CROMEMCO

FORTRAN MANUAL

ADDENDA 2

CROMEMCO, INC.
280 Bernardo Avenue
Mountain View, CA 94040

November 1978

RANDOM ACCESS DISK FILES

CROMEMCO FORTRAN IV now provides random disk accessing, i.e., a record may be specified with a disk READ or WRITE.

The record number is specified by using the REC=n option in the READ or WRITE statement. For example:

```
K = 10

WRITE (6,20,REC=K) X,Y,Z

.
.
.
```

This program segment writes record 10 on LUN 6. If a previous record 10 exists, it is written over. If no record 10 exists, the file is extended to create one. Any attempt to read a non-existent record results in an I/O error.

The record length of any file accessed randomly is assumed to be 128 bytes (1 sector). Therefore, it is recommended that any file you wish to read in random order be created via FORTRAN random access statements.

Random access files may be created via FORTRAN programs by using either binary or formatted WRITE statements. If the WRITE statement does not cause enough data to be transferred to fill the record (128 bytes), then the end of the record is filled with zeroes (NULL characters).

Linking FORTRAN Programs

The following additions are to be made to Section 3 (Linking FORTRAN Programs) of the CROMEMCO FORTRAN IV User's Manual.

1. Page 17-18

Add to Section 3.1.2 LINK Switches

/U and /M also print the origin and end of the program and data area in addition to selected globals. Example:

```
DATA      100       200

PROGRAM  1000      2000
```

The program information is only printed if a /D has been done. Othewise, the program is stored in the data area.

Switch	Action
N	When a <filename>/N is specified, the program will be saved on disk under the selected name (with a default extension of .COM) when a /E or /G is done. A jump to the start of the program is inserted if needed so the program can run properly at 100H.
P and D	/P and /D allow the origin(s) to be set for the next program loaded. /P and /D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is /P:<address> or /D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix is hex. /O sets radix to octal; /H to hex.) LINK does a default /P:<link origin>+3 (i.e., 103H) to leave room for the jump to the start address. If no /D is given, programs load as usual, except the area base is settable. If a /D is given, all Data and Common areas are loaded starting at the data origin and the Program area at the program origin. Example:

```
*/P:200,FOO
DATA 20      300
*/R
*/P:200 /D:400,FOO
DATA 400     480
PROG 200     280
```

2. Page 19-2Ø

Add to Section 3.2 Format of LINK Compatible Object Files

Loader type 9 is now in use; it is external+offset. Type 9 has only an A field; there is no B field as previously documented. The value for type 9 will be added to the two bytes starting at the current location counter. This addition is done after a /E or /G is given, so unless undefineds remain, the effect is external+offset.

This type can also be used to add program and data relatives or almost any other combination of relocation types. The assembler, however, only handles the case with externals.

3. Pages 21-22

Add to Section 3.3 LINK Error Messages

?Out of Memory — has replaced ?Fatal Table Collision

?<file> Not Found — has replaced ?File Not Found. The name of the file not found is printed.

%Overlaying {Program} Area
{Data }

A /D or /P will cause already loaded data to be destroyed.

?Intersecting {Program} Area
{Data }

The program and data area intersect and an address or external chain entry is in this intersection. The final value cannot be converted to a current value since it is in the area intersection.

?Start Symbol -<name> -Undefined

After a /E: or /G: is given, the symbol specified is not defined.

Ørigin {Above} Loader Memory, Move Anyway (Y or N)?
{Below}

After a /E or /G was given, either the data or program area has an origin or top which lies outside loader memory (i.e., loader origin to top of memory). If a Y <cr> is given, LINK will exit. In either case, if a /N was given previously, the image will already have been

saved.

?Can't Save Qbject File — A disk error occurred when the file was being saved. This indicates that the user is out of disk space or has too many disk entries.

Page 15

Add Section 2.5 Program Break Information

LINK stores the first free location in a symbol called $MEMRY if that symbol has been defined by a program loaded. $MEMRY is set to the top of the data area +1.

NOTE

If /D is given and the data origin is less than the program origin, the user must be sure there is enough room to keep the program from being destroyed. This is particularly true with the disk driver for FORTRAN which uses $MEMRY to allocate disk buffers and FCB's.

Compiling FORTRAN Programs

The following additions are to be made to Section 1 (Compiling FORTRAN Programs) of the Cromemco FORTRAN User's Manual.

1. Page 9
Add to Section 2.1.2 FORTRAN IV Compilation Switches

Switch	Action
M	Specifies to the compiler that the generated code should be in a form which can be loaded into ROMs. When a /M is specified, the generated code will differ from normal in the following ways: 1. FORMATs will be placed in the program area with a "jump" around them. 2. Parameter blocks (for subprogram calls with more than three parameters) will be initialized at runtime rather than being initialized by the loader.

NOTE

If a FORTRAN program is intended for ROM, the programmer should be aware of the following ramifications:

1. DATA statements should not be used to initialize RAM. Such initialization is done by the loader, and will therefore not be present at execution. Variables and arrays may be initialized during execution via assignment statements, or by READing into them.

2. FORMATs should not be read into during execution.

3. If the standard library I/O routines are used, DISK files should not be OPENed on any LUNs other than 6,7,8,9,10. If other LUNs are needed for Disk I/O, $LUNTB should be recompiled with the appropriate address pointing to the Disk driver routine.

A library routine, $INIT, sets the stack pointer at the top of available memory (as indicated by the operating system; i.e., the top of the CDOS user area) before execution begins.

The calling convention is:

```
LD      BC,<return address>

JP      $INIT
```

If the generated code is intended for a machine not using CDOS, this routine should probably be rewritten.

FORTRAN IV Library Routines

The FORTRAN IV library contains a number of subroutines that may be referenced by the user from FORTRAN or Assembly language programs. In the following descriptions, $AC refers to the floating accumulator; where $AC is the address of the exponent. $DAC refers to the DOUBLE PRECISION accumulator; where $DAC is the address of the low byte of the mantissa. $DAC+7 is the address of the DOUBLE PRECISION exponent.

All arithmetic routines (addition, subtraction, multiplication, division, exponentiation) adhere to the following calling conventions.

1. Argument 1 is passed in the registers:
   Integer in [HL]
   Real in $AC
   Double in $DAC

2. Argument 2 is passed either in registers, or in memory depending upon the type:

   a) Integers are passed in [HL], or [DE] if [HL] contains Argument 1.

   b) Real and Double Precision values are passed in memory pointed by [HL]. ([HL] points to the low byte of the mantissa.)

The following arithmetic routines are contained in the FORTRAN Library (FORLIB.REL) and may be called by user programs:

Function	Name	Argument 1 Type	Argument 2 Type
Addition	$AA	Real	Integer
	$AB	Real	Real
	$AQ	Double	Integer
	$AR	Double	Real
	$AU	Double	Double
Division	$D9	Integer	Integer
	$DA	Real	Integer
	$DB	Real	Real
	$DQ	Double	Integer
	$DR	Double	Real
	$DU	Double	Double
Exponentiation	$E9	Integer	Integer
	$EA	Real	Integer
	$EB	Real	Real
	$EQ	Double	Integer
	$ER	Double	Real
	$EU	Double	Double
Multiplication	$M9	Integer	Integer
	$MA	Real	Integer
	$MB	Real	Real
	$MQ	Double	Integer
	$MR	Double	Real
	$MU	Double	Double
Subtraction	$SA	Real	Integer
	$SB	Real	Real
	$SQ	Double	Integer
	$SR	Double	Real
	$SU	Double	Double

Additional Library routines are provided for converting between value types. The argument is always passed to and returned by these conversion routines in the appropriate register:

Logical in [A]

Integer in [HL]

Real in $AC

Double in $DAC

Name	Function
$CA	Integer to Real
$CC	Integer to Double
$CH	Real to Integer
$CJ	Real to Logical
$CK	Real to Double
$CX	Double to Integer
$CY	Double to Real
$CZ	Double to Logical