CONTROL DATA
CORPORATION

BULLETIN

APPLICATIONS DEVELOPMENT

6400/6600 FORTRAN CONVERSION GUIDE

6400/6600 FORTRAN CONVERSION GUIDE

CONTENTS

EXTENSIONS TO VERSION 1.1

6400/6600 FORTRAN Version 2.0 includes the ASA FORTRAN language and the extensions allowed in FORTRAN Version 1.1. The code produced is essentially the same as that produced under FORTRAN Version 1.1. The compiler and execution time routines operate under 6400/6600 SCOPE Version 2.0. Certain extensions and restrictions are provided for the additional features and requirements of SCOPE 2.0 and for ASA compatibility.

This document contains the information necessary to convert programs which run under Version 1.1 to Version 2.0.

The user may refer to Version 1.1 through the Chippewa FORTRAN Reference Manual Publication No. 60132700 and the 6000 Series System Bulletins No. 1 and No. 2.

1 SOURCE LANGUAGE EXTENSION

Source Language input 6400/6600 FORTRAN Version 2.0 is the same as allowed in Version 1.1 with the extensions for ASA FORTRAN. During compilation the user may set a switch either to select the ASA compatibility features of the execution time routines or to retain the present Chippewa FORTRAN method of format/list interaction and output format. The switch is initialized at the same time as the main program and therefore is active only when a program is being compiled.

1.1 Execution Extensions for ASA FORTRAN

In Version 2.0, unlimited group repeat is added according to the ASA specification. An innermost parenthetical group with no group repeat count specified in a FORMAT statement is assigned a group repeat count of one. If the last outer right parenthesis of the format specification is encountered before the I/O list is exhausted, control passes to the group repeat specification terminated by the last preceding right parenthesis. If there is no preceding right parenthesis, control passes to the first left parenthesis of the format statement.

1.2 Scale Factor For ASA Specifications

In Version 2.0 a scale factor is allowed for F,E,G, and D on input. This scale has no effect if there is an exponent in the external field. G output makes use of the scale factor only if E conversion is necessary to convert the data.

1.3 Gw.d Specifications

Gw.d format specification is allowed in Version 2.0 on both input and output.

Input

The input processing for this specification is the same as for the F conversion.

Output

The real data is represented by F conversion unless the magnitude of the data exceeds the range that permits effective use of F conversion; in which case, E conversion represents the external output. Therefore, the effect of the scale factor is not implemented unless the magnitude of the data requires E conversion.

Examples:

```
          Gw.d Output

          PRINT 101, XYZ          XYZ contains 77.132
101       FORMAT (G10.3)

                                  Result:  bbbb77.132

          PRINT 101, XYZ          XYZ contains 1214635.1
101       FORMAT (G10.3)          Result:  b0.121Eb07
```

1.4 Gw.d Scaling

Input

Gw.d scaling on input is the same as Fw.d scaling on input.

Output

The scale factor is effective only when the range of the data to be converted is outside the range of F conversion.

1.5 nP Scale Factor

The D, E, F, and G conversion may be preceded by a scale factor which is: External number = Internal number x $10^{\text{scale factor}}$ The scale factor applies to Fw.d and Gw.d on both input and output and to Ew.d and Dw.d on output only. A scaled specification is written as shown below; n is a signed integer constant.

```
nPDw.d          nPGw.d

nPEw.d          nP

nPFw.d
```

The scale factor is assumed to be zero if no other value has been given. Once a value has been given, however, it holds for all succeeding D,E,F, and G specifications within the same FORMAT specification. To nullify the scale factor, a zero scale factor, OP, must precede a D,E,F, or G specification. Scale factors for D,E,F, or G output specifications must be in the range $-8 \leq n \leq 8$.

Scale factors on D or E input specifications are ignored.

When the scaling specification nP appears independent of a D,E,F, or G specification, it holds for all subsequent D,E,F, or G specifications within the same FORMAT statement unless changed by another nP.

The specification (3P, 3I9, F10.2) is the same as the specification (3I9, 3PF10.2).

1.6 Compiler Extensions for ASA FORTRAN

1.6.1 EQUIVALENCE

The EQUIVALENCE statement allows multiply subscripted variables.

EQUIVALENCE (A,B,...),...

The above statement is an equivalence group of two or more simple or subscripted variable names.

1.6.2 Program Header Cards

If the first card of a program is not a PROGRAM header card, a main program with a blank name and files of INPUT, OUTPUT, is assumed, and a diagnostic is provided. If more files than INPUT, OUTPUT are necessary, the program card is required.

The program and subroutine header cards, MACHINE and ASCENTF, are not used in Version 2.0. The following header cards may be used in Version 2.0.

FORTRAN VI PROGRAM name $(f_1,\ldots,f_n)$

FORTRAN VI SUBROUTINE name $(p_1,\ldots,p_n)$

FORTRAN VI FUNCTION name $(p_1,\ldots,p_n)$

FORTRAN VI type FUNCTION name $(p_1,\ldots,p_n)$

The FORTRAN VI programs are compiled in FORTRAN IV mode except for DO loops and END statements. Under FORTRAN VI, the 3600 FORTRAN DO procedure is implemented. For example, a DO loop will not be executed if the initial value is greater than the terminal value. Also, under FORTRAN VI the appearance of an END statement in a function or subroutine will act as a RETURN statement.

1.6.3 Common Block Length

In Version 2.0, the length of a common block other than blank common may not be increased by subprograms using the block unless that subprogram is loaded first by the SCOPE loader.

1.7 Other Extensions

1.7.1 ENTRY

In Version 2.0, the ENTRY statement provides alternate entry points to a subprogram.

```
ENTRY name
```

The alphanumeric identifier, name, may appear within the subprogram only in the ENTRY statement. Each entry identifier must appear in a separate ENTRY statement. Format parameters appearing with the FUNCTION or SUBROUTINE statement do not appear with the ENTRY statement. ENTRY may appear anywhere in the subprogram except within a DO; ENTRY statements cannot be labeled. The first executable statement following ENTRY becomes an alternate entry point to the subprogram.

Reference to the entry name by the calling program is made in the same way as references to the FUNCTION or SUBROUTINE in which the ENTRY is contained. The name may appear in an EXTERNAL statement and, if a function entry name, in a type statement.

The ENTRY name must agree with the function name. A type may not be explicitly assigned to a name in the defining program; it assumes the same type as the name in the FUNCTION statement.

Examples:

```
      FUNCTION JOE (X,Y)
10    JOE=X+Y
      RETURN
      ENTRY JAM
      IF (X.GT.Y) 10,20
20    JOE=X-Y
      RETURN
      END
```

This could be called from the main program as follows:

```
      .
      .
      .
      Z=A+B-JOE (3.*P,Q-1)

      .
      .
      .
      R=S+JAM(Q,2.*P)
```

1.7.2 Overlays and Segments

Overlays

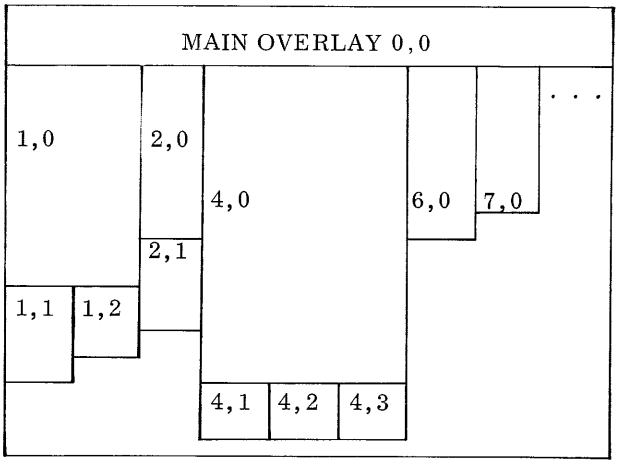
Overlay processing allows programs that exceed available storage to be divided into independent parts which may be called and executed as needed. Each overlay is numbered with a pair of relative priorities (I,J) called levels. Each level is in the range $0\text{-}77_8$. The first number, I, denotes the primary level; the second number, J, denotes the secondary level. An overlay with a non-zero secondary level, such as 1,1, is called a secondary overlay. It is associated with and subordinate to the overlay which has the same primary level and a zero secondary level (1,0), called the primary overlay. The initial or main overlay which remains in memory has levels 0,0. The levels determine the order in which overlays are loaded. A secondary overlay may be loaded only from the main overlay or from its associated primary overlay.

For any given program execution, all overlay identifiers must be unique. Overlay level numbers 0,1-0,n are illegal.

Primary overlays are all loaded at the same point immediately following the main overlay (0,0). Secondary overlays are loaded immediately following the primary overlay. The loading of any primary overlay destroys any other primary overlay. For this reason, no primary overlay may load other primary overlays.

Secondary overlays must be loaded only by the associated primary overlay, or by the main overlay. Thus two levels of overlay, one primary and one secondary, are available to the programmer at any one time.

Example:



Overlays (1,1) and (1,2) are secondary to overlay (1,0)

Overlay (2,1) is secondary to overlay (2,0)

Overlay 1,1 may be called by 1,0 or 0,0. Loading 1,2 destroys 1,1.

Overlays (1,0), (2,0), (4,0)... may be called only from the main overlay (0,0). Loading 2,0 destroys 1,0 and its associated 1,1 or 1,2.

Overlays are called by:

CALL OVERLAY (fn, il, i2, p)

OVERLAY FORTRAN execution time subroutine which translates the FORTRAN call into a call to the loader.

fn variable name of location which contains the name of the file (left justified display code) on which the overlay is stored.

il primary level of the overlay.

i2 secondary level of the overlay.

p recall parameter. If p equals 6HRECALL, the overlay is not reloaded if it is in memory.

Segments

Segmentation, the loading and delinking of a group of programs as a unit, allows the user to determine which programs are to be in memory during execution of a job.

A segment is defined by a SEGMENT card or user-parameter list and may contain both program names and SECTION names. A section is included in the loader scheme to reduce the number of program names in segment calls.

Segments, like overlays, are loaded at levels; however, a segment has only one level which may range from $0\text{-}77_8$. Level zero is reserved for the initial or main segment which remains in memory during execution. Segments may be loaded at any level. The number of segments in memory at any one time is limited only by available memory.

When the segment is loaded, external references are linked to entry points in previously loaded segments at a lower level.

Similarly, entry points in the segment are linked to unsatisfied external references in previously loaded segments. Unsatisfied external references may remain in the segment. Subsequent segment loading may include entry points to satisfy them; or the user may specify that the unsatisfied external references be filled from the system library. If a segment to be loaded has a specified level less than or equal to the level of the last loaded segment, all segments at levels down to and including the request level are delinked. Therefore, the last loaded segment always has the highest level. Delinking a segment destroys the linkage of entry points to external references in lower levels and they are unsatisfied once again.

References to labeled common blocks follow the same rules. Maximum blank common length is established in the first segment which declares blank common.

When delinking is complete the new segment is loaded. A user may effectively replace a previously loaded subprogram by naming it in another segment at a higher level. Thereafter all external references in higher levels would be linked to the new version but the linkages of the lower version remains until the segment containing the entry point is removed.

Example:

> The SINE routine is loaded in a segment at level 1. To try an experimental version of SINE, the user loads a segment containing the new SINE at level 4. Segments loaded at level 5 or higher will now be linked to SINE at level 4 until a new level 4 or a new SINE is loaded, but the old SINE remains at level 1 and its previously established linkages are not destroyed.

Common blocks may be loaded with any segment, but the maximum area for a given labeled common block must be declared in the first segment which refers to that common.

Segments may be called by:

> CALL SEGMENT (fn, e, a, lib, m)
>
> fn variable name of location which contains the file name (left justified display code) from which the segment is to be loaded.

e level of the segment load.

a simple or subscripted variable name of array containing a list of segments, sections and/or subprograms to be loaded with this call. This list must have the name in the upper 7 character positions with the lower 3 character positions zero and the list must be terminated by a zero entry. A first entry of zero signals a segment load of all subprograms remaining on the file fn.

lib if zero, or blank unsatisfied externals are to be satisfied from the system library.

m if zero or blank a map of the segment load is not produced.

lib and m need not be specified.

1.7.3 Compiler Output

The Version 2.0 compiler produces a binary object program and a source listing with diagnostics for any errors encountered. Upon option, it also produces a binary card deck of the object program, a source listing, even if no compilation errors have been detected, and an octal listing of the object program. Selection of the octal listing forces the listing of the source input.

These options are the same as those available under FORTRAN Version 1.1. However, the format of the binary object program when punched or written on another medium is that required by SCOPE. The diagnostics are expanded so that each two letter diagnostic appearing within the source listing results in the printing of a full line diagnostic after each subprogram has been compiled.

Compilation and Execution

The FORTRAN compiler is called by the control card:

RUN (cm,fl,bl,if,of,rf,lc,as,cs)

cm Compiler mode option: (if omitted, assumed G; if recognized, assumed S).

G compile and execute, no source list unless explicit LIST cards appear in the deck.

S compile with source list, no execute.

P compile with source list and punch deck on file PUNCHB, no execute.

L compile with source and object list, no execute.

M compile with source and object list, produce a punch deck on file PUNCHB, no execute.

fl object program field length (octal); if omitted, it is set equal to the field length at compilation time.

bl object program I/O buffer lengths (octal); if omitted, 2011B assumed

if file name for compiler input; if omitted INPUT assumed

of file name for compiler output; if omitted, OUTPUT assumed

rf file name on which the binary information is always written; if omitted, assumed to be LGO

lc line-limit (octal) on the OUTPUT file of an object program. If omitted, 10000 assumed. If line count exceeds specified line limit, the job is terminated.

as ASA switch. If non-zero or non-blank, produces the ASA I/O list/format interaction at execution time. It has no effect on the compilation method.

cs cross reference switch. If non-zero, a cross reference listing is produced.

Compiler output, except in the G mode, includes a reproduction of the source program, a variable map, and indications of fatal and non-fatal errors detected during compilation. If the G mode is selected, all output is suppressed unless fatal errors are detected; in which case the output is the same as for the S mode of compilation. If the L mode is selected, the output includes an octal list of the compiler instructions.

A copy of the compiled programs is always left in disk storage as a binary record on a file with the name LGO or the name specified as the rf parameter in the compiler call. The compiled program may be called and executed repeatedly by using the name of the load-and-go file. In the output file at the end of the compilation of each subprogram, the compiler indicates the amount of unused compilation space.

LIST and NOLIST control cards, which start after column 6, are free field and appear between subprograms. When the LIST card is detected, the source cards of the following programs are listed. If the compiler mode is L, the object code is also listed. When the NOLIST card is detected, no more listing takes place until a LIST card is detected. However, if fatal errors are detected within the range of the NOLIST, LIST pseudo ops, all subprograms in this range are listed. Since each LIST card causes time-consuming repositioning of the output file, excessive LIST-NOLIST sequences should be avoided.

When the FORTRAN compiler detects an OVERLAY or SEGMENT card between subprograms, it transfers these control cards to the load-and-go file and to the output file. If the P or M option is selected, the control cards are transferred to the PUNCHB file.

The following control cards which must start after column 6, are transferred to the load-and-go file.

OVERLAY (...

SECTION,...

SEGZGRO,

SEGMENT,

1.7.4 ASCENT Assembler

The FORTRAN compiler, RUN, loads the ASCENT assembler and transfers control to it upon the detection of a header card that has ASCENT or ASPER starting in column 11. When a header card is found that the ASCENT assembler does not recognize, control returns to the RUN compiler which continues processing. ASCENT produces the type of output specified by the compiler mode except in the case of an ASPER header card. When ASCENT detects an ASPER header card, binary information is written not on the load-and-go file but on the PUNCHB file providing the P parameter was selected.

II FORTRAN ERROR PRINTOUTS

During a FORTRAN compilation, two or three character error printouts follow the incorrect statements; other printouts may follow the end statement indicating types of errors in the program. These short printouts are expanded to a more descriptive full line diagnostic after the subprogram has been compiled.

A full line diagnostic contains a number to identify the statement in which the error was detected. The third letter of the short diagnostic is F as if the error is fatal.

Fatal errors force a listing of the subprogram with diagnostics and also inhibit the production of a relocatable record of the subprogram.

Non-fatal errors do not force a listing and appear only if some type of listing has been specified or if fatal errors have been detected.

The following diagnostics should be deleted from the two character error indicators as they are all types of diagnostics that will be provided by the SCOPE LOADER.

BI	MD	NC
BO	MF	SL
CO	ML	UE
FT	MR	UN
LR	MT	US
MC	MU	XM

III INFORMATIVE DIAGNOSTICS

In Version 2.0, additional diagnostics provide more informative error messages. They are non-fatal and the statement is processed.

Informative diagnostics occur if:

The type of variable is specified more than once.

The DIMENSION statement does not precede the first executable statement.

An array is dimensioned in more than one declaration statement.

The EQUIVALENCE statement appears after the first executable statement.

The statement function does not precede the first executable statement.

The following diagnostic has been deleted:

BC Syntax error in Boolean constant

The following diagnostics have been added:

AE Arithmetic statement function calls itself

Arithmetic statement function being compiled references itself.

AF Arithmetic statement function error

This statement has a number or it appears after the first executable statement.

DI DO terminator previously defined.

DL Declarative appears after first executable statement.

DN Illegal DO terminator

This statement cannot be used as a DO terminator.

DR Data range error

Attempt to store data out of range.

EX Syntax error in exponent

An attempt has been made to raise a value to a negative constant power.

ID Improperly nested DO loops.

IN Illegal Function name

Name of a function reference starts with a number.

IT Illegal transfer to DO terminator

A transfer to a DO terminator is not allowed if the terminator has already been defined and no transfer to it appeared before it was defined.

PT Syntax error in an entry statement

This statement is labeled, has more than one name, is in a DO loop, or the name begins with a number.

TT Variable given conflicting types

A variable has appeared in more than one type statement.

iV PROGRAM-SUBPROGRAM FORMAT

All programs are loaded beginning at RA+100_8; the first 77_8 locations contain file and loader information. A maximum of 50 files may be declared for any one program and the file names along with associated buffer addresses begin at RA+2. At execution time an object time routine, Q8NTRY, transfers the file information to RA+2+n, where n is the number of declared files. The I/O buffers are reserved as a portion of the main program. Q8NTRY also initializes the buffer parameters during execution.

The first word of a main program contains the name of the program in left justified display code and a parameter count greater than 77_8 in the right-most position. Since no more than 74_8 parameters may be passed to a subprogram, a count of 77_8 terminates trace back information.

The second word of a main program is the entry point which contains instructions for presetting the parameters for Q8NTRY. Q8NTRY is a subroutine which initializes the I/O buffer parameters. When an overlay is entered via a return jump to the entry point, the instructions are destroyed but the program is not affected because Q8NTRY performs no function after the first entry.

The addresses of the first six parameters to a subprogram are passed via B registers 1-6. One word is reserved for each parameter greater than six so that the address of the parameter is actually passed through this reserved word. Immediately following the reserved words is a location containing the name of the subprogram in left justified display code and parameter count in the lower six bits. Next is the entry/exit line for the subprogram. Therefore, a subprogram may have as few as two reserved words if the parameter count is less than six. Otherwise, there will be a reserved word for each parameter over six plus the name and entry words.

Traceback is accomplished by a return jump to a subroutine followed in the next line by 07XXADDRES

XX	number of parameters passed to the called routine
ADDRES	address of name of calling routine

The word containing the name of the calling routine must be followed by the entry/exit line. This specialized traceback is illustrated in the last two lines of the calling sequence to PEN in the example by:

```
RJ   PEN
0710L00002
```

Examples:

```
PROGRAM PETE (INPUT, OUTPUT, TAPE 1)

         CON  0          L00002      Name & parameter count plus 100(8)
         SB1  L00002     L00001      Entry/Exit line
```

```
        SB2   C00001
        RJ    Q8NTRY          L00003
```

SUBROUTINE PHD (A,B,C)

```
        Con   0               L00002    Name & parameter count
        Con   0               L00001    Entry/Exit line
```

SUBROUTINE PEN (A,B,C,D,E,F,G,H,I,J)

```
        CON   0               L00011    Reserved word for G
        CON   0               L00012                      H
        CON   0               L00013                      I
        CON   0               L00014                      J
        CON   0               L00002    Name & parameter count
        CON   0               L00001    Entry/Exit line
```

Calling Sequence to PEN

CALL PEN (M,N,O,P,Q,R,S,T,U,V)

```
        SB1   M
        SB2   N
        SB3   O
        SB4   P
        SB5   Q
        SB6   R
        SX6   Entry line of PEN
        SA1   X6-1            Name & parameter count
        SB7   X1-6            Number of parameter less 6
        SX6   S
        SA6   A1-B7           Reserved word for S
        SX7   T
```

```
SA7   A6+1                Reserved word for T
SX6   U
SA6   A7+1                Reserved word for U
SX7   V
SA7   A6+1                Reserved word for V
RJ    PEN
0710L00002                10 is parameter count and L00002 is the word
                          containing the name of calling routine.
```

V SYSTEM ROUTINE

The SYSTEM routine is an extension to FORTRAN 2.0, it is not included in version 1.1. For all FORTRAN mathematical routines SYSTEM: handles tracebacks, prints diagnostics, terminates output buffers, transfers to specified non-standard error procedures. The END processors also use SYSTEM to dump the output buffers and print an error summary. SYSTEM, like the initialization routine, Q8NTRY, and the end processors, END, STOP, EXIT, must always be available. Therefore, these routines are combined into one with multiple entry points.

Ordinarily an assembly language routine calls SYSTEM. The calling sequence passes the error number in X_1 and an error message in X_2. Therefore, one error number may have several different messages associated with it. The error summary given at the termination of the program lists the total number of times each error number was encountered.

A FORTRAN coded routine may call SYSTEM with a return jump to SYSTEMP, a special entry point. SYSTEMP must be called with eight parameters. The first six of which are from subprogram. If the subprogram does not have six parameters, dummy parameters must be supplied so that the addresses of the subprogram parameters may be passed to a non-standard recovery routine if one is specified. The seventh parameter to SYSTEMP is the error number - an integer constant or integer variable. The array or simple variable containing the diagnostic message is the eight parameter. After adjusting the arguments properly for SYSTEM, SYSTEMP transfers to SYSTEM for error processing.

If the error is non-fatal and no non-standard recovery address is specified, the error messages are printed and control is returned to the calling routine.

If the error is fatal and no non-standard recovery address is specified, the error messages are printed, an error summary is listed, all the output buffers are terminated, and the job terminates abnormally. If a non-standard recovery address is specified, control is transferred to the recovery routine.

CONTROL DATA CORPORATION
Documentation Department
3145 Porter Drive
Palo Alto, California

AUGUST 1966

PUB. NO. 60175500