REFERENCED BY THE FORTRAN
0006                     *COMPILER WHEN IT ENCOUNTERS A VARIABLE UNIT NUMBER
0007                     *E.G., "WRITE (N)"
0008   0000                      NAM   F:RUNN
0009                             LOAD  F:RU01    CALL UNIT 1 FCB
0010                             LOAD  F:RU02    CALL UNIT 2 FCB
0011                             LOAD  F:RU03    CALL UNIT 3 FCB
0012                             LOAD  F:RU04    CALL UNIT 4 FCB
0013                             LOAD  F:RU05    CALL UNIT 5 FCB
0014                             LOAD  F:RU06    CALL UNIT 6 FCB
0015                     *
0016                     F:RUNN END

0000   ERRORS
```

```
PAGE   0001   09/27/74   10:48:46   CALLER TO FORTRAN/OS INPUT FCB
LINE   LOC   INST ADDR   LABEL   MNEM OPERAND    COMMENT
0018                     *                  (F:RUIN)
0019                     *COPYRIGHT,1974 COMPUTER AUTOMATION INC
0020                     *
0021                     *THIS SEGMENT IS REFERENCED BY THE FORTRAN
0022                     *COMPILER WHEN IT ENCOUNTERS AN "INPUT"
0023                     *SOURCE STATEMENT. (STANDARD INPUT UNIT IS 5).
0024                     *
0025   0000                      NAM   F:RUIN    CALL INPUT UNIT FCB
0026                             LOAD  F:RU05    CALL UNIT 5 FCB
0027                     F:RUIN END

0000   ERRORS
```

Figure 5-3. Sample FCB Tables

```
PAGE   0001   09/27/74   10:48:46   CALLER TO FORTRAN /OS OUTPUT FCB

LINE   LOC    INST ADDR   LABEL   MNEM OPERAND    COMMENT
0029                      *                     (F:RUOT)
0030                      *COPYRIGHT 1974 COMPUTER AUTOMATION INC
0031                      *
0032                      *THIS SEGMENT IS REFERENCED BY THE FORTRAN
0033                      *COMPILER WHEN IT ENCOUNTERS AN "OUTPUT"
0034                      *SOURCE STATEMENT. (STANDARD OUTPUT UNIT IS 6).
0035                      *
0036   0000                       NAM  F:RUOT    CALL OUTPUT UNIT FCB
0037                              LOAD F:RU06    CALL UNIT 6 FCB
0038                      F:RUOT  END

0000   ERRORS
```

```
PAGE   0001   09/27/74   10:48:46   FORTRAN/OS FCB TABLES

LINE   LOC    INST ADDR   LABEL   MNEM OPERAND    COMMENT
0040                      *            (F:RU01 - F:RU06)
0041                      *COPYRIGHT 1974 COMPUTER AUTOMATION INC
0042                      *
0043                      *       THIS PROGRAM CONTAINS SEVERAL 21-WORD
0044                      *TABLES TO BE USED BY THE FORTRAN/OS RUNTIME
0045                      *INTERFACE FOR FILE CONTROL BLOCKS.
0046   0000                       NAM  F:RU01    UNIT 1
0047   0000                       REL  0
0048                      *
0049                      *       UNIT 1 FCB
0050                      *
0051   0000               F:RU01  CHAN F:RFCB    CHAIN NODE
0052   0001 0000                  DATA 0         ECB
0053   0002 B0B1                  DATA '01'      LUN
0054   0003 0000                  RES  18,0
0055                              END

0000   ERRORS
```

Figure 5-3. Sample FCB Tables (Cont'd)

```
PAGE  0001  09/27/74  10:48:46

LINE  LOC   INST ADDR  LABEL   MNEM OPERAND   COMMENT
0056  0000                     NAM  F:RU02    UNIT 2
0057  0000                     REL  0
0058                   *
0059                   *       UNIT 2 FCB
0060                   *
0061  0000             F:RU02  CHAN F:RFCB    CHAIN NODE
0062  0001 0000                DATA 0         ECB
0063  0002 E0B2                DATA '02'      LUN
0064  0003 0000                RES  18,0
0065                           END

0000  ERRORS
```

```
PAGE  0001  09/27/74  10:48:46

LINE  LOC   INST ADDR  LABEL   MNEM OPERAND   COMMENT
0066  0000                     NAM  F:RU03    UNIT 3
0067  0000                     REL  0
0068                   *
0069                   *       UNIT 3 FCB
0070                   *
0071  0000             F:RU03  CHAN F:RFCB    CHAIN NODE
0072  0001 0000                DATA 0         ECB
0073  0002 B0B3                DATA '03'      LUN
0074  0003 0000                RES  18,0
0075                           END

      ERRORS
```

Figure 5-3. Sample FCB Tables (Cont'd)

```
PAGE   0001   09/27/74   10:48:46

LINE   LOC    INST ADDR   LABEL    MNEM  OPERAND    COMMENT
0076   0000                        NAM   F.RU04     UNIT 4
0077   0000                        REL   0
0078                      *
0079                      *        UNIT 4 FCB
0080                      *
0081   0000               F:RU04   CHAN  F:RFCB     CHAIN NODE
0082   0001 0000                   DATA  0          ECB
0083   0002 B0B4                   DATA  '04'       LUN
0084   0003 0000                   RES   18,0
0085                               END

0000   ERRORS
```

```
PAGE   0001   09/27/74   10:48:46

LINE   LOC    INST ADDR   LABEL    MNEM  OPERAND    COMMENT
0086   0000                        NAM   F:RU05     UNIT 5
0087   0000                        REL   0
0088                      *
0089                      *        UNIT 5 FCB
0090                      *
0091   0000               F:RU05   CHAN  F:RFCB     CHAIN NODE
0092   0001 0000                   DATA  0          ECB
0093   0002 B0B5                   DATA  '05'       LUN
0094   0003 0000                   RES   18,0
0095                               END

0000   ERRORS
```

Figure 5-3. Sample FCB Tables (Cont'd)

ADDING A DISK DIB TO THE RTX LIBRARY FILE

The following discussion applies to the user who wishes to create his own RTX disk (or floppy disk) DIB (s) (Device Information Blocks) and to specify his own disk file boundaries.

The standard (Non-FORTRAN) disk DIB described in the RTX User's Manual differs somewhat from a disk DIB which is to be used in FORTRAN. Specifically, there exist within RTX two disk I/O handler routines, one for FORTRAN usage, and one for non-FORTRAN usage. The non-FORTRAN handler has no provision for writing or reading an end-of-file mark, and it also requires the user to maintain the current record number within the user's IOB. Since the FORTRAN user has no access to the IOB (all RTX IOB's are built and maintained within the I/O Interface module), a special disk handler for FORTRAN exists within RTX which allows for these differences.

Because the FORTRAN disk handler differs from the standard RTX disk handler, two additional considerations must be made by the FORTRAN user when creating a disk DIB:

1. The RTX Manual describes the disk DIB as a 15-word table. The FORTRAN disk handler in RTX requires an additional word (16 words in all) which is used to hold the current record number in the disk file. This word should contain a binary zero as its initial value.

2. The FORTRAN Disk DIB name, which is referenced in the Unit Assignment Table must be of the form "D:DKFx" (or "D:FDFx" if floppy disk), where x may be any alphanumeric character. This format notifies the RTX disk handler that the DIB refers to a FORTRAN disk file.

Figure 5-4 illustrates the proper format for a disk DIB for FORTRAN. The user should assemble one of these DIB's for each file he wishes to create on the disk. If more than one, each DIB should terminate with an assembler END directive, so that it may be linked to the FORTRAN program in library mode. Once the DIB has been created, the RTX FORTRAN Library file may be re-generated, following the procedure described in this section, with the new DIB (s) inserted in front of the RTX/IOX Segment 1 module, which is the segment containing the standard DIBs.

Alternatively, the RTX Library does not need to be permanently changed. The user may instead create the desired DIB (s), and include the module into the OS:LNK procedure at link edit time, by linking the RTX mainline and tasks, then the new DIB module, then the Library file.

Figure 5-4 is a listing of one of the standard FORTRAN disk DIB's which currently exist in RTX/IOX:

```
PAGE  0001  09/24/74  08:30:19  94500-10      I O X   T A B L E S
            D:DKF1 - FORTRAN DISK DIB

LINE  LOC   INST ADDR  LABEL   MNEM OPERAND  COMMENT
0365                   *       43 SERIES DISK, REMOVABLE PLATTER
0366                   *       CYLINDERS 0-199
0367  0000                     NAM  D:DKF1
0368                           EXTR C:DKF
0369        0000       D:DKF1  EQU  $
0370  0000                     CHAN X::
0371  0001  0000               DATA C:DKF,0,0,:15DD,'DK'
      0002  0000
      0003  0000
      0004  15DD
      0005  C4CB
0372  0006  C6B1               DATA 'F1',0,0,0,0,0,:C02,:1800,4800,0
      0007  0000
      0008  0000
      0009  0000
      000A  0000
      000B  0000
      000C  0C02
      000D  1800
      000E  12C0
      000F  0000
0373                           END

0000  ERRORS
```

Figure 5-4. Sample FORTRAN Disk DIB

USER-CREATED SUBPROGRAMS

The user who wishes to write his own subprograms in FORTRAN Assembly language and CALL them from his main program should follow the calling and receiving sequences shown below, as this is the object code generated by a CALL statement.

For execution under OS (RTX option not used),

```
CALL MYSUB (ARG1,ARG2...)
```

will generate the following object code:

```
JST    *BP (MYSUB)
DATA    n (where n is the number of arguments)
DATA    ARG1
DATA    ARG2
     etc.
```

For execution under RTX (RTX option used),

CALL MYSUB (ARG1,ARG2...) will generate the following object code:

```
JST *BP (SUBR: )
DATA MYSUB
DATA n                    (where n is the number of arguments)
DATA ARG1
DATA ARG2
DATA...
```

The SUBR: routine prevents re-entrance for RTX usage; the user's subprogram, to terminate the re-entrance-protecting effect of SUBR: , must include a call to SUBX: , as follows:

```
          JST    SUBX:
MYSUB     ENT
           •
           •
           •
          JMP    MYSUB-1          (on return from the routine)
```

instead of RTN MYSUB.

NOTE

The same assembly language subprogram may be used under both OS and RTX monitors, if it is set up using the SUBX: call shown above. The OS library contains a "dummy" SUBX: routine (within F:OSIO) to handle this situation.

Accessing Arguments

If the called subprogram is required to handle arguments passed to it by the calling program, then the user may access them using the F:RDMY library subprogram, which will move the arguments from the caller to the user's subprogram automatically:

```
CALL example         CALL FRED (UP, DOWN, MES1, N)
Subprogram example   FRED      ENT
                               JST *BP (F:RDMY)
                               DATA 4          (no. of arguments)
                     UP        RES 1
                     DOWN      RES 1
                     MES1      RES 1
                     N         RES 1
                               .
                               .
                               .
                               RTN FRED
```

Explanations:

1. The call to F:RDMY must immediately follow the subprogram's entry point;

2. The word following this call must contain the correct number of arguments, since this is checked by F:RDMY against the number supplied;

3. The following words, which may be labelled to correspond with the argument names, will be set by F:RDMY to the actual (base) address of each argument, the order corresponding to the order of arguments as shown;

4. The address contained in the entry location labelled FRED will be updated appropriately to point to the first instruction beyond the code generated for the CALL statement.

5. Even if no arguments are required, it is still necessary to put DATA 0 after the call to F:RDMY, which, having checked that no arguments were supplied and updated the return address, would return control to the subprogram at the instruction after the DATA 0 statement.

From the above, it can be seen that F:RDMY provides a safe and straightforward method for acquiring arguments and setting the correct return address. It can of course be programmed differently with the subprogram itself accessing the argument list via the address placed in the entry point. However the method shown is the recommended one.

APPENDIX A

DEBUGGING AIDS

DEBUGGING AIDS

During checkout of a FORTRAN program, the following aids are available to the user.

Fortran Trace Option

The Trace option, when requested prior to a compilation, will cause the compiler to generate, in addition to the normal object code, additional run-time calls which will cause the program to print a trace map onto unit 6 during execution. (Refer to compiler options section - Trace option).

OS:DBG, RTX ZBG

The OS:DBG and RTX ZBG utility programs may be used in conjunction with the executing program, for breakpointing and other debugging capabilities (refer to the OS:DBG description in the OS User's Manual or the ZBG description in the RTX User's Manual, for a complete description of these utilities). It will be necessary to include an object listing in the compilation, which may be used in conjunction with the OS:LNK memory map to follow the program flow during execution.

Normally, the link map is used to set DEBUG relocation registers, and then breakpointing may be done using the FORTRAN object listing(s). Observe the following precautions:

1. FORTRAN object code is generally organized with various data areas beginning at relative location zero, followed by the executable code; thus F:MAIN, the starting location, will not normally be at relative location zero. The relocation register should be set to correspond with relative location zero, rather than F:MAIN.

2. If the FORTRAN program to be debugged uses floating point values (Real, Double Precision or Complex), it will not be possible to breakpoint into a sequence of code which calls the Floating Point Interpreter. For example, the sample listing in Figure A-1 contains object code for both integer and floating point processing:

```
GE 0001   09/10/74   13:32:43    FORTRAN (X3) COMPILATION
          OPTIONS:   LO

 C
 C        INTEGER PROCESSING
 C
          J=-13
          K=IABS(J*9)
 C
 C        FLOATING POINT PROCESSING
 C
          A=-13.0
          B=ABS(A*9.0)
          OUTPUT J,K,A,B
          END
```

```
 0002   09/10/74   13:32:43    FORTRAN (X3) COMPILATION
        OPTIONS:   LO

ALAR ALLOCATION

CN   NAME    TYPE      WORDS   LOCN   NAME   TYPE      WORDS

000  J       INTEGER     1     :0001  K      INTEGER     1
002  A       REAL        2     :0004  B      REAL        2
```

Figure A-1. Integer and Floating Point Sample Listing

```
PAGE 0003   09/10/74   13.32:43    FORTRAN (X3) COMPILATION
            OPTIONS:   LO

0001 C
0002 C       INTEGER PROCESSING
0003 C
0004         J=-13
                :0006 :C70D              LAM    13
                :0007 :9E07              STA    J
0005         K=IABS(J*9)
                :0008 :F900 B            JST    *BP(F:RMPY)
                :0009 :0000 F            DATA   #IC1                       :0009
                :000A :3080 F            JAP    #M0
                :000B :0310              NAR
                 000C :9E0B    #M0       STA    K
0006 C
0007 C       FLOATING POINT PROCESSING
0008 C
0009         A=-13.0
                :000D :F900 B            JST    *BP(F:RINT)
                :000E :AA00 F            LDR    #RC0                       :C250:0000
                :000F :9E0D              STA    A
0010         B=ABS(A*9.0)
                :0010 :8200 F            MPM    #RC1                       :4210:0000
                :0011 :0005              ABS
                :0012 :9E0E              STA    B
0011         OUTPUT J,K,A,B
                :0013 :0000              XIT
                :0014 :F900 B            JST    *BP(F:ROUT)
                :0015 :F900 B            JST    *BP(F:RIOL)
                :0016 :0000              DATA   J
                :0017 :F900 B            JST    *BP(F:RIOL)
                :0018 :0001              DATA   K
                :0019 :F900 B            JST    *BP(F:RROL)
                :001A :0002              DATA   A
                :001B :F900 B            JST    *BP(F:RROL)
                :001C :0004              DATA   B
                :001D :F900 B            JST    *BP(F:RSIO)
0012         END
                 001E :F900 B            JST    *BP(F:RSTO)
                :001F :0000              DATA   0
                :0020 :C250    #RC0      DATA   -15792
                :0021 :0000              DATA   0
                :0022 :4210    #RC1      DATA   16912
                :0023 :0000              DATA   0
                :0024 :0009    #IC1      DATA   9

SUBPROGRAMS CALLED

NAME     TYPE     ARGS    NAME     TYPE     ARGS    NAME     TYPE     ARGS

IABS     INTEGER  1       ABS      REAL     1       F:ROUT   RUNTIME
F:RIOL   RUNTIME          F:RROL   RUNTIME          F:RSIO   RUNTIME
F:RSTO   RUNTIME          F:RUNN   RUNTIME          F:RREL   RUNTIME
```

INTERPRETED MACRO-INSTRUCTIONS

Figure A-1. Integer and Floating Point Sample Listing

```
AGE 0004  09/10/74  13:32:43   FORTRAN (X3) COMPILATION
         OPTIONS   LO

RFZ    RUNTIME         F:RFF    RUNTIME         F:RMPY   RUNTIME
:RINT   RUNTIME

ATEMENT LABELS

CN   LABEL   USE          LOCN   LABEL   USE          LOCN   LABEL   USE

000C #M0

ITRY=:0006
OGRAM SIZE=.0025 WORDS
ASE PAGE USED=:0007 WORDS
MPILATION COMPLETE  0 ERRORS
```

Figure A-1. Integer and Floating Point Sample Listing

The object code generated for the integer processing section (locations :0006-:000C) may be debugged using the breakpoint feature in the normal manner (note, however, that the data statement at location :0009 is a parameter to the F:RMPY routine and is not executed.

The object code generated for the floating point processing section (locations :000D-:0013), however, are not normal machine language instructions, but rather macro-instructions which are decoded by the floating point interpreter module (F:RINT), and a break point inserted in this sequence will cause incorrect operation of the FORTRAN program. It is the XIT macro instruction which causes the program to return from the "interpretive mode" of operation back to normal machine language instruction processing.

Thus it is permissible, in this example, to breakpoint from location :000D to location :0014, but not to breakpoint into this area.

The following FORTRAN routines cause "interpretive mode" processing:

F:RINT	(Floating Point Interpreter)
F:RCPX	(Complex Arithmetic Processor)
F:RDBL	(Double Precision Arithmetic Processor)
F:RREL	(Real Arithmetic Processor)

and should be recognized as such by the user.

The following macro-instructions signal termination of "interpretive mode" processing:

INT	(Convert to Integer and Exit from Interpretive Mode)
XIT	(Exit from Interpretive Mode)
XNL	(Exit from Interpretive Mode but do not unlock. Required by RTX, this function protects the contents of the floating point accumulator.)

They also indicate that the following instruction (not the exit instruction) may be used as a breakpoint.

APPENDIX B

SAMPLE JOB SEQUENCES

INTRODUCTION

The following sequences are to serve as sample control commands for various procedures in compiling, linking and executing FORTRAN programs. (Examples of System Generation procedures and alteration of the libraries are shown in section 5 under their related headings.) All examples assume card input. The compiled binary output is called PROG1, and the linked (executable) binary output is called PROG2. [ ] indicates optional parameters.

To transfer control from the teletype keyboard to the card reader, enter

```
/JOB
/BA CR
```

through the keyboard.

TO COMPILE, LINK AND EXECUTE UNDER OS

```
/AS BO=DO.PROG1
/EX FORT:4 [,option,option...]
(FORTRAN source deck(s), each terminated with the END statement)
/*
/AS BI=DO.PROG1,LI=DO.F:OSLB,BO=DO.PROG2
/EX OS:LNK,LL,TE
/AS SF=DC [, also assign any required FORTRAN unit numbers at this time]
/EX PROG2
Data Deck (if any), terminated with "/*"
/JOB (return CI control to teletype)
```

TO COMPILE, LINK AND EXECUTE UNDER OS, USING OS:DBG

```
/AS BO=DO.PROG1
/EX FORT:4,LOBJ [,option,option...]
(FORTRAN source deck(s), each terminated with an END statement)
/*
/AS BI=DO.PROG1,LI=DO.F:OSLB,BO=DO.PROG2
/EX OS:LNK,LL,TE
/LO PROG2
/AS CI=TK [, assign FORTRAN unit numbers at this time]
Data Deck (if any)
/*
```

Input via the keyboard:

```
/EX OS:DBG
```

At this time, OS:DBG is entered; OS:DBG's relocation register R0 is set to the start of the main program, which may not be the first executable instruction. (The execution address is noted on the OS:LNK memory map.) The FORTRAN object listing and OS:LNK memory map will serve as reference listings during the debugging process.

TO ASSEMBLE MAINLINE, COMPILE TASKS, LINK AND EXECUTE UNDER RTX

```
(LSI-2 example)
/JOB
/AS BO=DO,F:MAIN
/EX OS:ASM
(Mainline source deck)
/AS BO=DO.TASKS
/EX FORT:4,RT  [,option,option...]
(FORTRAN task(s), each terminated with an END statement)
/*
/AS BI=DO.F:MAIN,LI=DO.TASKS,BO=DO.PROG
/EX OS:LNK,NH,AB=100,SR=60,LL
/AS LI=DO.F:RXLB
LL,TE
/EX OS:ILD,DO.PROG

(LSI-3/05 example)
/JOB
/AS BO=DO.F:MAIN
/EX MACRO3
(Mainline source deck)
/AS BO=DO.TASKS
/EX FORT:4,T3  [,option,option...]
(FORTRAN task(s), each terminated with an END statement)
/*
/AS BI=DO.F:MAIN,LI=DO.TASKS,BO=FO.PROG
/EX OS:LNK,T3,AB=100,SR=20,SX=1,LL
/AS LI=DO.F3RXLB
LL,TE
```

At this time, the linked PROG or floppy F0 may be loaded into an LSI-3/05 processor using the directoried Load/Dump program (DLD).

Appendix C

FORTRAN RUN-TIME SUBPROGRAM LIST

FORTRAN BASIC EXTERNAL FUNCTIONS

Most of these functions reside in the F:EXTR (or F3EXTR) library module. Those preceded with an asterisk reside in the F:MATH (or F3MATH) module.

ABS	Real absolute value of a real argument
AIMAG	Convert imaginary part of a complex value to real
AINT	Truncate real argument to integer and back to real
*ALOG	Real natural logarithm of a real argument
*ALOG10	Real common logarithm of a real argument
AMAX0	Real maximum value of integer arguments
AMAX1	Real maximum value of real arguments
AMIN0	Real minimum value of integer arguments
AMIN1	Real minimum value of real arguments
AMOD	Real remainder of real modulus real
ATAN	Real arctangent of real argument
ATAN2	Real arctangent of two real coordinates
*CABS	Real absolute value of a complex argument
CCOS	Complex cosine of a complex argument
CEXP	Complex exponential of a complex argument
CLOG	Complex natural logarithm of a complex argument
CMPLX	Convert two real values to complex
CONJG	Conjugate a complex argument
*COS	Real cosine of a real argument
*COSH	Hyperbolic cosine of a real argument
CSIN	Complex sine of a complex argument
CSQRT	Complex square root of a complex argument
DATAN	Double prec. arc. tangent of a double prec. argument
DATAN2	Double prec. arctangent of two double prec. coordinates
DBLE	Convert a double prec. value to integer
DCOS	Double prec. cosine of a double prec. argument
*DEXP	Double prec. exponential of a double prec. argument
DFLOAT	Convert integer to double precision
DINT	Truncate double prec. value to integer and back to double prec.
*DLOG	Double prec. natural logarithm of a double prec. argument
*DLOG10	Double prec. common logarithm of a double prec. argument
DMAX0	Double prec. maximum value of integer arguments
DMAX1	Double prec. maximum value of double prec. arguments
DMIN0	Double prec. minimum value of integer arguments
DMIN1	Double prec. minimum value of double prec. arguments
DMOD	Double prec. remainder of double prec. modulus double prec.
DSIN	Double prec. sine of double prec. argument
DSQRT	Double prec. square root of double prec. argument

DTAN	Double prec. tangent of double prec. argument
DTANH	Double prec. hyperbolic tangent of double prec. argument
*EXP	Real exponential of real argument
FLOAT	Convert integer value to real
IDINT	Convert double prec. value to integer
IFIX	Convert real value to integer
INT	Convert real value to integer
MAX0	Integer maximum value of integer arguments
MAX1	Integer maximum value of real arguments
MIN0	Integer minimum value of integer arguments
MIN1	Integer minimum value of real arguments
MOD	Integer remainder of integer modulus integer
REAL	Real part of a complex argument
*SIN	Real sine of a real argument
*SINH	Hyperbolic sine of a real argument
SNGL	Convert double prec. value to real
*SQRT	Real square root of a real argument
TAN	Real tangent of real argument
TANH	Real hyperbolic tangent of real argument

FORTRAN MATH AND I/O ROUTINES

Most of these routines reside in the F:MATH (or F3MATH) library module. Those preceded with an asterisk reside in the F:EXTR (or F3EXTR) module. (Program name in parentheses following description is the first entry point in the routine.)

F:EATL	Argument too large
F:EBAZ	Both arguments zero (F:EATL)
F:EDVO	Division by zero (F:EATL)
F:EINA	Incorrect number of arguments (F:EATL)
F:ELOC	Error Location (F:RBPG)
F:ENGA	Negative argument (F:EATL)
F:EOVR	Overflow (F:EATL)
F:EQL1	Error Quote 1 (F:RBPG)
F:EQL2	Error Quote 2 (F:RBPG)
F:ERRC	Error print and continue (F:ERRC)
F:ERRS	Error print and TERM: (F:ERRC)
F:ESGL	Singularity (F:EATL)
F:IAIN	Internal aint (AINT)
F:IALG	Internal alog (ALOG)
*F:IAT2	Internal atan2 (ATAN)
F:ICAB	Internal cabs (CABS)
F:ICOS	Internal cos (SIN)
F:ICSH	Internal cosh
F:IDAD	Double add for functions (F:IDAD)
F:IDDV	Double divide for functions (F:IDAD)
*F:IDIN	Internal dint (DINT)
F:IDLD	Double load for functions (F:IDAD)

F: IDLG	Internal dlog (DLOG)
F: IDMV	Double move for functions (F: IDAD)
F: IDML	Double multiply for functions (F: IDAD)
F: IDNM	Double normalize for functions (F: IDAD)
F: IDSL	Double shift left one (F: IDAD)
F: IDST	Double store for functions (F: IDAD)
F: IDSB	Double subtract for functions (F: IDAD)
F: IDUN	Double unpack for functions (F: IDAD)
F: IDXP	Internal dexp (DEXP)
F: IEXP	Internal exp (EXP)
F: IFC1	Complex fetch and unpack one (F: IRAD)
F: IFD1	Fetch and unpack one (F: IDAD)
F: IFD2	Fetch and unpack two (F: IDAD)
F: IFI1	Integer fetch and unpack one (F: IIUN)
F: IFI2	Integer fetch and unpack two (F: IIUN)
F: IIUN	Integer fetch and unpack (F: IIUN)
F: IRAD	Real add for functions (F: IRAD)
F: IRDV	Real divide for functions (F: IRAD)
F: IRLD	Real load for functions (F: IRAD)
F: IRMV	Real move for functions (F: IRAD)
F: IRML	Real multiply for functions (F: IRAD)
F: IRSB	Real store for functions (F: IRAD)
F: IRST	Real subtract for functions (F: IRAD)
F: IRUN	Real unpack for functions (F: IRAD)
F: ISIN	Internal sin (SIN)
F: ISNH	Internal sinh
F: ISQR	Internal sqrt (SQRT)
F: RACE	Extended Accumulator Exponent (F: RBPG)
F: RACS	Extended Accumulator Sign (F: RBPG)
F: RAC1	Extended Accumulator Word 1 (F: RBPG)
F: RAC2	Extended Accumulator Word 2 (F: RBPG)
F: RAC3	Extended Accumulator Word 3 (F: RBPG)
F: RAC4	Extended Accumulator Word 4 (F: RBPG)
F: RARG	A register (interpreter) (F: RBPG)
F: RBPG	Base Page Definitions
F: RBSP	Backspace a record
F: RCAD	Complex add (F: RCPX)
F: RCBE	Cube A register
F: RCDV	Complex divide (F: RCPX)
F: RCGO	Computed Goto
F: RCIP	Complex to integer power
F: RCLD	Complex load (F: RCPX)
F: RCML	Complex multiply (F: RCPX)
F: RCNG	Complex negate (F: RCPX)
F: RCOL	Complex input/output element Formatted (F: RINP)
F: RCOM	Complex input/output element unformatted (F: RRU)
F: RCPX	Complex arithmetic package entry
F: RCRP	Complex repack (F: RCPX)
F: RCSB	Complex subtract (F: RCPX)

F:RCST	Complex store (F:RCPX)
F:RCTD	Complex to double (F:RCPX)
F:RCTI	Complex to integer (F:RCPX)
F:RCTR	Complex to real (F:RCPX)
F:RCUS	Complex input/output array element unformatted (F:RINP)
F:RCUT	Complex input/output array element unformatted (F:RRU)
F:RDAB	Double ABS (F:RDBL)
F:RDAD	Double add (F:RDBL)
F:RDBL	Double precision arithmetic package entry
F:RDDM	Double DIM (F:RDBL)
F:RDDV	Double divide (F:RDBL)
F:RDEN	Decode with optional N (F:RINP)
F:RDIP	Double precision to integer power
F:RDIV	Signed DIV
F:RDLD	Double load (F:RDBL)
F:RDML	Double multiply (F:RDBL)
F:RDMY	Setup argument addresses
F:RDOL	Double precision input/output element formatter (F:RINP)
F:RDOM	Double precision input/output element unformatted (F:RRU)
F:RDRP	Double precision to integer power (F:RIDP)
F:RDSB	Double subtract (F:RDBL)
F:RDST	Double store (F:RDBL)
F:RDTC	Double to complex (F:RCPX)
F:RDTI	Double to integer (F:RDBL)
F:RDTR	Double to real (F:RDBL)
F:RDUS	Double precision input/output array element formatted (F:RINP)
F:RDUT	Double precision input/output array element unformatted (F:RRU)
F:REND	End-of-file
F:RENN	Decode with Optional N (F:RINP)
F:RERR	Diagnostic error during compile formatted (F:RINP)
F:RFAA	Format argument address (F:RINP)
F:RFAD	Format skip asterisks and dollar (F:RFAD)
F:RFAF	Format asterisk flag (F:RINP)
F:RFD	Format conversion D (F:RFIR)
F:RFDA	Format back fill dollar and asterisks (F:RFAD)
F:RFDE	Format decimals count (F:RINP)
F:RFDF	Format dollar flag (F:RINP)
F:RFES	Format element size (F:RINP)
F:RFF	Format conversion F (F:RFIR)
F:RFFD	Format fetch from door (F:RINP)
F:RFFQ	Format fill with question marks (F:RFAD)
F:RFG	Format conversion G (F:RFIR)
F:RFI	Format conversion I (F:RFZ)
F:RFIR	Format conversion I Real (F:RFIR)
F:RFL	Format conversion L (F:RFZ)
F:RFPE	Format p scale factor exponent (F:RINP)
F:RFRA	Format return address (F:RINP)
F:RFRN	Format reset window no comma (F:RINP)
F:RFRW	Format reset window (F:RINP)

F: RFSF	Format stop flag (F: RINP)
F: RFSI	Format stop line IO (F: RINP)
F: RFSO	Format store output char (F: RINP)
F: RFSW	Format store in window (F: RINP)
F: RFTS	Format test sign (F: RFAD)
F: RFWB	Format store in window back (F: RFAD)
F: RFWD	Format set window door (F: RINP)
F: RFWE	Format window end (F: RINP)
F: RFWF	Format write flag (F: RINP)
F: RFWI	Format width (F: RINP)
F: RFWS	Format window start (F: RINP)
F: RFZ	Format conversion Z
F: RHFO	Format Hollerith free (F: RFIR)
F: RHUS	Hollerith input/output array element formatted (F: RINP)
F: RHUT	Hollerith input/output array element unformatted (F: RRU)
F: RIAU	Double add unpacked (F: RDBL)
F: RIDP	Integer to double precision power
F: RIDU	Double divide unpacked (F: RDBL)
F: RIIP	Integer to integer power
F: RIMU	Double multiply unpacked (F: RDBL)
F: RING	Double negate (F: RDBL)
F: RINP	Input statement
F: RINT	Integer arithmetic entry (F: RITP)
F: RIOL	Integer input/output element formatted (F: RINP)
F: RIOM	Integer input/output element unformatted (F: RRU)
F: RIRP	Real to integer power
F: RISG	Double SGN (F: RDBL)
F: RISU	Double subtract unpacked (F: RDBL)
F: RITC	Integer to complex (F: RCPX)
F: RITD	Integer to double (F: RDBL)
F: RITP	Runtime interpreter
F: RITR	Integer to real (F: RREL)
F: RIUN	Double unpack (F: RDBL)
F: RIUS	Integer input/output array element formatted (F: RINP)
F: RIUT	Integer input/output array element unformatted (F: RRU)
F: RLOL	Logical input/output element formatted (F: RINP)
F: RLOM	Logical input/output element unformatted (F: RRU)
F: RLUS	Logical input/output array element formatted (F: RINP)
F: RLUT	Logical input/output array element unformatted (F: RRU)
F: RMPY	Signed MPY
F: ROPE	Operand Exponent (F: RBPG)
F: ROPS	Operand Sign (F: RBPG)
F: ROP1	Operand Word 1 (F: RBPG)
F: ROP2	Operand Word 2 (F: RBPG)
F: ROP3	Operand Word 3((F: RBPG)
F: ROP4	Operand Word 4 (F: RBPG)
F: ROUT	Output statement (F: RINP)
F: RPAB	Parameter Block Adr (I/O) (F: RBPG)
F: RPAU	Pause

F: RRAB	Real ABS (F: RREL)
F: RRAD	Real add (F: RREL)
F: RRAU	Real add unpacked (F: RREL)
F: RRDM	Real DIM (R: RREL)
F: RRDP	Real to double precision power (F: RIDP)
F: RRDU	Real divide unpacked (F: RREL)
F: RRDV	Real divide (F: RREL)
F: RREL	Real Arithmetic package entry
F: RREW	Rewind
F: RRF	Read formatted (F: RINP)
F: RRFB	Read formatted with both options (F: RINP)
F: RRFN	Read formatted with END option (F: RINP)
F: RRFR	Read formatted with ERR option (F: RINP)
F: RRIP	Real to integer power
F: RRLD	Real load (F: RREL)
F: RRML	Real multiply (F: RREL)
F: RRMU	Real multiply unpacked (F: RREL)
F: RRNG	Real negate (F: RREL)
F: RROL	Real input/output element formatted (F: RINP)
F: RROM	Real input/output element unformatted (F: RRU)
F: RRPP	Parameter Pointer (Interpreter) (F: RBPG)
F: RRRP	Real to real power (F: RIRP)
F: RRSB	Real subtract (F: RREL)
F: RRSG	Real SGN (F: RREL)
F: RRST	Real store (F: RREL)
F: RRSU	Real subtract unpacked (F: RREL)
F: RRTC	Real to complex (F: RCPX)
F: RRTD	Real to double (F: RDBL)
F: RRTI	Real to integer (F: RREL)
F: RRTN	Trace return (F: RTRF)
F: RRU	Read unformatted (F: RINP)
F: RRUB	Read unformatted with both options (F: RRU)
F: RRUF	Read unformatted with END option (F: RRU)
F: RRUN	Real unpack (F: RREL)
F: RRUR	Read unformatted with ERR option (F: RRU)
F: RRUS	Real input/output array element formatted (F: RINP)
F: RRUT	Real input/output array element unformatted (F: RRU)
F: RSIO	Input/output end of list formatted (F: RINP)
F: RSIP	Input/output end of list unformatted (F: RRU)
F: RSMP	Script multiply
F: RSQR	Square A register
F: RSTN	Trace subprogram entry (F: RTRF)
F: RSTO	Stop
F: RTRF	Trace flow
F: RUAA	Get arg address (F: RUGN)
F: RUAV	Get arg value (F: RUGN)
F: RUGN	Get unit number adr (F: RUGN)
F: RUIR	IO return code process (F: RUGN)
F: RURE	Unlock and return (F: RBPG)

F:RURT	Restore temps (RTX) (F:RUGN)
F:RUST	Save temps (RTX) (F:RUGN)
F:RWF	Write formatted (F:RINP)
F:RWFB	Write formatted with both options (F:RINP)
F:RWFN	Write formatted with END option (F:RINP)
F:RWFR	Write formatted with ERR option (F:RINP)
F:RWU	Write unformatted (F:RRU)
F:RWUB	Write unformatted with both options (F:RRU)
F:RWUN	Read unformatted with END option (F:RRU)
F:RWUR	Read unformatted with ERR option (F:RRU)
F:RXRG	X register (interpreter) (F:RBPG)

<u>LSI-3/05 FORTRAN INSTRUCTION EMULATOR</u> (F3EMUL)

CNSOL:	Software Console Routine
EMUL:	Emulator Mainline
F:RLS3	Emulator Load Caller
MD1A:	Register Change Instructions Module 1
MDASH:	Arithmetic Shift Instructions Module
MDBOV:	Bit to Overflow Instructions Module
MDLSH:	Long Shift Instructions Module
MDMDN:	Multiply/Divide/Normalize Instructions Module
MDRRG:	Register Change Instructions Module 2

<u>FORTRAN RUN-TIME I/O INTERFACE ROUTINES</u> (F:OSIO, F:RXIO and F3RXIO)

F:RU01	Unit 1 FCB Table
F:RU02	Unit 2 FCB Table
F:RU03	Unit 3 FCB Table
F:RU04	Unit 4 FCB Table
F:RU05	Unit 5 FCB Table
F:RU06	Unit 6 FCB Table
F:RUIN	Standard Input Unit FCB Table reference
F:RUNN	Reference to all FCB Tables
F:RUOT	Standard Output Unit FCB Table reference
F:XBSP	Backspace one record
F:XCLS	Close all files
F:XDLL	De-allocate an I/O block
F:XEOF	Write an end-of-file mark
F:XERR	Output an error message
F:XINP	INPUT a record
F:XOUT	OUTPUT a record
F:XPSE	Output a PAUSE message
F:XRCS	Find maximum record size and allocate an I/O block
F:XRDS	Read a record
F:XRWD	Rewind a unit
F:XSTP	Output a STOP message
F:XWTS	Write a record

Appendix D

ERROR MESSAGES/HALTS

COMPILER DIAGNOSTICS DURING SCAN PHASE

Message	Error/ Warning	Comments
ALLOCATION	E	A name appearing in a declaration statement is invalid because of previous usage. For example: COMMON name already in COMMON or not scalar or array. Adjustable dimension not scalar dummy. Name dimensioned or typed twice. Dummy in COMMON, EQUIVALENCE, or EXTERNAL. EQUIVALENCE or DATA array subscript out of range.
ARGUMENT CONVERTED	W	Subprogram argument is wrong type and is converted to right type. This can happen on a library function (proper type is known to the compiler), a statement function (type was determined at the definition), or an ordinary external function (if a previous call is made with different type arguments). Logical cannot be converted to numeric or vice versa; this gets a TYPE CONFLICT error.
ARGUMENT COUNT	E	Wrong number of arguments to subprogram. This can happen in the same cases as ARGUMENT CONVERTED.
ARRAY SIZE	E	Array dimensioned greater than 32K.
BLOCK DATA ONLY	E	This statement may not appear in a BLOCK DATA subprogram.
BLOCK OVERFLOW	E	Working storage has overflowed at a critical point in the processing of an optimization block, where recovery is impossible. All of the source lines in the block will be printed followed by a FORT ER 321 and abort. Get around this problem by juggling the program around, e.g. by inserting a jumped-to label

Message	Error/ Warning	Comments
		to shorten the block. Note that this is a rare occurrence. Normally long blocks will be shortened automatically with no error message.
CONSTANT SIZE	E	Floating constant >1.7E38 or <1.5E-39; or Hexadecimal or Hollerith constant too long for context or more than 255 or less than 1; or DATA repeat count not integer >0.
DIMENSION OUT OF BOUNDS	E	Negative or zero dimension or upper bound less than lower.
DUPLICATE DUMMY	E	Same name used twice as dummy in definition of FUNCTION, SUBROUTINE, or statement function.
DATA COUNT	E	Number of constants not same as number of variables. (Long Hollerith strings may act as several constants.) This will usually be followed by a SYNTAX error.
DATA TYPE	E	Constant not same type as variable. This does not apply to hexadecimal or alphanumeric constants.
EXTRA COMMA	W	Two consecutive commas in a list of items.
FORMAT LABEL	E	Label previously referenced as a FORMAT (e.g. in a READ/WRITE statement).
ID CONFLICT	E	Name can not be used in this context, due to previous usage. See also MISUSED IDENTIFIER.
ILLEGAL ARGUMENT STATEMENT	E	Logical IF may not control a DO or another logical IF.
ILLEGAL DO CLOSE	W	A DO loop may not terminate on a GO TO, DO, arithmetic IF, RETURN, or STOP. If DOs are also improperly nested, this message may not appear. Instead, the label will appear under OPEN DO LOOPS.

Message	Error/ Warning	Comments
ILLEGAL LABEL	E	Label not 1-99999; or DO terminal label has already appeared; or Label on SET op-code not #Xn.
ILLEGAL NUMBER	E	Integer 32767; or format count value of zero; or integer in complex constant; or negated alphanumeric string. See also CONSTANT SIZE and RANGE.
ILLEGAL OP-CODE	E	In-line assembly op-code not recognized. May be caused by "FORTRAN" op-code with an operand or by #Xn label with op-code other than SET.
ILLEGAL SIGN	E	Must be unsigned integer value (e.g. as unit number or ENCODE/DECODE character count).
INDEX NOT ALLOWED	E	In-line assembly op-code cannot be indexed. This appears only on MPY, DIV, NRM: others will get SYNTAX error.
JUMPED TO LABEL	E	This label has previously appeared on a statement that was not a FORMAT.
LABEL MISSING	W	Unlabeled FORMAT statement, or unlabeled statement follows a jump and cannot be reached. Although this is a warning, an unlabeled FORMAT statement will not be generated.
MISSING COMMA	W	Comma needed between two items.
MISSING LABEL	W	A SET op-code has no #Xn label.
MISUSED IDENTIFIER	E	Similar to ID CONFLICT. This name cannot be used this way because of previous usage. For example: DO index is array; or name left of equal sign not scalar or array; or Intrinsic function name used as in-line assembly operand.
MISUSED NAME	E	A system name (containing a colon) was referenced improperly (e.g., as an in-line assembly language operand without a base page (BP) reference preceding it).
MULTI DEFINED	E	Statement label previously defined.

Message	Error / Warning	Comments
NOT ARRAY	E	FORMAT reference name not array.
NOT INTEGER	E	This expression must be integer (e.g. a subscript), but contains at least one non-integer element. The $ marks the end of the expression, but the erroneous element may not be the last one in the expression.
NOT SUBROUTINE	E	Name following CALL is not a subroutine name.
NUMBER OF SUBSCRIPTS	E	Too many or too few subscripts. On the left of an equal sign, an array with no subscripts will have the message UNSUBSCRIPTED.
POSSIBLE ERROR	W	Format stored in integer or logical array probably won't work in ANSI mode. See reference manual.
RANGE	E	In-line assembly operand out of range; or unit number not 1-99. See also CONSTANT SIZE and ILLEGAL NUMBER.
STATEMENT ORDER	E	Certain statements must appear before other statements. In general, declaration statements must come at the beginning. See appendix A of the reference manual.
SYNTAX	E	This is by far the most common error message. It indicates improper sequencing of operands, operators, or punctuation. In a FORMAT, it may be caused by incorrect Hollerith fields.
TYPE CONFLICT	E	Complex expression appears in arithmetic IF or improper assignment, relational, or exponentiation; or Logical operand or argument appears where numeric should or vice versa.
UNDEFINED CONDITIONAL	E	#Xn label has not been defined by a previous SET.
UNDIMENSIONED	E	Name followed by left parenthesis on left of equal sign has not been dimensioned.

Message	Error/ Warning	Comments
UNRECOGNIZABLE	E	More serious than SYNTAX. The compiler cannot determine what kind of statement this is supposed to be. Questionable appearances of this message should be reported to us.
UNSUBSCRIPTED	E	Array appears at beginning of statement (i.e. to left of equal sign) without subscripts.

COMPILER DIAGNOSTICS DURING ALLOCATE PHASE

Message	Comments
ALLOCATION ERRORS	Followed by a list of variable names. These names are involved in illegal EQUIVALENCEs: either a conflict in storage assignment or an extension of COMMON. This message appears at the end of the storage allocation map.
FUNCTION NAME NOT REFERENCED	The name of a FUNCTION, which is supposed to return the result, has never been referenced. This message appears at the beginning of the allocation map.
OPEN DO LOOPS	Followed by lines of the form: 44 OPENED AT LINE 140 This indicates a "DO 44" on line 140, but the terminal statement with label 44 was not found. Sometimes the label may have actually appeared, but was not found due to incorrect nesting of DO loops. This message appears at the beginning of the allocation map.
STORAGE OVERFLOW	One of the storage areas (local, blank COMMON, labeled COMMON) has overflowed 32K. This message appears following the map of the corresponding storage area.
UNDEFINED LABELS	Followed by lines of the form: 17 FIRST REF AT LINE 9 The statement number 17 was never defined, and there is at least one reference to it, on line 9. There may be overlap between this message and OPEN DO LOOPS. This message appears at the beginning of the allocation map.

COMPILER DIAGNOSTICS DURING GEN PHASE

Message	Error/ Warning	Comments
LITERAL POOL	E (or blank)	A literal pool has been created in the object code. If the message is not followed by "E*E*E", the pool has been necessitated by FORTRAN statements, and is guaranteed not to adversely affect any adjacent machine language instructions. If "E*E*E" appears in the message, the literal pool has been caused by the user's in-line ASSEMBLER language statements referencing out of range operands. The pool is preceded by a jump around, which may or may not work correctly, depending on where the pool appears. Examine the object listing to determine whether the pool is acceptable. If it is not acceptable, use an LPOOL directive to elicit the literal pool somewhere earlier in the in-line assembly language sequence. Note that if you supply your own LPOOL directives in your assembly language sequences, they will not generate a jump around them, nor will a "LITERAL POOL" diagnostic be output.
RANGE ERROR	E	An in-line assembly operand is out of range for the op-code it has been used with. Most of these will be caught by the RANGE error in Pass 1. This message appears when the range is not known until pass 2 (e.g. forward references). The error may refer to the operand of the line it appears on, or it may refer to the label, in which case there was a previous line that referenced this label and it is the previous line whose operand is out of range.

COMPILER ERRORS (ABORT CONDITION)

All abort-condition compiler Errors are of the form

FORT ER ptt

where p identifies the phase of the compiler that was operating:

p =	1	Scan
	2	Allocation
	3	Gen

and tt identifies the type of error:

tt =	11	Pointer overflow
	18	I/O error during overlay loading
	21	Working storage overflow
	28	Memory overflow during overlay loading
	31	Compiler error
	38	Illegal type code during overlay loading
	41	Compiler error
	51	Compiler error during collapse.

Except for 21 and 28, all of these result from hardware or software errors. If they occur in a reproducible way, they are probably software errors, which should be reported. 28 indicates that the compiler will not fit in memory. 21 indicates that the program cannot be compiled in the given amount of memory.

OS: LNK ERRORS

During the link process, various error conditions may occur. These errors may be grouped into three types:

Diagnostics

Diagnostics are messages output to the LO device as they are encountered. They indicate memory usage conflict of various forms, and are usually caused by scratchpad or main memory overflow, or an attempt to store data into a scratchpad location which is already occupied. These errors do not cause termination of OS: LNK, but may produce erroneous results during program execution. The specific error messages are described below.

"COMMON SIZE CONFLICT, IGNORED" (followed by program name, COMMON name, first defined size, subsequently defined size). A labeled COMMON definition has been encountered, whose size differs from that of a previous labeled COMMON definition of the same name. Since OS: LNK allocates memory according to the size in the first definition, no problem should occur as long as the first defined length is greater than the subsequent definition. However, if the subsequent definition is of greater size, a reference to the excess portion of the COMMON area may produce invalid results during execution. If this is the case, re-compilation is advisable using identical sizes for both definitions.

"MEMORY OVERFLOW, IGNORED" (followed by program name). Memory location :7FFF has been passed, and more memory is required. Allocation will continue at location zero. The program must either be shortened and then recompiled, or relocated to a lower memory location and then re-linked.

"SCRATCHPAD LITERAL OVERFLOW, IGNORED" (followed by program name). The literal pool address pointer has decremented to zero. Additional literals will not be assigned; references to any further unassigned literals will reference location zero. This error can often be corrected by re-linking with a different SR and/or SP option, or by re-compilation using the "NS" (no scratchpad) option.

"SCRATCHPAD PROGRAM/LITERAL OVERLAP, IGNORED" (followed by program name and scratchpad overlap address). The two pointers for scratchpad literals and scratchpad relocatable data have passed each other at the location shown. This is not necessarily a problem; however, the situation may sometimes by avoided by re-linking with a different SR and/or SP option, or by re-compilation using the "NS" (no scratchpad) option.

"SCRATCHPAD PROGRAM OVERFLOW, IGNORED" (followed by program name). Scratchpad relocatable data has passed the high scratchpad limit. OS:LNK will continue to store data into higher locations. This problem may be corrected by re-linking with a different SR and/or SP option, or by re-compiling using the "NS" (no scratchpad) option.

"SCRATCHPAD USAGE CONFLICT, IGNORED" (followed by program name and scratchpad location). Input data has been encountered that would be placed in a scratchpad location already occupied by a literal or other input data. If a literal occupies the cell, the input data will be lost. If the cell is occupied by input data, it will be overlayed by the new data. This problem may be corrected by re-linking with a different SR and/or SP option, or by re-compiling using the "NS" (no scratchpad) option.

Termination Errors

These are messages output to the CO and LO devices, indicating an error which prevents OS:LNK from completing the link operation. A memory map is printed at this time, and OS:LNK terminates. These messages are:

"BAD TYPE CODE". An invalid type code was recognized in the input data. The user should restart OS:LNK one time. If it fails again, re-compilation is probably required.

"LINK ERROR n" (where n may range from 1 to 5). This error indicates various types of logic failure within either the compiler (error No. 1-4) or OS:LNK itself (error No. 5). Computer Automation should be notified of such an occurence with as much information as possible regarding the program and procedure which elicited the error.

NOTE

> Currently, LINK ERROR 2 indicates that a variable in blank COMMON was given a value in a DATA statement. This is actually a source program error, but is not diagnosed by the compiler.

"TABLE FULL". An overflow condition has occurred in the link edit table. OS: LNK requires more memory for its working storage.

I/O Errors

I/O error messages are output to the CO device, and reflect an error status received from OS following an I/O operation.

"I/O ERR". An irrecoverable error status has been returned. OS: LNK will terminate; however, the user may re-execute OS: LNK to retry the I/O operation.

"INPUT CK". The BI or LI device is not ready for input. The user should ready the device, then continue with a /RESUME command.

FORTRAN RUN TIME ERROR MESSAGES

Form: (Routine Name), (message) ERROR at :xxxx

Message	Routine Name	Comments
ARGUMENT TOO LARGE	COS, DCOS, DSIN, DTAN, SIN, TAN	All significance to result lost. Zero returned.
ARGUMENT TOO LARGE	DEXP, EXP, IDINT, IFIX, INT, I**R, R**R, D**R, I**D, R**D, D**D	Result would overflow. Maximum value returned.
BOTH ARGUMENTS ZERO	ATAN2, DATAN2	Zero returned.
BOTH ARGUMENTS ZERO	CLOG	Real and imaginary parts both zero. Minus maximum value returned.
LINE dddd, COMPILATION	Program name	A statement has been reached that had a compilation source error. dddd is the source line number which will always have been marked with an error message except in the case of an undefined label reference.
DIVISION BY ZERO	Many	This condition is automatically tested for in a large number of routines, but is not expected to occur. If it does, let CAI know.
END OF FILE	ENDFILE, FORMATTED, UNFORMATTED	On a READ this means that an end-of-file mark has been encountered. On a WRITE or ENDFILE it means that end-of-tape or end-of-media has been reached (but the requested WRITE has been done). If an END= was specified, this message will not appear. Otherwise it will abort.
FORMAT INTEGER	FORMATTED	Number in FORMAT statement is greater than 32K. This should only happen on FORMATs stored in arrays, because normal FORMATs will be caught at compile time. Abort.
ILLEGAL FORMAT CHAR	FORMATTED	Syntax error in FORMAT statement. Only on FORMATs stored in arrays. Abort.

Message	Routine Name	Comments
ILLEGAL INPUT CHAR	FORMATTED	Illegal character in numeric input field. Abort.
ILLEGAL OPERATION	BACKSPACE, ENDFILE, FORMATTED, REWIND, UNFORMATTED	This operation cannot be performed on the requested device. Abort. Please refer to the following OS diagnostics for the various reasons this can occur: WRITE PROTECT, MULT WRITE ERROR, I/O BLOCKING OVERFLOW, and ILLEGAL OPEN.
ILLEGAL REPEAT COUNT	FORMATTED	FORMAT repeat count of zero. Only on FORMATs in arrays. Abort.
ILLEGAL UNIT	BACKSPACE, ENDFILE, FORMATTED, REWIND, UNFORMATTED	The unit number is not in the logical unit table. Abort. Under OS, this will be preceded by the message "yy NOT FOUND". Note that if yy is in the table, but is not assigned to a device, this will cause the UNASSIGNED error (under OS).
INCORRECT NUMBER OF ARGUMENTS	Many	A library routine has been called with the wrong number of arguments. Abort. FORTRAN compiled routines get the message NUMBER OF ARGUMENTS.
INTEGER INPUT OVERFLOW	FORMATTED	Input value exceeds 32K. Maximum value returned.
I/O	BACKSPACE, ENDFILE, FORMATTED, REWIND, UNFORMATTED	Hardware error. Under OS, this will usually be preceded by DATA ERROR or HDWR ERROR, identifying the physical device. Abort, unless ERR= exit specified.
NEGATIVE ARGUMENT	ALOG, ALOG10, DLOG, DLOG10, DSQRT, SQRT	Absolute value used instead.
NUMBER OF ARGUMENTS	-----	A FORTRAN compiled subprogram has been called with the wrong number of arguments. Abort.
NUMERIC MISMATCH	FORMATTED	A numeric value is associated with a logical format, or vice-versa. Abort.

Message	Routine Name	Comments
OUT OF RANGE	COMPUTED GO TO	The variable (v) is less than 1 or greater than n (the number of labels). Abort.
OVERFLOW	CABS, CCOS, CEXP, CSIN, CSQRT, DMOD, DTAN, DTANH, EXP, TAN, TANH	Maximum value returned.
OVERFLOW	I**I, R**I, D**I, C**I, I**R, R**R, D**R, I**D, R**D, D**D	Exponentiation overflow or underflow. Maximum value or zero returned, respectively.
PAREN NESTING	FORMATTED	More than eight levels of nesting. Only possible on FORMATs stored in arrays. Abort.
REAL INPUT OVERFLOW	FORMATTED	Floating point input value too large. Maximum value returned.
SINGULARITY	DTAN, TAN	Tangent of $(n+\frac{1}{2})\pi$ cannot be expressed Maximum value returned. Arguments near the singularity point may get the message OVERFLOW.
UNDEFINED SECONDARY REFERENCE	-----	The library is out of order or there is an error in the library or the generated code. Report this to CAI.

OS RUN TIME ERROR MESSAGES

Message	Return/ Suspend	Comments
xx DATA ERROR	Ret	Checksum or parity error in I/O transmission. xx is a physical device. This will be followed by an "I/O" error from FORTRAN, and the ERR= exit, if any.
zzzzzz DUPLICATE FILE	Sus	File name to be opened for WRITE already exists, possibly from your job, but more likely from a previous job. Choose a different name or delete the old file. zzzzzz is the file name.
xx HDWR ERROR	Ret	Hardware error. xx is the physical device. The record may or may not have been transmitted (e.g. a card moved from the hopper to the stacker); it may be possible to determine this by the status indicated on the device. Like DATA ERROR (above), this will be followed by a FORTRAN I/O error and possibly ERR= exit.
xx ILLEGAL OPEN	Sus	A device to be opened for input or binary is an output-only or ASCII-only device, respectively, or vice versa. xx is the physical device. This error will only occur on the first use of a unit number (when it is opened). Subsequent uses would get the FORTRAN ILLEGAL OPERATION error.
I/O BLOCKING OVERFLOW	Ret	Not enough unused memory for blocking buffers. Program is too large. This will be followed by a FORTRAN ILLEGAL OPERATION error.
xx MULT WRITE ERROR	Ret	Two unit numbers are assigned to files on the same tape unit, namely xx. (Disks can support multiple files open for writing, but tapes cannot.) If you need to do this, you must call a machine language subroutine to close the old file when you are through with it. This message will be followed by a FORTRAN ILLEGAL OPERATION error.

Message	Return/ Suspend	Comments
yy NOT FOUND	Ret	The unit number yy is not in the logical unit table. (Only units 1-6 are included in the standard delivered system.) This will be followed by a FORTRAN ILLEGAL UNIT error.
zzzzzz NOT FOUND	Sus	A file name to be opened for reading does not exist. zzzzzz is the file name.
xx NOT READY	Sus	The physical device xx is not ready.
yy UNASSIGNED	Sus	The unit number yy is in the logical unit table, but is not assigned to a physical device.
xx WRITE PROTECT	Ret	The device xx is either a write-protected tape or disk or else a disk that is full. This error can come out on any WRITE, not just when opened. It will be followed by a FORTRAN ILLEGAL OPERATION error. Note that files used during FORTRAN execution are not automatically deleted, and could accumulate until a disk was full. It is good practice, therefore, to delete files when you are through with them.

ERROR HALTS

Error halts are used to indicate a serious hardware or system software malfunction. When one of these occurs, Computer Automation should be notified. Each halt is coded with an identifying value in the low-order 8 bits of the instruction, and may be observed, via the Console, in the I-register.

FORTRAN Halts

I=:08DC	The floating point interpreter has encountered an unrecognized instruction during run-time. Report the condition to Computer Automation with all related program information (Contents of A, X, I, P registers, program listing, and, if possible, source input on cards or paper tape).
Console Data Register = :3CCO	An LSI-3/05 Uninstalled Memory Trap has occurred. This halt code was output by the Software Console routine. Locations :88 and :89 should be examined for the address and instruction, respectively, which caused the trap.

Console Data Register = :3CC2	An LSI-3/05 Unimplemented Instruction Trap has occurred. Using the Console panel, inspect locations :84 and :85 for the address and instruction, respectively, which caused the trap.

OS System Halts

I=:0801	The CI device does not respond. Correct the problem and reload OS.
I=:0802	The CO device does not respond. Correct the problem and reload OS.
I=:0803	The Real-time Clock does not respond. Correct the problem and reload OS.
I=:0804	Unrecoverable disk error. Notify Computer Automation.
I=:0805	Unrecoverable disk error. Notify Computer Automation.

RTX System Halts

None.