HEWLETT PACKARD

Manual Update

HP 9826/9836 BASIC 2.0 Language Update

Manual Part Nos. 09826-90055 and -90020

This update has two purposes: first, it describes additions to HP 9826 BASIC 1.0 Language; second, it describes the features of the HP 9836 Computer which are additions to those of the HP 9826. Additions to both the *BASIC Language Reference* and *BASIC Interfacing Techniques* are provided as part of this Update; *BASIC Programming Techniques* already includes all BASIC 2.0 features.

Summary of Additional Features

BASIC 2.0 Language can be used with both the HP 9826 and 9836 Computers. The language contains all of the features of BASIC 1.0 language, plus several additions to the language. These additions fall into two categories: those applicable to both HP 9826 and 9836, and those particular to only the HP 9836.

Additions to the BASIC 1.0 Language

The following capabilities have been added to HP 9826 BASIC 1.0 Language. **All** of these additional language capabilities are **common to both** HP 9826 and HP 9836.

- Structured-programming constructs: LOOP, REPEAT...UNTIL, SELECT...CASE, and WHILE.
- Additional graphics statements: PLOT, IPLOT, and RPLOT.
- External mass storage and file-transfer capabilities, provided by the COPY statement and additional mass storage unit specifiers.
- Enhanced keyboard control provided by ON KBD, KBD$, and SUSPEND/RESUME INTERACTIVE.
- Powerfail protection (optional).
- The capability[1] of using the HP 98628 Data Communication Interface.

1 No additional keywords have been added to the language to support the use of this interface; only the definitions of existing language features have been updated.

Supplement to 09826-90055
January 1982
Printed in U.S.A.

Features of the HP 9836

The following hardware features are particular to the HP 9836.

- CRT enhancements: eighty-character CRT width, with inverse, underlined, and blinking character capabilities.
- A second, built-in disc drive.

Documentation Overview

The documentation provided with these additional language and computer features is in the form of pages, sections, and a chapter describing each new capability. This additional text is to be inserted into the appropriate place within the specified, currently existing manual.

Techniques for using the structured-programming constructs, additional graphics statements, and mass storage and file-transfer capabilities are already included in the first edition of *BASIC Programming Techniques*; the corresponding language-reference descriptions are included in this update.

Updates to the *BASIC Language Reference*

Descriptions of the following new BASIC statements have been provided. These pages may be inserted into the appropriate place in the manual.

- **Additional Keyword Descriptions** — provides descriptions of the following keywords/ constructs.

  COPY
  IPLOT
  LOOP ... EXIT IF ... END LOOP
  KBD$
  OFF KBD/ON KBD
  PLOT
  RESUME INTERACTIVE
  REPEAT ... UNTIL
  RPLOT
  SELECT ... CASE ... CASE ELSE ... END SELECT
  SUSPEND INTERACTIVE
  WHILE ... END WHILE

Updates to *BASIC Interfacing Techniques*

Descriptions of the following new BASIC statements have been provided in this update. All of this text may be inserted into the appropriate place in the manual; it will be incorporated into the manual in the second edition.

- **HP 9836 CRT Enhancements** — describes the additional features of the CRT: eighty-character screenwidth, and enhanced characters (underlined, blinking, and inverse-video characters, in any combination).
- **Enhanced Keyboard Control** — describes the additional keyboard control available to BASIC programs. The new capabilities include keystroke trapping and suspending interactive keyboard operations.
- **Chapter 14: Powerfail Protection** — describes the powerfail protection capabilities of HP 9826 and 9836 computers. Both interface operation and programming are described.

Current Interface Manuals

The following chapters are not provided with this update; however, each chapter is automatically shipped with the interface it describes and is currently available as a separate chapter of the interfacing manual by ordering the specified HP part number.

- **Chapter 12: Data Communication Programming (09826-90021)** describes how the HP 98628 Data Communications Interface operates and provides several BASIC programming techniques.
- **Chapter 13: RS-232 Serial Interface (09826-90022)** provides the same information for the HP 98626 Serial Interface.
- **Chapter 15: The GPIO Interface (09826-90023)** provides the same information for the HP 98622 GPIO Interface.

Using the BASIC 2.0 Language

The BASIC 2.0 language may be loaded into either the HP 9826 or 9836 computer in the same manner in which BASIC 1.0 is loaded. First, insert the system disc into the disc drive (the right-hand drive of the 9836). Either cycle power off and the on again, or turn power on if previously off. The system is automatically loaded into the computer, which takes a few moments. When the language system is completely loaded, the following message is displayed:

```
        BASIC Ready 2.0
```

Many of the new enhancements provided by BASIC 2.0 were previously available with the BASIC Enhancements Binary program (BEB). However, it is no longer necessary (or possible) to load the BEB program into the computer while BASIC 2.0 is resident.

BASIC 1.0 Compatibility

All programs written on the HP 9826 equipped with BASIC 1.0 can be run on the HP 9836, including those programs written on the 9826 equipped with the BASIC Enhancement (BEB) capabilities.

Conversely, all programs written on either the HP 9826 or 9836 with BASIC 2.0 may be run on the HP 9826 with BASIC 1.0 and BEB, with the following **exception**. The mass storage unit specifier:

```
":INTERNAL,4,0"
```

cannot be used on the 9826 with BASIC 1.0 and BEB. The solution is to specify ":INTERNAL" when the 9826 drive (or 9836 right-hand drive) is to be used, if software is to be transported in this manner.

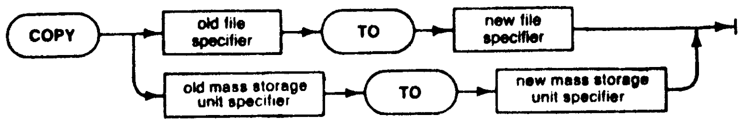
Additional Keyword Descriptions

This section contains descriptions of the additional keywords provided by BASIC 2.0. These pages may be inserted into the appropriate places in the BASIC Language Reference.

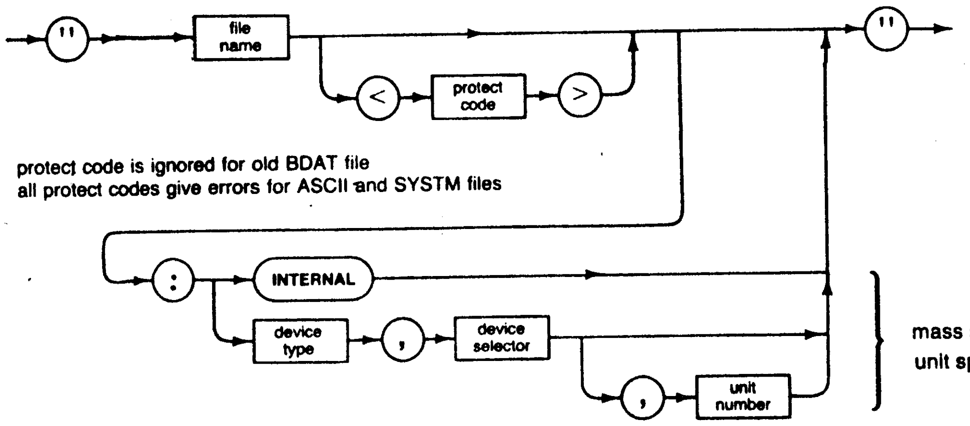
COPY

Minimum Requirement	BASIC 2.0
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

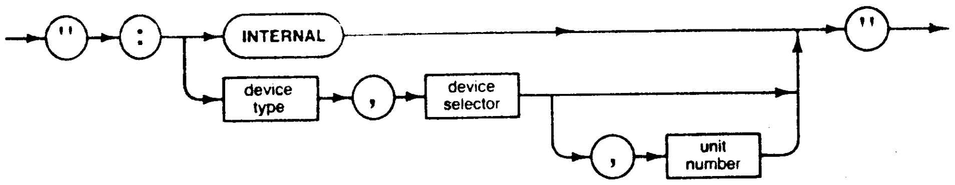
This statement allows copying of individual files or entire discs.


COPY
old file specifier
TO
new file specifier
old mass storage unit specifier
TO
new mass storage unit specifier

literal form of file specifier


"
file name
<
protect code
>
protect code is ignored for old BDAT file
all protect codes give errors for ASCII and SYSTM files
:
INTERNAL
device type
,
device selector
,
unit number
"
mass storage unit specifier

literal form of mass storage unit specifier



Item	Description/Default	Range Restrictions
file specifier	string expression	(see drawing)
mass storage unit specifier	string expression	(see drawing)
file name	literal	any valid file name
protect code	literal; only the first two characters are significant	—
device type	literal	INTERNAL HP9895 HP82901 HP82902 HP8290X
device selector	integer constant; when only INTERNAL is specified with no device selector, the default value is 4.	(see Glossary)
unit number	integer constant; Default = 0	0 thru 255 (often device dependent)

Example Statements

```
COPY "OLD_FILE" TO "New_file"
COPY File$ TO File$&Msus$
COPY ":INTERNAL,4,0" TO ":INTERNAL,4,1"
COPY Int_disc$ TO Ext_disc$
```

Semantics

Copying a File

The contents of the old file is copied into the new file, and a directory entry is created. A protect code, to prevent accidental erasure, may be specified for the new file. The old file and new file may exist on the same device, but the new file name must be unique.

COPY is canceled and an error returned when there is not enough room on the destination device.

If the mass storage unit specifier (msus) is omitted from a file specifier, the MASS STORAGE IS device is assumed.

Copying the Entire Disc

Discs may be duplicated between identically sized media or when the destination media is larger. Any attempt to copy from a larger capacity media to a smaller capacity media generates an error.

When copying a disc, msus's must be specified and unique. File names are not allowed. Disc-to-disc copy time is dependent on media type and interleave factors.

HP 9836 CRT Enhancements

The HP 9836 CRT display has all of the features of the HP 9826 but also has additional capabilities. The 9836 CRT has an 80-character screen width, while the 9826 has a 50-character width. The 9836 CRT has the ability of underlining characters, making them blink, and displaying the characters in inverse video. Access to these enhancement features is discussed in these sections, which may be added to Chapter 8 of the *BASIC Interfacing Techniques* manual.

Display-Enhancement Characters

The HP 9836 CRT also has the ability of displaying underlined, blinking, and inverse-video characters. These features are accessed by displaying special characters. Both the Output Area and the Display Line have these abilities.

There are eight special bit patterns that control the use of these features. `CHR$(128)` through `CHR$(135)` cause the following display actions.

Character Code	Action Resulting from Displaying the Character
128	All enhancements off.
129	Inverse mode on.
130	Blinking mode on.
131	Inverse and Blinking modes on.
132	Underline mode on.
133	Underline and Inverse modes on.
134	Underline and Blinking modes on.
135	Underline, Inverse, and Blinking modes on.

When one of these characters is sent to the CRT, it turns on the corresponding enhancement(s). All subsequent characters on the CRT are also displayed in the specified enhancement mode; if only a few characters are to be enhanced, a CHR$(128) must be sent to the display after the last character to be enhanced, which turns off **all** enhancements.

From the preceding table, you may have deduced that certain bits within the character bytes turn on these display modes. The following bit pattern and individual bits control these features.

Most Significant Bit | Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	0	0	0	Underline On	Blinking On	Inverse On
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Notice that the upper five bits (7 through 3) must be in the pattern shown (numeric value = 128). Thus, adding the values 4, 2, or 1 enable the Underline, Blinking, and Inverse features. Several examples follow.

```
100    PRINTER IS 1
110    Off=128
120    Underline=4
130    Blinking=2
140    Inverse=1
150    !
160    PRINT CHR$(Off);"Normal"
170    PRINT
180    PRINT CHR$(128+Inverse);"Inverse"
190    PRINT "carries over onto"
200    PRINT "subsequent lines"
210    PRINT
220    PRINT CHR$(128+Underline);"Underline"
230    PRINT "also remains on until turned off"
240    PRINT
250    PRINT CHR$(128+Blinking);"Blinking"
260    PRINT "is the same"
270    PRINT
280    PRINT CHR$(Off);"Back to normal"
290    PRINT
300    END
```

These same features can also be placed in strings by using the ANY CHAR key while initially assigning the string variable its value. Keep in mind that, even though these characters are not shown on the screen, they are counted in the length of the string. Dimension string variables accordingly.

Determining Screenwidth

All programs written and stored on the 9826 can be run on the 9836, and vice versa. Programs that use the display extensively have probably been written with the 50-character screenwidth in mind. Since all programs are transportable between these computers, the program should have the ability to distinguish in which computer it is being executed. The BASIC language system provides this capability.

Interface select code 1 is used to access the CRT from BASIC programs. Several registers are associated with this interface which allow interrogating and controlling the CRT through its interface. In particular, STATUS register 9 of the CRT interface is dedicated to storing the current screenwidth. The following statement is an example of determining the current screenwidth.

```
STATUS 1,9;Screenwidth
```

The resultant value of Screenwidth differs for each computer: the value of 80 is returned in the 9836, and 50 is returned in the 9826.

Enhanced Keyboard Control

This section is to be added to Chapter 9, ''The Internal Keyboard Interface'', of the *BASIC Interfacing Techniques* manual.

Enhanced Keyboard Control

Normally, the BASIC operating system handles all keyboard inputs. Several BASIC statements allow programs to handle inputs from the keyboard; examples are the INPUT, LINPUT, ENTER, ON KEY, and ON KNOB statements. Additional keyboard statements provide BASIC programs with a means of intercepting both ASCII and non-ASCII keystrokes for processing by the program. The statements are:

`ON KBD`	sets up and enables keystrokes to be trapped.
`ON KBD,ALL`	includes PAUSE, STOP, CLR I/O, and softkeys.
`KBD$`	returns keystrokes trapped in the buffer.
`OFF KBD`	resumes normal keystroke processing.

ON KBD allows terminal emulation, keyboard masking, and special data inputs. Each keystroke produces unique code(s) that allow the program to differentiate between different keys being pressed. The program can also determine whether the SHIFT or CTRL keys are being pressed with a particular key, but these keystrokes cannot be detected by themselves. Also, the RESET key cannot be trapped by ON KBD.

Trapping Keystrokes

The ON KBD statement sets up a branch that is initiated when the keyboard buffer becomes "non-empty". The service routine may then interrogate the buffer as desired, processing the keystrokes as determined by the program. The HP 9826's keyboard buffer may contain up to 100 characters, while the HP 9836's buffer may contain up to 160 characters. Calling the KBD$ function does two things: it returns all keystrokes trapped since the last time the buffer was read and then clears the keyboard buffer.

The following program uses ON KBD, KBD$, and OFF KBD to trap and process keystrokes, rather than allowing the operating system to do the same. The program defines each keystroke to print a complete word.

```
100     OPTION BASE 1
110     DIM String$(26)[6]
120     READ String$(*)
130     !
140     DATA A,BROWN,CAT,DOG,EXIT,FOX,GOT
150     DATA HI,IN,JUMPS,KICKED,LAZY,MY
160     DATA NO,OVER,PUSHED,QUICK,RED,SMART
170     DATA THE,UNDER,VERY,WHERE,XRAY,YES,ZOO
180     !
190     PRINTER IS 1
200     PRINT "Many ASCII keys have been"
210     PRINT "defined to produce words."
220     PRINT
```

```
230    PRINT "Press the following keys."
240    PRINT "T Q B F J O T L D ."
250    !
260    ON KBD GOSUB Process_keys
270    !
280    LOOP
290      EXIT IF Word$="EXIT"
300    END LOOP
310    !
320    STOP
330    !
340 Process_keys: Key$=KBD$   ! Read buffer.
350    !
360    REPEAT ! Process ALL keys trapped.
370      Key_code=NUM(Key$[1;1])! Calculate code.
380      !
390      SELECT Key_code          ! Choose response.
400      !
410        CASE 65 TO 90          ! CASE "A" TO "Z",
420          Word$=String$(Key_code-64)
430          Key$=Key$[2]         ! Remove processed key.
440          !
450        CASE 97 TO 122         ! CASE "a" TO "z".
460          Word$=String$(Key_code-96)
470          Key$=Key$[2]         ! Remove processed key.
480          !
490        CASE 255               ! CASE non-ASCII key.
500          IF Key$[2;1]<>CHR$(255) THEN
510            Word$=Key$[1,2]    ! Non-ASCII key alone,
520            Key$=Key$[3]       ! so take 2 codes.
530          ELSE
540            Word$=Key$[1,3]    ! Non-ASCII w/ CTRL,
550            Key$=Key$[4]       ! so take 3 codes.
560          END IF
570        CASE ELSE              ! CASE all others.
580          Word$=""
590          Key$=Key$[2]         ! Remove processed key.
600          !
610      END SELECT
620      !
630                               ! Execute response.
640      Defined=LEN(Word$)<>0
650      IF Defined THEN
660        PRINT Word$;" ";
670        DISP
680      ELSE
690        BEEP 100,.05
700        DISP "Key undefined."
710      END IF
```

```
720    !
730    UNTIL LEN(Key$)=0  ! Until ALL Keys processed.
740    !
750    RETURN
760    !
770 Quit:    END
```

Notice that all non-ASCII keys produce two-character sequences: CHR$(255) followed by an ASCII character. Pressing the (CTRL) key with non-ASCII keys produce three-character sequences: another CHR$(255) character preceding the two-character sequence produced by pressing the non-ASCII key by itself. See the tables in "Outputs to the Keyboard" for a listing of the sequences produced by non-ASCII keys.

BASIC programs can output ASCII keystrokes to the keyboard, via `OUTPUT 2`, without initiating an ON KBD branch; however, outputting non-ASCII "closure" keys will initiate the ON KBD branch. For example, executing the following statement (in a program line):

```
    OUTPUT 2;"32*2";CHR$(255);"E";"KBD";
```

causes the characters `KBD` which follow the closure key to be placed in the KBD$ buffer, which also initiates the ON KBD branch. The (EXECUTE)-key sequence which was sent to the keyboard executes the numeric expression `32*2` before the branch is initiated. This type of operation may result in unpredictable results and is therefore not recommended while ON KBD is in effect.

ON KBD branching is disabled by DISABLE, deactivated by OFF KBD, and temporarily deactivated when the program is executing `LINPUT`, `INPUT`, or `ENTER 2` statements.

Softkeys and Knob Rotation

When ON KNOB is not in effect, knob rotation is also trapped by ON KBD. Rotation will produce the "cursor" keystrokes; clockwise rotation produces CHR$(255)followed by "^", while counter-clockwise rotation produces CHR$(255) followed by "V".

`ON KBD,ALL` allows softkey trapping ("overrides" `ON KEY`) but does not change the softkey labels.

Disabling Interactive Keyboard

Another group of statements is used to disable the interactive keyboard functions:

`SUSPEND INTERACTIVE`	ignores the (EXECUTE), (PAUSE), (STOP), (STEP), and (CLR I/O) keys.
`SUSPEND INTERACTIVE,RESET`	ignores (RESET) too.
`RESUME INTERACTIVE`	returns to normal operation

SUSPEND INTERACTIVE can be used to prevent interruption of programs which gather data or which control other systems.

Special care should be taken when using SUSPEND INTERACTIVE,RESET. If an "infinite loop" is executed while interactive keyboard functions are disabled, only the power switch will stop execution of the program.

```
110     ! This program cannot be stopped by
120     ! PAUSE, STOP, OR RESET
130     ! before its normal completion
140     !
150     !
160     SUSPEND INTERACTIVE,RESET ! ignore Keyboard
170     !
180     PRINT "COUNTDOWN IS "
190     PRINT
200     I=10                        ! Initial value.
210       REPEAT
220         PRINT " T minus ";I     ! Print count.
230         I=I-1                   ! Decrement count.
240         WAIT 1                  ! Wait one second.
250       UNTIL I<0
260    !
270    PRINT
280    BEEP 100,1
290    PRINT "Done"
300    RESUME INTERACTIVE           ! Return to normal.
310    !
320    END
```

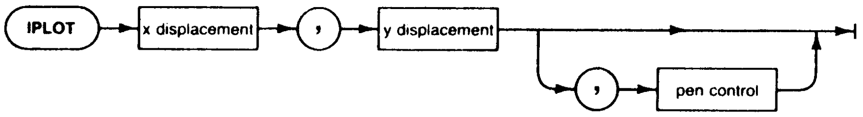
Powerfail Protection

This chapter may be added to the *BASIC Interfacing Techniques* manual.

IPLOT

Minimum Requirement	BASIC 2.0
Keyboard Executable	Yes
Programmable	Yes
In an IF... THEN	Yes

This statement moves the pen from the current pen position to a position calculated by adding the specified X and Y displacements to the current pen position using the current line type.



Item	Description/Default	Range Restrictions
x displacement	numeric expression, in current units	—
y displacement	numeric expression, in current units	—
pen control	numeric expression, rounded to an integer; Default = 1 (down after move)	−32768 thru 32767

Example Statements

```
IPLOT 0,Up,-1
IPLOT Right,0,Down_after_move
```

Semantics

The specified X and Y displacement information is interpreted according to the current unit-of-measure.

The line is clipped at the current clipping boundary. The PIVOT statement rotates the coordinates for the IPLOT, but the logical pen position receives the value of the unpivoted coordinates. The logical pen may bear no obvious relationship to the physical pen's position.

If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR.

The optional pen control parameter specifies following plotting actions; the default value is + 1 (down after move).

Pen Control	Resultant Action
− Even	Up before move
− Odd	Down before move
+ Even	Up after move
+ Odd	Down after move

KBD$

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This function returns the contents of the keyboard buffer.

Example Statements

```
Keys$=KBD$
Process$=Process$&KBD$
```

Semantics

When an ON KBD statement is in effect, all subsequent keystrokes are trapped and held in the keyboard buffer. The first keystroke also initiates the branch to the service routine. The KBD$ function returns the keystrokes and clears the buffer. A null string is returned if the buffer is empty or the ON KBD statement has not been executed.

Non-ASCII keys are stored in the buffer as two bytes; the first byte has the decimal value of 255, and the second specifies the key. Pressing CTRL and a non-ASCII key simultaneously generates three bytes; the first two bytes have the decimal value of 255, and the third specifies the key. The tables on pages 306-307 in the "Useful Tables" Appendix show the code and character produced by each non-ASCII key.

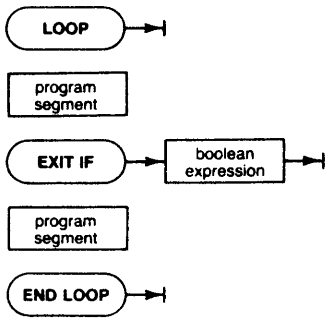
The 9826's keyboard buffer can hold up to 100 keystrokes, while the 9836's can hold up to 160. Further keystrokes are not saved and produce beeps. An overflow flag is set after the buffer is full, which is cleared by KBD$, SCRATCH A, and RESET. The flag can be checked by reading STATUS register 5 of interface select code 2.

The keyboard buffer is cleared by OFF KBD, INPUT, LINPUT, ENTER 2, RESET, RUN, STOP, END, SCRATCH, SCRATCH A, LOAD, and GET.

LOOP

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the boolean expression in an EXIT IF statement evaluates to be logically true (evaluates to a non-zero value).



Item	Description/Default	Range Restrictions
boolean expression	numeric expression; evaluated as true if non-zero and false if 0	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
460   LOOP
470     DISP "Looping"
480   END LOOP

390   LOOP
400     PRINT Count
410     EXIT IF Count>=Max
420     Count=Count+1
430   END LOOP
```

Semantics

The LOOP...END LOOP construct allows continuous looping with conditional exits which depend on the outcome of relational tests placed within the program segments. The program segments to be repeated start with the LOOP statement and end with END LOOP. Reaching the END LOOP statement will result in a branch to the first program line after the LOOP statement.

Any number of EXIT IF statements may be placed within the construct to escape from the loop. The only restriction upon the placement of the EXIT IF statements is that they must not be part of any other construct which is nested within the LOOP...END LOOP construct.

If the specified conditional test is true, a branch to the first program line following the END LOOP statement is performed. If the test is false, execution continues with the next program line within the construct.

Branching into a LOOP...END LOOP construct (via a GOTO) results in normal execution from the point of entry. Any EXIT IF statement encountered will be executed. If execution reaches END LOOP, a branch is made back to the LOOP statement, and execution continues as if the construct had been entered normally.

Nesting Constructs Properly

LOOP...END LOOP may be placed within other constructs, provided it begins and ends before the outer construct can end.

OFF KBD

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement cancels the event-initiated branch previously defined by an ON KBD statement.

Example Statements

```
OFF KBD
IF NOT Process_Keys THEN OFF KBD
```

Semantics

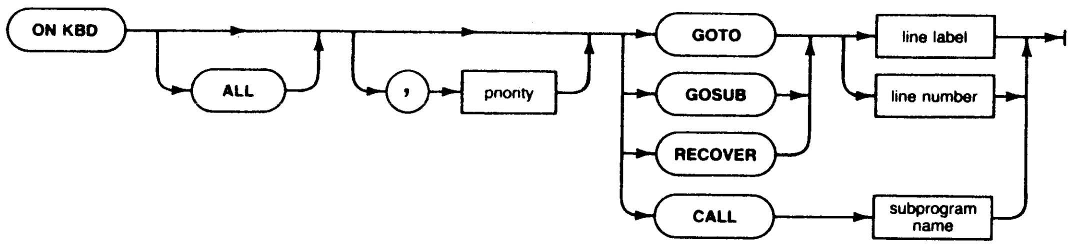
When this statement is executed, any pending ON KBD branch is cancelled, and the keyboard buffer is cleared.

If OFF KBD is executed in a subprogram such that it cancels an ON KBD in the calling context, the cancelled ON KBD definition is restored when the calling context is restored. However, the keyboard buffer's contents are not restored with the calling context, because the buffer was cleared with the OFF KBD.

ON KBD

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement defines an event-initiated branch which occurs when a key is pressed.



Item	Description/Default	Range Restrictions
priority	numeric expression, rounded to an integer	1 thru 15
line label	name of program line	any valid name
line number	integer numeric constant identifying a program line	1 thru 32766
subprogram name	name of a SUB subprogram	any valid name

Example Statements

```
ON KBD GOSUB Get_Keys
ON KBD,9 CALL Process_Keys
```

Semantics

Specifying the secondary keyword ALL causes all keys except (RESET), (SHIFT), and (CTRL) to be trapped. When ALL is omitted, the softkeys, (PAUSE), (STOP), and (CLR I/O) keys retain their normal functions. When the softkeys are trapped, ON KBD branching will override ON KEY branching.

The first keystroke triggers a keyboard interrupt and initiates a branch to the service routine when priority allows. Subsequent keystrokes are stored in the buffer until read with the KBD$ function. Any keystrokes made after reading the keyboard buffer will be entered into the buffer, and the branch will be initiated when priority allows.

Knob rotation will generate ON KBD interrupts unless an ON KNOB statement has been executed. Clockwise rotation of the knob produces Up-Arrow keystrokes; counterclockwise rotation produces, Down-Arrow keystrokes. Since one rotation of the knob is equivalent to 20 keystrokes, keyboard buffer overflow may occur if the BASIC service routine does not process the keys rapidly.

Live keyboard, editing, and display control functions are suspended during ON KBD. To restore a key's normal function, the keystroke may be OUTPUT to select code 2.

The most recently executed ON KBD definition replaces any previous definitions, except when changing program segments.

The priority can be optionally specified, with highest priority represented by 15. The highest priority is still less than the priority for ON ERROR, ON END, and ON TIMEOUT. ON KBD can interrupt other ON INTR, ON KNOB, ON KEY, or ON KBD service routines if the ON KBD priority is higher than the priority of the current routine. CALL and GOSUB service routines get the priority specified in the ON KBD statement which set up the branch that invoked them. The system priority is not changed when a GOTO branch is taken.

CALL and GOSUB will return to the line immediately following the one during which the interrupt occurred. RECOVER forces the program to return directly to the context in which the ON KBD was defined.

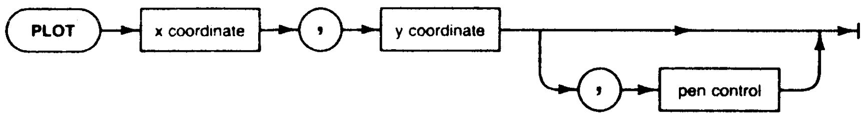
CALL and RECOVER remain active when the context changes to a subprogram, unless the change in context is caused by a keyboard-originated CALL. GOSUB and GOTO remain active when the context changes to a subprogram, but the branch is not initiated until the calling context is restored.

ON KBD is disabled by DISABLE, deactivated by OFF KBD, and temporarily deactivated when the program is executing LINPUT, INPUT, or ENTER 2.

PLOT

Minimum Requirement	BASIC 2.0
Keyboard Executable	Yes
Programmable	Yes
In an IF...THEN...	Yes

This statement moves the pen from the current pen position to the specified X and Y coordinate position using the current line type.



Item	Description/Default	Range Restrictions
x coordinate	numeric expression, in current units	—
y coordinate	numeric expression, in current units	—
pen control	numeric expression, rounded to an integer; Default = 1 (down after move).	−32768 thru +32767

Example Statements

```
PLOT X,Y,Down_before
PLOT 0,100
```

Semantics

The specified X and Y coordinate information is interpreted according to the current unit-of-measure.

The line is clipped at the current clipping boundary. The PIVOT statement rotates the coordinates for the PLOT, but the logical pen position receives the value of the unpivoted coordinates. The logical pen may bear no obvious relationship to the physical pen's position.

If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling.
Note 2: The starting point for labels drawn after other labels is affected by LDIR

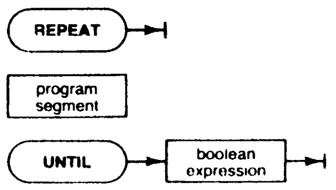
The optional pen control parameter specifies following plotting actions; the default value is + 1 (down after move).

Pen Control	Resultant Action
− Even	Up before move
− Odd	Down before move
+ Even	Up after move
+ Odd	Down after move

REPEAT...UNTIL

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the boolean expression in the UNTIL statement evaluates to be logically true (evaluates to non-zero).



Item	Description/Default	Range Restrictions
boolean expression	numeric expression; evaluated as true if non-zero and false if zero	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
530    REPEAT
540      PRINT Count
550      Count=Count+1
560    UNTIL Count>10


590    REPEAT
600      Char$=CHR$(65+RND*26)
610      PRINT Char$;
620    UNTIL Char$="N"
```

Semantics

The REPEAT...UNTIL construct allows program execution dependent on the outcome of a relational test performed at the **end** of the loop. Execution starts with the first program line following the REPEAT statement, and continues to the UNTIL statement where a relational test is performed. If the test is false a branch is made to the first program line following the REPEAT statement.

When the relational test is false, program execution continues with the first program line following the UNTIL statement.

Branching into a REPEAT...UNTIL construct (via a GOTO) results in normal execution up to the UNTIL statement, where the test is made. Execution will continue as if the construct had been entered normally.

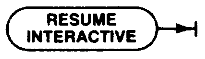
Nesting Constructs Property

REPEAT...UNTIL constructs may be nested within other constructs provided the inner construct begins and ends before the outer construct can end.

RESUME INTERACTIVE

Minimum Requirement	BASIC 2.0
Keyboard Executable	Yes[1]
Programmable	Yes
In an IF...THEN...	Yes

This statement enables the (EXECUTE), (PAUSE), (STOP), (STEP), (CLR I/O), and (RESET) after a SUSPEND INTERACTIVE statement.



Example Statements

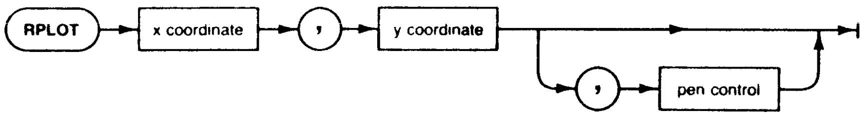
```
RESUME INTERACTIVE
IF Kbd_flag THEN RESUME INTERACTIVE
```

1 This statement is executable from the keyboard, but only while SUSPEND INTERACTIVE is **not** in effect.

RPLOT

Minimum Requirement	BASIC 2.0
Keyboard Executable	Yes
Programmable	Yes
In an IF... THEN	Yes

This statement moves the pen from the current pen position to the specified X and Y coordinate position relative to a **relative origin** using the current line type.



Item	Description/Default	Range Restrictions
x relative coordinate	numeric expression, in current units	—
y relative coordinate	numeric expression, in current units	—
pen control	numeric expression, rounded to an integer; Default = 1 (down after move).	−32768 thru +32767

Example Statements

```
RPLOT Rel_x,Rel_y
RPLOT 0,Up,Down_before_move
```

Semantics

The specified X and Y coordinates are interpreted according to the current unit-of-measure from the **current relative origin**; the current relative origin is the last point resulting from any of the following statements:

```
AXES      DRAW      FRAME     GRID      IDRAW
IMOVE     IPLOT     LABEL     MOVE      PLOT
```

The current relative origin is not changed by the RPLOT statement.

The line is clipped at the current clipping boundary. The PIVOT statement rotates the coordinates for the RPLOT, but the logical pen position receives the value of the unpivoted coordinates. The logical pen may bear no obvious relationship to the physical pen's position. If none of the line is inside the current clipping limits, the pen is not moved, but the logical pen position is updated.

Applicable Graphics Transformations

	Scaling	PIVOT	CSIZE	LDIR
Lines (generated by moves and draws)	X	X		
Characters (generated by LABEL)			X	X
Axes (generated by AXES & GRID	X			
Location of Labels	Note 1			Note 2

Note 1: The starting point for labels drawn after lines or axes is affected by scaling
Note 2: The starting point for labels drawn after other labels is affected by LDIR

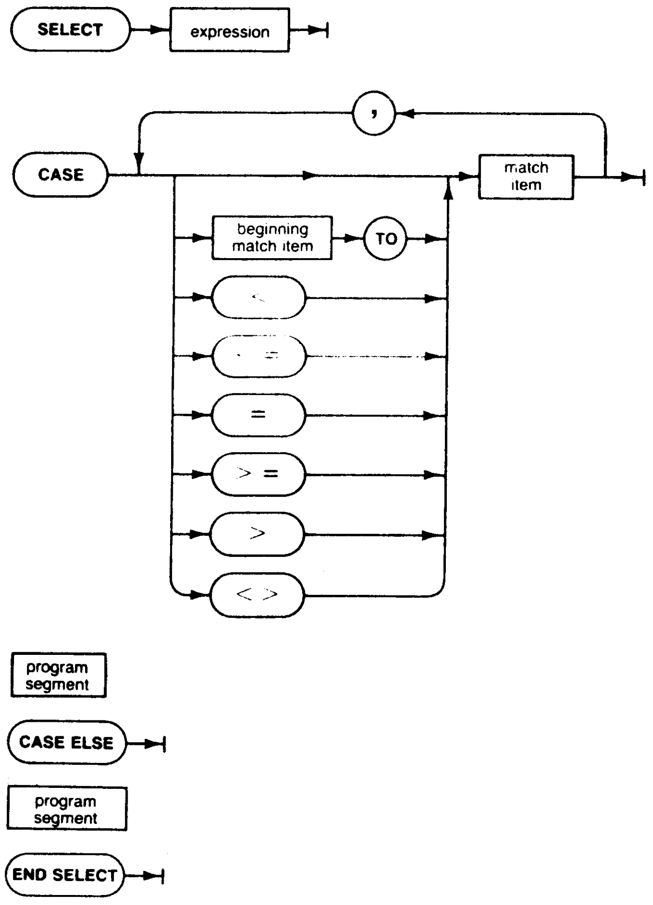
The optional pen control parameter specifies following plotting actions; the default value is + 1 (down after move).

Pen Control	Resultant Action
− Even	Up before move
− Odd	Down before move
+ Even	Up after move
+ Odd	Down after move

SELECT...CASE

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct provides conditional execution of one program segment of several.



Item	Description/Default	Range Restrictions
expression	a numeric or string expression	—
match item	a numeric or string expression optionally preceded by a relational operator	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
650    SELECT Expression
660      CASE <0
670        PRINT "Negative number"
680      CASE ELSE
690        PRINT "Non-negative number"
700    END SELECT

750    SELECT Expression$
760      CASE "A" TO "Z"
770        PRINT "Uppercase alphabetic"
780      CASE ":",";",",","."
790        PRINT "Punctuation"
800    END SELECT
```

Semantics

SELECT...END SELECT is similar to the IF...THEN...ELSE...END IF construct, but allows **several** conditional program segments to be defined; however, **only one segment will be executed** each time the construct is entered. Each segment starts after a CASE or CASE ELSE statement and ends when the next program line is a CASE, CASE ELSE, or END SELECT statement.

The SELECT statement specifies an expression, whose value is compared to the list of values found in each CASE statement. When a match is found, the corresponding program segment is executed. The remaining segments are skipped and execution continues with the first program line following the END SELECT statement.

All CASE expressions must be of the same type, (either string or numeric) and must agree in type with the corresponding SELECT statement expression.

The optional CASE ELSE statement defines a program segment to be executed when the selected expression's value fails to match any CASE statement's list.

Branching into a SELECT...END SELECT construct (via GOTO) results in normal execution until a CASE or CASE ELSE statement is encountered. Execution then branches to the first program line following the END SELECT statement.

Errors encountered in evaluating CASE statements will be reported as having occurred in the corresponding SELECT statement.

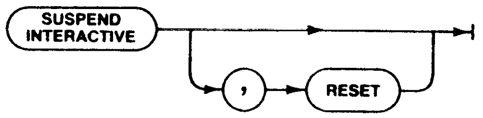
Nesting Constructs Properly

SELECT...END SELECT constructs may be nested, provided inner construct begins and ends before the outer construct can end.

SUSPEND INTERACTIVE

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	Yes

This statement disables the EXECUTE, PAUSE, STOP, STEP, CLR I/O, and (optionally) RESET key functions.



Example Statements

```
SUSPEND INTERACTIVE,RESET
IF NOT Kbd_flag THEN SUSPEND INTERACTIVE
```

Semantics

Execution of a PAUSE statement, a TRACE PAUSE statement, or a fatal execution error temporarily restores the suspended key functions. CONTINUE after a PAUSE will again suspend their normal operation.

SUSPEND INTERACTIVE is canceled by RESUME INTERACTIVE, STOP, END, RUN, SCRATCH, GET, or RESET.

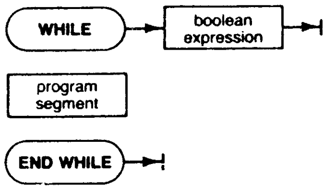
Note

Specifying the RESET key will prevent you from stopping a program before it ends.

WHILE

Minimum Requirement	BASIC 2.0
Keyboard Executable	No
Programmable	Yes
In an IF...THEN...	No

This construct defines a loop which is repeated until the boolean expression in the WHILE statement evaluates to true (evaluates to a non-zero value).



Item	Description/Default	Range Restrictions
boolean expression	numeric expression; evaluated as true if non-zero and false if zero.	—
program segment	any number of contiguous program lines not containing the beginning or end of a main program or subprogram, but which may contain properly nested construct(s).	—

Example Program Segments

```
840   WHILE Boolean=True
850     DISP "Boolean is still true"
860     Boolean=INT(.1+RND)
870   END WHILE
880   DISP "Boolean is no longer true"

920   WHILE NOT True
930     PRINT "Not true"
940   END WHILE
```

Semantics

The WHILE...END WHILE construct allows program execution dependent on the outcome of a relational test performed at the **start** of the loop. If the condition is true, the program segment between the WHILE and END WHILE statements is executed and a branch is made back to the WHILE statement. The program segment will be repeated until the test is false. When the relational test is false, the program segment is skipped and execution continues with the first program line after the END WHILE statement.

Branching into a WHILE...END WHILE construct (via a GOTO) results in normal execution up to the END WHILE statement, a branch back to the WHILE statement, and then execution as if the construct had been entered normally.

Nesting Constructs Properly

WHILE...END WHILE constructs may be nested within other constructs, provided the inner construct begins and ends before the outer construct can end.

Chapter 14
Powerfail Protection

The HP 9826 and 9836 have the optional capability of up to about one minute of powerfail protection. This feature is available as Option 050 on either computer. This chapter describes the capabilities provided by this optional internal interface, which has been **permanently assigned to interface select code 5**.

This optional feature is discussed in this Interfacing Techniques manual because of the nature of its access from BASIC programs. If you need additional explanation regarding interface registers or interface interrupt events, refer to Chapters 6 and 7 of this manual, respectively.

Overview of Capabilities Provided

The powerfail protection provided by the internal battery-backup circuitry is as follows.

- Up to one minute of operation **after powerfail** may be specified.
- The interface may optionally interrupt the computer when a powerfail has occurred. A delay time before interrupt may also be programmed to allow the computer to ignore power "glitches".
- The program can read both the powerfail interrupt cause and determine current powerfail status information, including ac power status, battery time remaining, and time elapsed since power was returned.
- The real-time clock and 64 bytes of memory registers are maintained after power has been down for greater than one minute.

The Computer's Reaction to Powerfails

There are two general categories of computer reactions to powerfail situations. The default response is to continue running as before the failure for up to one minute. The alternate response is to interrupt the current routine's execution to service the failure. In either case, the computer beeps and the following warning message is displayed on the CRT when the powerfail is detected.

```
Power failed
```

If power remains off for more than one minute, or if the computer turns itself off, only a real-time clock and 64 bytes of low-power memory registers are maintained. If power is restored, the computer powers on in its normal powerup sequence.

Continuous-Memory Registers

The sixty-four, single-byte registers on the interface are maintained after power has failed. The contents of these registers can be written with CONTROL statements and read with STATUS statements. The registers are numbered 8 through 71.

Real-Time Clock

The clock on the powerfail interface is read at powerup and is used to set the BASIC system clock. However, the system clock, not the powerfail clock, is read by the `TIMEDATE` function.

Executing either `SET TIME` or `SET TIMEDATE` sets **both** clocks to the specified value. Thus, the two clocks may drift apart temporarily but may be synchronized by setting time with either of these statements. See Chapter 10 of *BASIC Programming Techniques* for further detail.

Powerfail-Protection Timers

Three additional timers are used by the interface to keep track of times between different powerfail events. These timers allow the program to keep track of Powerfail events so that the desired service response may be initiated.

When a powerfail occurs, the **Powerfail Timer** is cleared and begins to count the seconds elapsed since the powerfail occurred. After waiting the **Powerfail Delay Time**, the interface may generate a Powerfail interrupt, if enabled to do so. If and when the Powerfail Timer timer reaches the value of the **Protection Time**, the computer automatically powers down.

When power is returned, the **Power Back Timer** is cleared and begins counting seconds elapsed since the power back occurred. When this timer reaches the value of the **Power Back Delay**, the computer is no longer in the Powerfail State; a Power Back interrupt is generated, if enabled.

When a powerfail occurs, the **Overheat Protection Timer** begins to increment, counting the seconds elapsed since the powerfail event occurred. When power is restored, this timer is **decremented one second for every two seconds that power is back**. If power remains on long enough, the timer decrements to 0. However, if the timer reaches 60 seconds, the computer automatically powers down. These actions ensure that the fan adequately cools the computer during continuous power fluctuations.

Further description of delay times, timer actions, and enabling interrupt events are described in the remainder of this chapter.

Interrupt Events

Interrupts can be generated by the powerfail-protection controller when **three different events are sensed**: when power fails, when power is returned, and when approximately one second of battery power remains. Enabling these events to initiate interrupts and typical responses to these events are explained in this and in the following section.

Setting Up and Enabling Interrupts

The desired interrupt condition(s) may be enabled by specifying the appropriate numeric mask value. The bits of the Interrupt Enable register enable the following interrupts.

Powerfail Interrupt Enable Mask

Most Significant Bit — Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used					One Second Left	Power Is Back	Power Has Failed
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

One Second Left — When this bit is set (1), an interrupt to the computer is generated when approximately one second of battery power remains.

Power Is Back — When this bit is set (1), an interrupt to the computer is generated when power has been returned (after a previous powerfail).

Power Has Failed — When this bit is set (1), an interrupt to the computer is generated when a powerfail has been detected.

The branch to the powerfail service routine is set up and enabled in the same manner as are other interrupt service routines. A typical example is as follows.

```
200   ON INTR 5 GOSUB Power_down
210   Mask=1  ! Enable Powerfail Interrupt.
210   ENABLE INTR 5;Mask
```

Service Routines

The service routine must determine which type of event initiated the interrupt branch. The bits of the Interrupt Cause register have the same definitions as those of the Interrupt Enable Mask register.

STATUS Register 1 **Powerfail Interrupt Cause**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used					One Second Left	Power Is Back	Power Has Failed
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

```
100    STATUS 5,1;Interrupt_cause
```

If more than one interrupt cause has occurred, more than one bit will be set in this register. Also, the register's contents must be stored in a variable which is not used until all causes have been determined, because **reading this register clears its contents**.

Also keep in mind that when the "One Second Left" bit is a 1, the computer will power down **regardless** of whether on not power is back before the end of the one second.

The action performed by the service routine is usually to store critical data. The internal disc drives remain fully operational for this purpose. External drives usually lose power when the computer loses its power; if so, they should not be used for this purpose. Other external devices may also be affected by the failure and therefore may not respond to the request to transfer the data. Therefore, all attempts to communicate with external devices should have ON TIMEOUT branches set up and enabled so that the program will not spend the entire minute waiting for the device to respond.

Powerfail Status and Timers

The Powerfail Status register and Timer registers provide useful information describing the state of computer power. The following example service routine reads these STATUS registers and displays the information on the CRT.

```
100    ON INTR 5 GOSUB Pfail_service
110    ENABLE INTR 5;7   ! Enable all three causes.
120    !
130    Pback_delay=300   ! Delay 3 s before Pback interrupt.
140    Protection=2000   ! 20 s max. of Pfail protection.
150    Pfail_delay=100   ! Delay 1 s before Pfail interrupt.
160    CONTROL 5,5;Pback_delay,Protection,Pfail_delay
170    !
180    LOOP
190      CONTROL 1;1,1    ! Upper-left corner.
200      OUTPUT 1;Number
210      Number=Number+1
220    END LOOP
```

```
230    !
240 Pfail_service: CONTROL 1;1,3  ! Begin on third line.
250                OUTPUT 1;"  Powerfail Interface Registe
rs"
260                OUTPUT 1;" ----------------------------
---"
270                !
280    REPEAT
290      CONTROL 1;1,5  ! Begin printing on line 5.
300      STATUS 5,3;Pf_status
310        Pfail=BIT(Pf_status,0)
320        Ac_down=BIT(Pf_status,1)
330        Batt_on=BIT(Pf_status,2)
340        One_sec=BIT(Pf_status,3)
350        S_test=BIT(Pf_status,7)
360        OUTPUT 1
370        OUTPUT 1;"STATUS Register 3 - Powerfail Status:"
380        OUTPUT 1;" Test Fail    1 Sec.    Batt. On    Ac
Down   In Pfail"
390        OUTPUT 1 USING "#,5X,D,5X";S_tst,One_sec,Batt_on
,Ac_down,Pfail
400        OUTPUT 1 USING "/"
410        !
420      STATUS 5,4;Ovheat
430        OUTPUT 1;"STATUS Register 4 - Overheat Timer:  "
;
440        OUTPUT 1 USING "DD.D,/";Ovheat/100
450        !
460      STATUS 5,5;Pback
470        OUTPUT 1;"STATUS Register 5 - Power Back Timer:"
;
480        OUTPUT 1 USING "DD.D,/";Pback/100
490        !
500      STATUS 5,6;Pf_timer
510        OUTPUT 1;"STATUS Register 6 - Powerfail Timer: "
;
520        OUTPUT 1 USING "DD.D,/";Pf_timer/100
530        !
540      STATUS 5,4;Ov_heat
550    UNTIL Ov_heat=0     ! UNTIL Overheat timer expires.
560    !
570    ENABLE INTR 5             ! Use same mask.
580    RETURN
590    !
600    END
```

Type in and run the program. Alternately remove and replace the power cord while watching the status values and timers change. You are highly encouraged to experiment with the parameters until you are familiar with how the computer responds to power failures. The next section presents several simple examples of service routines.

Typical Service Routines

The Powerfail Protection option allows programming several types of service responses. A few typical examples are shown in this section. All STATUS and CONTROL registers are summarized at the end of the chapter.

Using the Continuous-Memory Registers

The most common function of service routines is to store any critical data and then turn the computer off to conserve battery power. The following simple example shows the use of the continuous-memory registers for storing a message.

```
100    ON INTR 5 GOTO Pfail_serve
110    ENABLE INTR 5;1   ! Pfail interrupts only.
120    !
130    ! Use defaults of: 500 ms    Pback Delay,
140    !                  60    s    Protection Time,
150    !                  100 ms    Pfail Delay.
160    !
170    LOOP
180      DISP Number
190      Number=Number+1
200    END LOOP
210    !
220    STOP
230    !
240  Pfail_serve: ! Write message in Cont-Mem. Registers.
250                !
260      Message$="Adios, amigos."
270      Message$=Message$&CHR$(10) ! Add LF.
280      No_bytes=LEN(Message$)
290      !
300      FOR Reg=8 TO 8+No_bytes-1
310        CONTROL 5,Reg;NUM(Message$[Reg-7;1])
320      NEXT Reg
330      !
340      CONTROL 5;1 ! Shut down when finished.
350      !
360    END
```

Type in the program and press (RUN). The CRT shows a counter running continuously. Unplugging the power cord initiates the Powerfail interrupt after the default delay of 100 milliseconds. Thus, if power had failed for a duration of less than 100 milliseconds, the interrupt would not have been generated. Similarly, the Power Back Delay determines how long the computer will delay after power has been restored before generating a Power Back interrupt, when enabled.

The program did not allow the Powerfail Timer to reach the default Protection Time (60 seconds). Instead, it powered itself down after storing a message in the registers in order to save battery power. If power is subsequently restored, the computer powers on in the normal powerup sequence. If an Autostart routine exists, it will be run automatically.

The following program shows a method for reading the message stored in the continuous-memory registers by the preceding program. The program makes use of the fact that the message was terminated by a line-feed character, CHR$(10).

```
100    PRINTER IS 1
110    !
120    ! Read message in Continuous-Memory Registers.
130    DIM Registers$[64],Message$[64]
140    !
150    FOR Register=8 TO 71        ! Read all 64 registers.
160      STATUS 5,Register;Byte
170      Registers$[Register-7]=CHR$(Byte)
180    NEXT Register
190    !
200    ENTER Registers$;Message$ ! Enter and stop at LF.
210    !
220    PRINT Message$
230      !
240    END
```

Storing Data on Disc

Service routines can be programmed to take many other actions, such as to store data on an internal disc. The following program shows a technique for storing the ALPHA and GRAPHICS displays and the value of the clock at the time the powerfail occurred.

```
100    INTEGER Crt_graphics(1:12480) ! (1:7500) for 9826.
110    DIM Crt_alpha$(1:57)[80]       ! [50] for 9826.
120    !
130    ON INTR 5 GOTO Pfail_serve
140    ENABLE INTR 5;1   ! Pfail interrupts only.
150    !
160    Pback_delay=100   ! Delay 1 s before Pback interrupts
.
170    Protection=3000   ! 30 s max. of Protection Time.
180    Pfail_delay=200   ! Delay 2 s before Pfail interrupts
.
190    CONTROL 5,5;Pback_delay,Protection,Pfail_delay
200    !
210    FOR Crt_line=1 TO 57
220      OUTPUT 1;"Output Area line";Crt_line
230    NEXT Crt_line
240    !
250    GCLEAR
260    GRAPHICS ON
270    FRAME
280    MOVE 50,50
290    LABEL "GRAPHICS DISPLAY"
300    !
310    LOOP
320      DISP Number
330      Number=Number+1
340    END LOOP
```

```
350   !
360   STOP
370   !
380 Pfail_serve: ! First, store GRAPHICS display.
390     GSTORE Crt_graphics(*)
400     !
410     ! Then store ALPHA display.
420     STATUS 1,3;Lines_above
430     CONTROL 1;1,-Lines_above+1 ! Move print position
440                                ! to "top" of display.
450     ENTER 1 USING "K";Crt_alpha$(*) ! Enter screen.
460     !
470     ON ERROR GOTO Already
480       CREATE BDAT "Pfail_data:INTERNAL,4,1",116
490 Already: OFF ERROR ! File already created.
500     ASSIGN @File TO "Pfail_data:INTERNAL,4,1"
510     OUTPUT @File;Crt_graphics(*),Crt_alpha$(*)
520     !
530     CONTROL 5;1 ! Shut down when finished.
540   END
```

The INTEGER array used to store the graphics display was dimensioned for the 9836's display (12 480 INTEGER elements). Exactly 7 500 INTEGER elements are required to store the 9826's graphics display.

The size of the BDAT file was chosen for the "worst case" storage requirement. In order to calculate the maximum number of of disc sectors required to store both displays, you must determine three facts: the maximum number of data elements to be stored, the data type of each item, and the number of bytes required to store one element of each data type.

The 9836 display's Output-Area memory can hold up to 57 lines of 80 characters each (4 560 bytes). The 9836's graphics display requires 12 480 INTEGERs (24 960 bytes). A total of 29 520 bytes of storage is required. Since BDAT files contain default records of 256 bytes each, the file "Pfail_data" was dimensioned to 116 256-byte records.

The following program gives a method of restoring the alpha and graphics displays and real-time clock. Actual program would probably also restore other variables and resume program execution that was interrupted by the powerfail.

```
100   ! This program for use on a 9836; change
110   ! array sizes and msus for use on a 9826.
120   !
130   INTEGER Graphics(1:12480) ! (1:7500) for 9826.
140   DIM Crt_alpha$(1:57)[80]  ! [50] for 9826.
150   !
160   ASSIGN @File TO "Pfail_data:INTERNAL,4,1"
170   !                         ":INTERNAL,4,0" for 9826.
180   ENTER @File;Graphics(*)
190   ENTER @File;Crt_alpha$(*)
200   ENTER @File;Clock
210   !
```

```
220    GRAPHICS ON
230    GLOAD Graphics(*)
240    !
250    OUTPUT 1;Crt_alpha$(*)
260    !
270    SET TIME Clock
280    DISP "Powerfail occurred at";Clock
290    !
300    !
310    END
```

A very important consideration for the powerfail service routine is that it has enough battery time to store all the specified data. If there is insufficient battery time to allow storing all desired data, the service routine should be able to record exactly how far it got into the backup when battery power went down. The next example shows how to enable interrupts to signal that power is back or that only one second of battery power is left.

Power-Is-Back and One-Second-Left Interrupts

The powerfail-interface controller has the ability to sense when power is back and when approximately one second of battery power remains; it can optionally generate interrupts to the BASIC program when these events occur. The following example program shows how to enable and service these types of interrupts.

```
100    COM Important_data$(1:8192)[28]
110    DIM Random$[28]
120    !
130    ON INTR 5,14 CALL Pfail_response
140    ENABLE INTR 5;1 ! "Power Has Failed" interrupts.
150    !
160    !
170    REPEAT

                         Main Program.


370   UNTIL Error<1.E-12
380   !
390   END
400   !
```

```
410  ! ******** Powerfail Service Routine ********
420  SUB Pfail_response
430  COM Important_data$(1:8192)[28]
440  DIM Message$[64]
450  !
460  ! Set up and enable service routine for
470  ! "One-Sec-Left" and "Power-Back" interrupts;
480  ! priority 15 allows data storage to be interrupted.
490  ON INTR 5,15 GOSUB Stop_storing
500  ENABLE INTR 5;4+2
510  !
520  ! Assume BDAT file (1024 records) exists.
530  ASSIGN @Storage TO "PFAIL_DATA"
540  ! Store elements individually to permit interrupts.
550  FOR Element=1 TO 8192
560    OUTPUT @Storage;Important_data$(Element)
570  NEXT Element
580  !
590  ! Power Down after all data stored.
600  CONTROL 5;1
610  !
620  ! ********* New service routine. *********
630 Stop_storing: STATUS 5,1;Intr_cause
640              !
650  IF BIT(Intr_cause,2) THEN  ! One Second Left.
660    ! Define Message.
670    Message$="Only the first "&VAL$(Element)
680    Message$=Message$&" elements have been stored."
690    Message$=Message$&" Error="&VAL$(Error)
700    Message$=Message$&CHR$(10) ! End with LF.
710    ! Write to Continuous-Memory Regs.
720    FOR Reg=8 TO LEN(Message$)+7
730      CONTROL 5,Reg;NUM(Message$[Reg-7;1])
740    NEXT Reg
750    ! Power Down.
760    CONTROL 5;1
770  END IF
780  !
790  IF BIT(Intr_cause,1) THEN  ! Power Is Back.
800    ! Re-enable "Power Has Failed" interrupts.
810    ENABLE INTR 5;1
820    ! Then return to interrupted context.
830    SUBEXIT
840  END IF
850  !
860  SUBEND ! **********************************
```

The service routine first enables two types of interrupts; one is generated when power is back after the powerfail, and the other is generated when approximately one second of battery power remains. Then, the service routine attempts to store the specified data. Notice that the service routine stores the data one item at a time so that either interrupt may be serviced while the data are being stored.

If the Power-Is-Back interrupt is generated, the service routine ends and returns to the main program. You may want to expand the service routine to sense recurring power flutuations and to respond accordingly. If the One-Second-Left interrupt is generated, the program stores a message to show how much of the desired data have been stored. Keep in mind that once this interrupt is generated, the computer powers down, **regardless** of whether power is restored before the end of the one second.

Register Summary

This section lists all STATUS and CONTROL registers of the Powerfail-Protection Int:rface, which is permanently assigned to interface select code 5.

STATUS Registers

STATUS Register 0 — Card Identification is always 5.

STATUS Register 1 **Powerfail Interrupt Cause**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Not Used					One Second Left	Power Is Back	Power Has Failed
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

STATUS Register 2 — Interrupt Mask has bit definitions identical to the preceding register (Powerfail Interrupt Cause).

STATUS Register 3 **Powerfail Status**

Most Significant Bit Least Significant Bit

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Failed Self Test	Not Used			One Second Left	Currently Using Battery	Ac Is Down	In the Powerfail State
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

Bit 7 — Failed Self Test indicates the outcome of the self test: a 1 indicates failure, and 0 indicates successful results.

Bit 3 — One Second Left indicates that approximately one second of battery power remains. The computer will automatically power itself down, even if power is restored before one second has expired.

Bit 2 — Currently Using Battery indicates whether or not the battery is being used: 1 indicates it is currently being used for computer power, and ndicates that it is not.

Bit 1 — Ac Is Down indicates the current status of ac-line power: a 1 indicates that ac power is completely gone. If bit 2 is a 1 and this bit is 0, the battery is being used because ac power is not completely gone but has dropped below an acceptable level; in this case, a "brown-out" condition is indicated.

Bit 0 — In the Powerfail State indicates whether or not the computer is currently in the Powerfail State: a 1 indicates Powerfail State, and 0 indicates that the computer is not currently in the Powerfail State. The Powerfail State is exited when power is back and the Power Back Timer reaches the value of the Power Back Delay.

STATUS Register 4 — Overheat Protection Timer contains the amount of battery time used during this Powerfail State (in tens of milliseconds). For every second the power is down, it must be back for two seconds to ensure adequate cooling for the machine. Thus, the value of this register bounds the maximum amount of time that can be obtained from the battery, even though 60 seconds may have been specified as the protection time (CONTROL Register 6).

STATUS Register 5 — Power Back Timer contains the time elapsed since power was restored after the last powerfail (in tens of milliseconds).

STATUS Register 6 — Powerfail Timer contains the time elapsed since the last powerfail (in tens of milliseconds).

STATUS Register 7 — is not used.

STATUS Registers 8 thru 71 — Continuous-Memory Registers contain the 64 bytes of data written by the last CONTROL statement directed to these registers.

CONTROL Registers

CONTROL Register 0 — Shut Down. Any non-zero value written to this register will turn off both battery and ac-line power to the computer, which conserves battery power after the service routine has finished responding to the powerfail. If ac-line power is on when this statement is executed, the computer will be turned back on in the normal powerup sequence.

CONTROL Registers 1 thru 4 — are not used.

CONTROL Register 5 — Power Back Delay. The value of this register determines the amount of time (in tens of milliseconds) that the computer will delay, after power is back, before leaving the powerfail state (i.e., before generating a "Power Is Back" interrupt). The power-on default value is 50 (500 milliseconds).

CONTROL Register 6 — Protection Time. The value of register determines the maximum amount of time (in tens of milliseconds) that the computer is to have battery backup. Power-on default is 6000 (60 seconds).

CONTROL Register 7 — Powerfail Delay Timer. The contents of this register determine the amount of time (in tens of milliseconds) that the Powerfail-Protection Interface will wait, after a powerfail, before generating a "Power Has Failed" interrupt. Power-on default is 10 (100 milliseconds).

CONTROL Registers 8 thru 71 — Continuous-Memory Registers. These sixty-four, single-byte registers can be filled with any desired data, one byte (ASCII character) per register.