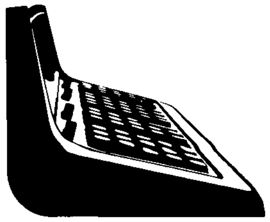
HPL to BASIC Translator



Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Important

The tape cartridge or disc containing the programs is very reliable, but being a mechanical device, is subject to wear over a period of time. To avoid having to purchase a replacement medium, we recommend that you immediately duplicate the contents of the tape onto a permanent backup tape or disc. You should also keep backup copies of your important programs and data on a separate medium to minimize the risk of permanent loss.

Table of Contents

Chapter 1 General Product Description

Introduction

The HPL to BASIC translator is a high-level HP BASIC program. The program tape supplied will operate on the appropriate HP BASIC machine. The Translator allows direct 9825 program input and then executes a two-pass, line-by-line translation of that HPL source to create a BASIC program that will run on your HP BASIC computer.

The HPL to BASIC Translator is designed as an aid for the conversion of programs from the 9825 HPL language to the HP BASIC language. The two languages are different enough that the conversion process may not be complete, and the translation process will require manual correction and optimization after the translator has converted the majority of the code. The translator can translate up to 85-90 per cent of the HPL statements. You must hand-translate the remainder of the statements as well as optimize the translated code, where possible, to utilize the added power of the BASIC language. The translator itself should be viewed as the first step in the translation process, and, thus viewed, can be a very useful tool. The time saving of translating a program versus re-writing can be up to 1:10.

Two listings are produced by the translator. The HPL source is listed during the first pass, and the translated BASIC is listed during the second pass. Any statements that are not translated, or statements where the code might need checking, are flagged on the BASIC listing. The BASIC code is also listed into the file selected by the operator. The translated BASIC program can be entered into the computer memory by executing a **GET** on that file.

NOTE

The translator is a tool to assist in the conversion of HPL programs to HP BASIC. It normally will not translate the programs 100 per cent. Therefore, the translation process requires the following three steps:

- Running the translator to convert the majority of the code and create the output file.
- Editing the translated program to correct the syntax errors noted by the translator and the BASIC syntaxer.
- Running the program to verify proper operation and detect any run-time errors not detectable from the syntax.

The translation process is not complete until all these steps have been taken.

A detailed breakdown on the capability of the translator is given later for all the mainframe and option ROM instructions.

Chapter 2
Operating Instructions

General Operating Intructions

To run the Translator, the following are needed:

- HP Desktop Computer with 64K minimum Read/Write memory.
- A hardcopy output device such as the 9866, 9871, or 2631 printer.
- The HP 9885M Flexible Disk can be used in conjuction with the Mass Storage ROM to increase the operating speed.

To run the Translator program, execute the following statement:

LOAD "INTRO",10

The **INTRO** routine will give you instructions on how to set up the mode of 9825 source transfer, how to set up the Destination file name, and how to set up the other program parameters. Each of the options is explained later in this section.

The **INTRO** routine will link you automatically to the program file translator or to the data file translator.

A utility program named **MAKETP** is supplied for use in copying the translator files to a flexible disk. The translator program will run significantly faster using the flexible disk, and the source and output files can be much larger. To run the **MAKETP** program, execute the following statement:

LOAD "MAKETP",10

The routine will ask for the source and destination Mass Storage Unit Specifiers, and will then copy the files from the translator tape to the disk.

Translator Tape Organization and File Description

The HPL to BASIC Translator is a series of programs that translate an HPL source to a BASIC program. The various segments of the translator are:

- **INTRO**: Introduction and system set-up. This is the program that is loaded and run to start the translator. All other program segments are automatically called by **INTRO** (except for the routine **MAKETP**).
- **PASS1**: First pass of the translation process. The majority of the functions and statements are translated during this pass.
- **PASS2**: Second pass of the translation process. The conversion to BASIC is completed and the BASIC program is listed. Any error messages are also listed.
- **PASS1B**: Additional pass between PASS1 and PASS2 used for the complete translation of subprograms and functions.
- **LISTER**: Routine to print out several different cross-reference tables to aid in program debug and optimization.
- **TAPE**: Routine to read the 9825 cartridge tape using the internal tape drive. The 9825 program file is also decompiled and listed.
- **READIO**: Subprogram driver that is appended to **PASS1** if the program input is via HP-IB. The I/O ROM is needed to input via HP-IB.
- **TCMPL**: Routine to translate 9825 data files on a 9825 tape to HP BASIC data files. This routine also lists key files and can provide a **tlist** of the 9825 tape.

The only operator interaction is in setting up the program parameters, selecting the track and file number of the 9825 program tape to be translated, and in selecting, after the translation process is complete, which cross-reference tables are desired. An Interactive Mode is allowed during the processing of subprograms and functions during **PASS1B**, if desired, to increase the overall effectiveness of the translation of those segments.

Several other files exist on the translator tape. Their purpose is as follows:

- **INBUF**: This is used as the default file to store the HPL source as it is read in from the 9825 tape. It is also used during **PASS1B** to buffer the subprogram program segments.
- **OUTBUF**: This is used as the default file to store the translated BASIC program. At the completion of the translation process, the translated program can be loaded into memory by performing a **GET** on this file.

NOTE

The files selected for use as the source and destination can be changed during the introduction phase.

- **ADD-FN**: Contains subprograms and functions that simulate several HPL functions that do not have a direct translation (such as **stf**, **fti**, **add**, etc). These subprograms and functions are added automatically at the end of the translated program if needed.
- **INTROD**: Data file used by several portions of the translator containing the set-up parameters, the HPL to BASIC mnemonic translation table, and the error messages.
- **MAKETP**: Program to transfer the translator files to a flexible disk.

Detailed Operating Instructions — Program Set-Up

The operation of the HPL to BASIC Translator program is as follows:

A. Load the **INTRO** program and run it by executing:

LOAD "INTRO",10

B. The system will display the setting of the following parameters:

- HPL Source (File Name, **9825**, **HP-IB**, or **KEY**)

  Buffer File Name if the input specified is **9825**.

  Select Code if the input specified is **HP-IB**.
- Destination File Name
- HPL / Data File Translator Listing Selection
- System Printer Select Code
- Interactive Mode Selection
- BASIC / HPL Starting Line numbers

If all the settings are satisfactory, then press the special function key 0 (**K0**) for program translation, or Key 1 (**K1**) for data file translation. If any of the parameters require changing then select the appropriate special function key. The program will then explain what to do to modify that parameter. After all the parameters are set properly, press the special function key **K0** or **K1**.

The data file translator can be loaded at this time. Note that the settings of the system printer and the listing selection flag are used in the data file translator.

The explanation of each parameter follows:

Changing The HPL Source

Four options exist for HPL source input:

- Direct input from a 9825 program tape: The translator, using the program **TAPE** and a special binary, will read automatically the 9825 program file from the 9825 tape, decompile it, and list that source on the system printer and also into the designated HP BASIC data file. This is the most convenient method for transfer. To select this, type in **9825** and hit **CONTINUE**.
- Input from a mass storage data file: The 9825 program must exist in a serial access file in string form, stored as 1 program line per string. The section on the flexible disk transfer explains how to use the 9825 and the disk to create such a file. To select a file as the input, type in the file name. Note that the names **9825**, **KEY**, and **HP-IB** are reserved for the other methods of transfer and thus cannot be used as file names.
- Transfer via HP-IB: The program can be transfered via HP-IB from a 9825 list. HP-IB transfer is covered more in the next section. To enter via HP-IB, type in **HP-IB** and hit **CONTINUE**.
- Entry via keyboard: The program or program segments can be keyed in from the keyboard. Note that no syntax checking is done, so the syntax of the entered line must be exactly the same as the 9825 listing. To enter via keyboard, type in **KEY** and hit **CONTINUE**.

Changing the Input Buffer

If the input specified is **9825** the system will request the name of an input buffer file. Reply with the name of a valid mass storage data file. If the file is not on the default mass storage device, then append to the name the **msus** (**m**ass **s**torage **u**nit **s**pecifier). Examples of valid replies are:

```
INBUF
INBUF:F8,1
```

Changing the HP-IB Select Code

If the input source specified is **HP-IB**, then the system will ask for the select code of the HP-IB interface card.

Changing the Destination File Name

To change the destination file name, reply with a valid file name. If the file does not exist on the default mass storage device, then append the **msus** to the file name. Examples of valid responses are:

```
OUTBUF
OUTBUF:F8,1
```

NOTE

The Input Buffer and Output Buffer must be accessible at the same time.

NOTE

The translator tape is shipped out with two files, **INBUF** and **OUTBUF**. These files are sufficiently large to allow the translation of at least an 8K HPL program. The default buffer and destination file selections are set to these file names. Refer to the section on translator limitations for information on translating long programs and file size requirements.

Changing the HPL Listing Selection

The HPL source can be listed on the system printer during the process of the translation by replying **Y** or hitting **CONTINUE**. To inhibit the HPL listing, reply **N**. This selection is also used to suppress the listing during the data file translation.

Changing the System Printer Select Code

The select code of the printer for the HPL and BASIC listings can be input. To list the program to the CRT, use select code 16. An HP-IB device can be used for the system printer. Examples of the proper format are shown on the CRT. This selection also determines the listing device for the data file translator.

Changing the Interactive Mode

The interactive mode can either be enabled or disabled by replying **Y** or **N**. The interactive mode is invoked in PASS1B when the boundaries of subprograms or functions are not well defined. If the interactive mode is not selected, the program will make an assumption and continue. If the interactive mode is selected, the program will prompt you and ask you to make a decision.

The interactive mode also allows you to specify if an un-dimensioned string variable is an array or simple variable.

The interactive mode is covered more in the section on Advanced Programming.

Starting BASIC and HPL Line Numbers

The translator allows you to specify the starting line number for the translated BASIC program. With the default setting, the BASIC program starts with line number 20. This feature is useful when translating linked programs as the linked programs can be translated starting with different line numbers and then be loaded into the computer memory concurrently.

The translator also allows the starting HPL line number to be specified when reading the 9825 program tape. This is used if a portion of a linked program is stored in a file, and the portion does not start with HPL line number 0.

Source and Destination File Checking

After the translation parameters are set up, the system will check the destination and source files. If they are not available, the system will print a warning and allow you to update the file name. The source and destination files must be available during all phases of the translation process. However, the source and destination files can exist on a separate cartridge and be switched in and out of the tape drive. In the following program segments the operator will receive a prompt if the proper file is not available to the program.

Adding Additional Dimension Statements

The system then asks you if this is a linked program. If not, then hit special function key **K0** to continue. Otherwise, hit key **K1**. This feature allows additional dimension statements to be input.

Normally this feature will NOT be used. However, if the following circumstances exist, additional dimension statements should be used:

- There are either string variables or arrays used in the program that are not dimensioned in the program itself (i.e. they are dimensioned in a program that then links to this program). All string and array variables must exist in a dimension statement to be properly converted and placed into a COMmon statement if needed.
- If this program is to be called by another program through the **LOAD** statement, all variables which must retain their value must be placed in a COMmon statement in both routines. The COMmon statement should be created after the first segment is translated and then entered into this program.

In entering additional **dim** statements, they can either be entered by hand or the file name can be entered. Individual statements are entered by typing in **dim** or **COM** followed by the dimension statement (follow the format shown on the CRT). The name of a file containing **DIM** or **COM** statements from a previous segment can be entered instead, and the translator will automatically search out all **DIM** or **COM** statements. The file name entered cannot contain **DIM**, **COM**, **dim**, or % since they are used as keywords.

HPL comments can be added at the start of the program by using the HPL free-syntax operator % before the line.

Main Translator Operation

After the introduction and set-up stage, either **PASS1** or the **TAPE** routine is loaded depending on the source of the HPL code:

If the input was specified as **9825**, then the **TAPE** routine is loaded. The **TAPE** routine will prompt you for the track number and file number of the 9825 program file. It will then prompt you to insert the 9825 tape and hit **CONTINUE**. After the tape is read, you will be prompted to reinsert the translator cartridge tape. If flexible disk is being used, simply hit **CONTINUE**. The HPL program will then be decomplied, listed if desired, and saved in the buffer file. **PASS1** will then be called directly.

For input from **HP-IB**, **KEY**, or a File, the **PASS1** routine is loaded. If the input is **HP-IB**, then the **READIO** subprogram is also loaded.

After the introduction phase (**INTRO**) and the 9825 tape has been read, if desired, **PASS1** is loaded into the computer and the line-by-line translation begins. The intermediate translated line is displayed in the display area of the CRT.

After the HPL source has been completely converted by the first pass, either **PASS1B** or **PASS2** is loaded depending on whether any subprograms or functions exist in the HPL program. If **PASS1B** is loaded, all the subprograms and functions are relocated to the end of the program, subprogram and function headers are created, and all variables that are used in subprograms are put into COMmon. The program then calls **PASS2**.

NOTE

PASS1B requires either a file named **SRC1** or the file **INBUF** to buffer the subprogram modules during the rearrangement process. If the file **SRC1** exists, it is used. If not, the file **INBUF** is used. The translator tape, in order to make **INBUF** as large as possible, does not contain a file **SRC1**. If the flexible disk is used or the input and output files are on another tape, then a file **SRC1** can be created and used. This allows the input buffer file to remain unchanged in case the HPL source must be used again.

PASS2 completes the translation process, printing each line including any warnings or errors, and listing the BASIC into the destination file. Any translator warnings appear above the line containing the error. If the line is subdivided due to the BASIC translation or an implied assignment, the warning may be printed above the first statement of that sequence.

Cross-Reference Tables and Program Termination

At the end of **PASS2**, the translation in complete. The routine **LISTER** is automatically called. This routine allows the following cross-reference tables to be printed:

- HPL Line Number to BASIC Line Number
- HPL Labels to BASIC Labels
- Format to Image Statements
- Format Statement References
- Subprogram and Function Calls

In addition, the translator can be restarted, or the translated program can be fetched. After the fetch is complete, due to the linkage in BASIC, the translated program may be run and an error may occur (this is normal).

When the program is fetched, all statements that do not syntax correctly are printed on the system printer and converted into comments. Remember that the translation process is not complete until the translated program has been fetched and all the errors corrected. During the correction phase, the BASIC listing produced by the translator must be consulted as many warnings produced by the translator are not flagged when the **GET** is executed.

NOTE

When the program is being fetched, all option ROMs needed to run that program (such as I/O, Plotter) must be in place. Otherwise the translated lines will not syntax correctly and will be converted into comments.

Chapter 3
Transferring the 9825 Program To Your Computer

Introduction

Several alternatives exist for the transfer of the 9825 program to your computer. In each case, the program must be transfered as ASCII string characters. Each of the following methods is aimed at that result. Choose the easiest method for your circumstance.

Program Transfer Via 9825 Tape Cartridge

Direct reading of the 9825 program tape is the most direct method of program transfer. This method requires the special binary program that is present on the translator tape in the **TAPE** program.

The **TAPE** routine is called by the **INTRO** routine whenever **9825** is specified as the source of the HPL program. You are prompted for the track number and file number of the HPL program file. You are then prompted to insert the 9825 tape into the drive, and the tape file is then read.

After the program file is read, the code is de-compiled into the same format as a listed HPL program. This is necessary because the 9825 program file is stored in an internal compiled format.

As the program is decompiled, it is listed, if desired, and stored into the buffer file named in the **INTRO** routine. When the entire HPL program has been decompiled, the **PASS1** routine is called.

Program Transfer Via HP-IB

The program can be transfered from the 9825 to the computer via HP-IB. The two machines are connected by 2 HP-IB 98034[1] interface cables. While the program is being translated during **PASS1**, the 9825 must list the program to the interface card.

To transfer via HP-IB, do the following:

- The interface card that goes into the computer must be modified so that it is HP-IB bus device #20 and is NOT the system controller. Refer to the 98034 Interface Manual for instructions.
- Specify during the **INTRO** routine to transfer via **HP-IB** and specify the proper select code.
- After **INTRO** is loaded, load the program into the 9825 and execute the following statement:

list #720

NOTE

In order to do the list on the 9825, all Option ROMs must be installed and the flexible disk, if accessed by any of the instructions, must be active. Otherwise, list errors will occur on the 9825.

NOTE

If the system printer is also an HP-IB device, two HP-IB cards must be used within your computer and the busses must be kept separate.

---

1 The HP-IB interface card that is plugged into your computer MUST be an HP 98034A **Revised** interface card.

Program Transfer Via Flexible Disk

The program can be transfered by using a flexible disk data file. A disk that is initialized on the 9825 can be directly used on your computer (Note that a disk that has been initialized on your computer lacks the necessary prompts to work on the 9825).

The data file must be opened on the 9825 and then the HPL program must be stored in it as a series of strings. To do that, append the following program to the HPL program to be translated while in memory and then execute a **run "STORE PROGRAM"** :

```
0: "STORE PROGRAM":dim A$[100],F$[20]
1: buf "LINE",100,1;ent "Enter the Dest. File Name",F$
2: ent "Enter the Last Line Number to be stored",E
3: asgn F$,1
4: for I=0 to E
5:  list#'A', I,I
6: red "LINE",A$;dsp A$
7: sprt 1,A$
8: next I
9: dsp "HPL Program Saved";end
10:  "A":ret"LINE"
11: end
```

The 9825 must be configured with all necessary option ROMs in order to create the list. The 12 line program can be saved in a special file and loaded by executing a **get**. This simplifies the creation of the HPL source file.

Chapter 4
Translating Linked Programs

General Considerations

The translation of linked HPL programs is somewhat more difficult than the translation of a single stand-alone program. A program that was segmented in HPL may not, due to the increased memory size of your computer, have to be segmented. The segmentation might change. In addition, variables that are dimensioned in the first segment will need to be dimensioned for the translator when translating following segments so that the string variables can be converted correctly and COMmon statements correctly created. Thus, the translation of a linked or segmented program is not as straight forward as for the stand-alone program, but the translator contains some tools to make the task easier.

How To Translate A Segmented Program

In general, the first segment that should be translated should be the segment that dimensions the variables. After that segment is translated, it should be edited to correct the dimension statements. If the following segments contain subprograms and some variables are used much more often than others, the COMmon statement should be generated at this time to optimize the placement of variables within the COMmon statement.

When the following segments are translated, allow the translator to insert the dimension statements from the first file into the program being translated. To do that, hit special function key **K1** when asked if it is a linked program, then enter the name of the file that contains the first segment. The translator will search that file and insert all the dimension statements at the front of the segment being translated.

After the translation of each following segments, the dimension statements can be deleted. If the COMmon statements were updated, they should be updated for the first segment so that all programs use the same COMmon statement.

When all segments are completed, check and make sure that all the COMmon statements are the same.

There are two ways to link BASIC programs, with the **LINK** and with the **LOAD**. The **LINK** offers the same capability as the 9825 (although slower) with the ability to locate the source being entered anywhere. Thus, a segment can be retained in the present program while overwriting a portion with higher line numbers. The **LOAD** executes faster, but the program overwrites all of the former program. Variables that must be preserved between segments (including single variables) must be stored in COMmon (another reason for creating the COMmon statement).

Other Linked Program Considerations

Branches To Locations Outside The Program Being Translated

If a branch occurs to an HPL line that is not in the segment being translated, the translator cannot determine what to convert it to. It replaces the line as follows:

```
gto 560
```

with

```
GOTO H-560
```

After the translation, those lines will need to be corrected, but that is easily done by doing line-number cross-reference tables for each segment as it is translated.

To aid in this problem, it may be desirable to translate several program segments at the same time.

Subprograms and Functions

In order to translate a section of code as a subprogram or a function, it is necessary for that section to be called within the program being translated. If it is not called, it will not be translated as such. It might be necessary to add additional statements on the end of the program to call the subprograms and functions so they will be correctly translated.

BASIC requires all subprograms and functions to be placed at the end of the program. This creates a problem if a series of subprograms or functions are used by several linked programs and thus kept in the lower part of a program. If possible, they should be converted into simple subroutines. Otherwise, they will have to be saved as a data file and loaded at the end of the program each time a link is performed.

Chapter 5
9825 Data File Translator

Introduction

This program provides for the translation of 9825 files to HP BASIC Mass Storage files. The 9825 file is read, and its contents are translated and/or displayed. The file type is used to determine the action taken. File types 0,1,4 and 6 (Binary Program, Read/Write memory, Program, Null or Empty) are indicated with a message. Type 5, Special Function Key, files are printed on the print device in a format that shows the key definition and key number. File types 2 and 3 (numeric data, data file including string) are translated according to a **dim** and **rcf** or **ldf** statements provided by the operator.

Operating Procedures and Limitations

Program Files

The main HPL to BASIC Program Translator should be used to translate HPL programs.

tlist Option

To obtain a **tlist** of the files on a 9825 tape, LOAD and RUN program INTRO. Select the data file translator by pressing special function key 1. When the translator is loaded use special function key 1 to select this option.

Special Function Keys

To obtain a listing of the special function key definitions (type 5) - LOAD and RUN the program **INTRO**. Select the data file translator using special function key 1. After the data file translator is loaded use special function key 0 to select data file translation. Answer the questions about track number (0 or 1) and the file number. Install the 9825 tape, and press **CONTINUE**. The listing will be printed on the print device (PRINTER IS n) as specified by **INTRO**.

HPL keys definitions containing an immediate execute for one statement only, or an immediate continue can be directly translated to your computer's key. If the key definition involves several executable statements, it is replaced by a BASIC level subroutine and an "ON KEY" statement within the program linking to that subprogram.

Translation of Data Files

To translate data files (types 2 and 3) - LOAD and RUN program **INTRO**. Select the data file translator by using special function key 1. After the data file translator is loaded use special function key 0 to select data file translation. Answer the questions about the track number (0 or 1) and the file number (0<=n). Install the 9825 tape and press **CONTINUE**. When the tape file has been read a prompt to switch back to the computer tape will appear. This prompt includes a query for the computer file name where you want the translated data to be placed. If the file does not exist, the program will create the file automatically. The program will then accept and syntax check a **dim** statement. The **dim** statement must not use variables to specify the size of arrays (not acceptable: dim A[B]). When the **dim** statement is accepted the prompt for the **rcf** or **ldf** statement will appear. In response to this enter the statement that would normally be used to reference the data in the file. The rcf/ldf statement syntax and data type will be checked as the data is translated. Translated data is displayed on the CRT as the translation takes place. It is also printed on the system printer if enabled during the **INTRO** routine.

Exiting the Data File Translator

To exit the data file translator and return to **INTRO** use special function key 2.

Limitations

1. The dimension used in the dim statement must not use a variable (**dim A[B]**)
2. HPL arrays of more than 6 dimensions can not be translated as a single array. The HP BASIC limit is 6 dimensions. You must segment these arrays in order to translate them.

Messages and Prompts

Prompts

Prompt	Response
Enter track number.	Type the track number 0 or 1, then terminate input with **CONTINUE**
Enter file number.	Type the file number (0<=n) then terminate input with **CONTINUE**
Install the 9825 tape and hit **CONT**	Exchange the tapes when necessary and press **CONTINUE**.
Install computer tape. Enter the destination file name.	Exchange the tapes when necessary. Type in the destination file name and terminate input with **CONTINUE**.
Enter the HPL dim statement used in the program that accesses this file.	Type in the HPL **dim** statement and terminate input with **CONTINUE**. If r variables are to be translated then only press **CONTINUE**.
Enter the HPL rcf or ldf statement used in the program that accesses this file.	Type in the HPL **rcf** or **ldf** statement and terminate input with the **CONTINUE** key. Include in the statement the file number (rcf 0,A$). This is required for syntax only.
Does this file contain r variables?	This prompt is given when the **dim** is not given. If r variables are to be translated then type Y and **CONTINUE**. If not type N and **CONTINUE**.
Select Special Function Key for action desired.	A display of available program action is printed on the CRT and corresponding Special Function Key numbers. You should make a selection.

Non-Error Messages

(title message): 9 8 2 5 T A P E F I L E T R A N S L A T O R

** File translation	This message is printed upon selecting file translation.
** tlist	This message is printed upon selecting 9825 **tlist**.
TRACK number	Is printed after the track number is accepted
FILE number	Is printed after the file number is accepted.
Length of file is X bytes.	This message is printed and gives the length in bytes of the 9825 file.
The selected file is a program file.	This message is printed and explains the file type.
The selected file is a Special Function key file.	This message is printed and explains the file type.
The selected file is a string and numeric data file.	This message is printed and explains the file type.
The selected file is a numeric data file.	This message is printed and explains the file type.
The selected file is a Binary Program file.	This message is printed and explains the file type.
The selected file is a Read/Write Memory file.	This message is printed and explains the file type.
The selected file is an empty file.	This message is printed and explains the file type.
The selected file is a null file.	This message is printed when the file header indicates an absolute file size of zero. NOTE: The last file on each track would be of this type.

File translation - Reading 9825 tape.	This message is displayed while reading the 9825 tape during file translation.
tlist - Reading 9825 tape	This message is displayed while reading a 9825 tape during a tlist.
** BASIC destination file is __________	This message is printed upon entering the HP BASIC destination file name.
Loading INTRO.	This message is displayed upon termination of the data file translator.

Error Messages

Improper track number -- Enter track number.	This error message is displayed when the entered track number was not 0 or 1. Enter a 0 or 1 and **CONTINUE**.
Selected file not found--Is 9825 tape installed? Correct and hit **CONT**.	This error message is displayed when the requested file number probably does not exist or the tape may be defective.
End of tape hole detected. To re-try, hit **CONT**.	This error message is displayed when a hole is detected in the 9825 tape. These holes are at both ends of the tape.
Checksum error--Is 9825 tape installed? Correct and hit **CONT**.	This error message is displayed when a checksum error is detected in the file header or in the file body.
The selected file would exceed the available memory space.	This error message is displayed when reading a file of more than 25,000 bytes.
The absolute file size is 0. **CONT**.	This error message is displayed when the file header indicates an absolute file size of zero. NOTE: The last file on each track would be of this type.

Is 9825 tape installed? Correct and hit **CONT**.	This secondary error message is displayed when a possible cause of error was the presence of a BASIC tape.
The BASIC tape is write protected. Correct and **CONT**.	The protect switch on the cartridge tape. Remove and unprotect. Reinstall and **CONTINUE**.
File ———————— is protected. Correct or enter new name and hit **CONT**.	The destination file name was not found or the file is protected. You should CREATE the file or select another file name. Terminate input with **CONTINUE**.
Improper dim statement. Hit **RECALL**, correct and **CONT**.	This error message is displayed when a syntax error is detected in the dim statement. **RECALL** the last line entered, correct and **CONTINUE**.
Improper rcf or ldf statement. Hit **RECALL**, correct and **CONT**.	This error message is displayed when a syntax error is detected in the rcf or ldf statement. **RECALL** the last line entered, correct and **CONTINUE**.

Chapter 6
Program Translator Limitations

Introduction

There are several limitations to the size of HPL program that can be translated. In general they are:

Limitations Due To Source and Destination File Size

The size of the input, output, and subprogram buffer file (if it exists) determines how large an HPL program can be translated. In general, the input file should have 8 records for every 1K of HPL program length (i.e. a 2K program would need **INBUF** to be 16 records long). The output file should contain 16 records for every 1K of HPL program. The increase in size is due to the larger program size needed for BASIC (an average BASIC program will take 2 to 3 times as many bytes as the corresponding HPL program) and for internal representation during the translation process.

If the file **SRC1** exists for buffering subprograms, it should contain about 8-10 records for every 1K of HPL program length. The above lengths are maximum numbers for the vast majority of cases. Most programs will translate with much smaller files. If the input is via 9825 tape, the translator will check the length of the files to make sure that they are large enough.

The translator tape, when shipped, contains large enough files to translate an 8K program. If a larger program is to be translated, then a flexible disk can be used or the **INBUF**, **OUTBUF** and **SRC1** files can be created on a separate tape and switched in and out. The suggested size for these files would be (for a 16K HPL program):

File	**Suggested Size**
INBUF	128 records of 256 bytes/record.
OUTBUF	256 records of 256 bytes/record
SRC1	160 records of 256 bytes/record

Maximum Size of HPL Program That Can Be Read From Tape

The maximum size of HPL program that can be read from a 9825 tape is 20K bytes. This is a limitation due to having 64K bytes R/W memory. Longer programs must be segmented in order to translate.

Maximum Number of HPL Lines Allowed

The maximum number of HPL lines allowed is 750.

Maximum Number of HPL Label Characters Allowed

The maximum number of HPL labels referenced by **gto**,**gsb**, and subprogram and function calls varies depending on the number of characters in each label, but the total number of characters allowed in the symbol table **Label$** is 1400(each HPL label requires 4 bytes of cross-reference information plus the number of characters in the label). An HPL label is entered into symbol table only if it is referenced in a **gto**, **gsb**, **cll**, function call, or a **dev**, **equ**, or **buf** statement. If a program overflows the symbol table (as explained in Appendix A), then start the translator again and sometime during the **INTRO** routine change the value of the variable **Label—limit** from 16 to a lower value. Do that by executing the following statement after **INTRO** has started:

```
Label—limit=X
```

where X is a number less than 16 (see Appendix A for details).

This will truncate the HPL labels more but will allow the program to be translated.

Limitation of the Number of Subprograms and Functions

The maximum number of HPL subprograms (not to be confused with mainframe subroutines) and functions allowed is 100.

Chapter 7
Optimizing and Running the Translated Program

Optimizing the Translated Program

There are some cases where BASIC can offer some improvements in the length of code over HPL. Some of the things to look for are:

- Places where a **jmp** followed by a series of **gtos** or **gsbs** could be replaced by the computed **GOTO** or **GOSUB**.
- Places where the built-in power of the matrix commands could be used.
- Places where, instead of accessing a number of HPL files, a single file could be used with random access.
- Places where the added power of the I/O ROM could be used.

Running the HPL Program

When running the HPL program, there are various defaults that the translator does not put into the program because they may be carried from one program to another. Among them are:

- The 9825 powers up with **fxd 2**. HP BASIC powers up with **FIXED 0**.
- The 9825 powers up in **deg** mode. The translator inserts a **DEG** statement in the front of each program, but each subprogram reverts back to **RAD** mode.
- The printer for the 9825 **prt** and **spc** is always the strip printer. HP BASIC powers up with the printer select as 16, but that can be modified with the **PRINTER IS** statement.
- The free-form printing and display format of the 9825 **prt** may be somewhat different than that for BASIC.

Correcting Translator Warnings and Cautions

The translator attempts to issue warnings with every statement that is not translated or that contains a potential bug. For example, every **not** is flagged, although only a fraction ever require any changing. It is your resonsibility to verify the proper operation of the program after translation. HPL contains some subtle differences that can never be detected during the translation process, and thus the program should be verified for proper operation.

The BASIC Utility Library supplied with your computer contains some BASIC level routines that operate on a program when stored as a data file. Among those routines that will be found useful are:

- SEARCH: Routine that searches for the occurrence of a ASCII string within a program. This is most useful for finding occurrences of labels, line numbers, strings, or arrays. It is not especially useful in finding occurrences of single variables.
- SEARPL: Routine that does the search as described above and in addition allows that string to be replaced by another string. A second data file is required. This routine will be useful in renaming strings, arrays, or **r** variables to make the program more readable. It is not very useful in renaming simple variables (such as in renaming the variable **A** to **Printer—sel**).
- XREFAL: Routine that does a complete cross-reference on the program, including simple variables, arrays, strings, and line labels and references.

Chapter 8
Translation Considerations — Mainframe Instructions

Labels

Labels in HPL are characters within quotes. Any character is allowed. Labels in BASIC consist of a capital letter followed by 0 to 14 lowercase letters, numbers, or the underscore. Labels are converted according to the following rules:

- A label not referenced by a **gto**, **gsb**, **cll**, or a function call is converted to a BASIC remark (preceded by a **!**)
- The HPL label is truncated to 16 or fewer characters for cataloging and comparison (depending on the value of the BASIC variable **Label—limit**). If the first character is not a letter, the letter **A** is inserted in the first position. All following characters are numbers, lower case letters, or the underscore character.
- The formed BASIC label is compared against all existing BASIC labels stored in the symbol table. If the label already exists with a different HPL label, numbers are added on the end to make it unique. In the unlikely event that a label cannot be made unique, an error message is printed.
- HPL allows the legal use of multiple-defined labels which is not allowed in BASIC. Only the first encountered line label is left as a line label.

The **LISTER** routine enables you to print out the label symbol table including the HPL label, the corresponding BASIC label, the line number where it was found, and the type of label.

jmp Statement

The **jmp** statement cannot be fully translated, since the **jmp** may depend on program activity. A **jmp** followed by a simple number is translated similar to a **gto** statement. Otherwise, it is not translated and will have to be manually translated to a BASIC computed **GOTO** or **GOSUB**.

In some extreme cases where a general **jmp** occurs (such as in **jmp G-500**) the jmp expression will need to be analyzed to determine the bounds of the expression. This may have to be determined by running the translated program.

Calculated gosub branching with the **jmp** is also not translated. Whenever a **jmp** instruction is noted, it should be checked for a computed **GOSUB** as it is not specifically flagged. The **GOSUB** statement appears after the **jmp** statement in the BASIC listing.

units Statement

The **units** operator is not translated. If the **units** function is required, the translated program will have to be modified to keep track of the current units in a string and then display that string.

r Variables

HPL **r** variables are converted to an array **Rv**. During the first pass the source is searched for the maximum value of r variable used. This value is then used to set up the dimension statement. If the program contains an instruction such as:

```
rA→B
```

where the limit of r variables cannot be determined, a caution is printed at each location in the program where the maximum value of r variables was inserted.

not Operator

The mathematical priority of the **not** operator is different in HPL and BASIC. Although this is not corrected, each occurrence of the **not** operator is flagged.

Imbedded Assignments

Imbedded assignments which can occur in HPL are translated into as many lines as necessary in BASIC. In the cases where the variable is used in the same statement as where it is assigned, the results might not be correct due to the way the 9825 interprets the line. For example:

```
(25→A)+1→B translates to A=25
                         B=(A)+1
```

giving a result of 26 in both HPL and BASIC

```
A+(25→A)→B translates to A=25
                         B=A+(A)
```

giving a result of 50 in both HPL and BASIC, but

```
1A+(25→A)→B translates to A=25
                          B=1*A+(A)
```

giving a result of 25 in HPL and 50 in BASIC.

The problems resulting from defining a variable and then using it later in that statement are not resolved within the translator, and you should be prepared to resolve any conflicts. No error is produced.

In many cases, because BASIC allows only one statement per line, one HPL line is translated into several BASIC lines. This may affect the operation because more end-of-line cycles are produced. An end-of-line interrupt can now occur in BASIC where in HPL it could not.

HPL Flags

Flags are used on the 9825 which do not exist in BASIC. The Translator converts these to an array **Flag**. Setting and clearing Flag14 on the 9825 corresponds to setting and disabling the defaults. The default values correspond very closely. In the 9825, however, Flag13 and Flag15 have special meaning that cannot be translated. Therefore, if an instruction references either of those flags, a warning is produced.

If an instruction accesses two flags on a **cmf** statement, two BASIC statements are produced. If the same flag is referenced twice, it is complemented twice instead of once as in HPL.

Arrays

BASIC arrays are limited to 6 dimensions. HPL arrays are not limited. The execution of the **GET** will detect that error.

HPL allows variable dimensions within a dimension statement, such as:

```
dim A[N]
```

Such statements are not allowed in BASIC except in subprograms. These statements are flagged, and you will need to dimension them to the maximum expected value.

All string variables and arrays that are used must appear in a dimension statement. This is necessary to properly convert string arrays and to properly create the COMmon statements.

min and max Functions

HPL allows the inclusion of entire arrays in the **min** and **max** statements. BASIC does not allow that and the line is flagged upon the execution of the **GET** on the destination file.

Cartridge Tape Commands

The Cartridge Tape System of the 9825 is file-by-number and is more primitive than the file-by-name approach of BASIC. The Translator changes all tape commands to work on the file-by-name approach even though some statements do not translate and all will require manual changing after the translation is complete.

When a file name is needed in a translated statement, a warning is issued showing the file number that needs replacement. The program inserts the characters:

> "FILE--" followed by the file number accessed

You must insert the new file name after translation. Note that the file name can be a string variable so that different files can still be accessed from one instruction. As an example, both of the programs below reference 20 files. The HPL example references files numbered 0 to 19. The BASIC example references files named Ft0 through Ft19. Note that BASIC files can also be randomly accessed, unlike HPL tape files.

HPL	Translated Basic	Modified Basic
For I=0 to 19	FOR I=0 TO 19	FOR I=0 TO 19
ldf I,A$	ASSIGN #9 TO "File--I"	ASSIGN #9 TO "Ft" &val$(I)
	READ #9;A$	READ #9;A$
prt A$	PRINT A$	PRINT A$
Next I	NEXT I	NEXT I

In most cases converting to the file-by-name method simplifies the translated code and increases the readability of the translated program.

Using the file-by-name approach, the cartridge tape commands translate as follows:

fdf and trk Statements

The operations **fdf** and **trk** are no longer meaningful and are thus changed into a comment or assignment. A warning notes that the statement was deleted.

mrk and ert Statements

The statements **mrk** and **ert** are not translated. They can be replaced by a series of file create/file purge commands.

idf Statement

The operation **idf** does not translate. This operation is used to identify the contents of a file. The directory is not available from within a BASIC program.

Other Tape Operations

The following operations translate as shown and create a warning to indicate that a file name is needed:

- ldb → LOAD BIN "FILE--X"
- ldk → LOAD KEY "FILE--X"
- ldm → LOAD ALL "FILE--X"
- rck → STORE KEY "FILE--X"
- rcm → STORE ALL "FILE--X"

where "X" is the referenced file number.

rcf Statement

The **rcf** is translated into either a **SAVE** if it is a program save or to a **PRINT #9** sequence if it is a data save. If the **rcf** is a data store, the statement is translated to the following series of lines:

- ASSIGN #9 to "FILE--X"
- PRINT #9;Data List,END
- ASSIGN #9 to *

If the data save is a save of **r** variables, the statement is converted into a **PRINT** of the array **Rv**. In HPL, however, the data elements are saved in reverse order of how an array is saved. This should cause no problem for the translated program if the **r** variables are stored into a file and then read back into **r** variables. However, if the file is read back into simple variables or into an array, the order will be reversed.

If the **rcf** is a program store, the statement is translated as follows:

- SAVE "FILE--X"(,First line number(,Second line number))

If line numbers are contained in the **rcf** statement they are translated to the corresponding BASIC line number(s). A warning is generated that the line numbers were converted.

ldp Statement

The **ldp** is translated to a **GET** statement. The HPL line numbers, if any exist, are updated to the equivalent BASIC line number in this program. It must be remembered that the line numbers in this case may correspond to a different program, in which case the updated numbers will be wrong.

ldf Statement

The **ldf** statement translates into either a **LINK** if it is a program load or into a **READ #9;Data List** if it is a data load. In some cases, where the file number is not specified or where no data list follows, the syntax does not make it clear whether it is a program load or a data load. The 9825 solves the problem by reading the tape header. The Translator assumes that all questionable statements are program loads (nearly always correct). All **ldf** statements are flagged with a caution to check the translation.

If the statement is a data read, it translates to the following:

- ASSIGN #9 TO "FILE--X"
- READ #9;DATA LIST

If the statement reads **r** variables, it translates to a read of the array **Rv**. Note that if the file was stored from simple variables or arrays, the order will be reversed. If the load is from a file that was created by the Data File Translator from an **r** variable file, or the file was created in a BASIC program from an **Rv** variable store, the order will be correct.

In loading simple variables and arrays, the default conditions for HPL and BASIC in loading data files are different and not reconcilable. They are:

- If there are fewer data values in the tape file than variables in the data list, HPL loads the variables that can be loaded. BASIC generates an EOF or EOR error.
- If there are more data values in the tape file than variables in the data list, HPL generates an error. BASIC loads all the variables and continues.

If the **ldf** translates into a program load, a **LINK** is used. The HPL line numbers, if any, are translated, but the same cautions apply that applied to the **ldp** statement.

BASIC allows two different types of programs storage: as data files with GET, LINK, and SAVE statements and as program files with STORE and LOAD statements. All program statements are translated into GET, LINK, and SAVE statements which incorporate the same features as HPL. The complementary instructions STORE and LOAD execute much faster that the data file instructions but do not have the relocation capability nor a LINK capability. Linking must be done through COMmon storage similar to how the translator programs are linked.

Chapter 9
Translation Considerations — Option ROMS

String ROM

All statements implemented in the String ROM are implemented in BASIC and are translated.

The str Function

The **str** function in HPL and the **VAL$** in BASIC operate slightly different. In HPL, a positive expression outputs a leading blank that is not output in BASIC. A caution is issued. The translator outputs a space followed by a VAL$, making the positive value case correct. A negative value argument will have an extra space before the minus sign.

String Arrays

String arrays are represented differently in BASIC and HPL. A string array that is represented in HPL as A$[1] is represented in BASIC as A$(1). In order to properly convert strings, the string must be dimensioned within the program. The **INTRO** program allows you to enter any **dim** statements that might be necessary.

NOTE

If a string ARRAY is not dimensioned, it is not translated correctly. A string variable is.

NOTE

If either a string, string array, or array are not dimensioned in the program or dimensioned during **INTRO**, they cannot be entered into any COMmon statement used for Subprograms or Functions. An undimensioned string variable **A$** is entered into the COMmon statement as **A$[]**. An undimensioned array **A** is entered as **A()**.

Matrix ROM

All statements with this ROM are translated. The HPL instruction **inv** is broken down into two statements if the determinant of the matrix is desired.

Plotter ROM

The BASIC Plotter ROM supports communication with the 9872A Plotter (NOT the 9862A). HPL programs that run with the 9862A Plotter are converted to the appropriate BASIC commands, where possible, but will NOT operate with BASIC.

HP BASIC Plotter ROM Limitations

The BASIC Plotter ROM uses the AGL language for high level commands. The high-level commands differ much from the 9825 commands and therefore many statements are not translated directly.

Extended Plotter Statements Using Output Statements

Much of the plotter status that was stored in the plotter with the 9825 is now stored internal to the computer with the BASIC ROM. This means that changing the plotter status manually or with an extended plotter command through an **OUTPUT** statement may not have any effect on the plotter output. The **CSIZE** and **LABEL** are two examples.

It is not feasible, in general, to mix plotting modes in translated programs (i.e. translating 9825 plotter commands to BASIC plotter commands and 9825 **wrt** statements to **OUTPUT** statements because:

- The character size and shape may not agree.
- Plotter ROM statements in BASIC affect, in general, the internal state and not the physical plotter state. Thus, the pen, when moved manually or via an **OUTPUT** statement, will move back to where the computer last placed it prior to executing the next instruction.

The Translator does NOT resolve the conflicts that arise out of mixing plotter and extended plotter statements in the same program.

Axis Statements

The **axe** statement (9862A only) translates only if all parameters are present (intersection and tic-marks).

The 9872A **xax** and **yax** statements will have to implemented manually with the one BASIC axis instruction. To label the graph, a series of **LABEL** and **MOVE** commands will have to be used.

psc Statement

The **psc 0** statement translates to **PLOTTER Sc IS OFF**. The select code inserted is the last processed **psc** select code, or 705.

The **psc** statement is translated for the 9872A/HP-IB case. Thus, the statement:

```
psc A
```

translates to

```
PLOTTER IS INT(.01*A),A MOD 100,"9872A"
```

which is correct for the 9872A but not for the 9862A.

csiz Statement

The **csiz** statement is translated to **CSIZE**, but it produces no output to the plotter. The internal state of the computer is modified.

lbl Statement

Moving the pen prior to executing a **LABEL** has no effect as the pen jumps back to the position stored internally prior to plotting. The pen must be moved by a MOVE or PLOT statement prior to labeling.

In order to eliminate the CR and LF that is generated by the LABEL statement, an image of "#,K" is used. This will simulate very closely the 9825 **lbl** statement. When a positive number is printed on the 9825, a leading blank is output that is not output in BASIC (similar to the **str** function).

Other Non-Translatable Plotter Commands

The following statements do not translate directly:

- pclr – Plotter clear
- xax – X Axis-Will have to be combined with the **yax** into an AXIS statement.
- yax – Y Axis
- ofs – Origin offset-will have to be implemented with a new **SCALE** statement.
- cplt – Character position set. This can be implemented with a MOVE, but it depends on the aspect ratio, etc.

Flexible Disk ROM

Mass Storage Unit Specifier

The Mass Storage Unit Specifier is not added to the file names of flexible disk files. In most cases all transfers within a program will all be to either the cartridge tape or to the flexible disk. If the program contains flexible disk commands, then the following statement is added at the beginning of the program (unless a **drive** statement is the first flexible disk command encountered):

```
MASS STORAGE IS ":F8"
```

If both cartridge tape and flexible disk commands are used within the same program, one device will have to set up as the default mass storage device and the appropriate **msus** added to the file names of the other device.

cat Statement

A **cat** to a buffer or a string is NOT allowed and results in an error.

asgn and Assign Statements

The return variable returned from the HPL statement **asgn** and the BASIC statement **ASSIGN** do not match totally. A return of 0 or 1 is the same, and they are the primary values for which one is testing. In BASIC, a return value of 2 indicates that the file was either protected, the wrong file type, or was not available. HPL returns a value of 3 to 8 to indicate the status of the file that was not assigned.

ens Operator

The **ens** operator following a **sprt** or **rprt** does not have an equivalent in BASIC and is deleted. It is not possible, in BASIC, to replace a single line within a serial file. To replace a single line, the rest of the file will need to be rewritten.

type Function

The **type** function corresponds in value to the BASIC function **TYPE** except for the following:

HPL	BASIC	Meaning
• 2.1	8	Starting portion of a string
• 2.2	9	Middle portion of a string
• 2.3	10	End portion of a string

BASIC also has several more return values that correspond to SHORT and INTEGER variables that are not used by HPL.

Other Non-Translated Statements

The following statements do not translate directly into BASIC:

• copy	(Total disk copy)
• copy	(Partial disk copy)
• dump	(Disk dump)
• load	(Disk load from tape)
• repk	(Repack)

All statements associated with the system tape--those statements in the system tape, along with any binary keyword, are decompiled to **nxrom**.

General I/O ROM

Translation of fmt Statements to IMAGE Statements

The internal format of **fmt** and **IMAGE** statements are very different and require extensive decoding. In addition, the format of input and output **IMAGE** statements in the BASIC I/O ROM is different. This creates a potential problem if a format number is referenced by both an input and output statement. If that occurs, the format statement is flagged.

In HPL, a programmer has ten format statements that can be active at any one time. Program flow, not position in the program, determines which one is active. A BASIC program can have an unlimited number of **IMAGE** statements, but each input or output statement must reference a specific **IMAGE** statement. The translator keeps a cross-reference table of format numbers versus IMAGE statements. A format reference references the closest IMAGE statement directly above the reference, or, if only one **fmt** statement exists for a particular format number, it references that converted **IMAGE** statement regardless of its position in the program. If there is a potential problem with multiple format statements, the translator flags the format references. If the **fmt** statement is on the same HPL line as where it is referenced, it is not flagged. However, since that format statement might be used later on, it should be checked.

Some format specifiers cannot be completely translated. They are:

- An f, fz, or e statement that does not specify the number of decimal places reverts to the current fixed or float setting. Since that is not available in BASIC, a warning is generated and the number of decimal places is set to 0.
- The **c** specifier is replaced by the **K** specifier. A **c5** is also replaced by **K**.
- If the width of the field used with a f,fz, or e specifier is not specified, HPL left justifies the number. That is replaced by the BASIC **K** specifier, which does not allow a specific format (decimal places or exponential formatting).

Write Binary Statement wtb

The **wtb** is converted to the BASIC statement **WRITE BIN** if something is output and no strings or characters are output. Otherwise, it is converted to a **OUTPUT USING "#;---";---** statement. Thus, the statement:

wtb 7

translates to:

OUTPUT 7 USING "#"

and a complex statement such as:

wtb 7,A,32,C$,D

translates to:

OUTPUT 7 USING "#,B,B,K,B";A,32,C$,D

The translator translates all **wtb** statements for byte output. If words are to be output, then the specifier **W** or **Y** must replace the **B**.

Write Control Statement wtc

The **wtc** statement is replaced by a **WRITEIO** statement with a register number of 5.

Read Status (rds) and Read Interface (rdi) Functions

The Read Status and Read Interface functions are converted into statements in BASIC. Thus, an expression of the form:

```
rds(7)→B
```

is replaced by:

```
STATUS 7;B
```

However, if the statement is more complicated and cannot be directly replaced, the line is broken down into two BASIC lines. For example:

```
if rds(7);2→R          or  1+rds(7)→E
```

translates to:

```
STATUS 7;Temp          or  STATUS 7;Temp
IF Temp THEN R=2           E=1+Temp
```

The **STATUS** statement returns the I/O Interface card Status bit as bit 8 for the main return variable.

List Statement

In BASIC, the **LIST** is a command only and is not a statement that can appear in a program. Therefore, it is not translatable. The **list** can be simulated by **SAVE**ing the program in a data file and then printing out that data file. However, in many cases the ability to store the program as a data file eliminates the need for the ability to list from a program.

Extended I/O ROM

Octal/Decimal Operating Modes

BASIC does NOT contain an OCTAL operating mode for I/O. The **moct** and **mdec** statements are translated as follows:

	moct	mdec
to	mode=1	Omode=0

The Omode specifier is used in the utility function that simulates the **add** operator. Otherwise, octal mode is not translated. The translator generates a warning when the **moct** statement is translated.

Octal, Decimal, and Addition (add) Functions

The octal and decimal functions (**oct** and **dec**) are translated to the BASIC I/O ROM functions OCTAL and DECIMAL.

The binary **add** contained in HPL is translated to a function call. Included in the function call is the current operating mode (decimal or octal). If the program contains an **add** function, the utility function is automatically appended at the end of your program after translation.

Interface Control Statements

The following interface control statements and functions are translated into BASIC:

- wti — Write interface
- rdi — Read interface
- iof — I/O Flag
- ios — I/O Status

The **rdi** function is converted into a statement as was the **rds** function.

The **wti 0,<Select Code>** is translated to the statement:

```
Wti—sc=<Select Code>
```

Wti—sc is used for both the **wti** and **rdi** translations.

Table Conversion and Parity Statements

In BASIC, the **CONVERT** statement is intended to replace the **conv**, **ctbl**, and **par** HPL instructions. In addition, the **CONVERT** statement is capable of setting up a separate cross-reference table for each device. In HPL, the three commands are separate but apply to all devices. In converting the three HPL statements to BASIC, the following rules are followed:

conv Statement

When the **conv** is translated, a warning is issued that a select code is needed. Then it is broken down into as many separate **CONVERT** statements as are needed. The BASIC does not clear the convert table with each new **CONVERT** statement as does the 9825. The table must be manually cleared if desired.

ctbl Statement

When a **ctbl** is translated, a warning is issued that the select code needs to be entered. Then the statement is translated to a utility subprogram call to the subprogram **Ctbl** which is appended to the end of your program.

HPL starts the convert table with the character 0. BASIC starts the table with the character 1, with character 0 overlapping at location 256 in the input string. To correctly translate this statement, the entire 256 byte string must be generated by the utility subprogram. The translated **ctbl** statement will run much slower.

par Statement

When the parity statement is translated, it prints a warning that the select code is needed.

If a **ctbl** or **conv** statement is executed that will clear the parity, a warning will be issued that the parity may need to be reset.

Error Functions

The **rom** error function is not translated. The **ern** and **erl** functions are translated but are flagged, since the line number and/or error number returned will not match the error number and line number returned in HPL. Any routine that checks for a specific error number or error line will need to be updated to check for the corresponding BASIC error number and line number.

Device Names

HPL device names (**dev**) are translated to simple variables. The following statement:

```
dev "DVM",710
```

translates to:

```
Dvm—=710
```

Device names should be defined in the program being translated. If they are not defined in the program with a **dev** statement, they are still converted. However, because of the process of making a label, the BASIC label for that device name may not correspond to the name created in another program (although it is not likely that will happen). If the device names are unique (i.e. there are no names with the same letters but capitalized differently), there will be no problems.

HPL Command equates (**equ**) convert to string variables in a similar manner. They must be defined in the translated program to be interpreted correctly.

Note that, for the **dev**, **equ**, and **buf** statements, the symbols are converted in the same manner (truncation, capitalization, etc) as HPL line labels. Also, the strings used for command equates are not dimensioned but are implicitly dimensioned to 18 characters by BASIC. If the string is longer than 18 characters, it must be dimensioned by use of a **DIM** statement.

Buffered I/O

BASIC does not contain the buffer statements or a transfer statement. However, each transfer in BASIC is buffered and each input or output statement allows you to define the transfer mode (i.e. DMA, Fast Handshake with bytes or words, etc.). Thus, much of the power of the 9825 buffered I/O is built into the BASIC I/O structure.

The buffered I/O characteristics are partially simulated as described here:

- All buffers are translated to string variables. If a **buf** statement equates a buffer to a string variable, then that statement has the effect of substituting the string name for the buffer name.

- All reads and writes from/to a buffer are translated to output statements from/to the value area as described in the I/O Manual. Any outputs to a buffer have the effect of writing to that entire string. Any inputs from a buffer have the effect of reading from that entire string. Any input or output statement is flagged when the destination or source is a buffer.
- No buffer pointers are kept. The statement:

  ```
  rds ("buffer")
  ```

  translates to:

  ```
  LEN(Buffer—$)
  ```

  which only simulates partially the operation of the buffer pointer.
- The **tfr** statement is not translated. The transfer operation could be replaced by the suitable **ENTER** or **OUTPUT** statement using the various modes allowed by the computer I/O ROM, or each individual read or write to the buffer could be changed.

Timeout Statement

The HPL timeout statement **time** is translated to **SET TIMEOUT**. A warning is generated noting that the select code and also the timeout linkage are needed. The timeout operates differently in HPL and BASIC. In HPL, the timeout creates an error that is trapped by the **on err** statement. BASIC creates an interrupt when a timeout occurs. Thus, the code to process a timeout must be transfered from an error routine to an interrupt routine. Interrupts must also be enabled.

Interrupts

Abortive interrupts are not allowed in BASIC. If an abortive byte exists, the byte is deleted, and a warning is generated.

Interrupts will likely appear different in BASIC due to the different operating speed. Also, more end-of-lines occur in BASIC because multiple statements per line are not allowed.

HP-IB Programming

Multiple Bus Addressing

The Extended I/O ROM of the 9825 allows multiple devices to be addressed on either input or output statements. Input statements are translated to two statements, a **CONFIGURE** statement followed by the **ENTER** statement. **OUTPUT** statements are directly translated.

Other HP-IB Statements

All other statements involving HP-IB communication are translated.

The **trg** statement asserts the attention line but does not bring it back down. An additional statement might be needed after the trigger, such as a **STATUS**, to bring the line back down.

The I/O ROM for BASIC is needed for all General and Extended I/O ROM statements of the 9825 as well as for HP-IB transfer.

Advanced Programming ROM

For/Next Loops

The FOR/NEXT loops translate directly into BASIC. The **by** is replaced by **STEP**.

Subprograms and Functions

Subprograms and functions for HPL are translated into subprograms and functions in BASIC. A maximum of 100 subprograms are allowed. The subprograms in BASIC are vastly different in the following ways, however:

- All HPL variables are global. Variables in the BASIC routines are local. Therefore, all variables used must be placed into COMmon storage.
- BASIC does not allow jumps outside the subprogram environment. HPL does.
- HPL allows subprograms to exist anywhere. Subprograms in BASIC must be at the end of the program.
- HPL considers format references as global. BASIC requires the IMAGE to be within the subprogram.

From the above list, it is clear that there are many differences between HPL and BASIC subprograms.

Changing Subprograms to Subroutines

In many cases in HPL a **cll** is used instead of a **gsb** because the **cll** is executed immediately and the **gsb** is executed at the end of the line. HPL lines can be conserved using this technique. During the translation process separate BASIC lines are formed anyway. Therefore, a more efficient translation will result if all unnecessary **cll**s are changed to **gsb**s. Subprograms that are good candidates to change to simple subroutines are:

- Routines that have no passed parameters.
- Routines that use no local **p** variables.
- Routines that are not called from another routine that will be translated as a subprogram or function.

Even some subprograms that use local **p** variables or passed parameters can be more efficiently translated to a normal subroutine.

Subprogram and function calls are executed much slower in BASIC than in HPL. If speed is critical, they should be changed to subroutines. Functions can be changed also to subroutines, although the modification required is greater than for a subprogram.

Since HPL subprograms are identified by their **cll** references, it is necessary to change any unnecessary **cll**s to **gsb**s. Either of these two methods can be used: Before starting translation, change the selected **cll** to a **gsb**. This may require creating new HPL lines if the **cll** is on the same line as a **gto** or **gsb**. For example, the line:

```
1: cll subprog;gto "end"
```

would have to change to:

```
1: gsb "subprog" 2: gto "end"
```

Select the interactive mode. Then, when **PASS1B** has been loaded and before the actual subprogram translation begins, you will be given the name of each subprogram and asked if it should remain a subprogram or be changed to a simple subroutine. If you desire it changed to a subroutine, all **cll**s referencing that subprogram will be changed to **GOSUB**s.

Creation of Common Statements for Subprograms

HPL considers every variable to be global. A subprogram or function can use a variable defined in the main program. In BASIC, all variables are considered local within the subprogram unless passed or in COMmon. While this is an advantage for structured programming and modularity, it creates difficulty in translating HPL subprograms to BASIC. To aid you, **PASS1B** does the following:

- All variables used within the subprograms and functions are put in a COMmon statement. An appropriate COMmon statement is put at the start of the program and also at the start of each subprogram, if necessary.
- If you, during the **INTRO** routine, input a COMmon statement(s) as part of the file linkage, that COMmon statement is used as the basis for building the COMmon statements in the present program, thus simplifying the linking of programs.

Determining Subprogram and Function Limits

HPL subprogram endings may be very difficult to determine. To determine where a subprogram ends, the following rules are applied:

- Each **gto** or **gsb** is cataloged and the destination line tagged as belonging to the present environment.
- Potential terminators are:

  ret
  end
  ldf
  ldp
  get
  chain
  gto

  Whenever one of the above terminators is encountered, the program looks ahead to see if any future lines are tagged with the present environment. If so, the subprogram or function is continued. If not, it is terminated.

- If a new subprogram entry point is encountered and an old subprogram has not been terminated, one of three things will happen:

  1. If the operator has selected Interactive Mode, the translator will beep twice and ask you to indicate whether the present subprogram should be terminated.

2. If the Interactive Mode is not active and the two subprogram entry points are adjacent, the present subprogram is continued. The second entry point is flagged, and the calls to that entry point are flagged as invalid calls.

3. If the Interactive Mode is not active and the two entry points are not adjacent, then the first subprogram is flagged and terminated and the second subprogram begun. A linkage will need to be manually created from the first subprogram to the second.

- If a line tagged as belonging to the main program is encountered within a subprogram, the subprogram is terminated. If the Interactive Mode has been selected, you will be asked to define how the translation should continue.

- If the subprogram contains an untranslated **jmp** statement, the subprogram is not terminated until a line is found that is flagged with another environment (such as a line starting another subprogram). In the Interactive Mode you are allowed to define the end of the subprogram if the program flow is not clear.

Adjustment of Subprogram and Function Calls
HPL allows calls to a subprogram or function to have a variable number of parameters. The translator determines the maximum number of parameters passed and makes all calls be that length. Zeros (00) are passed if necessary.

If a subprogram uses **p0**, which is set to the number of parameters passed, the number of passed parameters is added to the pass parameter list of each call to that subprogram, and the variable **P0** is added to the subprogram header.

Relocation of Subprogram and Functions
BASIC subprograms and functions must exist at the end of a program. In HPL they can exist anywhere and may not even be contiguous. The translator relocates contiguous subprograms and functions (i.e. subprograms whose code is all in a straight line) to the end of the program.

Identifying Subprograms and Functions
In HPL, subprograms and functions are identified by their calls. During **PASS1** all calls to subprograms and functions are tabulated. If a subprogram is not called, it is not translated as a subprogram.

Relocation of Comment Lines
Comment lines (stand alone labels) and blank lines just before the starting line of a subprogram or function are relocated to within the subprogram body.

Subroutines within a Subprogram

Subroutines within a subprogram are not detected, although **gsb** statements within a subprogram are flagged. Any **ret** within that subroutine is changed to **SUBEND** or **SUBEXIT** and will have to be converted back.

p VARIABLES

p variables are converted to the BASIC variables **Pv0**, **Pv1**,etc. If a reference is made to a **p** variable in the form **pp0** or **pN**, the variable is converted to an array form similar to the **r** variables. You will need to equate that array variable to the passed **p** variables if necessary.

Other Non-Global HPL Parameters

Several of the HPL parameters are global in HPL but will not be global in the BASIC subprograms. Among them are IMAGE statments (an IMAGE must be in the same environment as the subprogram), device names, and equate strings. IMAGE statements will need to be duplicated in the subprogram. Equate names and buffer names can be passed or placed into COMmon. The translator will detect device names and place them into common storage. If a COMmon statement was input during the **INTRO** routine containing a device name, that device name is equated within the program to the current device name being processed only if all letters of the names are the same so that a one-to-one correspondence between the BASIC name and the HPL name can be determined.

Several other parameters are reset when entering a BASIC subprogram. The trigonometric mode is reset to RADIANS.

Optimizing the Translation of Subprograms and Functions

To optimize the translation effectiveness of subprograms and functions, the following should be done:

- Make sure, either by modifying the HPL source prior to translation or by using the interactive mode as explained, that no subroutines are needlessly called as subprograms.
- If the subprograms and functions involve a large number of variables in COMmon, the COMmon statement can be optimized to reduce the length of the COMmon statement contained in each subprogram.

Split and Integer Precision Storage Functions

The functions **fts** (Full to Split), **stf** (Split to Full), **fti** (Full to Integer), and **itf** (Integer to Full) are converted to BASIC function calls that call the utility routines contained in **ADD-FN**. Those utility routines are appended at the end of the translated program if needed by the translator. BASIC offers a variety of ways to store variables (such as INTEGER, SHORT) that HPL does not offer except through these functions. The translated program will run much faster if the storage functions are converted to numeric expressions using the "INTEGER" or "REAL" storage forms. The **fti** and **itf** can also be converted to single line functions with an increase in the execution speed.

Cross Reference Statement

The **xref** statement is not translated. The utility library contains a BASIC program that will perform a cross-reference on a program stored as a data file (**SAVE**d).

Systems Programming ROM

Intelligent Terminal Instructions

The **on key** and **kret** statements are translated. The **key** statement is translated to the **KBD$** function, although the action of the two is different. The HPL **key** function returns only the next character in the buffer while the **KDB$** returns the entire buffer.

The functions **asc** and **bred** are not translated.

End-of-Line Specification

The **eol** function is translated to the BASIC I/O ROM statement **EOL**. However, for the end-of-line specification to have effect, it must be referenced in the IMAGE statement with the **L** specifier.

Serial Interface Control Statements

The following commands are translated:

- wsc to WCONTROL
- wsm to WMODE
- rss to RSTATUS

The remainder of the Systems Programming statements are not translated, with the exception of the free-syntax operator. They are:

- asc
- bred
- rkbd
- store
- nal
- avm
- cln

H-P Basic System Enhancements

The enhanced BASIC language makes programming much easier. The ability to indent lines allows a program to be segmented to aid understanding. Use of multiple-character names also allows the program to be better documented.

BASIC offers the unified mass storage concept in storing data on tape or on disk. This allows the program to run with either mass storage device without program modification.

BASIC offers a variety of ways that data can be stored internally and on tape. Common storage for subprograms, integer, and short precision are all allowed.

The plotting capability has been greatly enhanced with BASIC, although the translation from HPL to BASIC is not direct. There is a much larger set of instructions for use in plotting.

BASIC automatically buffers all I/O. Also, each file for the mass storage device can be buffered. This can be used to advantage, especially in the **OVERLAP** mode, to speed up the processing of an I/O or Mass Storage intensive program.

Appendix A

Error Messages and Explanations

The Translator prints out WARNINGS and CAUTIONS within the BASIC listing where difficulty was encountered. In general, a WARNING means that the statement was not completely or accurately translated and will need to be edited manually. A CAUTION means that the statement was translated, but it needs to be examined for possible difficulty. Below is a list of the translator CAUTIONS and WARNINGS and their meanings:

Translator Cautions

CAUTION ABORTIVE BYTE DELETED

Abortive Interrupts are not allowed in BASIC. Only end of line interrupts exist. Therefore, the abortive byte was deleted.

CAUTION CALL CONVERTED TO GOSUB

Using the Interactive Mode, a Subprogram is now being translated as a simple subroutine. This CALL is being changed to a GOSUB.

CAUTION CHECK TRANSLATION

The following statement belongs to the **ldf** or **cmd** class where the exact intent of the statement is determined by the 9825 at run time. The statement may not have been translated correctly. Several other instructions are flagged with this warning. Subtle differences in the way BASIC and HPL operate may require modification of the line or following lines.

CAUTION CONVERT MODE ASSUMED OUTPUT

The **conv** statement is assumed to apply to the output direction.

CAUTION ens DELETED

The **ens** at the end of the following statement was deleted.

CAUTION ERROR FUNCTION

The line contains either a **erl** or **ern** function which, although they are translated, may need checking in their use.

CAUTION FUNCTION NOT ALLOWED

A function call is not allowed at this position in BASIC (usually following a **prt** or **wrt**). The function call will have to be moved to a preceding line and a temporary variable defined.

CAUTION gsb IN SUBPROGRAM

A **gsb** was detected in this subprogram/function. The **ret** associated with that inbedded subroutine may have been changed to **SUBEXIT** or **SUB-END**.

CAUTION HPL LINE NUMBERS BEING TRANSLATED

This program load or store command contains line number references which have been translated to the corresponding BASIC line number for this program. That may not be correct for the application and thus should be checked.

CAUTION ILLEGAL SUBPROGRAM ENTRY POINT

This line is called by a **cll** or a function call but was not translated into a subprogram entry point. In BASIC, the only point that a subprogram/function can be entered is through the first line. The statements that call this line will be flagged as invalid calls.

CAUTION IMAGE ERROR AT LOCATION

The **fmt** statement had an item that was not recognized at the specified character position in the line.

CAUTION **K** SUBSTITUTED FOR

The BASIC image format specifier **K** was substituted for the indicated HPL format specifier. This is done when the width is not specified (as if **f.2**) and the result should be left justified. Note that the **K** outputs the number in standard format.

CAUTION MULTIPLE DEFINED LABEL

The line label has already been defined previously.

CAUTION MAXIMUM r VARIABLE USED UNDETERMINED

It may not have been possible, during PASS 1, to determine the maximum **r** variable used, and thus the dimensioning of the array **Rv** may not be correct.

CAUTION MULTIPLE DEFINED FORMAT REFERENCE

This **red** or **wrt** statement references a format number for which there is more than one corresponding IMAGE statements. You should, using the cross-reference tables, verify that the right IMAGE statement is referenced.

CAUTION not OPERATOR

The following line contains the operator **not**. Since the mathematical priority of the **not** operator is different in HPL and BASIC, the line should be inspected to see if any parenthesis are needed.

CAUTION NULL PROMPT

The following **ent** or **enp** contains a null prompt. In HPL this means that any previously displayed comment will remain on the screen for the variable prompt. To operate correctly, a ; will need to be inserted after the **DISP** statement that generates the desired prompt. This will require looking at the program flow to determine which **DISP** should be changed.

CAUTION PARITY MAY BE NEEDED

This statement resets any parity condition that might be set, and thus the parity might need to be reset.

CAUTION PLOT CONTROL OF 0

A plot control parameter of 0 does not have the same effect in BASIC.

CAUTION Rd MISSING IN

The Rd parameter is missing in the **fmt** statment parameter listed. HPL takes the default setting of the **fxd** or **flt** statement for the decimal positions. The translated version will have 0 decimal places.

CAUTION REFERENCE TO INPUT OR ERROR FLAG

Either Flag 13 or Flag 15 is referenced. Neither Flag has a counterpart in BASIC.

CAUTION RETURN VALUE MAY BE DIFFERENT

The value returned by the corresponding BASIC function may be different from that returned by the HPL function. (i.e. **asgn** vs. **ASSIGN**)

CAUTION rnd SEED DELETED

The **rnd** seed was deleted from the **rnd** function and was negative. If you want to initialize the seed, then use the **RANDOMIZE** statement.

CAUTION SE OR DB AFTER rcf DELETED

The debug and secure flags were stripped from the program record command.

CAUTION str FUNCTION

The **str** function was translated into the BASIC **VAL$**. However, if the function expression is positive, **str** outputs a leading blank that is not output in BASIC. Therefore, the length of the string returned in HPL is one character longer than that returned in BASIC. An extra space is inserted by the translator which may make the negative condition incorrect.

CAUTION SUBPROGRAM ENVIRONMENT PROBLEM

This HPL line is reference by a previous HPL line in another subprogram/function or main program segment.

Translator Warnings

WARNING ENVIRONMENT BRANCH ERROR

This line branches to a line outside the current program segment (i.e. outside the current subprogram or function or into a subprogram or function if in the main program)

WARNING eol REQUIRES IMAGE REFERENCE

For the following **eol** sequence to have effect in BASIC, if must be referenced in an IMAGE statment with the **L** operator.

WARNING FORMAT USED FOR BOTH INPUT AND OUTPUT

The format number of this **fmt** statement is referenced by both **wrt** (output) and **red** (input) statements. In BASIC there are slight differences in an input and output image statement. This image statement has been translated for output. You should check and make sure the translation is correct.

WARNING FILE NAME NEEDED TO REPLACE FILE

A file name is needed to replace the dummy file name **FILE — X** that was inserted by the translator. The referenced file number is given.

WARNING FORMAT NOT FOUND

The format number referenced was not found in the current program segment being translated.

WARNING I/O TO BUFFER

The following statement does input or output to a buffer. Since I/O to a buffer will not work the same in BASIC as it does in HPL, the line should be reworked.

WARNING ILLEGAL BUFFER TYPE

The buffer type specified in the **buf** statement was not 0,1,2,3, or 4.

WARNING INPUT f FORMAT ERROR

The specified format element is invalid for an input **IMAGE** statement.

WARNING INPUT EXP FORMAT AT

The specified format element is invalid for an input **IMAGE** statement.

WARNING INTERRUPT LINK TO STRING

The **on int** statement set up a link to a string. BASIC requires the link to be to a line number or label.

WARNING INVALID CALL

This is either a **CALL** or function call that calls a line that was not translated as a Subprogram/Function entry point, or it was a call that was converted to a **GOSUB** through the interactive mode and passed parameters.

WARNINGINVALID dim

One of the arrays was dimensioned using another variable or array. BASIC requires that all arrays in the main program be dimensioned with integers.

WARNING jmp NOT TRANSLATED

The **jmp** instruction was not a simple integer jump. It will have to be hand translated and replaced with a BASIC computed **GOTO** or computed **GOSUB**.

WARNING LABEL CANNOT BE MADE UNIQUE

The mentioned label could not be made unique.

WARNING moct NOT VALID MODE

The BASIC language does not allow for a decimal and octal operating modes. Decimal is the only valid mode.

WARNING OPERAND DELETED

The stated operand was deleted and made into a comment.

WARNING OPERAND NOT TRANSLATED

An operand in the following line was not translated at all, or the translation is not correct for BASIC. You will have to hand translate that instruction.

WARNING p VARIABLE ERROR

A p variable in the following line was accessed as an array and thus was left in that form. All p variables are passed as simple variables. You will either need to change the line or else define the elements of the array **Pv** from the various passed p variables.

WARNING PASS2 SYNTAX PROBLEM

The PASS2 routine encountered a syntax that caused a fatal error. That error was trapped and printed and the translation continued.

WARNING SEGMENT ILLEGALLY TERMINATED

The above program segment was not properly terminated before a new program segment was begun. Generally this means that there was a **gto** or **gsb** to a line that has not been encountered, but the next line either starts a new subprogram/function or is jumped to from the main program.

WARNING SELECT CODE NEEDED

The statement requires a select code in BASIC to operate. Replace the **Sc** with the desired select code.

WARNING STRING VARIABLE — MIS-DIMENSIONED

The listed string variable was either never dimensioned in the translated program, or was dimensioned more than once. This problem can be cured in linked programs by supplying the dimensioning information before the translation begins (during the **INTRO** routine).

WARNING SYNTAX ERROR

The following line contains a syntax problem that caused a fatal error during either PASS1 or PASS1B. The fatal error was trapped, printed, and the translation continued.

WARNING TIMEOUT LINKAGE NEEDED

The timeout linkage is different in BASIC and HPL. in HPL, the **time** statement creates an error and can be trapped by an **on err** statement. In BASIC, the **SET TIMEOUT** creates an interrupt, and thus an interrupt routine must be set up.

Translator Fatal Errors and Remedies

Listed below is a summary of the problems that could cause the translator to halt. In most cases the problem is caused by an array being too small and overflowing, or by the input or output file being too small in size. The arrays used in the translator are large and should be sufficient for the vast majority of programs, but in order to fit the program into 64K of memory the size had to be constrained. Listed are the potential problems and their solutions:

1. Error 59 in Program segment **TAPE**:
   The input file is too small — re-create with a larger size and start over.
2. Error 59 in Program segment **PASS1**:
   The destination file is too small — re-create with a larger size and start over.
3. Error 59 in Program segment **PASS1B**:
   The buffer file used by **PASS1B** was too small. If the file **SRC1** existed, then it was used as the buffer file. Otherwise, the INBUF file was used. Re-create with a larger size and start over.
4. Error 59 in Program segment **PASS1B**:
   The destination file, when attempting to add the subprograms, was too small. Re-create with a larger size and start over.
5. Error message — **Label$ array overflow-translator fails**
   The string variable **Label$** has overflowed. Either the program being translated must be segmented into two smaller parts, or the Translator can be re-run from the **INTRO** routine and the following line executed from the live keyboard after the verbage has been printed:

   Label—limit=X

   where X is some number less than 16. This truncates the long labels as they are compiled to X characters. If the program was nearly through the first pass when the problem occured, then 14 should be ok. If a large number of HPL lines were left to be translated when the error occured, then 12 or 10 should be used.
6. Error 17 in Program Segment **PASS1**:
   The line number cross-reference table **Line—xref** overflowed. The HPL program contained more than 750 lines. To correct, the program must be segmented and run in two parts.
7. Error 17 in Program segment **PASS1B**:
   The program to be translated contains more than 100 subprograms or functions. The program will have to be segmented to allow translation.

Subject Index

HEWLETT
PACKARD

Part No. 09835-10031
Rev. B

Printed in U.S.A.
2nd Edition, Mar. 1981