MICROSOFT FORTRAN-80

user's manual

Microsoft FORTRAN-80 User's Manual

CONTENTS

ADDENDA TO: Microsoft FORTRAN-80 User's Manual.
Section 3.3.1
CALL FCHAIN

3.3.1 CALL FCHAIN

Programs may be loaded and executed (CHAINed) by a Fortran program through the CALL FCHAIN facility. (See the "Microsoft FORTRAN-80 Reference Manual", Section 9.13). Operating system dependent syntaxes are:

CP/M:

```
CALL FCHAIN ('Filename ',Drive)
```

WHERE: Filename is a valid CP/M filename. Drive is the number of the disk on which the file exists. See Drive and Filename discussion in 3.3 above.

Examples:

CALL FCHAIN ('PROG2 COM',0)	PROG2.COM is loaded and executed from the logged disk.
CALL FCHAIN ('PROG2 OLD ',2)	PROG2.OLD is loaded and executed from Drive "B".

ISIS-II:

Refer to the ISIS Filename rules in 3.3 above.

Examples:

CALL FCHAIN (':F1:PROG2 ')	Load and execute PROG2 from Drive 1.
CALL FCHAIN (':F0:PROG 2 ')	Results in a **IO** Error. (Filename contained imbedded blank).

SECTION 1

Compiling FORTRAN Programs

1.1 FORTRAN-80 Command Scanner

To tell the FORTRAN compiler what to compile and with which options, it is necessary to input a "command string," which is read by the FORTRAN-80 command scanner.

1.1.1 Format of Commands

To run FORTRAN-80, type F80 followed by a carriage return. FORTRAN-80 will return the prompt "*" (with the DTC operating system, the prompt is ">"), indicating it is ready to accept commands. The general format of a FORTRAN-80 command string is:

```
objprog-dev:filename.ext,list-dev:filename.ext=
    source-dev:filename.ext
```

objprog-dev:
The device on which the object program is to be written.

list-dev:
The device on which the program listing is written.

source-dev:
The device from which the source-program input to FORTRAN-80 is obtained. If a device name is omitted, it defaults to the currently selected drive.

filename.ext
The filename and filename extension of the object program file, the listing file, and the source file. Filename extensions may be omitted. See Section 4 of the Microsoft Utility Software Manual for the default extension supplied by your operating system.

Either the object file or the listing file or both may be omitted. If neither a listing file nor an object file is desired, place only a comma to the left of the equal sign. If the names of the object file and the listing file are omitted, the default is the name of the source file.

Examples:

`*=TEST`	Compile the program TEST.FOR and place the object in TEST.REL
`*,TTY:=TEST`	Compile the program TEST.FOR and list program on the terminal. No object is generated.
`*TESTOBJ=TEST.FOR`	Compile the program TEST.FOR and put object in TESTOBJ.REL
`*TEST,TEST=TEST`	Compile TEST.FOR, put object in TEST.REL and listing in TEST.LST
`*,=TEST.FOR`	Compile TEST.FOR but produce no object or listing file. Useful for checking for errors.

1.1.2 FORTRAN-80 Compilation Switches

A number of different switches may be given in the command string that will affect the format of the listing file. Each switch should be preceded by a slash (/):

Switch	Action
O	Print all listing addresses, etc. in octal. (Default for ALTAIR DOS)
H	Print all listing addresses, etc. in hexadecimal. (Default for non-ALTAIR versions)
N	Do not list generated code.
R	Force generation of an object file.
L	Force generation of a listing file.
P	Each /P allocates an extra 100 bytes of stack space for use during compilation. Use /P if stack overflow errors occur during compilation. Otherwise not needed.

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 "JMP" around them. 2. Parameter blocks (for subprogram calls with more than 3 parameters) will be initialized at runtime, rather than being initialized by the loader.

Examples:

*,TTY:=MYPROG/N	Compile file MYPROG.FOR and list program on terminal but without generated code.
*=TEST/L	Compile TEST.FOR with object file TEST.REL and listing file TEST.LST
*=BIGGONE/P/P	Compile file BIGGONE.FOR and produce object file BIGGONE.REL. Compiler is allocated 200 extra bytes of stack space.

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 addresses 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) before execution begins.

The calling convention is:

```
LXI     B,<return address>
JMP     $INIT
```

If the generated code is intended for some other machine, this routine should probably be rewritten. The source of the standard initialize routine is provided on the disk as "INIT.MAC". Only the portion of this routine which sets up the stack pointer should ever be modified by the user. The FORTRAN library already contains the standard initialize routine.

1.2 Sample Compilation

```
A>F80

*EXAMPL,TTY:=EXAMPL

FORTRAN-80 Ver. 3.2 Copyright 1978 (C) By Microsoft - Bytes: 4524
00100                PROGRAM EXAMPLE
00200                INTEGER X
00300                I = 2**8 + 2**9 + 2**10
00400                DO 1 J=1,5
*****   0000'        LXI     H,0700
*****   0003'        SHLD    I
00500   C            CIRCULAR SHIFT I LEFT 3 BITS -- RESULT IN X
00600                CALL CSL3(I,X)
*****   0006'        LXI     H,0001
*****   0009'        SHLD    J
00700                WRITE(3,10) I,X
*****   000C'        LXI     D,X
*****   000F'        LXI     H,I
*****   0012'        CALL    CSL3
*****   0015'        LXI     D,10L
*****   0018'        LXI     H,[      03       00]
*****   001B'        CALL    $W2
00800   1            I=X
*****   001E'        LXI     B,X
*****   0021'        LXI     D,I
*****   0024'        LXI     H,[      01       00]
*****   0027'        MVI     A,03
*****   0029'        CALL    $IO
*****   002C'        CALL    $ND
00900   10           FORMAT(2I15)
*****   002F'        LHLD    X
*****   0032'        SHLD    I
*****   0035'        LHLD    J
*****   0038'        INX     H
*****   0039'        MVI     A,05
*****   003B'        SUB     L
*****   003C'        MVI     A,00
*****   003E'        SBB     H
*****   003F'        JP      0009'
01000                END
*****   0042'        CALL    $EX
*****   0045'        0100
*****   0047'        0300

Program Unit Length=0049 (73) Bytes
Data Area Length=000D (13) Bytes

Subroutines Referenced:

$IO                         CSL3                         $W2
$ND                         $EX
```

```
Variables:

X        0001"            I        0003"            J        0005"

LABELS:

1L       002F'            10L      0007"

*^C
A>
```

See Section 1.8 of the Microsoft Utility Software Manual for a listing of the MACRO-80 subroutine CSL3.

1.3 FORTRAN Compiler Error Messages

The FORTRAN-80 Compiler detects two kinds of errors: Warnings and Fatal Errors. When a Warning is issued, compilation continues with the next item on the source line. When a Fatal Error is found, the compiler ignores the rest of the logical line, including any continuation lines. Warning messages are preceded by percent signs (%), and Fatal Errors by question marks (?). The editor line number, if any, or the physical line number is printed next. It is followed by the error code or error message.

Example:

```
?Line 25: Mismatched Parentheses

%Line 16: Missing Integer Variable
```

When either type of error occurs, the program should be changed so that it compiles without errors. No guarantee is made that a program that compiles with errors will execute sensibly.

Fatal Errors:

Error Number	Message
100	Illegal Statement Number
101	Statement Unrecognizable or Misspelled
102	Illegal Statement Completion
103	Illegal DO Nesting
104	Illegal Data Constant
105	Missing Name
106	Illegal Procedure Name
107	Invalid DATA Constant or Repeat Factor
108	Incorrect Number of DATA Constants
109	Incorrect Integer Constant
110	Invalid Statement Number
111	Not a Variable Name
112	Illegal Logical Form Operator
113	Data Pool Overflow
114	Literal String Too Large
115	Invalid Data List Element in I/O
116	Unbalanced DO Nest
117	Identifier Too Long
118	Illegal Operator
119	Mismatched Parenthesis
120	Consecutive Operators
121	Improper Subscript Syntax
122	Illegal Integer Quantity
123	Illegal Hollerith Construction
124	Backwards DO reference
125	Illegal Statement Function Name

126	Illegal Character for Syntax
127	Statement Out of Sequence
128	Missing Integer Quantity
129	Invalid Logical Operator
130	Illegal Item Following INTEGER or REAL or LOGICAL
131	Premature End Of File on Input Device
132	Illegal Mixed Mode Operation
133	Function Call with No Parameters
134	Stack Overflow
135	Illegal Statement Following Logical IF

Warnings:

0	Duplicate Statement Label
1	Illegal DO Termination
2	Block Name = Procedure Name
3	Array Name Misuse
4	COMMON Name Usage
5	Wrong Number of Subscripts
6	Array Multiply EQUIVALENCEd within a Group
7	Multiple EQUIVALENCE of COMMON
8	COMMON Base Lowered
9	Non-COMMON Variable in BLOCK DATA
10	Empty List for Unformatted WRITE
11	Non-Integer Expression
12	Operand Mode Not Compatible with Operator
13	Mixing of Operand Modes Not Allowed
14	Missing Integer Variable
15	Missing Statement Number on FORMAT
16	Zero Repeat Factor
17	Zero Format Value
18	Format Nest Too Deep
19	Statement Number Not FORMAT Associated
20	Invalid Statement Number Usage
21	No Path to this Statement
22	Missing Do Termination
23	Code Output in BLOCK DATA
24	Undefined Labels Have Occurred
25	RETURN in a Main Program
26	STATUS Error on READ
27	Invalid Operand Usage
28	Function with no Parameter
29	Hex Constant Overflow
30	Division by Zero
32	Array Name Expected
33	Illegal Argument to ENCODE/DECODE

SECTION 2

FORTRAN Runtime Error Messages

Code	Meaning
Warning Errors:	
IB	Input Buffer Limit Exceeded
TL	Too Many Left Parentheses in FORMAT
OB	Output Buffer Limit Exceeded
DE	Decimal Exponent Overflow (Number in input stream had an exponent larger than 99)
IS	Integer Size Too Large
BE	Binary Exponent Overflow
IN	Input Record Too Long
OV	Arithmetic Overflow
CN	Conversion Overflow on REAL to INTEGER Conversion
SN	Argument to SIN Too Large
A2	Both Arguments of ATAN2 are 0
IO	Illegal I/O Operation
BI	Buffer Size Exceeded During Binary I/O
RC	Negative Repeat Count in FORMAT
Fatal Errors:	
ID	Illegal FORMAT Descriptor
F0	FORMAT Field Width is Zero
MP	Missing Period in FORMAT
FW	FORMAT Field Width is Too Small
IT	I/O Transmission Error
ML	Missing Left Parenthesis in FORMAT
DZ	Division by Zero, REAL or INTEGER
LG	Illegal Argument to LOG Function (Negative or Zero)
SQ	Illegal Argument to SQRT Function (Negative)
DT	Data Type Does Not Agree With FORMAT Specification
EF	EOF Encountered on READ

Runtime errors are surrounded by asterisks as follows:

```
**FW**
```

Fatal errors cause execution to cease (control is returned to the operating system). Execution continues after a warnikg error. However, after 20 warnings, execution ceases.

SECTION 3

FORTRAN-80 Disk Files

3.1 Random Disk I/O

In the current release of FORTRAN-80, only the CP/M and ISIS-II versions provide random disk I/O capability.

3.2 Default Disk Filenames

A disk file (random or sequential) that is OPENed by a READ or WRITE statement has a default name that depends upon the LUN and the operating system:

CP/M and ISIS II:	FORT06.DAT, FORT07.DAT,..., FORT10.DAT
ALTAIR:	FOR06DAT, FOR07DAT, ..., FOR10DAT
DTC:	FOR06D, FOR07D,..., FOR10D

In each case, the LUN is incorporated into the default file name.

3.3 CALL OPEN

Instead of using READ or WRITE, a disk file may be OPENed using the OPEN subroutine (see the FORTRAN-80 Reference Manual, Section 8.3.2). The format of an OPEN call under CP/M, Altair and DTC is:

```
CALL OPEN (LUN, Filename, Drive)
```

where:

LUN = a Logical Unit Number to be associated with the file (must be an Integer constant or Integer variable with a value between 1 and 10).

Filename = an ASCII name which the operating system will associate with the file. The Filename should be a Hollerith or Literal constant, or a variable or array name, where the variable or array contains the ASCII name. The Filename should be blank-filled to exactly the number of characters allowed by the operating system:

CP/M:	11 characters
ALTAIR:	8 characters
DTC:	6 characters

Drive = the number of the disk drive on which the file exists or will exists (must be an Integer constant or Integer variable within the range allowed by the operating system). If the Drive specified is 0, the currently selected drive is assumed; 1 is drive 0 (or A), 2 is drive 1 (or B), etc.

The form of an OPEN call under ISIS-II is:

```
CALL OPEN (LUN, Filename)
```

where:

LUN = a Logical Unit Number to be associated with the file (must be an Integer constant or Integer variable with a value between 1 and 10).

Filename = an ASCII name which the operating system will associate with the file. The Filename should be a Hollerith or Literal constant, or a variable or array name where the variable or array contains the ASCII name. The Filename should be in the form normally required by ISIS-II, i.e., a device name surrounded by colons, followed by a name of up to 6 characters, a period, an extension of up to 3 characters, and a space (or other non-alphanumeric character). <u>The Filename must be terminated by a non-alphanumeric character</u>.

The following are examples of valid OPEN calls under ISIS-II:

```
CALL OPEN (6, ':F1:FOO.DAT ')

CALL OPEN (1, ':F5:TESTFF.TMP ')

CALL OPEN (4, ':F3:A.B ')
```

3.4 Record Length

The record length of any file accessed randomly under CP/M or ISIS-II is assumed to be 128 bytes (1 sector). Therefore, it is recommended that any file you wish to read randomly be created via FORTRAN (or Microsoft BASIC) random access statements. Random access files created this way (using either binary or formatted WRITE statements) always have 128-byte records. If the WRITE statement does not transfer enough data to fill the record to 128 bytes, then the end of the record is filled with zeros (NULL characters).