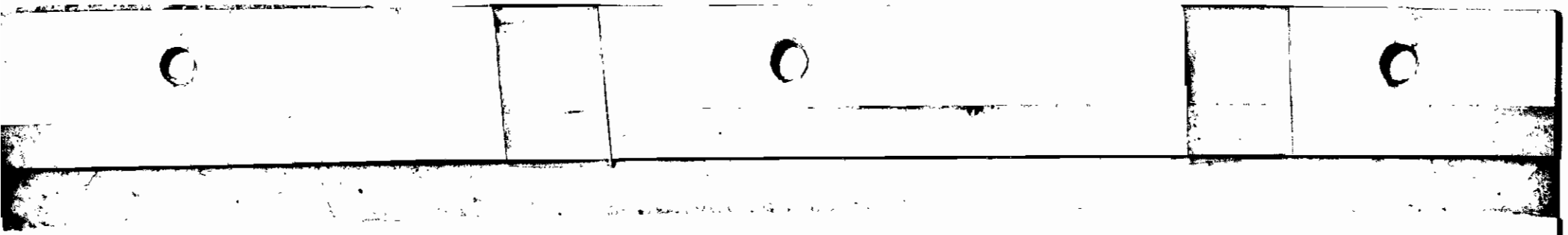
BASIC/CAPS-11

USER'S MANUAL

DEC-11-LIBCA-A-D

digital



BASIC/CAPS-11

USER'S MANUAL

DEC-11-LIBCA-A-D

SWINBURNE COLLECE OF TECHNOLOGY
ELECTRICAL ENGINEERING DEPT.

Order additional copies as directed on the Software Information page at the back of this document.

digital equipment corporation · maynard. massachusetts

Printed July, 1974

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this manual.

The software described in this document is furnished to the purchaser under a license for use on a single computer system and can be copied (with inclusion of DIGITAL's copyright notice) only for use in such system, except as may otherwise be provided in writing by DIGITAL.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

The HOW TO OBTAIN SOFTWARE INFORMATION page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

CDP	DIGITAL	INDAC	PS/8
COMPUTER LAB	DNC	KA10	QUICKPOINT
COMSYST	EDGRIN	LAB-8	RAD-8
COMTEX	EDUSYSTEM	LAB-8/e	RSTS
DDT	FLIP CHIP	LAB-K	RSX
DEC	FOCAL	OMNIBUS	RTM
DECCOMM	GLC-8	OS/8	RT-11
DECTAPE	IDAC	PDP	SABR
DIBOL	IDACS	PHA	TYPESET 8
			UNIBUS

CONTENTS

PREFACE

This document should be used in conjunction with the BASIC-11 Language Reference Manual (DEC-11-LIBBA-A-D). The BASIC/CAPS bootstrapping, loading, running, linking and assembling instructions, the file structure, the error messages, and the system-dependent features are described in this manual. A summary of the BASIC/CAPS system-dependent features may be found in section 2.1 of this manual.

A description of the BASIC-11 Documentation system and conventions may be found in section 1.1 of the BASIC-11 Language Reference Manual. This should be read first by all users. The CAPS-11 User's Guide (DEC-11-OTUGA-A-D) should also be available for reference as the CAPS-11 Monitor System will be used to zero cassettes, copy files, write assembly language routines for BASIC, and to do related file maintenance tasks.

CHAPTER 1

INTRODUCTION

BASIC/CAPS is a single-user BASIC running under the CAPS-11 Cassette Monitor System. BASIC/CAPS provides powerful sequential file capabilities and allows the user to save and retrieve files from cassettes. BASIC/CAPS allows user-defined functions, user-written assembly language routines, and chaining between BASIC programs with data passed in cassette files and/or in memory.

The BASIC/CAPS system is provided on three cassettes. Two running versions are provided on one cassette (DEC-11-OTBAA-A-TC1). These may be loaded by either the bootstrap loader (see section 1.1.2) or the CAPS-11 Monitor (see Section 1.1.1). One version is for PDP-11 systems with more than 8K of memory (referred to in this document as the "greater than 8K version") and the other will run in any PDP-11 with at least 8K (referred to as the "8K version") of memory. The 8K version employs overlaying of the BASIC system from cassette and does not support string variables or the PRINT USING statement. The greater than 8K version is a non-overlaying BASIC and supports string variables and the powerful PRINT USING statement.

BASIC/CAPS runs on any PDP-11 system with 8K or more of memory, a dual drive cassette and a console terminal. BASIC/CAPS also supports the following optional hardware: a line printer (LP11 or LS11), a high-speed paper tape reader/punch (PC11), a low-speed paper tape reader/punch (LT33), special arithmetic hardware (EAE, FPU, PDP-11/40 Extended Processor, and PDP-11/45 Processor), and up to 28K of memory.

For systems with more than 8K of memory, support is also provided for the Laboratory Peripheral System (LPS) and the GT40 display processor (VT11). Support is provided in the form of subroutines that may be linked with BASIC. The procedure to build a load module of BASIC containing LPS or GT support may be found in Appendix B of this document. The use of the subroutines in BASIC programs is described in Appendices D (LPS) and C (GT40) of the BASIC-11 Language Reference Manual.

1.1 LOADING AND STARTING INSTRUCTIONS

1.1.1 Running BASIC/CAPS from the CAPS-11 Monitor

BASIC/CAPS may be loaded and started by the bootstrap loader (see section 1.1.2) or by the CAPS-11 Monitor. As the BASIC program overlays the Monitor there is no reason to load the Monitor for the sole purpose of loading BASIC. However, the Monitor will often be in memory as it will be used to zero cassettes and copy files. When the Monitor is already in memory, the greater than 8K version of BASIC may be loaded by typing (<CR> represents carriage return):

```
R BASIC<CR> (if the BASIC system cassette is on drive 0)
  or
R 1:BASIC<CR> (if the BASIC system cassette is on drive 1)
```

The 8K version of BASIC may be loaded by typing:

```
R BAS8K<CR>   (for cassette drive 0)
  or
R 1:BAS8K<CR> (for cassette drive 1)
```

At this point the initial dialogue, which is described in Section 1.2, will occur.

1.1.2 Bootstrapping BASIC/CAPS into Memory

BASIC/CAPS, in addition to being run from the CAPS-11 monitor, can also be bootstrapped directly into memory, in a manner similar to the way the CAPS-11 Monitor is bootstrapped (see section 3.1 in the CAPS-11 User's Guide).

The two running versions of BASIC are supplied on cassette DEC-11-OTBAA-A-TC1. The following files are on the cassette:

CTLOAD.SYS	Cassette Bootstrap File
BASED.OVL	8K Version Edit Overlay
BASEX.OVL	8K Version Execute Overlay
BAS8K.SLO	8K Version
BASIC.SLO	Greater than 8K Version
FTBLA.OBJ	(Used for BASIC/CAPS demonstration)
PROG2.OBJ	(Used for BASIC/CAPS demonstration)
PROG2.BAS	(Used for BASIC/CAPS demonstration)
FPMPEA.OBJ	Math package for EAE hardware
FPMPEI.OBJ	Math package for EIS hardware
FPMPFP.OBJ	Math package for FPU hardware

NOTE

The BASIC/CAPS demonstration is described in GETTING STARTED WITH BASIC/CAPS (DEC-11-DTBIA-A-D).

To bootstrap either version of BASIC into memory from the cassette provided, follow this procedure:

1. Press and Raise the HALT switch.

2. Mount BASIC cassette on unit 0. If the PDP-11 system includes a hardware cassette bootstrap go to step 4.

3. Toggle in one of the software bootstraps listed in Table 3-1 of the CAPS-11 User's Guide.

4. Set the Switch Register to the start address of the bootstrap (hardware or software) and press LOAD ADDRESS.

5. Set the Switch Register to 000007 (octal) to bootstrap BAS8K.SLO or to 000011 (octal) to bootstrap BASIC.SLO and then press START. The cassette on unit 0 will spin forward and then will halt.

6. Set bit 0 of the Switch Register to 0. Bits 1 to 15 of the Switch Register must not all be 0; at least one bit must be 1!

7. Press CONTINUE and BASIC.SLO (or BAS8K.SLO) will be read in and then the initial dialogue will begin (see section 1.2 for a description of the initial dialogue).

1.1.2.1 Description of Bootstrap Procedure - The hardware or software bootstrap causes the file CTLOAD.SYS (which must be the first file on the cassette) to be read into memory and started. The Switch Register settings in step 5 are interpreted by CTLOAD as described in Table E-2 in Appendix E of the CAPS-11 User's Guide. Setting bit 0 to 1 signifies a special load and bits 1-14 specify the position of the file to be loaded. If the file to be loaded is the nth file after CTLOAD (or the 1+nth file on the cassette) bits 1-14 should be set equal to n. Alternatively, the user may create a cassette on which the first file is CTLOAD and the second is BASIC.SLO or BAS8K.SLO. This cassette may be bootstrapped exactly as the CAPS-11 Monitor System is bootstrapped (see section 3.1 of the CAPS-11 User's Guide.)

1.2 INITIAL DIALOGUE

When BASIC is first loaded by either the R command or the Bootstrap Loader, the following once only dialogue occurs:

BASIC prints:

```
BASIC/CAPS V01-01 (or current version)
OPT FNS?
```

and awaits specification on inclusion of the optional functions. Refer to BASIC-11 Language Reference Manual, Chapter 3 for information on these functions. Depending on the response (Carriage Return, A, N, or I) to this message, all functions (Carriage Return or A) are included, none of the functions (N) are included, or the functions are listed and may be individually selected for inclusion (I).

Selectively excluding functions provides space for additional user program lines. Reply with one of the following codes (<CR> represents the Carriage Return key):

CODE	EXPLANATION
A<CR> or <CR>	Loads all of the optional functions
N<CR>	Loads none of the optional functions
I<CR>	Allows the functions to be specified individually.

If any character other than a Carriage Return, A, N, or I is typed, the message is repeated. If the reply is I, BASIC prints:

```
Y - YES    N - NO
RND:
```

to allow specification of each function to be loaded as part of BASIC/CAPS.

Reply with a Y<CR> or N<CR> for the RND function and each additional function as the names are printed. The optional functions are:

String BASIC (Greater than 8K)	No String (8K)
RND	RND
ABS	ABS
SGN	SGN
BIN	BIN
OCT	OCT
TAB	
LEN	
ASC	
CHR$	
POS	
SEG$	
VAL	
TRM$	
STR$	

Each exclusion of a function other than POS or SEG$ provides room for between two and five additional program lines. Excluding the POS and SEG$ functions provides approximately ten additional lines each.

If any user assembly language routines had been linked into BASIC, to be referenced by a CALL statement (See Chapter 4), BASIC prints:

```
USER FNS LOADED
```

Next BASIC requests the terminal type by printing:

```
TERM?
```

Reply with one of the following responses:

0<CR> or <CR>	if the terminal is not a fast serial terminal.
1<CR>	if the terminal is a 300-baud LA30.
2<CR>	if the terminal is a VT05 with a data rate of 600 baud or greater.

BASIC then prints

```
DATE:
```

requesting the current date in the form

```
DATE:dd-mmm-yy
```

where dd is the day of the month (may be one or two digits), mmm is the first three letters of the month and yy is the last two digits of the year.

If the date is not in this format, BASIC prints

```
BAD DATE
DATE:
```

requesting the date again. After the user has typed in a date in correct format, BASIC prints the message

```
READY
```

which indicates that the system has been successfully initialized and that a command or program line may be typed (see BASIC-11 Language Reference Manual, Chapter 1).

If the computer is turned off while BASIC is operating, the ?PWF (Power Fail) error message is output when power is turned on again. The user program is not destroyed and BASIC returns to the READY message with all files closed.

BASIC can be restarted by setting the switch register to all 0's and by pressing and releasing the HALT, LOAD ADDR, and START KEYS. The user program is not preserved.

CHAPTER 2

SYSTEM-DEPENDENT FEATURES

2.1 SUMMARY OF SYSTEM-DEPENDENT LANGUAGE FEATURES

The following table summarizes the system-dependent language features and messages

CTRL/C	A CTRL/C key command stops execution of the BASIC program, prints a STOP or STOP AT LINE message and then returns BASIC/CAPS to the READY message. A second CTRL/C immediately after the first causes BASIC/CAPS to print the REBOOT? message.
REBOOT?	Prompts the user to place the CAPS-11 System Cassette on cassette unit 0 and confirm the reboot request.
B?	In 8K version only. Indicates that the cassette on unit 0 has been searched and that the needed system overlay file has not been found. Prompts user to place a cassette which contains the 8K BASIC/CAPS system overlay files on unit 0.
ASSIGN	Not in BASIC/CAPS
BYE	Not in BASIC/CAPS
DEASSIGN	Not in BASIC/CAPS
KILL	Not in BASIC/CAPS
NAME AS	Not in BASIC/CAPS
UNSAVE	Not in BASIC/CAPS

2.2 CONTROL/C KEY COMMANDS (CTRL/C)

Typing a CTRL/C terminates program execution, closes all open files, and produces one of the two following messages:

Message	Meaning
STOP AT LINE xxxx	BASIC was executing a program when the CTRL/C was pressed and has executed line xxxx (unless line xxxx is a multi-statement program line).
STOP	BASIC was in Edit mode (able to accept commands or program lines typed in) at time of CTRL/C.

In either case the program in memory is retained and BASIC then prints the READY message and waits for a command or program line to be typed in.

A CTRL/C typed during the initial dialogue has no effect on BASIC/CAPS.

Response to CTRL/C in BASIC/CAPS is not immediate if cassette input or output is in progress. This includes the BASIC system overlaying that occurs only in the 8K version.

A second CTRL/C immediately after the first causes BASIC to print:

```
REBOOT?
```

This feature allows the CAPS-11 Monitor or another version of BASIC to be loaded without having to go through the system bootstrap procedure.

To load the CAPS-11 Monitor, first place the CAPS-11 system cassette on cassette unit 0; set bit zero of the switch register to zero, and then type any response beginning with Y and ending with a <CR>. This reboots the CAPS-11 System.

To reboot BASIC, first mount the supplied BASIC cassette, set the switch register as described in step 5 of the instructions to bootstrap BASIC in Section 1.1.2, and then type any response beginning with Y and ending with a <CR>. When the cassette halts set bit zero equal to zero and press the CONTinue switch.

Any response not beginning with a Y cancels the reboot request and returns BASIC to the READY message leaving the stored program unchanged.

2.3 OVERLAYING IN THE 8K VERSION

To provide maximum space for user programs, the 8K version of BASIC/CAPS (BAS8K.SLO) employs overlaying. Overlaying is the method by which sections of the BASIC system program that are never required simultaneously can occupy memory alternatively. This feature requires that a cassette containing the BASIC/CAPS overlay files (BASEX.OVL and BASED.OVL) be mounted on cassette unit 0 when overlaying is necessary.

Whenever BASIC must overlay to continue operation, the cassette currently mounted on unit 0 is searched for the necessary overlay file. If the needed file is found it is read into memory and then BASIC continues. If the cassette mounted on unit 0 does not contain the overlay file or if no cassette is mounted on unit 0 BASIC prints the prompt:

```
B?
```

At this point a cassette containing the BASIC overlay files should be mounted on unit 0. Then any character on the keyboard should be typed. BASIC searches the cassette for the needed overlay file and either reads the file into memory and continues or, if the file is not found, prints another B? prompt.

When the cassette mounted on unit 0 contains the two overlay files the BASIC overlay mechanism is invisible except for the additional time required for transitions from Edit to Execute mode and the fact that an open file on cassette unit 0 is closed when overlaying is needed.

When BASIC programs and commands are doing cassette file Input/Output, it is convenient to have several scratch cassettes with the BASIC/CAPS overlay files on them. These cassettes may be created by using the CAPS-11 PIP program to copy the BASIC/CAPS overlay files onto zeroed cassettes. These cassettes can then be used for data and program file storage, and it will not be necessary to remount the BASIC system cassette when overlaying is required.

The following commands and statements require overlaying in the 8K version:

```
RUN
RUNNH
RUN file descriptor              to go from Edit to Execute mode.
RUNNH file descriptor
Any immediate mode statement

STOP
END
OVERLAY                          to go from Execute to Edit mode.
CHAIN
CTRL/C
After fatal error message
After execution of any immediate mode statement
```

Execution of any of these commands or statements will close any open file on cassette unit 0. A file on unit 0 may be opened in a multiple statement immediate mode line but the file is closed after the line is executed.

CHAPTER 3

FILES IN BASIC/CAPS

3.1 BASIC/CAPS FILES

BASIC/CAPS has the ability to utilize cassettes for data file storage and for saving BASIC programs. Data files are controlled by the OPEN, CLOSE, PRINT#, PRINT# USING, INPUT#, IF END#, and RESTORE# statements described in section 2.11 of the BASIC-11 Language Reference Manual. BASIC program storage and retrieval is accomplished by use of the OVERLAY and CHAIN statements, described in section 2.12 of the BASIC-11 Language Reference Manual, and the OLD, SAVE, REPLACE, APPEND, RUN, and RUNNH commands described in Chapter 4 of the BASIC-11 Language Reference Manual. The format of the file descriptor used in these statements and commands and the system-dependent features of cassette files are described in this chapter. In addition to cassettes, BASIC/CAPS supports a line printer, high-speed paper tape reader, and a high-speed paper tape punch via the same commands and statements used to manipulate cassette files. All I/O devices except the console terminal are treated as sequential files; however, the RESTORE # statement is meaningful only if the input file is on a cassette.

The OPEN statement opens files for input or output by the BASIC/CAPS program.

The format is:

OPEN string [FOR INPUT / FOR OUTPUT] AS FILE #expr [DOUBLE BUF]

where string may be a string constant or scalar string variable and is a file descriptor.

expr is the logical unit number in the range 1-7.

See section 2.11.1 for a more complete description of the BASIC-11 OPEN statement.

Only one file may be open on one device at any time. (Each cassette unit is considered a separate device.) If an attempt is made to open a file on a device that has a file open on it, a ?DCE (Device Channel Error) message is printed, all files are closed, and BASIC returns to the READY statement.

All open files are closed by execution of any of the following: CTRL/C, SCR, NEW, OLD, RUN[NH] [file descriptor] and CLEAR commands, STOP, CHAIN, and END statements, and any program termination, including error messages, which causes BASIC to print the READY message. Any statement that causes the 8K version of BASIC/CAPS to overlay (see section 2.3) closes any open file on cassette unit 0.

Inclusion of DOUBLE BUF in the OPEN statement is meaningful only if the device is a cassette. If specified, a second 64-word Input/Output buffer is allotted to the file. Using DOUBLE BUF improves the execution speed of a program with cassette I/O but requires more memory.

The Line Printer and High-Speed Paper Tape Punch can only be opened for output. If one of them is opened for input an ?ILR (Illegal Read) error message is printed. The High-Speed Paper Tape Reader can only be opened for input. If it is opened for output a ?WLO (Write Lock) error message is printed.

If an attempt is made to open a file for output on a write-locked cassette the WRT LCK? error message is printed. Saving or replacing a program onto a write-locked cassette also produces the WRT LCK? error message.

3.1.1 File Descriptor

The file descriptor is used in the OPEN, OVERLAY, and CHAIN statements and the RUN, APPEND, OLD, SAVE, and REPLACE commands.

The general form of the file descriptor in BASIC/CAPS is:

```
dev:filnam.ext
```

where

dev: represents the device that the statement or command is operating on. The devices that are supported by BASIC/CAPS are:

dev:		device
CT0:	or 0:	Cassette unit 0
CT1:	or 1:	Cassette unit 1
LP:		Line Printer
PR:		High-Speed Paper Tape Reader
PP:		High-Speed Paper Tape Punch

filnam — is an alphanumeric string with a maximum of six characters; characters after the sixth are ignored. When the device is a cassette unit filnam is the name of the file accessed. When filnam is in an OLD, RUN or RUNNH command or CHAIN statement, filnam becomes the program name.

.ext — is an alphanumeric string with a maximum of three characters in addition to the period. Characters after the third are ignored. When the device is a cassette unit .ext is the extension of the file accessed.

The following are the default conventions:

dev: — whenever the dev: is omitted CT0: is assumed.

filnam — when the filnam is omitted in OPEN statements and SAVE and REPLACE commands, the current program name is assumed.

.ext — when the extension is omitted in the OPEN statement .DAT is assumed.

when the extension is omitted in the OVERLAY or CHAIN statement or OLD, SAVE, REPLACE, APPEND, RUN, or RUNNH command, .BAS is assumed.

NOTE

In order for BASIC programs to be upward compatible to a disk-based system, BASIC/RT11, it is recommended that cassettes not be referred to as "1:" and "0:" in BASIC programs but rather as "CT1:" and "CT0:".

In BASIC statements requiring a file descriptor (OPEN, OVERLAY and CHAIN) the file descriptor may be a scalar string variable or a string constant.

3.1.2 Examples of File Statements

```
OPEN "CT0:ABC" FOR OUTPUT AS FILE #1
```

Creates ABC.DAT on cassette 0 as file #1.

```
LET A$ = "CT1:XYZ"
LET B  = 1+1
OPEN A$ AS FILE B DOUBLE BUF
```

Opens file XYZ.DAT on cassette 1 as file #2 for input and allocates two 64-word I/O buffers.

```
OPEN "CT0:" AS FILE #5
```

If current program is named FIND, this statement opens file FIND.DAT on cassette 0 as file #5 for input.

```
OPEN "1:ALTO.MAC" FOR INPUT AS FILE #4
```

Opens ALTO.MAC on cassette 1 for input as file #4.

```
OPEN "LP:" FOR OUTPUT AS FILE #7
```

Opens the specified device, line printer, for output as file #7. If FOR OUTPUT is not specified, input is assumed and an ?ILR (Illegal Read) error message results since the line printer is a write-only device.

```
OVERLAY "CT1:OVL1"
```

Merges the program in memory with the program in OVL1.BAS on cassette unit 1.

```
CHAIN "PR:"
```

SCRatches the program in memory and then loads and runs the program in the high-speed paper tape reader.

3.1.3 Examples of File Commands

```
SAVE LP:
```

Lists the program in memory on the line printer.

```
SAVE SNOOPY
```

Saves the program in memory as file SNOOPY.BAS on cassette unit 0.

```
OLD PR:TEST1
```

Scratches the program in memory and loads the program in the high-speed reader. The program name is changed to TEST1.

```
SAVE PP:
```

Punches the program in memory on the high speed punch.

```
OLD CT1:FOOBAR
APPEND CT0:SUBR1
```

The OLD command SCRatches the program in memory, loads the program in file FOOBAR.BAS on cassette unit 1, and changes the program name to FOOBAR. The APPEND command merges this program with the program in file SUBR1.BAS on cassette unit 0 and does not change the program name.

```
REPLACE PROGRM
```

Deletes the file PROGRM.BAS from cassette unit 0 and then saves the program in memory as file PROGRM.BAS on cassette unit 0.

```
RUN DEMO
```

SCRatches the program in memory and then loads and runs the program in the file DEMO.BAS on cassette unit 0. The program name is changed to DEMO but no header is printed.

CHAPTER 4

USING ASSEMBLY LANGUAGE ROUTINES WITH BASIC

BASIC/CAPS has a facility which allows experienced PDP-11 assembly language programmers to interface their own assembly language routines to BASIC. This facility permits the user to add functions to BASIC which can operate directly on special purpose peripheral devices. This chapter describes in some detail the internal characteristics of BASIC during the execution of a BASIC program, and is intended to serve as a programming guide for the creation of such user-coded assembly language functions. This material assumes the user is familiar with PDP-11 assembly language. For additional information on this subject, refer to the CAPS-11 System Users Manual.

The CALL statement is used to execute these assembly language routines from the BASIC program. See Section 2.10.8 of the BASIC-11 Language Reference Manual for a description of the CALL statement.

4.1 SYSTEM FUNCTION TABLE

For a routine to be accessible from the CALL statement, it must be defined in the special System Function Table. This table allows BASIC to associate the name of an assembly language routine with the actual location of the routine in memory. The function table consists of a list containing the name and address of every routine that may be called from BASIC.

The software provided with the BASIC kit includes a function table, FTBL.PAL. User-written assembly language routines must be defined in this module. The routines provided with the BASIC peripheral support for graphics (GT) and for the Laboratory Peripheral System (LPS) are conditionally defined in FTBL.PAL. See Appendix B for instructions to define the peripheral support routines in the function table.

The table consists of a series of 3-word entries. The first two words of the entry contain the ASCII characters of the routine name to be used in the CALL Statement. Those names with less than four characters are followed by 0 bytes to fill the remainder of the two words. A 4-character call name should have no 0 bytes, a 3-character call name should be followed by one 0 byte, a 2-character call name should be followed by two 0 bytes (or one 0 word), and a 1-character call name should be followed by three 0 bytes.

The third word of the entry contains the address of the function which must also be declared a global.

The following instructions to the EDIT program of CAPS-11 will produce a new FTBL which contains the names of three assembly language routines (AND, OR, and REV) and their addresses (ANDFN, ORFN, and REVFN, respectively). The sample routines in section 4.2 are the three assembly language routines that would be called by AND, OR, and REV.

```
                (Mount scratch cassette on unit 1)
                (Mount CAPS-11 system cassette on unit 0)

.Z 1:
.R EDIT
*EW1:FTBL.PAL $$
                (Mount cassette with FTBL.PAL on unit 0)

*ER0:FTBL.PAL $ R $$
*FFTBL: $$
*I
                .GLOBL    ANDFN,ORFN,REVFN
                .ASCII    'AND'
                .BYTE     0
                .WORD     ANDFN
                .ASCII    'OR'
                .BYTE     0,0
                .WORD     ORFN
                .ASCII    'REV'
                .BYTE     0
                .WORD     REVFN
$ EX $$
```

$ represents the ALTMODE key.

The FTBL.PAL on cassette unit 0 should be assembled and linked with the other BASIC object modules and the user functions as described in sections A.2.1 and A.2.2.

4.2 WRITING ASSEMBLY LANGUAGE ROUTINES

The user's assembly language routine must interface with the BASIC system to pass its arguments to and from the calling BASIC program.

If the user's routine does not accept a variable number of arguments, then the general subroutines GETARG, STORE, and SSTORE, which are listed in section 4.4., should be used to interface the user routines with BASIC. The routine GETARG checks the syntax of the CALL statement and the argument types. It accesses the routine arguments as specified in the CALL statement and stores references to them in a table addressed by R0.

Argument Type	Stored in table at (R0)
1 - Input numeric expression	two words, the expression value
2 - Output numeric target variable	three words, used by STORE subroutine
3 - Input string expression	zero words are stored in table, string pointer is returned on the stack
4 - Output string target variable	three words, used by SSTORE subroutine

To store target variables (argument types 2 and 4), the user routine addresses the corresponding 3-word entry in the table set up by GETARG and calls the subroutine STORE for numeric target variables, and SSTORE for string target variables. The examples in section 4.2.1 show how these routines are used.

Once the user routine has called GETARG to access its arguments, it may use any registers except R5 for calculations. The routine must return via an "RTS PC" instruction, with the stack unchanged.

The GETARG, STORE, and SSTORE subroutines assume that all arguments to the user routines are in the CALL statement. In the case of a user routine which handles optional arguments, the user routine may use the system subroutines described in section A.3.1 to pass the arguments to and from BASIC. Each of the routines named is a .GLOBL symbol.

When the CALL statement is executed, the user's assembly language routine is called by the instruction:

```
JSR PC, routine address
```

When the user routine is entered, these registers contain information about the calling sequence:

R1 is a pointer to the translated code of the CALL statement. (See section A.3.3 for the format of the translated code.)

If the routine has an argument list, R1 points to the 1-byte token (refer to section A.3.1 for an explanation of tokens) which represents the left parenthesis in the calling sequence. This token has the value .LPAR.

```
          R1
          ▼
CALL "AND" (A,B,C)
```

The 1-byte values of code bytes (tokens) .LPAR, .COMMA and .RPAR (right parenthesis) are global symbols. These are not the same as the ASCII representation of these characters.

R4 Contains the low limit of the stack. If the stack is used heavily, the function must check that it never goes below this limit. (If it does, transfer control to ERRPDL, a global location in BASIC.)

R5 Contains the address of the "user area", which must be preserved for all calls to BASIC subroutines.

Once the argument references are no longer required by the function, R0 through R5 may be used in any way. R0, R2, and R3 need not be preserved in any case.

The function may use the stack, but must return via an

```
RTS PC
```

instruction with the stack unchanged.

The user routine can not use the TRAP instruction, as it is reserved for use by the BASIC system program.

A user routine which does not use the GETARG subroutine should verify the syntax of the invoking CALL statement by checking that the left parenthesis, comma and right parenthesis tokens are contained in the code where expected. (.LPAR, .COMMA and .RPAR are the global values of these 1-byte tokens, respectively.)

In general, arguments which are expression values are passed to the user by the subroutine EVAL, as described in section A.3.1.The program can then obtain the value of the expression from the floating accumulator or FAC (FAC1(R5) and FAC2(R5)).

Arguments are passed from the user routine back to BASIC by first calling GETVAR to address the target variable and then calling STOVAR for numeric results and STOSVAR for string results to store the new value in the BASIC variable. These routines are also described in section A.3.1.

4.2.1 Sample User Functions

The following source program shows how the routines AND, OR and REV in the function table created by the editing instructions in section 4.1 would interface with the BASIC system to pass their arguments to the calling program. These programs use the general subroutines GETARG, STORE, and SSTORE listed in section 4.4.

```
; FUN2 - SAMPLE USER FUNCTIONS
        .TITLE  FUN2
        .GLOBL  ANDFN, ORFN, REVFN
        .GLOBL  GETARG, STORE, SSTORE
R0=%0
R1=%1
R2=%2
R3=%3
R4=%4
R5=%5
SP=%6
PC=%7
FAC1=40
FAC2=42
;
;
; "AND"  (A,B,C)
ANDFN:  MOV     #TABLE,R0       ;ADDRESS VARIABLE STORAGE AREA
        JSR     PC,GETARG       ;CHECK SYNTAX AND SET ARGS
        .BYTE   1,1,2,0         ;(ARG TYPES)
        .EVEN
        MOV     #FAC1,R3
        ADD     R5,R3           ;ADDRESS FAC1(R5) IN R3
        MOV     A1,R2
        COM     R2
        MOV     B1,(R3)
        BIC     R2,(R3)+        ;FAC1(R5) IS A1 (AND) B1
        MOV     A2,R2
        COM     R2
        MOV     B2,(R3)
        BIC     R2,(R3)         ;FAC2(R5) IS A2 (AND) B2
        MOV     #C,R0           ;ADDRESS C
        JSR     PC,STORE        ;STORE FAC1,FAC2 IN C

        RTS     PC
; "OR"   (A,B,C)
ORFN:   MOV     #TABLE,R0       ;ADDRESS ARGUMENT TABLE
        JSR     PC,GETARG       ;CHECK SYNTAX AND GET ARGS
        .BYTE   1,1,2,0         ;(ARG TYPES)
        .EVEN
        MOV     #FAC1,R3        ;ADDRESS FAC1(R5) IN R3
        ADD     R5,R3
        MOV     A1,(R3)
        BIS     B1,(R3)+        ;FAC1(R5) IS A1 (OR) B1
        MOV     A2,(R3)
        BIS     B2,(R3)         ;FAC2(R5) IS A2 (OR) B2
        MOV     #C,R0           ;ADDRESS C
        JSR     PC,STORE        ;STORE FAC1,FAC2 IN C
        RTS     PC
; "REV"  (A$,B$)
REVFN:  MOV     #TABLE,R0       ;ADDRESS ARG AREA
        JSR     PC,GETARG       ;CHECK SYNTAX AND GET ARGS
        .BYTE   3,4,0           ;(ARG TYPES)
        .EVEN
        CMP     (SP),#-1        ;CHECK NULL STRING
```

```
        BEQ     REVX
        CLR     R2
        MOV     (SP),R3
        BISB    (R3)+,R2        ;R2 IS STRING LENGTH
        CMPB    (R3)+,(R3)+     ;R3 ADDRESSES CHARS
REV1:   DEC     R2              ;ADDRESS NEXT PAIR OF BYTES
        MOV     R3,R0           ;TO SWITCH
        ADD     R2,R0
        CMP     R0,R3           ;CHECK DONE--REACHED MIDDLE
        BLOS    REVX
        MOVB    (R0),R1         ;EXCHANGE ANOTHER PAIR
        MOVB    (R3),(R0)       ;OF BYTES
        MOVB    R1,(R3)+
        DEC     R2
        BR      REV1
REVX:   MOV     #B$,R0          ;ADDRESS B$

        JSR     PC,SSTORE       ;STORE STRING ON STACK

        RTS     PC
;
; ARGUMENT AREA
TABLE:
A1:     .WORD   0               ;VALUE OF A (2 WORDS)
A2:     .WORD   0
B1:     .WORD   0               ;VALUE OF B (2 WORDS)
B2:     .WORD   0
C:      .WORD   0,0,0           ;ADDRESS OF C (3 WORDS)


;
.=TABLE
                                ;POINTER TO A$ IS ON STACK
B$:     .WORD   0,0,0           ;ADDRESS OF B$ (3 WORDS)


;
        .END
```

4.3 BACKGROUND ASSEMBLY LANGUAGE ROUTINES

BASIC provides for the execution of a "background" assembly language subroutine during its idle-time, that is, when it is waiting for terminal input. An example of such a background routine is one that displays data from an array on a CRT. The background routine must be defined in the file BASINT.PAL. To define a background routine named BKG in the BASINT Module this procedure is followed:

```
                    (S) represents the ALTMODE key
                    (Mount scratch cassette on unit 1)
                    (Mount CAPS-11 system cassette on unit 0)
        .Z 1:
        .R EDIT
        *EW1:BASINT.PAL$$

                    (Mount cassette with BASINT.PAL on unit 0)
        *ER0:BASINT.PAL (S) R (S)(S)
        *FBKGI: (S) I
                    .GLOBL  BKG
                    .WORD   BKG
        (S)(S)
        *EX (S)(S)
        .
```

The background source file should be in the following format.

```
        R0=%0
        R1=%1
        R2=%2
        R3=%3
        R4=%4
        R5=%5
        SP=%6
        PC=%7
        .GLOBL  BKG

BKG:    ;START OF BACKGROUND ROUTINE
        .
        .
        .
        RTS     PC

        .END
```

4.4 GENERAL INTERFACE SUBROUTINES

```
        ; GETARG, STORE, SSTORE :  SUBROUTINES FOR
        ; LINKAGE OF ASSEMBLER SUBROUTINES TO BASIC
        ;
                .TITLE  GETARG  29-AUG-73
                .GLOBL  GETARG, STORE
                .GLOBL  EVAL, GETVAR, ERRARG, ERRSYN
                .GLOBL  .LPAR, .COMMA, .RPAR, .EOL
                .GLOBL  STOVAR, .SQUOT, .DQUOT
                .IFNDF  $NOSTR
                .GLOBL  SSTORE, STOSVAR
                .ENDC   ;$NOSTR
                .CSECT
        ;
        ;$NOSTR =        1       ;DELETE ";" TO ASSEMBLE FOR
        ;                        ;BASIC WITH NO STRINGS
        R0=%0
        R1=%1
        R2=%2
        R3=%3
        R4=%4
        R5=%5
        SP=%6
        PC=%7
        NVAL=4
                .IFDF   $NOSTR
        NVAL=3
                .ENDC   ;$NOSTR
        .TEXT=377
        FAC1=40
        FAC2=42
        VARSAV=22
```

```
;-------------------------------------------------------------
; SUBROUTINE "GETARG"   CALLED BY MOV  #TABLE,R0
;                                 JSR  PC,GETARG
;                                 .BYTE N1,N2,...,0
;                                 .EVEN
;                       WHERE TABLE IS THE ADDRESS OF A
;                       TABLE TO HOLD THE ARG REFERENCES.
;                       N1,N2,ETC. INDICATE THE ARG TYPES:
;                       1       INPUT NUMERIC EXPRESSION. 2
;                               (THE EXPRESSION VALUE) ARE
;                               STORED IN TABLE.
;                       2       OUTPUT NUMERIC VARIABLE. 3 WORDS
;                               ARE STORED IN TABLE.
;                       STRING VERSION ONLY:
;                       3       INPUT STRING EXPRESSION. NO WORDS
;                               ARE STORED IN TABLE. THE STRING
;                               POINTER IS ON THE STACK.
;                       4       OUTPUT STRING VARIABLE. 3 WORDS
;                               ARE STORED IN TABLE.
;                       NO STRING VERSION:
;                       3       INPUT STRING LITERAL. 2 WORDS
;                               ARE STORED IN TABLE. WORD 1 CON-
;                               TAINS THE START OF THE ASCII STRING.
;                               WORD 2 CONTAINS THE LENGTH OF THE
;                               STRING IN BYTES.
;                       CHECKS THE SYNTAX OF THE CALLING
;                       STATEMENT AND FINDS THE REQUESTED
;                       ARGUMENT REFERENCES, STORING THEM
;                       CONSECUTIVELY IN TABLE.
GETARG: MOV     (SP)+,R3        ;ADDR OF CALL IN R3
        MOVB    (R3)+,R2        ;GET 1ST BYTE IN R2
        BLE     GETX            ;NO ARGS, EXIT
        CMPB    (R1)+,#.LPAR    ;CHECK STARTING "("
        BNE     GETERS          ;NO, SYNTAX ERROR
        BR      GET2            ;ENTER LOOP
GET1:   CMPB    (R1)+,#.COMMA   ;CHECK "," BETWEEN ARGS
        BNE     GETERS          ;NO, SYNTAX ERROR
GET2:   CMP     R2,#NVAL        ;CHECK VALID BYTE
        BHI     GETERA
        ASL     R2
        MOV     R0,R0S          ;SAVE REGS
        MOV     R3,R3S
        MOV     BRTAB-2(R2),PC  ;BRANCH TO ROUTINE
```

```
; NUMERIC EXPRESSION
NUMEXP: JSR     PC,EVAL         ;EVALUATE!
        BCS     GETERA          ;STRING IS BAD
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        MOV     FAC1(R5),(R0)+
        MOV     FAC2(R5),(R0)+  ;SAVE VALUE
        BR      NXTARG
; STRING EXPRESSION
STREXP:
        .IFNDF  $NOSTR
        JSR     PC,EVAL         ;EVALUATE!
        BCC     GETERA          ;NUMERIC IS BAD
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        BR      NXTARG
        .ENDC   ;$NOSTR
        .IFDF   $NOSTR
        MOVB    (R1)+,-(SP)     ;LOOK FOR STRING LITERAL
        CMPB    (SP),#.SQUOT    ;CHECK QUOTE CHAR.
        BEQ     STR1
        CMPB    (SP),#.DQUOT
        BNE     GETERS
STR1:   CMPB    (R1)+,#.TEXT    ;CHECK .TEXT TOKEN NEXT
        BNE     GETERS
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        MOV     R1,(R0)+        ;SAVE STRING ADDRESS IN TABLE
        CLR     R2              ;NOW FIND LENGTH
STR2:   TSTB    (R1)+           ;END OF STRING IS BYTE 00
        BEQ     STR3
        INC     R2              ;COUNT
        BR      STR2
STR3:   MOV     R2,(R0)+        ;SAVE LENGTH IN TABLE
        CMPB    (SP)+,(R1)+     ;CHECK MATCHING CLOSE QUOTE
        BNE     GETERS
        BR      NXTARG
        .ENDC   ;$NOSTR
; NUMERIC TARGET VARIABLE
NUMVAR: CLR     -(SP)           ;REMEMBER IT'S NUMERIC
        .IFDF   $NOSTR
        BR      VAR1
; STRING TARGET VARIABLE
STRVAR: MOV     R2,-(SP)        ;REMEMBER IT'S STRING
        .ENDC   ;$NOSTR
VAR1:   MOVB    (R1)+,R2        ;GET SYNTAX REF IN R2
        BMI     GETERS
        SWAB    R2
        BISB    (R1)+,R2
        ADD     (R5),R2
        JSR     PC,GETVAR       ;ADDRESS VARIABLE
        MOV     R0S,R0          ;RESTORE TABLE POINTER
        MOV     R5,R2           ;ADDRESS VARSAV
        ADD     #VARSAV,R2
        MOV     (R2),R3         ;SAVE A COPY
        MOV     (R2)+,(R0)+     ;MOVE 3 WORDS INTO TABLE
        MOV     (R2)+,(R0)+
        MOV     (R2),(R0)+
        TST     (SP)+           ;STRING OR NUM
        BNE     VAR2
        CMP     (R3),#-1        ;NUMERIC, CHECK TYPE AGREES
        BEQ     GETERA
        BR      NXTARG
VAR2:   CMP     (R3),#-1
        BNE     GETERA
```

```
; GO TO NEXT ARGUMENT
NXTARG: MOV     R3S,R3
        MOVB    (R3)+,R2        ;GET NEXT BYTE IN R2
        BGT     GET1            ;LOOP TILL BYTE IS 0
        CMPB    (R1)+,#.RPAR    ;CHECK CLOSING ')'
        BNE     GETERS
GETX:   CMPB    (R1)+,#.EOL     ;AND END-LINE TOKEN
        BNE     GETERS
        INC     R3              ;MAKE SURE R3 IS EVEN
        ASR     R3
        ASL     R3
        JMP     (R3)
R0S:    .WORD   0
R3S:    .WORD   0
GETERA: JMP     ERRARG
GETERS: JMP     ERRSYN
BRTAB:  .WORD   NUMEXP
        .WORD   NUMVAR
        .WORD   STREXP
        .IFNDF  $NOSTR
        .WORD   STRVAR
        .ENDC   ;$NOSTR


;-----------------------------------------------------------
; SUBROUTINE 'STORE'    CALLED BY JSR PC,STORE
;                       R0 POINTS TO 3-WORD ARG REFERENCE
;                       SET UP BY GETVAR
;                       SAVES THE VALUE OF THE FAC
;                       IN THE SPECIFIED NUMERIC VARIABLE
STORE:  MOV     R5,R2           ;ADDRESS VARSAV
        ADD     #VARSAV,R2
        MOV     (R0)+,(R2)+     ;MOVE FROM TABLE TO USER AREA
        MOV     (R0)+,(R2)+
        MOV     (R0),(R2)
        JSR     PC,STOVAR       ;STORE IT
        RTS     PC
;
        .IFNDF  $NOSTR
;-----------------------------------------------------------
; SUBROUTINE 'SSTORE'   CALLED BY JSR PC,SSTORE
;                       R0 POINTS TO 3-WORD ARG REFERENCE
;                       SET UP BY GETVAR
;                       STRING POINTER IS AT THE TOP OF STK
;                       SAVES THE STRING AT TOP OF STK
;                       IN THE SPECIFIED STRING VARIABLE
SSTORE: MOV     R5,R2
        ADD     #VARSAV,R2      ;ADDRESS VARSAV
        MOV     (R0)+,(R2)+     ;MOVE FROM TBL TO USER AREA
        MOV     (R0)+,(R2)+
        MOV     (R0),(R2)
        MOV     (SP),R3         ;SWITCH RETURN & STRING PTR
        MOV     2(SP),(SP)
        MOV     R3,2(SP)
        JSR     PC,STOSVAR      ;STORE STRING
        RTS     PC              ;RETURN
        .ENDC   ;$NOSTR
        .END
```

CHAPTER 5

ERROR MESSAGES

When BASIC encounters an error, execution of the command or statement in error halts and an error message is printed. A fatal error message indicates that after printing the error message BASIC ends all execution and then returns to the READY message. At this point the condition causing the error message may be fixed and the program may be started from the beginning or a GOTO command may be used to continue execution (although files may have been re-opened in immediate mode before the GOTO command). Non-fatal error messages indicate that BASIC will allow the error condition to be fixed and then will continue execution.

All error messages are printed in one of the following formats

```
          message
or
          message AT LINE xxxxx
```

where xxxxx is the line number of the statement containing the error. Error messages produced in immediate mode statements or commands will not include AT LINE xxxxx. Non-fatal error messages do not include AT LINE xxxxx and do not return to READY.

Table 5-1 lists all BASIC-11 error messages. The message produced will be the abbreviation unless BASIC-11 has been assembled from the sources with longer error messages specified. All error messages are fatal unless the explanation specifies non-fatal.

Table 5-1
BASIC Error Messages

Abbreviation	Longer Message	Explanation
?ARG	ARGUMENT ERROR	Arguments in a function call do not match, in number or in type, the arguments defined for the function.
?ADC	Cannot issue ADC command while an ATS operation is underway *	
?ATL	ARRAYS TOO LARGE	Not enough memory is available for the arrays specified in the DIM statements. If the arrays can not be redimensioned to a smaller dimension, then reduce the size of the program by eliminating remarks or by using subroutines, user-defined functions, chaining or overlaying.
B?		Non-fatal, BASIC/CAPS system overlay file not found on cassette unit 0. Mount a new cassette and type a character to continue. Occurs only in 8K version.
BAD DATE		Non-fatal, an illegal date was typed during initial dialogue. Retype to continue.
?BDR	BAD DATA READ	Item input from DATA statement list by READ statement is bad.
?BOV	BAD OVERLAY FILE	Non-fatal, BASIC/CAPS system overlay file was found but contained an error. Also prints B? message. Retry or mount new cassette with overlay files. BASIC/CAPS may also be rebooted. Occurs only in 8K version.
?BRT	BAD DATA-RETYPE FROM ERROR	Non-fatal, item entered to input statement was bad. Retype and program will continue.
?BSO	BUFFER STORAGE OVERFLOW	Not enough room available for file buffers.
?BUF	Buffer name given in LPS command has not been previously defined in a USE statement. *	
?DCE	DEVICE CHANNEL ERROR	The device channel number, also called the logical unit number, specified in OPEN statement has been previously opened or is out of range (1-7).

* PAGE I-16 "BASIC - RT11" HANDBOOK.

Table 5-1 (Cont.)
BASIC Error Messages

Abbreviation	Longer Message	Explanation
?DNR	DEVICE NOT READY	A hardware I/O device referred to by an OLD, SAVE, or PRINT command is not on-line or the file does not contain any legal BASIC program lines.
?DV0	DIVISION BY 0	Program attempted to divide some quantity by 0.
?ETC	EXPRESSION TOO COMPLEX	The expression being evaluated caused the stack to overflow usually because the parentheses are nested too deeply. The degree of complexity that produces this error varies according to the amount of space available in the stack at the time. Breaking the statement up into several simpler ones eliminates the error.
?FIO	FILE I/O ERROR	A hardware I/O error occurred. All files are automatically closed.
?FNF	FILE NOT FOUND	The file requested was not found on the specified device. May be caused by an OPEN FOR INPUT, OVERLAY, or CHAIN statement or an OLD, APPEND, RUN[NH] file descriptor, or REPLACE command.
?FNO	FILE NOT OPEN	The logical unit number specified is not associated with an open file. May be caused by PRINT #, INPUT #, or CLOSE statements.
?FRM	FORMAT ERROR	Format string error occurred in PRINT USING statement or an attempt was made to print string in numeric field or vice versa.
?FSE	FILE SPECIFICATION ERROR	The file descriptor contained an illegal device or character. Legal characters are A through Z and 0 through 9.
?FWN	FOR WITHOUT NEXT	The program contains a FOR statement without a corresponding NEXT statement to terminate the loop.
?GND	GOSUBS NESTED TOO DEEPLY	Program GOSUB nested to more than 20 levels.
?IDF	ILLEGAL DEF	The DEFine function statement contains an error.
?IDM	ILLEGAL DIM	In either a DIMENSION or a COMMON statement, subscript is not an integer number or array has been dimensioned previously.

Table 5-1 (Cont.)
BASIC Error Messages

Abbreviation	Longer Message	Explanation
?ILN	ILLEGAL NOW	Execution of INPUT statement was attempted in immediate mode.
?ILR	ILLEGAL READ	A write-only device was opened for input or an attempt was made to input from a file opened for output.
?LTL	LINE TOO LONG	The line being typed is longer than 120 characters; the line buffer overflows.
?NBF	NEXT BEFORE FOR	The NEXT statement corresponding to a FOR statement precedes the FOR statement.
?NER	NOT ENOUGH ROOM	Cassette is full and there is not enough room to open file. May be caused by an OPEN FOR OUTPUT statement or a SAVE or REPLACE command.
?NOR	Number out of range*	
?NPR	NO PROGRAM	The RUN command has been specified, but no program has been typed in.
?NSM	NUMBERS AND STRINGS MIXED	String and numeric variables may not appear in the same expression, nor may they be set equal to each other; for example, A$=2.
OFFLINE{0,1}?		Non-fatal, cassette unit indicated was offline. Cassette should be mounted and operation will continue.
?OFFLINE{0,1}		Fatal cassette error, occurs when cassette becomes offline after I/O has started.
?OFO	OUTPUT FILE OVERFLOW	Reached end of cassette while outputting to file.
?OOD	OUT OF DATA	The data list was exhausted and a READ requested additional data.
?OVF	OVERFLOW	The result of a computation is too large for the computer to handle.
?PTB	PROGRAM TOO BIG	The line just entered caused the program to exceed the user code area. Reduce program size by eliminating remarks and through use of subroutines, user-defined functions, overlaying, and chaining.
?PWF	POWER FAIL	A power fail interrupt occurred while the specified program line was executing. All files are closed.
?RBG	RETURN BEFORE GOSUB	A RETURN was encountered before execution of a GOSUB statement.

* PAGE I-16 "BASIC-RTII" HANDBOOK

Table 5-1 (Cont.)
BASIC Error Messages

Abbreviation	Longer Message	Explanation
?RPL	USE REPLACE	File saved already existed on device. Caused by SAVE command.
?SOB	SUBSCRIPT OUT OF BOUNDS	The subscript computed is greater than 32,767 or is outside the bounds defined in the DIM statement.
?SSO	STRING STORAGE OVERFLOW	Not enough memory is available to store all the strings used in the program.
?STL	STRING TOO LONG	The maximum length of a string in a BASIC statement is 255 characters.
?SYN	SYNTAX ERROR	The program has encountered an unrecognizable statement. Common examples of syntax errors are misspelled commands, unmatched parentheses, and other typographical errors.
?TIMING{0/1}		Cassette hardware timing error.
?TLT	LINE TOO LONG TO TRANSLATE	Lines are translated as entered and the line just entered exceeds the area available for translation.
?UFN	UNDEFINED FUNCTION	The function called was not defined by the program or was not loaded with BASIC or there was a syntax error in the first keyword on a line and BASIC translated the line as an implied CALL statement.
?ULN	UNDEFINED LINE NUMBER	The line number specified in an IF, GO TO or GOSUB statement does not exist anywhere in the program.
?WLO	WRITE LOCKOUT	Tried to write on a file opened for input or tried to open for output a read-only device.
WRT LOCK{0/1}?		Non-fatal, cassette on unit indicated was write-locked. Write enable cassette and continue (removing cassette to write enable it will cause the non-fatal OFFLINE message).

Table 5-1 (Cont.)
BASIC Error Messages

Abbreviation	Longer Message	Explanation
?WRT LOCK	{0 / 1}	Cassette on unit indicated was write-locked while I/O was in progress.
?↑ER	↑ ERROR	The program tried to compute the value A↑B, where A is less than 0 and B is not an integer. This produces a complex number which is not represented in BASIC.

When the message ?DNR AT LINE xxxxx is printed because the device referred to is not on-line, turn the device on and issue a GO TO xxxxx statement. Execution of the program resumes at the line (xxxxx) specified. This message may also indicate that a program file does not contain any legal BASIC program lines.

When the message ?OOD AT LINE xxxxx is printed because the file referred by an INPUT# statement is not ready, prepare the file and issue a GO TO statement to resume execution.

Function Errors

The following errors can occur when a function is called improperly.

?ARG	The argument used is the wrong type. For example, the argument was numeric and the function expected a string expression.
?SYN	The wrong number of arguments was used in a function, or the wrong character was used to separate them. For example, PRINT SIN(X,Y) produces a syntax error.

In addition, the functions give the errors listed below.

FNa(...)	?UFN	The function a has not been defined (function cannot be defined by an immediate mode statement).
	?SYN	A syntax error has been found in the expression in the DEF statement which defined the function. This error message is produced by statements evaluating user - defined functions, not by the statement defining the function.

RND or RND(expr)		No errors
SIN(expr)		No errors
COS(expr)		No errors
PI	?SYN	An argument was included
SQR(expr)	?ARG	Expression is negative
ATN(expr)		No errors
EXP(expr)	?↑ER	Expression is greater than 87
LOG(expr)	?ARG	Expression is negative or 0
LOG10(expr)	?ARG	Expression is negative or 0
ABS(expr)		No errors
INT(expr)		No errors
SGN(expr)		No errors
TAB(EXPR)	?ARG	Expression is not in the range 0<x<256
LEN(string expr)		No errors
ASC(string expr)	?ARG	String expr is not a string of length 1
CHR$(expr)	?ARG	Expression is not in the range 0<x<256
POS(string expr1,string expr2,expr)		No errors
SEG$(string expr,expr1,expr2)		No errors
VAL(string expr)	?ARG	String expr is not a valid number
STR$(expr)		No errors
TRM$(string expr)		No errors
BIN(string expr)	?ARG	Character other than blank, 0 or 1 in string
OCT(string expr)	?ARG	Character other than blank or 0 through 7 in string

APPENDIX A

BASIC/CAPS SYSTEM INFORMATION

A.1 LINKING BASIC/CAPS FROM THE OBJECT MODULES

This section contains instructions to link BASIC/CAPS from the object modules. The object cassettes contain the following files:

BASICR.OBJ FPMP .OBJ BASICE.OBJ BASICX.OBJ BASICS.OBJ BASINT.OBJ FTBL .OBJ BASICH.OBJ	BASIC OBJECT MODULES FOR THE GREATER THAN 8K VERSION
FPMPEA.OBJ FPMPEI.OBJ FPMPFP.OBJ	SPECIAL MATH PACKAGES FOR OPTIONAL HARDWARE (ONLY USED WITH GREATER THAN 8K VERSION)
BASING.OBJ FTBLG .OBJ PERVEG.OBJ GTNLPS.OBJ GTB .OBJ GTC .OBJ	OBJECT MODULES FOR GT40 DISPLAY SUPPORT.
BASRO .OBJ FPMP .OBJ BASE1 .OBJ BASE2 .OBJ BNOX .OBJ BASSO .OBJ BASINT.OBJ FTBL .OBJ BASHO .OBJ BASPT .OBJ	BASIC OBJECT MODULES FOR THE 8K VERSION

```
BASINL.OBJ
FTBLL .OBJ      OBJECT MODULES FOR LPS SUPPORT
PERVEC.OBJ
LPS0  .OBJ
LPS1  .OBJ
LPS2  .OBJ
LPS2C .OBJ
LPS3  .OBJ
LPS4  .OBJ
```

In addition, the following source files are provided:

```
PERPAR.PAL
PERVEC.PAL      FOR CUSTOMIZING LPS AND GT SUPPORT
BASINT.PAL
FTBL  .PAL
```

A.1.1 Linking BASIC/CAPS for Systems With More Than 8K of Memory

First, zero a scratch cassette; this cassette is used to hold the load module of BASIC/CAPS. Mount the BASIC object module cassette containing BASICR.OBJ on unit 1 and run the CAPS-11 Linker, using the following command string:

```
.R LINK
*BASIC.SLO/P,TT:=1:BASICR,FPMP/F,BASICE/F,BASICX/F,BASICS/F/B:400/C

,BASINT/F,FTBL/F,BASICH/F
```

When the Linker prompts with '0?' on PASS2, dismount the CAPS-11 system cassette from unit 0, mount the zeroed scratch cassette and type a character on the keyboard. When the Linker is finished, it prints an '*' to indicate it is ready for a new command string.

A.1.1.1 Linking with a User Function and/or a Background Assembly Language Routine

To link an assembly language user function and/or a background routine with BASIC, follow the instructions given in Chapter 4 for editing the BASINT and FTBL modules and creating the modules for the user function and/or background routine. The procedure given in Chapter 4 will produce a cassette with the following modules:

BASINT.OBJ	(From BASINT.PAL, edited if necessary to show the presence of a Background routine)
FTBL.OBJ	(From FTBL.PAL, edited if necessary to show the presence of a user function)

The cassette also contains some or all of the following modules:

GETARG.OBJ	(For use with user function if one is present)
FUN.OBJ	(User function module)
BKG.OBJ	(Background assembly language routine)

At this point follow the general procedure for linking BASIC for systems greater than 8K, but the second line of the linker command string should be;

For User Function and No Background Routine

```
,BASINT/P,FTBL/F,GETARG/F,FUN/F,BASICH/P
```

For User Function and Background Routine

```
,BASINT/P,FTBL/F,GETARG/F,FUN/F,BKG/F,BASICH/P
```

For Background Routine Only

```
,BASINT/P,FTBL/F,BKG/F,BASICH/P
```

All of these commands cause additional prompt messages as follows:

1?	(Mount cassette containing OBJ modules of BASINT,FTBL,GETARG, etc. on unit 1)
1?	(Mount BASIC object cassette containing BASICR.OBJ on unit 1)
PASS 2	(No action required)
0?	(Mount zeroed scratch cassette on unit 0)
1?	(Mount cassette containing OBJ modules of BASINT,FTBL,GETARG, etc. on unit 1)
1?	(Mount cassette containing BASICR.OBJ on unit 1)
*	(Linker is done)

A.1.2 Linking BASIC/CAPS for Systems with 8K of Memory

First, zero a scratch cassette; this cassette is used to hold the load module of BASIC/CAPS. Mount the BASIC object module cassette containing BASRO.OBJ on unit 1 and run the CAPS-11 Linker, using the following command string:

```
.R LINK
*BAS8K.SLO/P,TT:=1:BASRO,FPMP/F,BASE1/F,BASE2/F,BNOX/F,BASSO/F/B:400/C

,BASINT/F,FTBL/F,BASHO/F,BASPT/F
```

When the Linker prompts with '0?' on PASS2, dismount the CAPS-11 system cassette from unit 0, mount the zeroed scratch cassette and type a character on the keyboard. When the Linker is finished, it prints an '*' to indicate it is ready for a new command string.

A.1.2.1 Linking With a User Function and/or a Background Assembly Language Routine - This procedure is identical to that given for systems with more than 8K of core, except follow the general procedure for linking 8K systems and the second line of the command string should be:

For User Function and No Background Routine

```
,BASINT/P,FTBL/F,GETARG/F,FUN/F,BASHO/P,BASPT/F
```

For User Function and Background Routine

```
,BASINT/F,FTBL/F,GETARG/F,FUN/F,BKG/F,BASHO/P,BASPT/F
```

For Background Routine Only

```
,BASINT/P,FTBL/F,BKG/F,BASHO/P,BASPT/F
```

The prompt messages are identical to those for systems greater than 8K, but the BASIC object cassette used is the one containing BASRO.OBJ, not the one with BASICR.OBJ.

A.2 ASSEMBLING BASIC

BASIC/CAPS is also available separately in source form and may be customized and then assembled into object modules. This section contains all assembling instructions for BASIC/CAPS. A 16K system is required to assemble BASIC. After assembling BASIC/CAPS, the object modules should be linked as described in section A.1.

NOTE

The 8K version of BASIC cannot be assembled from the sources.

BASIC/CAPS may be purchased separately in source form. The thirteen source files are:

BASICR.PAL	Kernel module
BASICE.PAL	Edit module
BASX1.PAL	Execute module 1
BASX2.PAL	Execute module 2
BASICS.PAL	System dependent module
BASH1.PAL	High module 1
BASH2.PAL	High system dependent module 2
BASH3.PAL	High module 3
FPMPA.PAL	Floating Point math package Module 1
FPMPB.PAL	Floating Point math package Module 2
BASP1.PAL	Parameter module 1
BASP2.PAL	Parameter module 2
FTBL.PAL	Function Table module
BASINT.PAL	Interface module

The following modules are also supplied. These are sources for the BASIC/CAPS special device support:

LPS0.PAL	Laboratory Peripheral System module 0
LPS1.PAL	LPS module 1
LPS2.PAL	LPS module 2
LPS3.PAL	LPS module 3
LPS4.PAL	LPS module 4
GTNLPS.PAL	GT40 display module
GTB1.PAL	GT40 display module
GTB2.PAL	GT40 display module
GTC.PAL	GT40 display module
PERVEC.PAL	Vector address module (in binary kit)
PERPAR.PAL	Parameter module (in binary kit)

The BASIC sources may be assembled and linked to create a standard BASIC, i.e., including strings, PRINT USING statement, a line printer, high-speed paper tape reader/punch, and a power fail restart. Alternatively, a file BASPAR.PAL may be created with EDIT to include any of the following parameters; these parameters govern the assembly of BASIC/CAPS and allow the user to produce a customized version of BASIC/CAPS.

Parameter	Default Value	Description
$NOSTR	undefined	Define to eliminate code for string variables, e.g., $NOSTR = 1.
$NOPRU	undefined	Define to eliminate code for PRINT USING statement.
$NOPOW	undefined	Define to assemble without the power fail/restart routine.
$LONGER	undefined	Define to get long, explanatory error messages.
$NOPTP	undefined	Define to eliminate code for high-speed paper tape reader/punch.
$NOLPT	undefined	Define to eliminate code for the line printer.
$STKSZ	300 (octal)	Change value to adjust size of the program stack, in bytes, e.g., $STKSZ = 400 (must be greater than or equal to 300).
$PPBSZ	30 (octal)	Change value to adjust paper tape punch buffer size (bytes).
$PRBSZ	30	Change value to adjust paper tape reader buffer size (bytes).
$LPBSZ	40	Change value to adjust line printer buffer size (bytes).
$TPBSZ	20	Change value to adjust terminal teleprinter buffer size (bytes).
$KBBSZ	20	Change value to adjust keyboard buffer size (bytes).

A.2.1 Assembling the Standard Object Modules

The following procedure will produce BASIC/CAPS object modules that are equivalent to the object modules provided in the BASIC kit. These object modules can be linked to create the greater than 8K version of BASIC/CAPS. The linking instructions are provided in section A.1.1. The 8K version can not be assembled from the sources.

Mount the CAPS-11 System Cassette write-locked on unit 0 and bootstrap the CAPS-11 Monitor.

When the Monitor is loaded and responds with a dot, enter the date, then mount a scratch cassette on unit 1, write-enabled, and zero it by typing:

```
.Z 1:<CR>
```

Now run the assembler (must be a 16K assembler!) by typing:

```
.R PAL<CR>
```

When the command string Interpreter types an asterisk, enter the following command string:

```
*1:BASICR=0:BASP1/P,BASP2/F,BASICR/P<CR>
```

This command causes several prompt messages to be printed on the console terminal. The following list of prompt messages tells what action to take for each prompt:

PROMPT	USER SHOULD
0?	Dismount system cassette from unit 0. Mount BASIC source cassette containing BASP1.PAL and BASP2.PAL on unit 0. Type <CR>.
0?	Dismount cassette containing BASP1.PAL from unit 0. Mount BASIC source cassette containing BASICR.PAL on unit 0. Type <CR>.
PASS 2	(No action required)
0?	Dismount cassette containing BASICR.PAL from unit 0. Mount cassette containing BASP1.PAL and BASP2.PAL on unit 0. Type <CR>.
0?	Dismount cassette containing BASP1.PAL from unit 0. Mount cassette containing BASICR.PAL on unit 0. Type <CR>.
000000 ERRORS	(Assembly is done)
*	(Ready for next command string)

Following is a list of the remaining Assembler command strings required to assemble the sources of BASIC/CAPS. They should be entered in the order listed; the prompt messages and required actions follow the pattern of the first command string.

```
*1:FPMP=0:BASP1/P,BASP2/F,FPMPA/P,FPMPB/P<CR>

*1:BASICE=BASP1/P,BASP2/F,BASICE/P<CR>

*1:BASICX=BASP1/P,BASP2/F,BASX1/P,BASX2/P<CR>

*1:BASICS=BASP1/P,BASP2/F,BASICS/P<CR>

*1:BASINT=BASINT/P<CR>

*1:FTBL=FTBL/P<CR>

*1:BASICH=BASP1/P,BASP2/F,BASH1/P,BASH2/F,BASH3/F<CR>
```

A.2.2 Assembling Non-standard Object Modules

The following example builds a load module of BASIC for a PDP-11 cassette system with no line printer, no high-speed paper tape reader/punch and no power-fail restart option.

Mount the system cassette on unit 0 and load the monitor, if it is not already resident in memory. Mount a scratch cassette on unit 1, zero the scratch cassette, run EDIT, and create the parameter file by typing:

Ⓢ Represents ALTMODE Key

```
.Z 1:
.R EDIT
*EW1:BASPAR.PAL ⓈⓈ
*I$NOPOW=1
$NOPTP=1
$NOLPT=1
Ⓢ EX ⓈⓈ
.
```

Dismount the cassette on unit 1. Mount a second scratch cassette on unit 1, zero it, and run the assembler by typing the following:

```
.Z 1:
.R PAL
*1:BASICR=BASPAR/P,BASP1/P,BASP2/F,BASICR/P
```

This PAL command produces the following prompt messages:

Prompt	User Should
0?	Mount cassette containing BASPAR.PAL on unit 0, type any character.
0?	Mount cassette containing BASP1.PAL and BASP2.PAL on unit 0, type any character.
0?	Mount cassette containing BASICR.PAL on unit 0, type any character.

Prompt	User Should
PASS 2	
0?	Mount cassette containing BASPAR.PAL on unit 0, type any character.
0?	Mount cassette containing BASP1.PAL and BASP2.PAL on unit 0, type any character.
0?	Mount cassette containing BASICR.PAL on unit 0, type any character.
000000 ERRORS	
	(Assembly is done)
*	(Ready for next command string)

Type the following commands. For each "/P" switch the PAL assembler prints a "0?" on each pass and the user should mount the appropriate cassette for each prompt.

```
*1:FPMP=BASPAR/P,BASP1/P,BASP2/F,FPMPA/P,FPMPB/P<CR>

*1:BASICE=BASPAR/P,BASP1/P,BASP2/F,BASICE/P<CR>

*1:BASICX=BASPAR/P,BASP1/P,BASP2/F,BASX1/P,BASX2/P<CR>

*1:BASICS=BASPAR/P,BASP1/P,BASP2/F,BASICS/P<CR>

*1:BASINT=BASPAR/P,BASP1/P,BASP2/F,BASINT/F<CR>

*1:FTBL=BASPAR/P,BASP1/P,BASP2/F,FTBL/F<CR>

*1:BASICH=BASPAR/P,BASP1/P,BASP2/F,BASH1/P,BASH2/F,BASH3/F<CR>

*
```

When the PAL assembler prints the last "*" all the object modules are on the cassette on drive 1. These object modules must be linked into a load module (See section 4.1.1).

A.3 TECHNICAL DESCRIPTION OF BASIC/CAPS

A.3.1 System Routines in BASIC

The routines described below are all global symbols and are available to the user functions:

Routine Name (Global)	Call	Description
BOMB	TRAP 0 .ASCII 'MESSAGE' .EVEN	This routine stops execution of the BASIC program and types the message: ?MESSAGE AT LINE**** If the $LONGER option is specified when BASIC is assembled from the sources, the '?' character is omitted. BASIC then types the READY message.

Routine Name (Global)	Call	Description
BKGI	MOV @#BKGI,R0 JSR PC,@R0	Address of assembly language background routine.
ERRPDL	JMP ERRPDL	Called when the stack pointer (SP) goes below the value in R4. Causes execution to halt and types out %ETC AT LINE xxxxx. There are 20 extra "buffer" words on the stack. If the user routine will definitely not use more than this many words on the stack, the routine need not check for a stack overflow.
ERRSYN	JMP ERRSYN	Syntax error. Stops execution and prints out ?SYN AT LINE xxxxx.
ERRARG	JMP ERRARG	Argument error. Stops execution and prints out ?ARG AT LINE xxxxx.
EVAL	JSR PC,EVAL	Evaluate expression. R1 points to the start of the expression in the code. EVAL sets the carry bit as follows: carry = 0: The expression is numeric. The value of the expression is contained in the floating accumulator (FAC1 and FAC2). carry = 1: A string expression. If the string is non-null, the top of the stack is an indirect pointer to the string. (See section 8.6 for the format of string variables.) If the string is null, the top of the stack is the value 177777. In both cases, R1 is moved to point to the byte following the expression in the code. If it detects an error in the expression, EVAL branches to the appropriate error routine.
FTABI	MOV @#FTABI,R0	Contains address of the system function table described in Section 5.1. Function table may be examined by assembly language routine to find addresses of other assembly language routines.

Routine Name (Global)	Call	Description

GETVAR	JSR PC,GETVAR	Address variable or array element. R2 contains the address of the symbol table entry for the variable. GETVAR looks up and saves the address of the variable reference, so that a subsequent STOVAR or STOSVAR will store a value in the addressed variable. GETVAR destroys the FAC when addressing an array element; R1 is left unchanged. To address the symbol table entry, precede the GETVAR call with the code:
		`MOVB (R1)+,R2 ;FIRST BYTE OF ;OFFSET`
		`BMI ESYN ;IF NEGATIVE, ERROR`
		`SWAB R2`
		`BISB (R1)+,R2 ;GET 2ND HALF OF ;OFFSET`
		`ADD (R5),R2 ;ADD BASE OF SYMBOL ;TABLE`

MSG	JSR R1,MSG .ASCII 'MESSAGE' .BYTE 0 .EVEN	Print message on console. Prints the ASCII characters specified after the JSR instruction up to the 0-byte. MSG prints only those characters specified in the calling sequence plus padding characters specific to the terminal in use. The calling program must insert a carriage return where required. MSG clears the CTRL/O condition.
NUMSGN	JSR PC,NUMSGN .WORD ROUTINE	This subroutine converts the number contained in the FAC to ASCII, and saves it via the specified ROUTINE. The ROUTINE is called by a "JSR PC" instruction and preserves all register contents.
STOVAR	JSR PC,STOVAR	Store numeric variable. Stores the FAC in the variable or array element last referenced by GETVAR. If it was a string variable, STOVAR stops execution of the program and produces the ?NSM error message.
STOSVAR	JSR PC,STOSVAR	Store string variable. Stores the top of the stack in the variable or array element last referenced by GETVAR, and pops one word from the stack. If it was a numeric variable, STOSVAR stops execution of the program and produces the %NSM error message:

Routine Name (Global)	Call	Description
INT	JSR PC,INT	Integerize the FAC. Sets the value of the FAC to the greatest integer contained in the previous contents of the FAC. The number is expressed in the BASIC integer format if possible.
MAKEST	JSR PC,MAKEST	Make non-null string variable. The top of the stack contains the length of the string to be created. R2 must point to a word which contains 3 less than the address of the first of the characters to fill the string. MAKEST returns an indirect pointer to the string on the top of the stack. (Called MAKESTR in sources.) To create a string "ABCD" the following code could be used:

```
        MOV #4,-(SP)        ;LENGTH
                            ;TO TOP OF
                            ;STACK

        MOV #STRING-3,R2    ;ADDRESS
        JSR PC,MAKEST       ;OF FIRST
                            ;CHAR -3

        MOV (SP)+,R2        ;R2 NOW
             .              ;CONTAINS
             .              ;ADDRESS
             .              ;OF BASIC
                            ;STRING
STRING: .ASCII /ABCD/
```

In addition, the user program may call the following FPMP-11 routines, which are documented in the FPMP-11 User's Manual (DEC-11-NFPMA-A-D).

$POLSH	Enter "Polish Mode"
$IR	Integer-to-Real Conversion
$MLR	Multiply Real
$DVR	Divide Real
$ADR	Add Real
$SBR	Subtract Real
SIN	Sine Function
COS	Cosine Function
SQRT	Square Root Function
ALOG	Logarithm Function (Base e)
ATAN	Arctangent Function
EXP	Exponentiation Function

The following list contains all the .GLOBL symbols available to the user's assembly language routines in this release of BASIC.

GLOBL Symbol	Description
BKGI	Address of the background routine
BOMB	Error routine, called by TRAP 0
ERRARG	Argument error
ERRPDL	Stack overflow error
ERRSYN	Syntax error
EVAL	Evaluate expression
FTABI	Address of the system function table
GETVAR	Address variable
INT	Integerize floating accumulator
MAKEST	Create a string
MSG	Print a message on the terminal
NUMSGN	Convert from numeric to ASCII
STOSVAR	Store string variable
STOVAR	Store numeric variable
.COMMA	Comma token ,
.DQUOT	Double quote token "
.EOL	End-line token \
.LPAR	Left-parenthesis token (
.RPAR	Right-parenthesis token)
.SQUOT	Single quote token '

The offset of system variables in the "user area", which starts at the address contained in R5, will not change from release to release; if new ones are added, they will be inserted at the end of the user's area. Therefore, these values may be set by MACRO equate statements in the user's source program (e.g., FAC1 = 40). The most commonly-used user area offsets are described below.

User area offset			Description
SYMBOLS	=	0	Address of symbol table
CODE	=	16	Address of stored program
LINE	=	20	Address of input line buffer
VARSAVE	=	22	Saved symbol table entry address
SS1SAVE	=	24	Saved first array subscript
SS2SAVE	=	26	Saved second array subscript
LINENO	=	30	Line number being executed
FAC1	=	40	Floating accumulator, upper word
FAC2	=	42	Floating accumulator, lower word
PROGNM	=	112	Program name, 6 ASCII bytes

A.3.2 Representation of Numbers in BASIC

The value stored in the floating accumulator (FAC1(R5) and FAC2(R5)) by EVAL is always two words long: FAC1(R5) contains the high-order, and FAC2(R5), the low-order portion. If FAC1(R5) is non-zero, then the number is stored as a 2-word floating-point number, in this format:

Word	Bit(s)	Description
FAC1(R5)	15	Sign bit, set if the number is negative.
	14-7	Exponent, with a bias of 200 octal.
	6-0	The second through eighth significant bits of mantissa. The first significant bit is always an assumed 1.
FAC2(R5)	15-0	The 9th through 24th significant bits of mantissa.

If FAC1(R5) is zero then FAC2(R5) contains the integer value of the number in 2's complement form. Note that the integers from -32,768 to +32,767 do not have a unique representation: they may be stored in the floating-point or integer form. For example, the number represented by:

```
FAC1:     40640     ;Floating-point "5"
FAC2:     0
```

has the same value as:

```
FAC1:     0
FAC2:     5         ;Integer "5"
```

The subroutine INT, described in section 3.2.6 of the PDP-11 BASIC Language Reference Manual converts a number from the floating point representation to an integer.

A.3.2.1 Representation of Strings in BASIC - Non-null strings are represented as follows:

Byte(s)	Contents
0	The length of the string, N
1 and 2	An internal "back-pointer" used by BASIC. Do not change this value.
3x0(2+N)	The ASCII characters of the string
3+N	The length of the string, N

A null string is not stored in BASIC; rather, the indirect pointer to the string has the value 177777.

A.3.3 Format of Translated BASIC Program

When the user inputs a BASIC program, the BASIC system does not store the program exactly as it is typed or read from the input file. Instead, it translates the program to an intermediate form which can be used in two different ways. The intermediate code can be "un-translated" by the LIST or SAVE commands to produce an ASCII program which looks very similar to the input program, or the translated code can be very quickly interpreted by the RUN command to provide swift execution of a program under BASIC/RT-11.

A.3.3.1 Symbol Table Format - As the BASIC program is input, the system builds a symbol table in memory at the indirect address 0(R5). There are four different types of symbol table entries, as shown in Table A-1.

Table A-1
Symbol Table Entries

Symbol Table Definition	Description
Line Number	This entry is two words long, with this format: Word 1: Line number as an unsigned 16-bit integer. The highest number allowed is 77777 octal or 32,767 decimal. Word 2: The address of the specified line in the stored translated program.
Numeric Scalar	This is five words long, with this format: Word 1: Constant 177775 Word 2: High-order Scalar Value Word 3: Low-order Scalar Value Word 4: Constant 0 Word 5: ASCII scalar name, the second byte is 0 if the name is only one character.
Numeric Array	This entry is five words long, with this format: Word 1: Constant 177776 Word 2: Address of array Word 3: Maximum value of first subscript (SS1MAX below) Word 4: Maximum value of second subscript or -1 if the array is singly-dimensioned Word 5: ASCII array name The scalar with the same name as an array is stored internally as the first element of the array. The address of the array is actually the address of this element. The arrays are stored with the first subscript varying the fastest; each element of the array takes up two words. The address of the (M,N) element in the array is the array address plus the quantity: 4*(N*SS1MAX+M+1)

Table A-1 (Cont.)
Symbol Table Entries

Symbol Table Definition	Description
String	This entry is five words long, with this format: Word 1: Constant 177777 Word 2: Array Address, or string pointer, if Word 3=-1 Word 3: Maximum value of first subscript (SS1MAX below), or -1 if not a string array Word 4: Maximum value of second subscript, or -1 if the array is singly-dimensioned or scalar Word 5: ASCII string name, with the '$' character omitted Strings and string arrays are stored as 1-word pointers to the strings, or the flag 177777 for a null string. If a string is dimensioned or used as a string array, the scalar string with the same name is stored as the first entry in an array. Otherwise, the pointer to the scalar string is stored directly in the symbol table entry, as indicated above. The address of the pointer to the (M,N) element in the array is then the array address plus the quantity: 2*(N*SS1MAX+M+1)

A.3.3.2 Translated Code - After the line is input, the TRAN subroutine is called to translate it to the internal format. TRAN scans the input line from left to right, and translates it as described below.

All references to line numbers or variable names are stored as the two-byte offset into the symbol table of the entry for that variable name. The symbol table entries for all numeric variables are initially scalars, and are changed to dimensioned arrays when the RUN statement is executed. This two-byte offset is, of course, not negative; therefore, it may be distinguished from the "keyword tokens" described below. It is not necessarily aligned to a word boundary.

All sequences of characters used as a single unit by the BASIC language are defined as "Keywords". The following are examples of keywords:

```
LET
INPUT
STEP
+
(
)
SIN(
GO TO
RANDOMIZE
```

TRAN scans the characters in the program line for the occurrence of any of the keywords, disregarding blanks. When one is found, the corresponding 1-byte system "token" is stored in the saved program. Thus, only one byte in the stored program is required to store such keywords as GOSUB and RANDOMIZE. All of the tokens have the high-order bit set.

At the end of every line in the code, there is a special ".EOL" token. At the end of the program there is an ".EOF" token.

The values of the tokens may be found in a listing of BASIC. Since they are only used internally, some of the values may be different for different versions of BASIC.

When an integer literal is encountered in the program following a GOSUB, GO TO, THEN, LIST, or LISTNH keyword, or as the first element on a line, it is stored as a symbol table reference to a line number entry.

When TRAN finds any other literal numeric value in the input program line, it stores it in the translated program in one of the following forms:

1-Byte Literal — An integer constant in the range 0-255 is stored as two bytes in the translated program:

> Byte 1: constant 375
> Byte 2: 1-byte value

1-word Literal — An integer constant with an absolute value less than 32,768 which is not in the range 0-255 is stored as three bytes in the translated program:

> Byte 1: Constant 376
> Bytes 2-3: 2-byte value

2-word literal — Any other numeric constant is stored as five bytes in the translated program:

> Byte 1: constant 374
> Bytes 2-5: 4-byte floating point value of the literal, as described in section A.3.2.

Certain Keywords when they are translated into tokens have special "extra bytes" inserted after the token as described below.

Keyword	Translated code
' or "	When the first quote character is encountered, TRAN outputs the corresponding token, followed by a .TEXT token, with the value 377. Next follow all of the ASCII characters in the program line up to the closing quote character. Finally, TRAN outputs a 0 byte and a matching close-quote token to the translated program.
FN	A special byte is placed in the translated code after the FN token. It contains a function number to represent the function name, as follows:

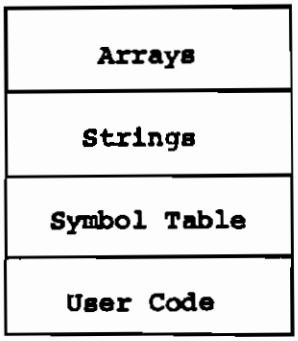
Function Number (octal)	Function Name
0	FNA
2	FNB
4	FNC
6	FND
.	.
.	.
.	.
62	FNZ

NEXT	Ten extra bytes are output to the translated code following the NEXT statement; these are required at execution time for the proper nesting of FOR-NEXT loops.
REM	The REM token in the code is followed by a .TEXT token, and then the remaining characters on the line.

Any sequence of characters which cannot be translated into a token and is not a symbol table reference or literal, is translated as the .TEXT token, followed by the remaining characters on the line. The BASIC language does not allow a program to have two adjacent variable names without an intervening character. If this occurs, the remainder of the line is translated as described above. When any such translated program line is executed, it produces a syntax error. If the first sequence of characters in a program statement is translated as a .TEXT token, BASIC assumes an implicit CALL statement (see Section 2.10.8 of the BASIC-11 Language Reference Manual). The characters are compared with the assembly language call statements defined in the system function table. If the characters match a call name the call name is executed and if a match is not found the ?UFN (Undefined Function) error message will be printed.

A.3.4 BASIC Memory Map

BASIC stores a user program in memory in the following format:



The symbol table and user code area are created when the program is entered. When the RUN command is given the user program is scanned and arrays are set up. The string buffer is created during program execution.

The SCRatch command (refer to paragraph 4.4 of the BASIC-11 Language Reference Manual) clears all the user code, symbol table, strings and arrays from memory. The CLEAR command clears the arrays and strings but does not affect the user code or symbol table.

Every cassette file open has a 64-word buffer associated with it. If DOUBLE BUF was specified when the file was OPENed, the buffer is 128 words.

A symbol table entry is created for each distinct line number (four bytes) or variable name (ten bytes) referenced in the program. These entries are not deleted, however, even when all references in the program to a particular line number or variable are removed. Thus, if the program in memory is heavily modified, it may be desirable to save with the SAVE command and then restore the program with the OLD command to obtain the largest possible user area.

All user-entered blanks that are not in REM statements or string constants, and all blanks produced by BASIC when listing or saving a program do not contribute to the size of the program in memory. The total amount of memory storage required to store a BASIC program (user code and symbol table) depends on the following parameters (2 bytes=1 word):

Parameter	Contribution (bytes)	Definition
L	7*L	Total number of lines in the BASIC program
T	T	Total number of tokens in the program; one BASIC token is generated for each occurrence of the following: BASIC keywords such as PRINT, IF and THEN, function references such as PI, SIN and SEG$ (the left parenthesis following a function name is considered to be part of the function), and special characters such as +, -, (,), ', ", etc. (The + or - preceding a numeric constant is considered to be a separate entity)
V	10*V	Total number of distinct variable names used in the program (in the context, a scalar and array variable with the same name are considered the same variable name)
R	2*R	Total number of occurrences of variable names and references to line numbers in the program (not including the line number at the beginning of each line)
I1	2*I1	Total number of integer constants in the range 0<x<255
I2	3*I2	Total number of integer constants in the range 255<x<32767
F	5*F	Total number of non-integer numeric constants and integer constants not in the above ranges
N	10*N	Total number of NEXT statements in the program
F	2*F	Total number of references to the name of a user-defined function, e.g., FNA (including the definition itself)
S	2*S	Total number of REM statements, implied CALL statements, and string constants
	C	Total number of characters in the above statements (number of characters following REM, number of characters in an assembly language routine name, number of characters between (and not including) the quotes in a string constant)

For example, each use of the multiple statement line saves six bytes. The program:

```
10 A=3
20 B=4
```

takes 6 bytes more of memory than the equivalent program:

```
10 A=3\B=4
```

When the BASIC program is running, the following additional array and string storage is required. For each numeric array, the number of bytes allocated is

```
4*(SS1MAX+2)
```

for a singly-dimensioned array.

or

```
4*[(SS1MAX+1)*(SS2MAX+1)+1]
```

for a doubly-dimensioned array, where SS1MAX and SS2MAX are the maximum values of the first and second array subscripts, respectively. For each string array, the number of bytes allocated is

```
2*(SS1MAX+2)
```

for a singly-dimensioned array or

```
2*[(SS1MAX+1)*(SS2MAX+1)+1]
```

for a doubly-dimensioned array, where SS1MAX and SS2MAX are the maximum values of the first and second array subscripts, respectively.

For each non-null string scalar or array element of length N currently defined in the BASIC program, N+4 bytes of string storage are required.

SWINBURNE COLLEGE OF TECHNOLOGY
ELECTRICAL ENGINEERING DEPT.

APPENDIX B

PERIPHERAL DEVICE SUPPORT

B.1 LABORATORY PERIPHERAL SYSTEM (LPS) SUPPORT

The Laboratory Peripheral System (LPS) support for BASIC is supplied in six binary relocatable files.

LPS0.OBJ	LPS kernel module	Required
LPS1.OBJ	Analog to digital conversion	Optional
LPS2.OBJ	Real-time clock(60hz line frequency)	Either is
LPS2C.OBJ	Real-time clock(50hz line frequency)	optional
LPS3.OBJ	Digital input/output	Optional
LPS4.OBJ	Display	Optional

The standard BASIC/CAPS binary kit contains all the object modules required to link a version of BASIC/CAPS that contains LPS support (with all four optional LPS modules). In addition to the LPS object modules already mentioned, the following files are supplied:

BASINL.OBJ	Interface module for LPS support.
FTBLL.OBJ	Function table for LPS support.
PERVEC.OBJ	Vector definition module for LPS support with LPS vectors set at location 340 (octal).

There are also the following files which are provided in source form in all kits:

FTBL.PAL	Function Table Module
PERVEC.PAL	Vector Definition Module
BASINT.PAL	Interface Module
PERPAR.PAL	Parameter file

NOTE

BASIC with LPS support requires a PDP-11 with 16K or more of memory. The procedures in this section assume the user has at least 16K of memory and has reconfigured his CAPS-11 system, along with PAL and LINK, for 16K.

To create a version of BASIC/CAPS with complete LPS support, no GT support, and a standard hardware configuration, it is only necessary to link the supplied object modules as directed in the example load building for a complete configuration in section B.1.1.

To create a version of BASIC/CAPS including support for both the LPS and GT, see the load building example in section B.2.1.

To create a customized version of BASIC/CAPS with LPS support, the parameter file PERPAR.PAL is edited and assembled with FTBL.PAL, PERVEC.PAL and BASINT. The three object modules produced are then linked with the LPS and BASIC object modules to produce a load module. The specific instructions that are given to the system programs (EDIT, PIP, PAL, and LINK) are given in section B.1.1.

NOTE

All of the procedures in this section assume that an unaltered PERPAR.PAL is being edited. It is recommended that a copy of the original PERPAR.PAL be made and saved for future use.

The BASINT.PAL interface module should be used with all versions of BASIC/CAPS. If the display module is not included in the LPS support to be linked, another background routine may be linked with BASIC but it must be defined in this module. See Section 4.3 for instructions to define the background routine.

For the LPS routines to be accessible from the BASIC CALL statement, the routine must be defined in a System Function Table as described in Section 4.1. FTBL.PAL is a function table in source form. If any user-written assembly language routines are also linked with BASIC the routines must be defined in this function table. See Section 4.1 for instructions to add the assembly language routine definitions to the Function Table.

PERVEC.PAL is the vector definition module. It defines the hardware addresses of the status registers and the interrupt vectors. The standard hardware address for the LPS interrupt vector is 340 (octal). In PDP-11E10 machines with LPS support, however, the interrupt vector is location 300 (octal). To assemble PERVEC with the interrupt vector at 300 (octal) it is necessary to delete the semicolon before the $V=0 definition in PERPAR.PAL. If the interrupt locations are at another location in memory then correct the interrupt addresses by using the system editor to define $V in PERPAR equal to the interrupt address minus 300 (octal). For example. if the LPS interrupt vectors start at 320 (octal) define $V=20 (octal). A listing of PERVEC.PAL is printed at the end of section B.1.2.

PERPAR.PAL is a parameter file; a listing follows:

```
.TITLE PERPAR -- PERIPHERAL SUPPORT PACKAGE PARAMETER MODULE.
;
; DEC-11-LSPAA-A-LA     BASIC KERNEL V02-01
;
; COPYRIGHT (C) 1974
;
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
; DEC ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT
; MAY APPEAR IN THIS DOCUMENT.
;
; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A
; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND
; CAN BE COPIED (WITH INCLUSION OF DEC'S COPYRIGHT
; NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT AS MAY
; OTHERWISE BE PROVIDED IN WRITING BY DEC.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE
; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
; WHICH IS NOT SUPPLIED BY DEC.

; THE CONDITIONALS CONTAINED IN THIS MODULE AFFECT THE ASSEMBLY
; OF THE FUNCTION TABLE MODULE "FTBL.MAC".
; TO OBTAIN THE DESIRED CONDITIONAL DEFINITION(S),
; REMOVE (USING AN EDITOR) THE
; SEMI-COLON APPEARING BEFORE THE CONDITIONAL.
;$DISK=0                 ;DEFINE FOR RT-11
        .IFNDF $DISK
$STRNG=0                ;DO NOT DEFINE FOR PTS BASIC WITHOUT
                        ;STRINGS,= DEFINED FOR PTS V01 WITH STRINGS
        .ENDC           ;

;$LPS=0                 ;DEFINE FOR LPS
        .IFDF $LPS
;$V=0                   ;DEFINE FOR LPS WITH VECTORS STARTING
;                       ;    AT 300.  DEFAULT SETTING IS VECTORS AT
;                       ;    340, SET $V = ANY OTHER DISPLACEMENT IF
;                       ;       VECTORS START AT DISPLACEMENTS
;                       ;       OTHER THAN 0 OR 40 FROM
;                       ;       VECTOR 300
```

```
$ADC=0                  ;INCLUDE A/D ROUTINES.
$CLK=0                  ;INCLUDE CLOCK ROUTINES.
$DIO=0                  ;INCLUDE DIGITAL IO ROUTINES
$DIS=0                  ;INCLUDE DISPLAY ROUTINES.
        .ENDC   ;$LPS
;
;
;$VT11=0                         ;FOR GT40 (GT44)
;
;


        .IFDF   $VT11
$CLOCK=0                        ;FOR SYSTEM CLOCK (KW11L)
        .ENDC

        .EOT
```

To link the LPS module with BASIC it is necessary to delete the semicolon (;) before the $LPS=0 statement in PERPAR.PAL. If any of the four optional modules are not to be included, a semicolon (;) must be inserted before the appropriate conditional.

Parameter	Insert ; before parameter if
$ADC=0	module LPS1 is not to be included.
$CLK=0	module LPS2 (LPS2C) is not to be included.
$DIO=0	module LPS3 is not to be included.
$DIS=0	module LPS4 is not to be included.

Using the system assembler, the sources are assembled in the following combinations to produce the needed LPS object modules:

Object File	Source Files
FTBL	PERPAR,FTBL
PERVEC	PERPAR, PERVEC
BASINT	PERPAR, BASINT

After these modules have been reassembled, the LPS support may be linked with the BASIC object modules with only the desired optional LPS modules included in the LINK command strings.

For extremely long programs that do not use string variables, the LPS support may be linked with the 8K overlay version object modules. This no-string version of BASIC with LPS support has more free memory for program and array storage, but will require overlaying of the BASIC/CAPS system.

After BASLPS has been linked it may be loaded by the following monitor command:

```
.R BASLPS
```

At this point the standard BASIC initial dialogue occurs. See Section 1.2 for a description of the initial dialogue.

As part of the initial dialogue, BASIC prints:

```
USER FNS LOADED
```

This message occurs whenever BASIC has been linked with LPS support.

When editing PERPAR.PAL, $LPS=0 should be enabled for BASIC with any LPS support, and $ADC=0, $CLK=0, $D10=0, and $DIS=0 should be disabled whenever the appropriate optional LPS module is not to be included. In addition, $V=0 should be enabled for any PDP-11 with LPS hardware interrupt located at 300 (octal) instead of 340 (octal). Most PDP-11E10 with LPS require the defining of the $V=0 assembly parameter. For hardware addresses other than 300 or 340, define $V as described in paragraph about PERVEC.PAL.

B.1.1 Linking BASIC/CAPS with LPS Support

The procedures for building the following load modules:

BASIC/CAPS	with complete LPS support.
BASIC/CAPS	with complete LPS support and LPS interrupt vectors at location 300 (octal).
BASIC/CAPS	with only the ADC and DIS optional display modules.

are described in this section. Instructions for linking both LPS and GT support with BASIC are given in Section B.2.1. Since all editing instructions assume an original PERPAR.PAL, a copy of the original file should be preserved to allow any future load modules to be built from an unedited PERPAR.PAL.

In all the following examples the user can substitute the module LPS2C for the module LPS2 if the line frequency is 50Hz instead of 60Hz.

(Ⓢ) represents the ALTMODE Key.

Complete Configuration

To build a load module BASLPS.SLO under CAPS-11 including LPS support and all four optional modules, enter the following command strings:

```
                    (Mount CAPS-11 system cassette on unit 0)
                    (Mount scratch cassette on unit 1)
    .Z 1:
                    (Mount BASIC object module cassette containing
                    BASICR.OBJ on unit 1)
    .R LINK
                    (When unit 0 has rewound, mount zeroed scratch
                    cassette on unit 0)

*BASLPS.SLO,TT:=1:BASICR,FPMP,BASICE,BASICX/F,BASICS/F/B:400/C
,BASINL/P,FTBLL/F,PERVEC/F,LPS0/F,LPS1/F,LPS2/F,LPS3/F,LPS4/F,BASICH/P

    1?              (Mount cassette with BASINL.OBJ on unit 1)

    1?              (Mount cassette with BASICR.OBJ on unit 1)

       (LOAD MAP PRINTED)
```

```
PASS 2

1?              (Mount cassette with BASINL.OBJ on unit 1)

1?              (Mount cassette with BASICR.OBJ on unit 1)

*               (Done.  Cassette on unit 0 contains a new
                BASLPS.SLO which is BASIC/CAPS plus all LPS
                support-interrupt vectors at 340 (octal)).
```

Complete Configuration-
Interrupt vectors at location 300 (octal)

These instructions are the same as the preceding instructions except that a $V=0 parameter definition in PERPAR.PAL is enabled.

To change the interrupt vector location, the file PERPAR.PAL must be edited to enable the $V=0 parameter definition. The new PERPAR is then used to reassemble PERVEC.PAL to redefine the vector locations.

```
                (Ⓢ represents the ALTMODE key
                (Mount CAPS-11 system cassette on unit 0)
                (Mount scratch cassette on unit 1)
.Z 1:
                (Mount a second scratch cassette on unit 1)

.Z 1:
.R EDIT
*EW1:PERPAR.PAL ⓈⓈ
                (Mount BASIC object cassette containing PERPAR.PAL
                on unit 0)
*ER0:PERPAR.PAL Ⓢ R ⓈⓈ
*F;$LPS=0 Ⓢ 0AD ⓈⓈ
*F;$V=0 Ⓢ 0AD ⓈⓈ
*EX$$
                (Mount CAPS-11 system cassette on unit 0)
.R PAL
*PERVEC/P=1:PERPAR/P,PERVEC/P

1?              Type any keyboard character

1?              (Mount cassette with PERVEC.PAL on unit 1)

PASS 2

0?              (Mount second scratch cassette on unit 0)

1?              (Mount cassette with new PERPAR.PAL on unit 1)

1?              (Mount cassette with PERVEC.PAL on unit 1)

000000 ERRORS
*↑C

                (Mount CAPS-11 system cassette on unit 0)
                (Mount cassette containing BASICR.OBJ on unit 1)

.R LINK
                (When unit 0 has rewound, mount cassette with new
                PERPAR.PAL on unit 0)
```

```
*BASLPS.SLO,TT:=1:BASICR,FPMP,BASICE,BASICX/F,BASICS/F/B:400/C
,PERVEC/P,BASINL/P,FTBLL/F,LPS0/F,LPS1/F,LPS2/F,LPS3/F,LPS4/F,BASICH/P

     1?              (Mount cassette with new PERVEC.OBJ on unit 1)

     1?              (Mount cassette with BASINL.OBJ on unit 1)

     1?              (Mount cassette with BASICR.OBJ on unit 1)

        (LOAD MAP PRINTED)
     PASS 2

     1?              (Mount cassette with new PERVEC.OBJ on unit 1)

     1?              (Mount cassette with BASINL.OBJ on unit 1)

     1?              (Mount cassette with BASICR.OBJ on unit 1)

     *↑C             (Done.  New version of BASLPS.SLO  with  LPS
                     interrupt vectors at 300 is on cassette 0)

     .
```

Partial Configuration

To build a load module BASLPS.SLO of BASIC/CAPS which includes only the ADC and display routines, enter the following command strings:

```
                     (S) represents the ALTMODE key
                     (Mount CAPS-11 system cassette on unit 0)
                     (Mount scratch cassette on unit 1)
     .Z 1:
                     (Mount second scratch cassette on unit 1)

     .Z 1:
     .R EDIT
     *EW1;PERPAR.PAL (S) R (S)(S)
     *F;$LPS=0 (S) 0AD (S)(S)
     *F$CLK=0 (S) 0AI; (S)(S)
     *F$DIO=0 (S) 0AI; (S)(S)
     *EX (S)(S)
                     (Mount CAPS-11 system cassette on unit 0)
     .R PAL
     *FTBLL/P=1:PERPAR/P,FTBL/P

     1?              (Type any keyboard character)

     1?              (Mount cassette with FTBL.PAL on unit 1)
```

```
PASS 2

0?              (Mount second scratch cassette on unit 0)

1?              (Mount cassette with new PERPAR.PAL on unit 1)

1?              (Mount cassette with FTBL.PAL on unit 1)

000000 ERRORS
                (Cassette on unit 0 now contains a new FTBLL.OBJ)
*↑C

                (Mount CAPS-11 system cassette on unit 0)
                (Mount cassette containing BASICR.OBJ on unit 1)

.R LINK

                (When unit 0 has rewound, mount cassette
                containing new PERPAR.PAL on unit 0)

*BASLPS.SLO,TT:=1:BASICR,FPMP/F,BASICE/F,BASICX/F,BASICS/F/B:400/C
,FTBLL/P,BASINL/P,PERVEC/F,LPS0/F,LPS1/F,LPS4/F,BASICH/P

1?              (Mount cassette with new FTBLL.OBJ on unit 1)

1?              (Mount cassette with BASINL.OBJ on unit 1)

1?              (Mount cassette with BASICR.OBJ on unit 1)

   (LOAD MAP PRINTED)
PASS 2

1?              (Mount cassette with new FTBLL.OBJ on unit 1)

1?              (Mount cassette with BASINL.OBJ on unit 1)

1?              (Mount cassette with BASICR.OBJ on unit 1)

*↑C             (Done.  New version of BASLPS.SLO with only ADC
                and display routines, interrupt vectors at 340, is
                on cassette 0)

.
```

B.1.2 Assembling LPS Support from the Sources

The Laboratory Peripheral System support may also be purchased in source form. The following nine source files are provided. (The source files for FTBL, BASINT, PERPAR, and PERVEC are provided with the binary kit.)

```
LPS0.PAL
LPS1.PAL
LPS2.PAL
LPS3.PAL
LPS4.PAL
FTBL.PAL
PERVEC.PAL
BASINT.PAL
PERPAR.PAL
```

The following chart lists the assembly parameters for each module.

Source File	Conditionals	Define for Systems with:
LPS0.PAL	None	
LPS1.PAL	None	
LPS2.PAL	CYC50	50 Hz line frequency (60 Hz is default)
LPS3.PAL	None	
LPS4.PAL	None	
FTBL.PAL	$ADC	LPS1
	$CLK	LPS2
	$DIO	LPS3
	$DIS	LPS4
	$LPS	LPS0 (all systems with LPS support)
	$VT11	GT40 or GT44 support
	$DISK	(only for RT-11 systems)
PERVEC.PAL	$LPS	LPS0
	$V	LPS interrupts not at location 340 (octal)
	$VT11	GT40 or GT44 support
BASINT.PAL	$LPS	LPS0
	$DIS	LPS4

NOTE

To change the number of buffers which may be defined, edit the source of LPS0 and change the value of NARRAY before assembling.

To assemble the LPS object modules from the sources, use the following command strings:

```
                    (Mount CAPS-11 system cassette on unit 0)
                    (Mount scratch cassette on unit 1)

.Z 1:
                    (Mount cassette with LPS0.PAL on unit 1)

.R PAL

*LPS0/P=1:LPS0

PASS 2
0?                  (Mount scratch cassette on unit 0)

000000 ERRORS
```

```
*LPS1=1:LPS1

PASS 2

000000 ERRORS

*LPS2=1:LPS2

PASS 2

000000 ERRORS

*LPS3=1:LPS3

PASS 2

000000 ERRORS
                    (Mount cassette with LPS4.PAL on unit 1)
*LPS4=1:LPS4

PASS 2

000000 ERRORS
*↑C
.
```

If the line frequency is 50Hz, use the following commands to create the object module LPS2C:

```
*↑C
                    Ⓢ represents the ALTMODE key
                    (Mount CAPS-11 system cassette on unit 0)
                    (Mount scratch cassette on unit 1)

.Z 1:

.R EDIT
*EW1:PARAM.PAL ⓈⓈ
*ICYC50=0
ⓈⓈ
*EX ⓈⓈ

.R PAL
*LPS2C/P=1:PARAM/P,LPS2/P
```

```
1?              (Type any keyboard character)

1?              (Mount cassette with LPS2.PAL on unit 1)

PASS 2

0?              (Mount cassette with newly-created object  modules
                of LPS0 and LPS1 on unit 0)

1?              (Mount cassette with PARAM.PAL on unit 1)

1?              (Mount cassette with LPS2.PAL on unit 1)

000000 ERRORS

                (Mount cassette with LPS3.PAL on unit 1)
*LPS3=1:LPS3    (Continue as in above example)
      .
      .
      .
```

Either of these procedures produces five object modules: LPS0.OBJ, LPS1.OBJ, LPS2.OBJ (or LPS2C.OBJ), LPS3.OBJ, LPS4.OBJ. The instructions to assemble the other LPS files and the instructions to link the LPS object modules with the BASIC object modules are given in the preceding section. Following is a listing of PERVEC.PAL which contains the interrupt vector location for the LPS and GT40(44) hardware:

```
.TITLE PERVEC VECTOR DEFINITION MODULE FOR BASIC SUPPORT PACKAGES.
;
; DEC-11-LBPVA-A-LA     BASIC KERNEL V02-01
;
; COPYRIGHT (C) 1974
;
; DIGITAL EQUIPMENT CORPORATION
; MAYNARD, MASSACHUSETTS 01754
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
; DEC ASSUMES NO RESPONSIBILITY FOR ANY ERRORS THAT
; MAY APPEAR IN THIS DOCUMENT.
;
; THIS SOFTWARE IS FURNISHED TO PURCHASER UNDER A
; LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND
; CAN BE COPIED (WITH INCLUSION OF DEC'S COPYRIGHT
; NOTICE) ONLY FOR USE IN SUCH SYSTEM, EXCEPT AS MAY
; OTHERWISE BE PROVIDED IN WRITING BY DEC.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE
; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
; WHICH IS NOT SUPPLIED BY DEC.

; THIS MODULE DEFINES THE HARDWARE ADRESSES USED BY
; SUCH HARDWARE AS THE "LPS", THE "VT11"(GT40) AND THE "LV11".
; IF THE VECTORS FOR THESE DEVICES SHOULD CHANGE
; THIS MODULE MUST BE EDITED TO REFLECT THE CHANGE.
```

```
        .IFDF $LPS
        .IFNDF  $V
        $V=40
        .ENDC
        .GLOBL LPSAD,LPSADB,LPSDR,LPSDMA
        .GLOBL LPSCKS,LPSPB,LPSDRS,LPSDIB
        .GLOBL LPSDOR
        .GLOBL LPDISS,LPDISX,LPDISY
        .GLOBL CKLIVA,CKLIP,DRSIVA,DRSIP,LPSIVA,LPSIP

; DEVICE EQUATES:

LPSAD   =       170400  ;LPS A/D STATUS REG.
LPSADB  =       170402  ;LPS A/D BUFFER LED REG.
LPSCKS  =       170404  ;LPS CLOCK STATUS REG.
LPSPB   =       170406  ;LPS CLOCK BUFFER PRESET REG.
LPSDR   =       170410  ;LPS DIGITAL I/O STATUS REG.
LPSDRS  =       LPSDR
LPSDIB  =       170412  ;LPS DIGITAL INPUT REG.
LPSDOR  =       170414  ;LPS DIGITAL OUTPUT REG.
LPDISS  =       170416  ;LPS DISPLAY STATUS REG.
LPDISX  =       170420  ;LPS DISPLAY REG, X
LPDISY  =       170422  ;LPS DISPLAY REG, Y
LPSDMA  =       170436  ;LPS DMA REGG.

; INTERRUPT VECTOR PAIRS:


CKLIVA  =       304+$V  ;ADR. OF CLOCK INTERRUPT VECTOR
CKLIP   =       306+$V  ;ADR. OF CLOCK INT.     PRIORITY

DRSIVA  =       310+$V  ;ADR. OF DRS INT. VECTOR
DRSIP   =       312+$V  ;ADR. OF DRS INT. PRIORITY.

LPSIVA  =       300+$V  ;ADR. OF THE A/D INT. VECTOR.
LPSIP   =       302+$V  ;ADR. OF THE INT.PRIORITY.

        .ENDC   ;$LPS
        .IFDF   $VT11   ;GT40
        .GLOBL  DPC,DSR,DISX,DISY,GTVECT

DPC     =       172000  ;VT11 DISPLAY PC
DSR     =       DPC+2   ;VT11 DISPLAY STATUS REG
DISX    =       DSR+2   ;VT11 X STATUS REG
DISY    =       DISX+2  ;VT11 Y STATUS REG
GTVECT  =       320     ;ADR. OF VT11 [GT40 (GT44)] INTERRUPT
                        ;VECTOR LIST. REDEFINING GTVECT
                        ;REDEFINES THE ENTIRE SET
                        ;OF DISPLAY PROCESSOR INT. VECTORS.
;GTVECT:                ;DISPLAY STOP VECTOR
;GTVECT+4:              ;LIGHT PEN HIT VECTOR
;GTVECT+10:             ;DISPLAY TIME OUT VECTOR
        .ENDC   ;$VT11


        .END
```

B.2 GT GRAPHICS SUPPORT

The GT Support is supplied in the BASIC/CAPS binary kit in the following files:

GTB.OBJ	Main GT object module
GTC.OBJ	Secondary GT object module
PERVEC.PAL	Vector definition source file
FTBL.PAL	Function table source module
BASINT.PAL	Interface Module (source)
PERPAR.PAL	Parameter file (source)
GTNLPS.OBJ	Module linked with GT when LPS support is not also linked
BASING.OBJ	Interface module (object) for BASIC with GT support.
FTBLG.OBJ	Function table (object) for BASIC with GT support.
PERVEG.OBJ	Vector definition module (object) for BASIC with GT support.

NOTE

BASIC with GT support has three graphic array calls (YGRA, XGRA, and FIGR) that allow the display of an entire array through one graphics call. Graphics created by these calls may be changed while they are being displayed by calls to APUT, AGET, and FPUT. Because the graphics created may be dynamically changed, extreme care must be taken in their use. Once a graphic array call (YGRA, XGRA, or FIGR) has been made a call to INIT must be made before the user array is zeroed by the BASIC system. The following BASIC statements and commands zero the user arrays: RUN, RUNNH, SCRATCH, CLEAR, NEW, OLD, and CHAIN. If the user's arrays are zeroed while a graphics array is being displayed, the display processor is not accessible by any program. Any attempt to execute any graphics call causes the processor to enter a closed loop. If this situation occurs, it is necessary to reload BASIC/CAPS with GT support from cassette.

This causes a hardware reset of the display processor and complete initialization of the BASIC system. The stored program will be lost.

To create BASIC/CAPS with GT support, without LPS support, and with a standard hardware configuration, it is only necessary to follow the linking instructions in section B.2.1

To create a version of BASIC/CAPS with GT and LPS support or with a non-standard GT vector location, the parameter file PERPAR.PAL must be edited and assembled with FTBL.PAL, PERVEC.PAL, and BASINT.PAL. The three object modules produced are then linked with the BASIC, GT, and, if desired, LPS object modules to produce a load module. The specific instructions that are given to the system programs (EDIT, PIP, PAL, and LINK) are given in section B.2.1

NOTE

All the procedures for editing PERPAR.PAL in this section assume that an unaltered PERPAR.PAL is being used. It is recommended that a copy of the original PERPAR.PAL be made and saved for future use.

A listing of PERPAR.PAL is given in section B.1.

The semicolon before the $VT11=0 statement must be deleted when building a customized version of BASIC/CAPS with GT support. If the LPS modules are also to be linked, it is necessary to delete the semicolon before the $LPS=0 statement.

If the GT vector addresses are not standard (at location 320 (octal)) then PERVEC.PAL, listed in section B.1.2, must be altered to have the correct vector addresses.

The BASINT.PAL interface module should be used with all versions of BASIC/CAPS. If a background routine is also linked with BASIC, it must be defined in the interface module. See section 4.3 for instructions to link a background routine with BASIC.

For the GT routines to be accessible by a BASIC "CALL" statement the routines must be defined in a System Function Table as described in Section 4.1. FTBL.PAL is a function table in source form. If any user-written assembly language routines (or LPS support) are also linked with BASIC the routines must be defined in this function table. See section 4.1 for instructions to add assembly language routine definitions to this function table.

PERPAR.PAL should then be assembled with FTBL.PAL, PERVEC.PAL, and BASINT.PAL. The three object modules produced should be linked with the BASIC, GT, and, if desired, LPS object modules.

B.2.1 Linking BASIC/CAPS with GT Support

This section contains two load-building examples:

BASGT.SLO	BASIC/CAPS with GT support only
BGTLPS.SLO	BASIC/CAPS with GT and LPS support

BASIC with GT support and no LPS support

The instructions for linking BASIC with GT and no LPS support in the CAPS-11 environment follow:

```
                    (S) represents the ALTMODE key

                    (Mount CAPS-11 system cassette on unit 0)
                    (Mount scratch cassette on unit 1)

    .Z 1:
                    (Mount cassette with BASICR.OBJ on unit 1)

    .R LINK

*BASGT.SLO/P,TT:=1:BASICR,FPMP/F,BASICE/F,BASICX/F,BASICS/F/B:400/C
,BASING/F,FTBLG/F,PERVEG/F,GTNLPS/F,GTB/F,GTC/F,BASICH

       (LOAD MAP PRINTED)

    PASS 2

    0?              (Mount scratch cassette on unit 0)

    *↑C             (Done.  Unit 0 contains new BASGT.SLO)

    .
```

BASIC with GT and LPS Support

To build a load module including both GT and LPS support (with all the optional LPS modules) the following instructions should be given to the CAPS-11 editor, assembler, and linker.

```
                    (S) represents the ALTMODE key
                    (Mount scratch cassette on unit 1)
                    (Mount CAPS-11 system cassette on unit 0)
    .Z 1:
                    (Mount second scratch cassette on unit 1)
    .Z 1:

    .R EDIT
    *EW1:PERPAR.PAL (S)(S)
                    (Mount cassette with PERPAR.PAL on unit 0)
    *ER0:PERPAR.PAL (S) R (S)(S)
    *F;$LPS=0 (S) 0AD (S)(S)
    *F;$VT11=0 (S) 0AD (S)(S)
    *EX (S)(S)
                    (Mount CAPS-11 system cassette on unit 0)
    .R PAL
    *FTBL/P=1:PERPAR/P,FTBL/P

    1?              (Type any keyboard character)

    1?              (Mount cassette with FTBL.PAL on unit 1)

    PASS 2

    0?              (Mount second scratch cassette on unit 0)
```

```
1?              (Mount cassette with new PERPAR.PAL on unit 1)

1?              (Mount cassette with FTBL.PAL on unit 1)

000000 ERRORS   (Cassette on unit 0 now contains a new FTBL.OBJ)
*PERVEC=1:PERPAR/P,PERVEC/P

1?              (Mount cassette with new PERPAR.PAL on unit 1)

1?              (Mount cassette with PERVEC.PAL on unit 1)

PASS 2.

1?              (Mount cassette with new PERPAR.PAL on unit 1)

1?              (Mount cassette with PERVEC.PAL on unit 1)

000000 ERRORS
*↑C
                (Mount CAPS-11 system cassette on unit 1)
.R 1:PIP
*BASINT.OBJ=1:BASINL.OBJ/P

1?              (Mount cassette with BASINL.OBJ on unit 1)

*LPS.OBJ=1:LPS0.OBJ/F,LPS1.OBJ/F,LPS2.OBJ/F,LPS3.OBJ/F,LPS4.OBJ/F

*GT.OBJ=1:GTB.OBJ/P,GTC.OBJ/F

1?              (Mount cassette with GTB.OBJ on unit 1)

*↑C
                (Mount CAPS-11 system cassette on unit 0)
.R LINK
                (Mount cassette with BASICR.OBJ on unit 1)

*BGTLPS.SLO/P,TT:=1:BASICR,FPMP/F,BASICE/F,BASICX/F,BASICS/F/B:400/C
,FTBL/P,PERVEC/F,BASINT/F,LPS/F/M,GT/F,BASICH/P

1?              (Mount cassette with new FTBL.OBJ on unit 1)

1?              (Mount cassette with BASICR.OBJ on unit 1)

   (LOAD MAP PRINTED)
PASS 2

0?              (Mount cassette with new PERPAR.PAL on unit 0)

1?              (Mount cassette with new FTBL.OBJ on unit 1)

1?              (Mount cassette with BASICR.OBJ on unit 1)

*↑C             (Done. Cassette on unit 0 contains edited
                PERPAR.PAL and new version of BASIC/CAPS with GT
                and LPS support - BGTLPS.SLO)
```

B.2.2 Assembling GT Support from the Sources

BASIC GT support may also be purchased in source form. The following files are provided in source form:

```
GTB1.PAL
GTB2.PAL
GTC.PAL
PERVEC.PAL
FTBL.PAL
BASINT.PAL
PERPAR.PAL
GTNLPS.PAL
```

FTBL.PAL is the function table for the GT support functions. It is always provided in source form in the binary kit and may be extended by the careful addition of the user's own functions as described in section 4.1.

GTB1.PAL and GTB2.PAL make up the main module containing the majority of code for the GT support functions. They are provided in the source kit.

GTC.PAL contains the rest of the GT support functions. It is provided in the source kit.

PERVEC.PAL is the vector definition module. If the hardware configuration is non-standard the address of the graphic vector should be changed in the source. PERVEC.PAL is listed at the end of section B.2.1 It is provided in the binary kit.

BASINT.PAL is the interface module that should be used with all versions of BASIC/CAPS. It is provided in the binary kit.

PERPAR.PAL is a parameter file that must be edited for assembling with all the remaining files. It is used to conditionally assemble the remaining files according to the user's special requirements. Some of the parameters that may be defined are:

$VT11	Always define.
$DISK	Do not define for BASIC/CAPS.
$LONGER	Define to give longer GT error messages.
$CRASH	Define to cause a halt with an ?ADR error message rather than change data that is too large for the display instructions to use. Otherwise data is transformed into the nearest legitimate value.
$CLOCK	Always define even when hardware does not include a real-time clock.
$LPS	Always define when assembling GTB and GTC.
$DEPTH	Define to change the default depth of nesting of subpictures. If not defined here, the default is 10(decimal).

PREPAR.PAL is provided in the binary kit and is listed in the preceding section; it is edited by the EDIT program. To assemble the GTB1.PAL, GTB2.PAL, GTC.PAL, and GTNLPS.PAL sources to duplicate the object modules provided, type the following CAPS-11 Monitor instructions:

```
                    (S) represents the ALTMODE key

                    (Mount scratch cassette on unit 1)
.Z 1:
                    (Mount second scratch cassette on unit 1)
.Z 1:
                    (Mount CAPS-11 system cassette on unit 0)
.R EDIT
*EW1:PERPAR.PAL (S)(S)
                    (Mount cassette with PERPAR.PAL on unit 0)
*ER0:PERPAR.PAL (S) R (S)(S)
*F;$LPS=0 (S) 0AD (S)(S)
*F;VT11=0 (S) 0AD (S)(S)
*EX (S)(S)
                    (Mount CAPS-11 system cassette on unit 0)
.R PAL
*GTB/P=1:PERPAR/P,GTB1/P,GTB2/P

1?                  (Type any keyboard character)

1?                  (Mount cassette with GTB1.PAL on unit 1)

1?                  (Mount cassette with GTB2.PAL on unit 1)

PASS 2

0?                  (Mount second scratch cassette on unit 0)

1?                  (Mount cassette with edited PERPAR.PAL on unit 1)

1?                  (Mount cassette with GTB1.PAL on unit 1)

1?                  (Mount cassette with GTB2.PAL on unit 2)

000000 ERRORS       (Done)

*GTC=1:PERPAR/P,GTC/P

1?                  (Mount cassette with edited PERPAR.PAL on unit 1)

1?                  (Mount cassette with GTC.PAL on unit 1)

PASS 2

1?                  (Mount cassette with edited PERPAR.PAL on unit 1)

1?                  (Mount cassette with GTC.PAL on unit 1)

000000 ERRORS       (Done)

                    (Mount cassette with GTNLPS.PAL on unit 1)
*GTNLPS=1:GTNLPS
```

```
PASS 2

000000 ERRORS
*↑C             (Done)

.
```

The cassette on unit 0 will contain new object modules of GTB, GTC, and GTNLPS which can be used to build load modules of BASIC/CAPS with GT support as described in section B.2.1.

GTB.PAL and GTC.PAL may be conditionally assembled with the optional $LONGER, $CRASH, and $DEPTH parameters defined in PERPAR.PAL. For example, to create a BASIC with GT support for CAPS-11 with longer GT error messages, halting after error messages, and a depth of nesting of 17(octal) (decimal=15) allowed, the following instructions should be given to the CAPS-11 Monitor.

```
Ⓢ represents the ALTMODE key
.R EDIT
*EW1:PERPAR.PAL ⓈⓈ
          (Mount cassette with PERPAR.PAL on unit 0)
*ER0:PERPAR.PAL Ⓢ R ⓈⓈ
*I$LONGER=0
$CRASH=0
$DEPTH=17
ⓈⓈ
*F;$LPS=0 Ⓢ 0AD ⓈⓈ
*F;$VT11=0 Ⓢ 0AD ⓈⓈ
*EX ⓈⓈ
```

The remaining instructions are the same as those above for assembling GTB, GTC, and GTNLPS.

These GTB and GTC object modules may be linked with FTBL, PERVEC, BASINT, and the BASIC and LPS object modules as described in section B.2.1.

B.2.3 Technical Description of Display Buffer Management

The following technical information should be read before any user-written assembly language routines alter the graphic files or before the sources of GT are altered.

1. Overall Structure

   a. Root portion

      i) Tracking object subpicture (with tag -1)
      ii) Default conditions; i.e., beam at (0.0) etc.
      iii) A display jump to the user space

b. User space

i) User data starts at address in global ACTST
ii) Data continues through address in global ACTEND
iii) A display jump and address of root code is stored in the two words ending at ACTEND
iv) The top of the display buffer is the address in global DCRASH

2. Subpictures

a. Call consists of 5 words:

i) A display stop (code:173400)
ii) Address of the rest of the file
iii) Address of the subpicture
iv) The tag of the subpicture
v) The pointer to the next tag or zero if the last subpicture.

b. Header of the subpicture consists of one spare word used in the SAVE routine. The address in 2a-ii points to the next word.

c. The end of a subpicture is given by 2 words:

i) A display stop
ii) A zero word

d. Subpictures are turned off and on by interchanging the display stop of 2a-i by a display jump respectively.

e. When subpictures are erased:

i) The display stop of 2a-i is replaced by a display jump.
ii) The tag of the subpicture and the tags of any sub-pictures contained in it are deleted from the linked list.

f. The first subpicture tag pointer is located at address global TAG1+2.

3. Display Stop Handler

Assume DPU interrupts with the address X in the display PC.

a. Call to subpicture if the contents of X is greater than X+2. In this case X+2 is stored in a runtime stack (the subpicture tag is at X+4). The address of X (see 2a-iii above) is moved to the DPC to start the display.

b. Addition to display file if the contents of X is less than X+2. This condition arises when the display jump at ACTEND-2 is replaced by a display stop. The words are inserted and the display is started at its beginning in the root code.

c. End of a subpicture if contents of X is zero. The top entry in the runtime stack is popped off into R0 and the display is started at the address contained in R0-2.

4. Light Pen Handler

   a. Checks the runtime stack to see if the hit occurred in a subpicture.

   b. If a hit occurred in the tracking object, move the object center to new coordinates weighted by 3 times the old x (y) plus the new X (respectively Y) divided by 4. If the tracking object is on the edge, move it to the center. Continue with 4d.

   c. If a hit occurred in a subpicture, store its tag; otherwise store zero unless a previous hit has not been queried, in which case, go to 4e.

   d. Unscale to user coordinates and if the hit was in the tracking object, update the variables passed in the call to the TRAK routine.

   e. Resume the display and exit.

5. Display Time-out

   a. Reset the runtime stack and restart the display.

6. Calls to XGRA, YGRA, FIGR

   a. The following code is generated for XGRA or YGRA:

      i) Set graphic mode to XGRA or YGRA
      ii) Load Status Register B with the graphplot increment.
      iii) A display jump
      iv) Address of the first word of A(0) where the screen-scaled array data starts.
      v) The actual first subscript of the array. Stored as negative if an LPS array.

   b. The display jump at the "end" of the array points to the word following 6a-v.

   c. For FIGR, 6a-i and 6a-ii are replaced by a single word to set the graphic mode to long vector.

HOW TO OBTAIN SOFTWARE INFORMATION

SOFTWARE NEWSLETTERS, MAILING LIST

The Software Communications Group, located at corporate headquarters in Maynard, publishes newsletters and Software Performance Summaries (SPS) for the various Digital products. Newsletters are published monthly, and contain announcements of new and revised software, programming notes, software problems and solutions, and documentation corrections. Software Performance Summaries are a collection of existing problems and solutions for a given software system, and are published periodically. For information on the distribution of these documents and how to get on the software newsletter mailing list, write to:

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

SOFTWARE PROBLEMS

Questions or problems relating to Digital's software should be reported to a Software Support Specialist. A specialist is located in each Digital Sales Office in the United States. In Europe, software problem reporting centers are in the following cities.

Reading, England	Milan, Italy
Paris, France	Solna, Sweden
The Hague, Holland	Geneva, Switzerland
Tel Aviv, Israel	Munich, West Germany

Software Problem Report (SPR) forms are available from the specialists or from the Software Distribution Centers cited below.

PROGRAMS AND MANUALS

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center.

Digital Equipment Corporation	Digital Equipment Corporation
Software Distribution Center	Software Distribution Center
146 Main Street	1400 Terra Bella
Maynard, Massachusetts 01754	Mountain View, California 94043

Outside of the United States, orders should be directed to the nearest Digital Field Sales Office or representative.

USERS SOCIETY

DECUS, Digital Equipment Computer Users Society, maintains a user exchange center for user-written programs and technical application information. A catalog of existing programs is available. The society publishes a periodical, DECUSCOPE, and holds technical seminars in the United States, Canada, Europe, and Australia. For information on the society and membership application forms, write to:

DECUS	DECUS
Digital Equipment Corporation	Digital Equipment, S.A.
146 Main Street	81 Route de l'Aire
Maynard, Massachusetts 01754	1211 Geneva 26
	Switzerland

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form (see the HOW TO OBTAIN SOFTWARE INFORMATION page).

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you do not require a written reply, please check here. ☐

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754