Surprise! Your HP 86 is more powerful than you think and all the extra power is free.

THERE'S GOLD IN THEM DISKS

BY GORDON S. BUCK

Unless you've studied the HP 86 demonstration disk, you're in for a pleasant surprise—your 86 is more powerful than you think. Best of all, this extra power is free, or at least won't cost you more than you've already spent.

The extra power comes from the binary routines on the demo disk.

WHAT ARE BINARIES?

A binary is a machine-language program. The demo disk binaries add Basic-like commands to the 86. They function like the internal ROM-based commands except the binaries are loaded from mass storage. Since they're written in machine language, binaries execute many times faster than programs written in Basic. Binaries let you do things you can't with Basic.

There are 47 predefined functions and 144 Basic commands and statements in the 86 manual. HP also provides 20 additional commands through the demo disk binaries. To use these commands, simply load the binary programs and execute the commands as instructed on the demo disk. For some reason, neither the operating manual nor the *Introduction to the HP 86* stresses using these binaries.

My system is an 86A with an additional 64k, an 82905B printer, 82913A monitor and single 9130A 5.25-inch disk drive. For the first few months, I didn't try to use the demo disk binaries—learning the operating manual was enough to keep me busy. I've found that most 86 owners have this in common. However, many owners never find out about the binaries or are hesitant to use them.

Binary programs are described briefly in the operating manual. Like Basic programs, binary programs are loaded from disk except that the command is LOADBIN instead of LOAD. Both a binary and a Basic program can be in memory at the same time. In fact, up to five binaries and one Basic program can be stored simultaneously. However, loading a Basic program will scratch the binaries from memory, so you must load Basic programs before loading binaries. (Note: the CHAIN command will load and execute a program without harming the binaries.)

The demonstration disk that comes with your 86 or 87 contains four binary programs: *Util/1*, *Getsave*, *Gdump* and *Askiob*.

Util/1 contains the largest number of commands and has been the most useful binary to me, providing additional control over the cursor and keyboard. It also helps manipulate strings and enhance graphics.

Getsave converts Basic program statements into string data and allows you to merge two programs. It can help transfer programs between different computers. An improved *Getsave* is built into the 86B.

Gdump prints the graphics display on a dot matrix printer.

Askiob determines the system configuration. The model number of the computer, space remaining on a disk, type of disk drive and plotter model number can be determined within a program. This helps to get the same program to run with different peripherals.

Now that you know the binary power available from the demo disk, you need to learn the commands to use the programs. The demo disk program *Cardfile* contains the instructions.

Place your demo disk in the disk drive and type LOAD CARDFILE, press END LINE and then RUN. When the *Cardfile* instructions are displayed, press L to load an existing file. You'll then be prompted for the file name. Enter BINARIES and press END LINE. *Cardfile* reads the disk file named BINARIES and displays a key label menu. K4 is labeled DISPLAY; K5 is labeled PRINT. Either key will get you the instructions for the binaries.

UTIL/1

This is the binary I use most often. I've be-

ILLUSTRATION BY HOWARD LEWIS

Demo disk binaries, written in machine language, add Basic-like commands to the 86 and let you do things you can't with Basic—and much faster.

come so accustomed to some of its commands that I routinely include them in my Basic programs. *Util/1* requires 2,922 bytes and contains 13 usable commands.

The first command is AREAD (string variable). AREAD reads the display. However, the cursor must be in the position of the first character to be read. This is done with the AWRITE [row, column, [string]] command listed after AREAD.

AREAD fills the string variable with characters from the screen. This means the string must be properly dimensioned or specified. I had some problems with AREAD until I realized that a clear display is made of CHR$(13)s (ASCII nulls) and not CHR$(32)s (spaces).

You can move the cursor around the display by using AWRITE and specifying the row and column. The instructions imply that AWRITE is limited to PAGESIZE 16; however, it works just as well for PAGESIZE 24. Notice that the first row is zero instead of one. Also, the first column is zero; the last column is 79. The example that follows moves the cursor to row 10, column 45, writes I MOVED IT, then moves to row 15, column 50 and writes AGAIN. It then moves the cursor to row 10, column 47 and reads X$[1,4] as MOVE.

```
LOADBIN "UTIL/1"
10 CLEAR
20 AWRITE 10,45, "I MOVED IT"
30 AWRITE 15,50, "AGAIN"
40 AWRITE 10,47
50 AREAD X$[1,4]
60 END
RUN
```

After running this example, notice that the cursor doesn't show on the monitor. Press END LINE. The cursor now appears at the left margin on the line just below I MOVED IT. Before pressing END LINE, the cursor was not visible but was positioned at row 10 and column 47 by line 40. Now type in X$[1,4] and press END LINE. The display will show MOVE.

START CRT AT (line) allows you to position the display window. Remember that the 86 uses a 54-line CRT memory in the Alpha mode and 204 lines in the Alphall mode. This command also gave me some problems. START CRT AT uses an absolute numbering system. AWRITE and AREAD use line numbers relative to the current display window. With START CRT AT you can prepare two display screens and flip back and forth between them in your programs.

The following example clears the entire CRT memory to provide 54 blank lines. It then writes PAGE 1 on the first 24-line window and, PAGE 2 on the second. The function keys let you flip between the windows.

```
10 ALPHALL
20 ALPHA
30 PAGESIZE 24
40 START CRT AT 0
50 AWRITE 0,65, "PAGE 1"
60 AWRITE 1,0, "THE FIRST PAGE."
70 START CRT AT 24
80 AWRITE 0,65, "PAGE 2"
90 AWRITE 1,0, "THE SECOND PAGE."
100 START CRT AT 0
110 ON KEY# 1, "PAGE 1" GOTO 160
120 ON KEY# 2, "PAGE 2" GOTO 180
130 ON KEY# 3, "EXIT" GOTO 200
140 KEY LABEL
150 GOTO 150
160 START CRT AT 0
170 GOTO 140
180 START CRT AT 24
190 GOTO 140
200 END
```

Flip between the two pages by pressing K1, then K2. Press K3 to exit from the program. Now hold down the ROLL key. You'll see the two pages rolling down the screen.

The next command, LINPUT, is used exactly as the normal INPUT command except that commas will be accepted as part of the string variable. This allows you to input dates and geographical data as they are normally written. Use caution when entering LINPUT, as syntax errors have been known to cause a crash.

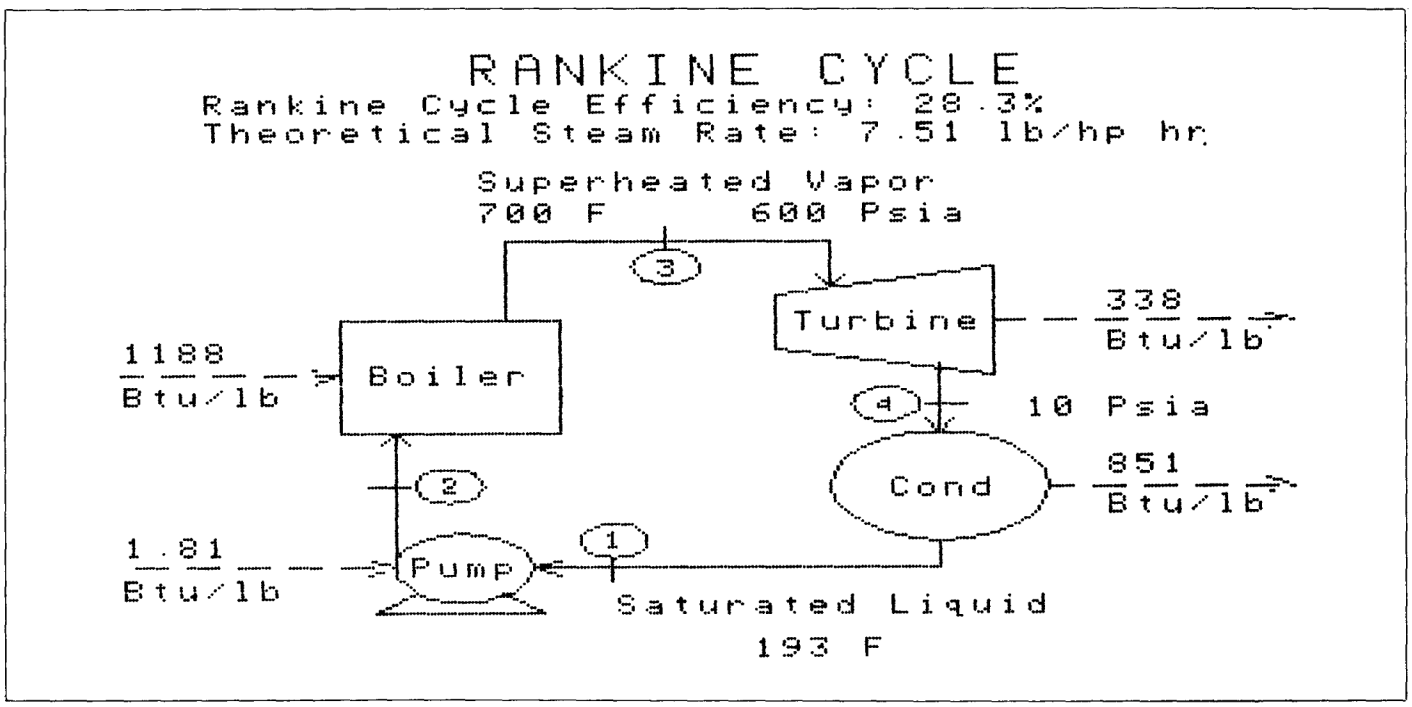
Entering the command UTIL/1 will return the revision number of the binary program.

TAKE KEYBOARD is used to restrict access to the keyboard during program execution. It locks out and puts in a buffer all keys except function keys K1 through 14 and RESET. The instructions say that you must define all function keys even if they're not used. But this doesn't seem to be true in all cases.

KEY$ returns one character from the buffer set up by TAKE KEYBOARD. If you use KEY$, you must use TAKE KEYBOARD first. Failure to do so can crash the system. Using KEY$ reduces the use of the END LINE key when responding to prompts. You can also redefine the entire keyboard if you wish.

RELEASE KEYBOARD is the reverse of TAKE KEYBOARD. If you use TAKE KEYBOARD in your programs, at some critical point you'll forget to put in a RELEASE KEYBOARD. When this happens, the keyboard will be dead. Chalk it up to experience and press RESET.

Here's an example using TAKE KEYBOARD, KEY$ and RELEASE KEYBOARD:




```
10 CLEAR
20 TAKE KEYBOARD
30 DISP "ENTER 'Q' WHEN TIRED OF A DEAD
   KEYBOARD"
40 IF KEY$ < > "Q" THEN 40
50 DISP "THE KEYBOARD IS WORKING NOW"
60 RELEASE KEYBOARD
70 END
```

FAST LABEL (column, row, string, highlight) is the next command. This is a nice feature to use in graphics displays. It's much faster than using LABEL when writing on the graphics screen.

The first parameter, column, is the alpha column for the first character of the string. Graph-mode columns are numbered from zero to 49. In the Graphall mode, columns are numbered from zero to 67.

The second parameter, row, is given in dots from the top of the screen. Because the row is specified in dots, sometimes more effort is required to position the label; however, the method does give greater flexibility.

The upper left-hand corner of the character block is positioned at the specified row and column. If the entire string can't be written on the specified row starting at that column, the excess characters are written on the next row (one dot lower) starting at column zero.

Remember, in both Graph and Graphall modes, the display has dots numbered from zero to 239. You have no control over character size when using FAST LABEL. The character size is the same as in the Alpha mode. The following shows how FAST LABEL is used:

```
10 GCLEAR
20 GRAPH
30 FAST LABEL 0,0,"0 COLUMN AND 0 ROW",0
40 FAST LABEL 0,10,"0 COLUMN AND 10
   ROW.",1
50 FAST LABEL 29,221, "29 COLUMN AND 221
   ROW.",0
60 FAST LABEL 47,231, "47 COLUMN AND 231
   ROW.", 1
70 END
```

Now change statement 20 to GRAPHALL and run the program again.

SGCLEAR (x1,x2,y1,y2) [, mask] clears a rectangular area of the graphics display. This can save time if you need to redraw only a portion of a graphics display. The x and y values are measured from the lower left corner. Note that the x value will be rounded off to the nearest multiple of eight. Remember that the Graph mode has 400 dots in the x direction; Graphall has 544. Try this example:

```
GCLEAR
FRAME
MOVE 0,0
DRAW 168,100
MOVE 0,100
DRAW 168,0
SGCLEAR 100,300,60,200
```

MSUS$ (".volume") simply returns the mass storage unit specifier of the disk with the

EXAMPLE OF **graphics dump using DUMP GRAPHICS command with the HP 82905B printer.**

U*til/1* provides additional control over the keyboard and helps you manipulate strings and enhance your graphics.

specified volume label. People with only one disk drive don't seem too interested in this command. Try the command with one of your disks and its label.

RPT$ (string, number) is most easily remembered and described as "repeat the string." It's a nice feature. An example is included with the example for TRIM$.

TRIM$ (string) turns out to be a handy command to use along with AREAD. Remember, AREAD fills the string. I prepare the screen by AWRITEing spaces in the areas to be AREAD. The extraneous spaces that are AREAD can then be removed by TRIM$.

```
10 CLEAR
20 AWRITE 5,20,RPT$(" ",18) ! Writes 18 spaces
30 AWRITE 5,20,"13 CHARACTERS"
40 AWRITE 5,20
50 AREAD X$
60 AWRITE 10,0
70 DISP "X$: ";X$
80 DISP "X$ LENGTH: " LEN (X$)
90 DISP "TRIMMED STRING: ";TRIM$ (X$)
100 DISP "TRIMMED STRING LENGTH: ";LEN
    (TRIM$ (X$))
110 END
```

Since string variables are dimensioned as 18 characters unless otherwise specified, the AREAD command reads the 13 written characters and then an additional five spaces. The LEN command, therefore, correctly shows a length of 18 characters. After trimming with TRIM$, the LEN is 13, the correct length for the intended purpose.

GETSAVE

Getsave lets you store and retrieve Basic programs as string data instead of tokenized (compiled instruction) code. *Getsave* requires 1,480 bytes and has two usable commands, GET and SAVE. With *Getsave*, you can convert an 86/87 Basic program into string data, load the data into an 85 and convert it to program instructions by using the 85 program *Getsav*. My 86 instructions for *Getsave* say that the 85 binary is named *Dgtsav*. However, I couldn't find *Dgtsav* in the HP Users Library. I found and used the 85 binary *Getsav* instead.

I found that the string data stored by *Getsave* can be read by the HP word processing program *Word/80* using the GET command. This means that *Word/80* can be used as a program editor. After editing, use *Getsave* to convert the *Word/80* text file back to a Basic program. However, since *Word/80* expects 80 characters per line and Basic program lines can have 159 characters, Basic statements will be truncated to 80 characters.

Another use for *Getsave* is to merge two Basic programs by SAVEing the first program, loading the second as a Basic program and then GETting the first while the second is in memory. Be certain you have renumbered one of the programs to avoid any conflicts.

The command GETSAVE returns the revision number of the binary.

WHAT MOTHER DIDN'T TELL YOU ABOUT *UTIL/1*

Util/1 was developed in a hurry some time after the final release of the Gemini (86/87) operating system to fill a need for extra screen support, initially in the demo disk programs.

While the program has been widely accepted by Series 80 users, most people don't realize there are some bugs lurking just beneath the surface. Here's a brief rundown of the most common bugs and the cures.

LINPUT (string$). If you mistakenly enter a $ before the string reference, you can crash on the spot. The problem is traceable to some expedient, but faulty, design in the parse routine that pushes bytes before adequate testing has been done on the proposed new program line. Avoid this syntax.

Remedy: Enter LINPUT statements with due care.

KEY$. A program reference to KEY$ without a prior call to TAKE KEYBOARD will crash the program clear back to power on.

Remedy: Make sure you have set up the key buffer first.

START CRT AT (numeric_ref). Executing this statement with a negative value will confuse the CRT controller and cause a sideways wrapped screen.

Remedy: Track the values you plan to use for screen START AT addressing and trap any value less than zero.

TRIM$ (string$). Fails to work on strings longer than 32,767 bytes because of incorrect register usage in the runtime code segment.

Remedy: Don't use strings longer than 32,767 bytes.

RPT$ (string$,number_of_repeats). If you're trying to fill a string longer than 32,767 bytes with a single byte, the function fails because there is a call at input to system routine ONEB when ONEX should have been used. It's probably the result of a hurried conversion of the HP 85 code.

Remedy: If you're filling a long string, do

GDUMP

Since I don't have a plotter, I appreciate *Gdump* more than any other binary. It requires 3,542 bytes and has one usable command, DUMP GRAPHICS. *Gdump* dumps the graphics display onto the printer. With the 82905B printer, my command is simply DUMP GRAPHICS when the graphics are done in Graph mode (see figure).

The full command is DUMP GRAPHICS [lower bound [,upper bound [,rotate [printer type]]]]. The instructions give three printer-type codes for five HP printers. I've been told that the Epson MX80 HP printer will give a good graphics dump if it's identified as an HP 82905A printer (use a negative number to indicate the type of printer). If you have a program that uses graphics, use DUMP GRAPHICS after program execution is complete. The next example shows how DUMP GRAPHICS can be executed from the keyboard.

```
LOADBIN "GDUMP"
SCALE 0,100,0,100
FRAME
PRINTER IS 701,80
DUMP GRAPHICS 0,0,0,0
```

Now put the printer into compressed mode by keying in PRINT CHR$(27)&"&k2S". Repeat the example. Try it again with −1 for the rotate code. This should be useful.

Notice that both the upper and lower bounds were specified as zero. If the lower bound of the scale is zero, this will cause the complete graphics screen to be dumped. If the screen is not scaled from zero, specifying zero as the lower bound will return an error.

The command GDUMP returns the revision number of the binary.

ASKIOB

With *Askiob*, you can determine the system configuration. This can be useful if you are writing programs that run on 86 systems other than your own. *Askiob* requires 1,172 bytes and has four usable commands.

DISK FREE (x,y,msus) will return with x equal to the number of free sectors on the disk at that mass storage unit specifier (msus). Also, y will be equal to the largest contiguous block. Therefore, if x and y are not equal, there are null files on the disk. (There are 1,040 sectors on an empty but initialized disk.)

Executing DISK FREE for a non-initialized disk produces a warning for null data. Assuming you have a disk in the drive at msus d700, try the following example:

```
LOADBIN "ASKIOB"
DISK FREE x,y,":d700"
x
y
```

Entering x then pressing END LINE returns the value for x. The y value is obtained similarly. The value for x is the number of free sectors on the disk. If y is not equal to x, you can PACK the disk to eliminate the null files.

DISK TYPE (msus) will return a one if the disk at that location is an eight-inch or a hard (Winchester) disk. A zero will be returned if the disk is a 5.25-inch or 3.5-inch disk. With your disk still at d700, try:

```
DISK TYPE (":d700")
```

PLOTTER ID (device selector) returns the model number of the plotter located at that address. Not having a plotter, I haven't needed this one, but I know different plotters have different physical limits, numbers of pens and capabilities.

ASKIOB returns the revision number of the binary.

You may not need all 20 of these new commands, but I'll bet some of them can help you right away. Your 86 is now more powerful than ever. □

Gordon Buck is a registered engineer in Louisiana and a software developer for the HP Series 80.

---

it twice with a RPT$ value less than the working maximum and concatenate.

Even though some of these problems could have been avoided, one of the most common areas for failure of string functions concerns the reserved memory requirement of these system and binary programs. For the same reason, you can experience a memory overflow error trying such innocuous things as adding a few more bytes to a very long string using the system concatenation operator &.

If you want to add just two bytes to a string 65,000 bytes long, you must have at least 65,000 bytes of free memory for the procedure to succeed. The wait for this relatively simple operation can be pretty annoying.

If you receive a memory overflow error when using these functions (and similar ones in the Advanced Programming ROM), thank the bugs.

Don Person