HEWLETT PACKARD

Competitive Analysis

For Internal Use only

HP BASIC

vs

MicroSoft® QuickBASIC

and

QuickC

Measurement Systems Operation

Table of Contents

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Introduction

The purpose of this application note is to make you more comfortable discussing test development languages. The comparison is written from the perspective of a test engineer learning each of these languages for the first time, and is divided into several key areas: development environment, I/O, computation, graphics, control structures, data types, learning products, and platforms.

The sections are organized according to their importance in making a presentation, so just read what you need. The summary is first. "The Main Argument for HP BASIC" presents a high-level discussion of key differences, and includes a comparison table at the end for a quick reference. "Sales Presentation Strategies" considers different customer types. And "Background" contains code examples and more detailed information about several of the comparison areas.

Note

HP BASIC = "RMB" (Rocky Mountain Basic)
QuickBASIC = "QB"
QuickC = "QC"

A Summary of the Comparison

Development Environment

HP BASIC, QB, and QC provide friendly environments for test development. QB and QC have more modern, window-oriented environments with pull-down menus and help facilities. HP BASIC and QB have more effective debugging tools than QC, i.e. syntax checking and the capability of testing program statements.

I/O	HP BASIC has unified I/O, optimized data transfer, extensive interrupt capability, data format control, and I/O paths. In QB and QC, you are dependent on the capabilities of your I/O library and must play the role of system integrator.
Computation	HP BASIC supports complex math and array operations; QB and QC do not. QB also lacks binary functions and hyperbolic trigonometric functions.
Graphics	HP BASIC has commands more relevant to test development than QB or QC.
Control Structures	The three languages are equivalent.
Data Types	HP BASIC does not have pointers, 32 bit integers, records, or user-defined types. QC has pointers and records; QB and QC both have 32 bit integers and user-defined types.
Learning Products	HP BASIC has clear manuals but a lot of them, an excellent self-study course, and customer courses. QB has clear, concise manuals aimed at the novice and a good demo. QC has mediocre documentation.
Platforms	HP BASIC runs on DOS, the HP Series 300 workstation, and HP-UX. QB and QC are limited to DOS.

The Key Advantage of HP BASIC

HP BASIC is an integrated system.

Your chances of cutting test development time, and therefore cost, are much better with HP BASIC. The computer, operating system, language, various interfaces and cards, as well as most instruments were all designed to work together using HP BASIC.

RMB is ready to go. It has everything you need in one integrated design. QB is a very good language, but it's not as powerful as RMB and you still have to attach a few parts--an I/O card and library, for example. QC is an excellent tool, but you have to assemble it before you can get going.

The Main Argument for HP BASIC

In addition to being an integrated system HP BASIC is easy to use and powerful. It has been created for instrument control. HP BASIC is superior to QB and QC in I/O, computation, and graphics.

For easier applications, QB may serve the purpose. If you're willing to master the C language and write the extra lines of code, then you can accomplish your goals with QC. With both languages, however, you'll have to be the system integrator, and it will be difficult to match the fast program development time of HP BASIC.

Let's look at the following key areas in more detail:

- Development Environment
- I/O
- Computation
- Graphics
- Control Structures
- Data Structures
- Learning Products
- Platforms

Development Environment

HP BASIC

You are probably familiar with HP BASIC's environment -- a black background and green lettering, eight softkeys at the bottom of the screen, which can display several menus. The large black area is divided into four areas: output area, display line, keyboard area, and message or results line. Nowadays HP BASIC's environment seems old-fashioned, but you can perform the essential tasks of filing, editing, searching, running, and debugging very effectively. It's fast and simple.

Two key traits of HP BASIC are syntax checking and the live keyboard. Syntax checking simply means that statements are checked for errors at entry. The live keyboard means that you can enter commands from the keyboard during program execution. For example, you might want to pause the program to check the value of a variable and then continue execution. The live keyboard allows you to use the computer as a calculator while programming, or you can execute some simple OUTPUT commands to see if an instrument is responding to the computer. Syntax checking and the live keyboard cut development time significantly.

QuickBASIC

QuickBASIC has a "window-oriented" environment because the screen is divided into separate areas in which you can perform different tasks. QB's default setup has a blue background and white lettering, which you can customize. A menu bar along the top includes seven menus for editing and debugging operations plus a help facility. A reference bar along the bottom tells what function keys to press for important commands, displays information about highlighted keys, and contains row and column indicators. You choose commands via the menus or through function keys (most commands). QB obtains additional information from the user through dialog boxes.

QB has four types of windows: the View window for programming, the Immediate window to test program statements, the Help window, and the Watch window to monitor variables. QB wakes up with the View window and Immediate window open.

QB performs all the essential development tasks of filing, editing, searching, running, and debugging. Its method is different from HP BASIC's -- windows and menus as opposed to the EDIT mode, the live keyboard, and so forth.

A key trait of QB is the excellent help facility, which can be accessed via the Help menu or through context-sensitive help. (Position the cursor under a keyword, hit F1, and you'll get a help screen on that word.) There is an effective cross-referencing scheme as well as plenty of coding examples. You can even insert a code segment from Help into your own program for editing.

Another key trait of QB is the option of two types of compilation. Compilation is the process by which the computer converts source code that you write into machine code which it executes. You can develop your program using syntax checking as in HP BASIC. Then, when you're done, you can compile it into a stand-alone file executable from DOS. (In HP BASIC you can also optimize your code by compiling when you're done, but you must run the program in the HP BASIC environment.)

QuickC

The environment in QC is almost identical to QB's with a few exceptions. There is no Options menu. An Error window replaces QB's Immediate window, since you cannot test the effect of program statements in QC. And the QB's Reference bar becomes a Status line in QC. Otherwise, you use QC's environment in the same way -- i.e., menus, dialog boxes, a Watch window. You also have a similar help facility.

A key trait of QC is that you can only compile C after you've finished the editing cycle. If the program has syntax errors (mis-spelled keywords, for example), you return to edit mode to fix them and then recompile. Once you've compiled the file you're editing you have to link it with other files or a library. If you get link errors, you return to editing mode again to fix them. Then you recompile, relink, and run. THEN you fix the errors of logic, whichyou check by going through the whole edit-compile-link-run cycle again. You really start to appreciate the luxury of syntax checking after you've programmed this way for a while. Figure 1 shows the difference between the development tasks in HP BASIC and QB, which have syntax checking, and QC, which doesn't.

Important Differences

HP BASIC vs. QB

HP BASIC's environment in combination with its wealth of keywords would make it a faster development environment for medium to difficult applications. Both languages would be equivalent in speed of development for easier applications. Both have syntax checking, which is a great advantage over QC in terms of development speed. They both have everything you need for editing and debugging, but the methods differ. Do you prefer using windows and menus or typing commands and using softkeys? It's up to you. Would you rather use QB's Watch window or HP BASIC's live keyboard? They can both be effective debugging tools.

QB's Help facility, menus, and Reference bar combined with excellent documentation and an online demo do make it easier to learn than HP BASIC in the beginning. HP BASIC has an excellent self-study course and customer classes. My feeling is that you could get going a little faster in the first week or two, but then the issue of ease of use would be more important. After the break-in period, HP BASIC's power in terms of relevant keywords would make it a faster language than QB for programming.

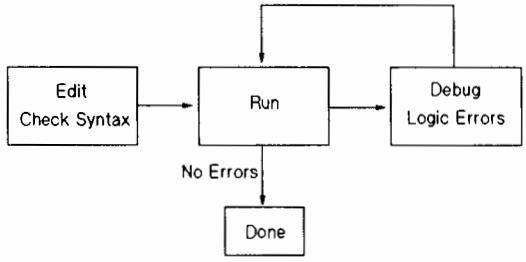
QB can be compiled into a file executable from DOS, and it does have some unique features like splitting the View window. However, HP BASIC has some unique debugging features of its own like the XREF command for a cross-reference listing of program identifiers.

In summary, HP BASIC is a faster environment for medium to difficult applications, but is equivalent to QB for easier programs. QB has a more "modern" look (window-oriented screen and a menu bar) as well as more aids for fast learning. And QB has an excellent help facility.

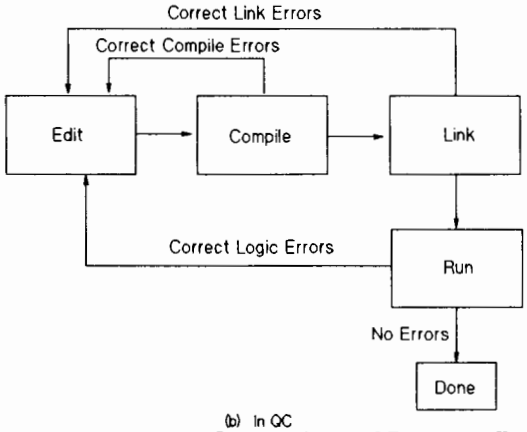
HP BASIC vs. QC

HP BASIC has a development speed advantage over QC because of syntax checking. In QC, you must go through a continual cycle of separate editing, compiling, linking, and running. Although QC provides one environment for all these tasks, they still are separate and take much longer than in HP BASIC.

QC is window-oriented and has menus, so its method of operation is different than HP BASIC. Which you prefer depends on your programming habits. HP BASIC and QC both have all the tools in place to get the job done.



(a) In RMB and QB



(b) In QC

Figure 1 - Comparison of Program Development Tasks

Concerning ease of learning, QC has a slight advantage. (I'm just referring to the environment here.) You can learn about HP BASIC's environment through the manual set, the self-study course, or the customer classes. QC teaches the environment in a single programmer's guide, and you also have the advantage of the menu bar and help facility.

As in the QB comparison, I think the issue of ease of use becomes more important after the first two weeks, and HP BASIC's environment combined with its rich command set would make it a faster development environment for more difficult test systems.

In summary, HP BASIC has the syntax checking over QC in terms of development speed, which is a very important distinction. QC has a window-oriented environment with menus and a help facility. These qualities impose a different style of editing-debugging and give QC an edge in ease of learning.

HP BASIC's environment along with its rich command set makes it significantly faster than QC's environment in total development time. There might be execution speed advantages and low-level coding capabilities in C, which have to be weighed against HP BASIC's superiority in development time and ease of use. Which advantages are more important to your customer?

I/O

You've often heard that HP BASIC is "optimized for I/O." What exactly does that mean?

The most important point about I/O is that it's a part of the integrated design of HP BASIC. This means the operating system, the programming language, the I/O card, and the I/O driver will all work together harmoniously.The principle advantage of this is hassle-free operation. The parts of the system in HP BASIC were designed to work as a whole.

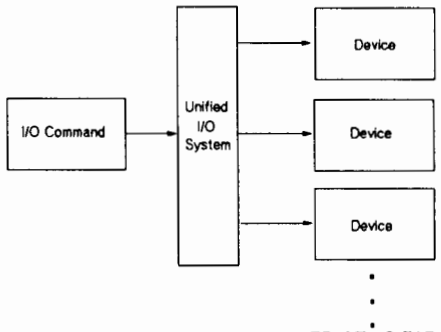


Figure 2 - Unified I/O in RMB

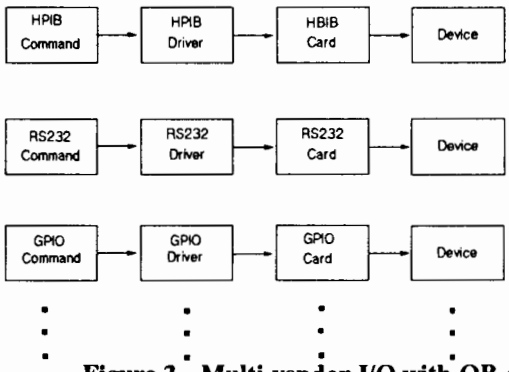


Figure 3 - Multi-vendor I/O with QB or QC

QB and QC don't have unified I/O. That means you'll have to watch over system integration yourself. You have to make sure the computer (a third-party vendor, for example), the operating system (DOS), the language (QB or QC, in this case), the interface card and HP-IB I/O library (by National Instruments or HP, say) all work together. And, if they don't work together for your application, who will you call to solve the problem? If you're working with HP BASIC, first of all you'll have less bugs, since HP BASIC was designed as an integrated unit, and secondly you can call HP engineers when you get stuck.

Let's examine five aspects of I/O: unified I/O, data transfer, events and interrupts, data formatting, and I/O paths.

Unified I/O

HP BASIC supports HP-IB, GPIO, RS-232, and SRM interfaces. You won't have to worry about compatibility or what commands to use. There are 29 device I/O commands in HP BASIC and they apply to all system resources. For example, you can use OUTPUT to send a string to an instrument or a file or the CRT. You can use ENTER to read data from a file, the keyboard, or an instrument. It's simple. Compare unified I/O with multi-vendor I/O in the two figures below. The job of system integrator is yours with QB and QC.

Unified I/O also makes it easy to change devices. The necessary drivers are a part of the language. For example, you might have a printer on your HP-IB bus, which you replace with a printer off your RS-232 port. In HP BASIC, you just specify the new address, and PRINT statements will use the new printer. If you want a hard copy of graphics that you've plotted on the screen, just use the PLOTTER IS statement to signify the target plotter, and it's a trivial exercise to plot your data.

In QB and QC, you're going to have to worry about interfaces and drivers yourself. Are they compatible? How do you configure them? Is it necessary to use DOS system calls? What is the new command set? For example, say you have QuickBASIC on a Vectra-compatible using a National Instruments I/O card and library talking to HP instruments. The elements of this system have never been tested together.

What if they don't work together on your application? Who will you call? Each vendor is only responsible for his product, not the entire system. But with HP BASIC the system has been designed as a whole, and it's likely HP support can solve the problem.

Also, don't forget that as soon as you use an I/O library with QB, you lose your syntax checking. In HP BASIC, it's still there. And what about errors on the HP-IB interface? Using an I/O library error checking will be dependent on the vendor. With most I/O libraries you would have to add a line of code to check for errors on every I/O call. In HP BASIC, you'll get error checking built in.

This kind of problem is not obvious when you first consider a language. But it stands to reason that a system designed as a whole, designed to interface with instruments, a system as mature as HP BASIC will operate a lot better than a mix-and-match collection from different vendors.

Data Transfer

HP BASIC has a method of optimizing data transfer. Using TRANSFER instead of OUTPUT, speed improvements of up to 66% are possible. TRANSFER transmits the data on the bus as a background operation, while the CPU proceeds with program execution. Transfers are buffered by the operating system, which relieves the program from the responsibility of handling individual transfer events. This is particularly useful when you have large blocks of data or when you have small pieces of data that the instrument sends to the CPU intermittently.

Events and Interrupts

Sometimes you want certain "events" to "interrupt" the normal program execution, so the CPU can perform some predefined task. This method relieves you from the task of periodically checking to see if a particular event has occurred. For example, an instrument is done taking readings and wants the CPU to know that data is ready to be sent to the controller. The instrument sends a signal to the CPU. The CPU stops what it's doing, acknowledges the interrupt, and goes to an appropriate service routine. In this case, it might transfer the data to memory. When the signal comes from a device on the bus, it's called a device I/O interrupt. Other types of events might include pressing a key or runtime errors.

HP BASIC supports 41 interrupt types, QB has 12, and QC has 0. Not only does HP BASIC support a much greater variety of events than QB, but it has

device I/O interrupts, which are very useful in test systems. In QB and QC, your ability to use device I/O interrupts is dependent on your I/O library. Some support interrupts; some don't.

If your library doesn't support them, you can check for errors on every I/O operation (tedious) or you can write assembly routines that control the DOS interrupt capabilities. But you need to be an experienced programmer, and there's no chapter in the manual that will show you the way.

Types of events HP BASIC supports but QB doesn't:

ON CDIAL	from turning a knob on a control dial box
ON CYCLE	periodic interrupts from the clock
ON DELAY	an interrupt by the clock after a delay
ON EOR	from end-of-record during data transfer
ON EOT	from end-of-transfer
ON HIL EXT	from Human Interface Link device
ON INTR	from an interface or device
ON KNOB	from turning a knob
ON SIGNAL	from SIGNAL statement (software event)
ON TIMEOUT	from interface or device when it's taken too long to respond to a data transfer handshake

Similar events that HP BASIC and QB support:

HP BASIC	**QB**
ON ERROR	ON ERROR (runtime errors)
ON TIME	ON TIMER (when specified time reached)
ON KBD / KEY	ON KEY (from keys and softkeys)
ON END	ON EOF (end-of-file reached)
ON INTR (RS232)	ON COM (serial port)

Notice the lack of an HP-IB device interrupt in QB.

HP BASIC also has prioritized interrupts, so you can decide the order in which they're serviced.

Data Formatting

In HP BASIC it's a simple matter to input data that comes from a device in various formats. For example, you might get an integer followed by a string followed by an array of complex numbers. Just use the ENTER statement with the three different variables and HP BASIC will parse the incoming string for you. HP BASIC can do I/O with all its data types.

In QB and QC, your control over data formatting is dependent on the I/O library. Instruments like the HP 3852A Data Acquisition and Control Unit can send data as integers or in an internal format. HP BASIC can do either one of these in a single statement. Using a typical I/O library with QB or QC you'd have to do number conversions or enter data in strings. A Spectrum Analyzer might send a 128 bit complex number, which HP BASIC could handle easily. In QB or QC you'd have to enter the data in strings, parse it, and store it in a user-defined data type.

I/O Paths

When data moves to or from the CPU, it follows a default I/O path. In HP BASIC, you can assign a name to that path and customize how data goes through it. For instance, you might want to enter data in words instead of bytes, or convert characters to other representations as they come in, or change the end-of-line sequence. Once you assign these attributes to an I/O path, the data will be customized in that way whenever you use the path, which reduces overhead. In this way you can avoid the task of taking in the data as strings and customizing it yourself.

In addition to this specific control you get up to 30% increased performance. This capability isn't possible using an I/O library in QB or QC. Furthermore, in RMB you have the flexibility of unified I/O as well, so that paths can be used to send data anywhere -- an HP-IB device, a serial interface, or a file.

I/O Summary

The benefits of I/O offer the most compelling argument for choosing HP BASIC over QB and QC.

HP BASIC has unified I/O, which relieves you from the task of being system integrator. You won't have to memorize new I/O commands for separate libraries; you won't have to wrestle with compatibility problems; you have a single place to call in case of problems. QB and QC do not have unified I/O.

HP BASIC has the TRANSFER statement for optimizing the transfer of large blocks of data. QB and QC do not have an equivalent statement.

HP BASIC has a wealth of interrupts and does support device I/O interrupts. In QB and QC, your interrupt capability is dependent on your I/O library. QB does support some interrupts from events like keystrokes and runtime errors.

HP BASIC provides complete control over data formatting. In QB and QC, you are dependent on your I/O library.

Finally, HP BASIC has I/O paths, which offer flexibility in talking to different system resources and produce performance gains of up to 30%.

Computation

HP BASIC has the significant advantage of providing a greater range of operations and a complex data type. QB and QC have a slight advantage in supporting 32 bit integers.

QB does not have complex functions, binary functions, array operations, or hyperbolic trigonometric operations. HP BASIC does.

The QC math library does not have matrix operations and only has one complex function (to calculate the absolute value of a complex number). HP BASIC has matrix operations and a complex data type, so that a broad range of complex math is supported.

If you need more extensive math capabilities in QB and QC, you can do one of two things. Either you could locate a math library that has everything you need, in which case you'd have compatibility concerns, new commands to memorize, and lose the benefits of syntax checking in QB. Or you could write your own library of routines, in which case you'd take on a lot of extra work, probably lose some execution speed, and you'd lose syntax checking in QB.

Graphics

HP BASIC provides a greater range of commands and more commands relevant to the job of graphing results. HP BASIC has 48, QB has 17, and QC has 42.

HP BASIC has keywords like AXES, GRID, and PLOT, which are useful to the test engineer for graphing results. AXES allows you to set up the axes of a graph, add tic marks scaled properly, and emphasize certain tic marks -- in a single statement. In QB and QC you have to write a routine to draw each coordinate, add the small tic marks one at a time, and then add the emphasized tic marks one at a time.

The only way to duplicate the functionality of HP BASIC would be to write routines in QB and QC or search for more complete graphics libraries. The DADiSP package might be a typical choice to add a more powerful analysis and graphics tool to QB or QC. (Refer to the Background section for a simple graphics program written in all three languages. Compare the total number of lines to accomplish the same result.)

HP BASIC can set up a grid or size your labels with single commands. Do you want a label at 90 degrees or 45 degrees? HP BASIC can do it. Do you want to position your label in reference to some point? HP BASIC can do it. How about switching plotters without changing a lot of code? You guessed it -- HP BASIC can do it. Easily. With single statements.

Control Structures

HP BASIC, QB, and QC are equivalent in this area.

Data Types

HP BASIC has some potential weaknesses in this area.

	RMB	QB	QC
Pointers	no	no	yes
User-defined Types	no	yes	yes
Complex Data Type	yes	no	no*
Records	no	no	yes
32 bit integers	no	yes	yes

*The math.h file defines a complex structure for one function.

The key point here is deciding whether or not you need pointers, records, or long integers in your test systems. In most test systems, you don't.

Learning Products

HP BASIC vs. QB

HP BASIC's documentation is clear and includes a number of examples, once you've found the right manual. The problem is that you've got a whole shelf full of manuals to contend with (and store). QB has two manuals -- the first oriented toward using their environment, the second focused on programming.

QB's documentation seems to be produced for the layman; HP BASIC's seems to be aimed at an engineer. The QB manuals are so geared to looking easy that they're annoying. Pictures of senior citizens and children abound, stuffed animals sit on top of the computers ... you begin to lose patience.

On the other hand, they do make it easy to learn. The instructions are clear. The material's organized in a logical way. In addition to the manuals there is an on-line tutorial that takes about a half hour to complete.

HP BASIC has an excellent self-study course called Using HP BASIC for Instrument Control, which takes about 30 hours to complete. It goes from very easy to more difficult material like handling interrupts. This is a considerable advantage for HP BASIC, because you get detailed instruction on how to control instruments. In addition there are two manuals on interfacing techniques as well as customer classes on HP BASIC and instrument control. HP BASIC is a unified system, so the instruction is available. In QB, you don't learn how to control instruments or perform device I/O, because it's not a part of the language.

HP BASIC vs. QC

QC has one paperbound manual (no cloy pictures this time) geared toward teaching you the environment and associated software tools. You get a one chapter review of C ("C Quick Start") and a one chapter "Graphics QuickStart". The documentation team likes to append the adjective Quick to

names. Three manuals for the Microsoft C Optimizing Compiler are included for reference, and they are very similar to our own in style. Microsoft just provides a language reference for C, which is dry and reads like a dictionary. They leave the task of teaching C to the many books on the market already. The bottom line is that you're not going to be up and running very fast unless you already know C.

HP BASIC's manuals are superior to QC's. There are more of them, but you can at least learn the language from them, and we also have classes and the self-study course. Unlike QB, QC has no online demo.

Platforms

HP BASIC programs run on three platforms: Vectra PC/IBM PC AT (or compatible) using HP BASIC Language Processor (BLP), an HP BASIC Workstation, and HP-UX. BLP Release II (BLP-II) requires only 295K of PC memory, and it uses about 600K of its own RAM. In background mode, BLP-II requires 103K of PC RAM.

QB requires a Vectra PC/IBM PC (or compatible) with at least 384K of RAM and a minimum of 720K disk-drive capacity. A hard disk and 640K of RAM are recommended for best performance.

QC requires a Vectra PC/IBM PC (or strict compatible) running MS-DOS or PC-DOS Version 2.1 or greater; two floppy-disk drives or one floppy-disk drive and a hard disk; and 448K RAM.

Many would argue that C is the most portable language, but in test programs you have to add an I/O library. That means that if you want to move a QC program to a UNIX platform you will have to rewrite the I/O calls. An HP BASIC program can run on DOS or HP-UX without change. A significant advantage for HP BASIC is that you can move programs from DOS to an HP-UX operating system, which protects the customer's software investment.

Comparison Table

	RMB	QB	QC
Development Environment			
Edit/debug tools	yes	yes	yes
Syntax checking	yes	yes	no
Line formatting	yes	yes	no
Window environment	no	yes	yes
Single statement execution	yes	yes	no
Monitoring variables	yes	yes	yes
Help facility	no	yes	yes
Menu bar	no	yes	yes
Multitasking*	yes	no	no

* True multitasking on HP-UX. For multitasking on BLP-II, consult field training manual.

	RMB	QB	QC
I/O			
Unified I/O	yes	no	no
Needs I/O library	no	yes	yes
# of interrupts	41	12	0
Optimized data transfer	yes	no	no
Data format control	yes	no	no
I/O paths	yes	no	no
Computation			
Complex math	yes	no	no (in general)
Array operations	yes	no	no
Binary operations	yes	no	yes
Hyperbolic trigonometric functions	yes	no	yes
Graphics			
# of commands	48	17	42
commands that graph results	yes	no	no
Control Structures			
all basic types	yes	yes	yes
Data Types			
Complex type	yes	no	no
Records	no	no	yes
32-bit integers	no	yes	yes
Pointers	no	no	yes
User-defined types	no	yes	yes

	RMB	QB	QC
Learning Products			
Demo	no	yes	no
Self-study course	yes	no	no
Clear manuals	yes	yes	sometimes
Concise manuals	no	yes	no
Customer courses	yes	no	no
Platforms			
Vectra compatibles	yes	yes	yes
HP BASIC workstation	yes	no	no
UNIX*	yes	no	no

*(C) UNIX is a Trademark of the AT&T Corporation.

Sales Presentation Strategies

The objective is to find out what your customers want and then support them accordingly. For example, find out the level of commitment to a particular platform and a particular language. If they want PC's and QuickBASIC, then show how we can support them along those lines. You might recommend our HP-IB card. For easier test systems this may be a perfectly valid solution. You might ask if they are aware of the differences between QB and HP BASIC, and then talk a bit about HP BASIC's integrated system, ease of use, and power.

In the long run it's better to be the trusted instrumentation consultant rather than just a salesman. Strategically it would be much better in terms of support, if the customer has a system organized around HP BASIC.

For instance, what if they have a problem with a system composed of an HP instrument, a National Instruments I/O card, a third-party vendor PC, and a software package like ASYST? The PC vendor says the problem is probably in the card. National Instruments says their card supports ASYST. The customer calls HP. In this case, nobody has checked to see if these products work together harmoniously. No support engineer can be expected to know everything on the market. Now, if this system were organized around HP BASIC, the problem could probably be solved. Just as the cartoon with the race cars implies, the winner of the T&M 500 will probably be the one with the integrated system.

We divide customers into three camps here: novice, somewhat-experienced, and computer scientist. If your customer is very experienced, the chances of convincing them to switch to a new language are very low, so it's probably not a good place to spend your time. You could, however, recommend a Vectra and an HP-IB Command Library.

For the Novice

Chief concerns -- ease of use/learning; time to get up and running; cost.

Suggestions:

1. Demonstrate simplicity of OUTPUT in controlling instruments.

Get an HP 3478A Multimeter and show customer how easily he can control the instrument with commands like:

```
OUTPUT 723; "F2"      !changes function to AC volts.
                      !"F1" for DC.
OUTPUT 723; "R1"      !30 V range. "R-2" for 30 mV range.
```

2. Write a simple program and run it. Graphics examples are usually more interesting, such as:

```
GRAPHICS ON
GCLEAR
GINIT                          !initializing for graphics
VIEWPORT 30,100,20,80
FRAME                          !framing graphics viewport
WINDOW -10,10,-10,10           !scaling area
AXES 1,1,0,0                   !setting axes and tic marks
END
```

You can also implement the same thing with the live keyboard, but if you're using the BLP card be sure to hit F12 after each command when you want graphics to remain on the screen.

You might also insert some PLOT statements in the program, too.

3. Put a bug in the program and show how EDIT goes right to the problem; correct it; show CONTINUE.

4. Stress availability of a great deal of turnkey software applications associated with particular instruments. For example, the EMI Measurement Software is a general purpose program for making automatic commercial and military emission measurements using a spectrum analyzer.

5. Prepare to leave lessons 1, 2, 3, and 19 from *Using HP BASIC for Instrument Control* by working through a few examples from one of the lessons.

You have now demonstrated ease of use and learning and shown how quickly they could teach themselves. In passing you have demonstrated syntax checking, the speed of the development environment, the live keyboard, and some debugging.

If they are considering QB or QC, tailor the main argument for HP BASIC to stress integrated system and ease of use issues. Particularly, if you are going against QC, stress the ease of getting up and running, because there are many potential traps waiting for the novice user of C. Against QB, ask him to compare the amount of code it would take to execute the axes on the

graph you just made. The need for writing such routines would not be unusual in QB.

Usually the main issue is price if they are considering a Microsoft language and a National GPIB card. Will the total development cost be less though? HP BASIC is a more powerful language which includes an HP-IB software driver as well as RS232, GPIO, and SRM drivers. You get multitasking with BLP-II, and you can eventually migrate to a UNIX (*) environment. The single-vendor support combined with all the tools available within the language should more than make up for a lower initial cost with QB or QC.

On the issue of QB or QC interfacing with powerful analysis and graphics packages such as DADiSP or LabWindows, HP BASIC on BLP-II can do that too. Furthermore, HP BASIC is powerful enough to handle most analysis and graphics requirements without going to an off-the-shelf package.

For the Experienced Programmer

Chief concerns -- ease of use; power and completeness; time to get up and running; cost; migration to UNIX platform; multitasking

1. Ease of use can be demonstrated using the multimeter again going into a little more depth on each point.

2. Stress the integrated system issues.

3. Stess the I/O benefits in HP BASIC as examples of power and completeness. How would you develop interrupts in QB or QC?

4. Availability of test and measurement applications software.

5. Ability to use HP BASIC on three platforms including HP-UX with all the UNIX development tools.

6. Mention the customer classes and prepare to leave lessons 1, 2, 3,13,14, and 19 from *Using HP BASIC for Instrument Control.*

* (C) Trademark of AT&T.

Background

The Background section provides more information in the key comparison areas such as code comparisons or specific data types offered.

Development Environment

I'm assuming you already have information on HP BASIC specifics, so I'll focus on some interesting features in QB and QC.

QB's Immediate Window

The Immediate window at the bottom of the screen can be used as a utility window as you program in the View Window. For example, if you want to clear the output screen, you can execute the CLS statement from the Immediate Window. You can also use the Immediate Window for doing calculations, or experimenting with programming ideas and previewing results.

The Watch Window in QB or QC

When you select Add Watch from the Debug menu, you will be prompted for the variables or expressions you want to monitor. Then the Watch window lets you monitor these values while your program is running, so you can be sure they are within the expected range. Breakpoints are automatically displayed in the Watch window. (The Watch window only monitors values in the procedure where they were added to the window.)

Output Screen Command in QB or QC

This selection in the View menu causes QB to toggle between the environment and output screens created by your program.

Creating Executable Files from DOS in QB or QC

Once your program runs QB or QC, you can use the Make EXE File command from the Run menu to make a version of the program that runs directly from the DOS command line without being loaded into the QB or QC environment. In QB, using the Stand-Alone EXE File option you can get a program that executes a little faster and can be run on any DOS computer. This is due to the fact that the programs will include necessary support routines from the run-time module BRUN45.EXE when they are compiled.

The QB Debugging Procedure

In QB you can pause program execution to fix an error and then continue without going back to the beginning. You can single step or trace through a program. You can set breakpoints. The difference from HP BASIC would be in using the Watch window to monitor variables instead of the Live Keyboard. In QB, you might use Instant Watch to check the value of a particular variable or expression. You would also use the Immediate Window to test expressions vs. the Live Keyboard in HP BASIC. You cannot simply check the value of a variable by typing it in during execution as you can in HP BASIC, but you have plenty of tools to aid debugging.

The QC Debugging Procedure

1. Compile the program with the Debug option in the Compile dialog box turned on.

2. Turn on any debugging features such as breakpoints, watch expressions, or screen swapping.

3. Run your program by using the Start, Restart, and Continue commands on the Run menu. Observe the order in which statements and functions are executed, the changes made to watch variables, and the program output.

4. Edit your program as needed to correct errors you found in step 3. Then recompile your program.

5. If you need to change any of the debugging information, go back to step 2. Otherwise, go back to step 3 and rerun your program.

6. Repeat steps 3-5 until your program runs correctly.

Not only is this process more complicated than HP BASIC, but you don't get a feeling for how long it takes to hunt down compile errors BEFORE you get to use the debugger in step 3.

You also have to experience the delay in recompiling a program every time you think you've corrected a bug, before you can really know what slow development means. Of course, if you're very experienced and don't make many mistakes to begin with, this would be less important.

Quick Libraries in QB and QC

A Quick Library contains nothing but procedures in compiled code. These procedures can be written not only in QuickBASIC, but also in other Microsoft languages as well (C, Pascal, FORTRAN, and MASM). Quick Libraries have several uses:

- They provide an interface to other languages.
- They allow designers to hide proprietary software.
- They load faster and are usually smaller than modules.

For example, the HP-IB Command Library is contained in a Quick Library, which must be loaded when the environment program is run:

```
qb /l QB4HPIB          for QuickBASIC

qc /l QCHPIB           for QuickC
```

In addition, in QC you have to create a QC HP-IB library module, and you must declare the types of the HP-IB I/O functions in your program. HP has printed an application note with instructions for this--82990A Application Note.

All of these hassles will be corrected in an upcoming version of the Command Library.

Pointer Check Option in QC

The Pointer Check option in the Compile Dialog Box protects your program from common pointer errors that do not otherwise cause run-time errors. When you select this option, the QC compiler generates run-time code that verifies that pointer values address program data before the pointers are used. If not, QC displays a run-time error in the error window. Pointer checking can prevent errors that might overwrite QC or DOS code or data and cause QC or DOS to stop running.

Maintaining Large Programs in QC

Maintaining large programs with large numbers of modules is automatic with the QC Compiler. As you edit the modules that make up a program, you add them to a "program list." When you rebuild the program, QC saves you time by recompiling only the modules that have been changed since the last program build. The program list is saved in a file that is compatible with the MAKE utility; consequently, programs can be updated automatically outside the QC environment.

I/O

I/O Using an I/O Library

Since you can get more detailed information on HP BASIC's I/O from the manuals, I thought it would be appropriate to include a few examples of handling I/O with QB and QC using the HP-IB Command Library.

In QB and QC you would include a standard routine for handling errors. Instead of using HP BASIC interrupts you would check for errors on every library call. For example:

```
CALL IORESET(ISC&)
' interface to start-up state
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
' if error, branch to error routine
CALL IOTIMEOUT(ISC&, 5!)
' setting timeout parameter to 5 s.
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
CALL IOCLEAR(ISC&)
' clearing all devices on interface
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
```

A similar procedure in QC:

```
error = ioreset (isc);
error_handler (error, "IORESET");
error = iotimeout (isc, 5.0);
error_handler (error, "IOTIMEOUT");
error = ioclear (isc);
error_handler (error, "IOCLEAR");
```

Error_handler is a routine that will tell where an error has been uncovered and what type of error it is. The user may then do something about it or continue program execution.

Speed

The issue of speed is an important one and will be addressed with a thorough set of benchmarks in the near future.

A few observations -- speed on the bus will be controlled by the slowest instrument regardless of how fast the language may be. In real-time programming, either the slowness of the test or the speed of the instrument will be the controlling factor.

The run-time performance of QB and QC are very good, particularly on 80386 PCs, and are directly proportional to the investment in the PC platform. The speed of BLP-II is significantly better than the first release: 11 times better on boot speed, 370% better in graphics, and 570% better in hard-disk performance.

In terms of cost, the speed of program development and maintenance may be the crucial factor rather than the execution speed of the program. In RMB you can typically develop the test system faster and operate it with less problems. If you do have problems, you get quality, single-vendor support.

Graphics

RMB has more graphics commands than QB and QC, and it has commands more relevant to the task of graphing results. To demonstrate this, I have written a code segment in all 3 languages to display a graph.

In RMB

```
GRAPHICS ON
GCLEAR
GINIT    ! initializing for graphics
VIEWPORT 30,100,20,80    ! establish graphics area
FRAME    ! frame the graph
WINDOW -10,10,-7.5,7.5! scale graph
AXES 1,1,0,0    !create axes with tic marks
CLIP OFF    ! turn off clipping region
MOVE 0,-10
LORG 6    ! specify label location
LABEL "INPUT VOLTAGE"    ! label x axis
MOVE -10,0    ! prepare to label y axis
DEG    ! using degrees instead of radians
LDIR    ! putting y label at 90 degrees
LORG    ! specify label location
LABEL "OUTPUT VOLTAGE"
END
/***********************************************/
```

In QB

```
SCREEN 1    ' 320 X 200 resolution
VIEW (53,10)-(253,160), ,1    ' defining graphics region
WINDOW (-10,7.5)-(10, -7.5)    ' scaling graph

LINE (-10,0)-(10, -7.5)    ' x axis
LINE (0, 7.5)-(0, -7.5)    ' y axis

PRESET STEP(-10,7)    ' x axis tic marks
FOR i% = 1 to 19
LINE STEP(1,1)-STEP(0,-1)
NEXT i%

PRESET (-.5,8)    ' y axis tic marks
FOR i% = 1 TO 15
LINE STEP(1,-1)-STEP(-1,0)
NEXT i%

LOCATE 24,14    ' label x axis
PRINT "INPUT VOLTAGE"
FOR i% = 1 TO 14
LOCATE 2 + i%,2    ' label y axis
PRINT MID$("OUTPUT VOLTAGE", i%, 1)
NEXT i%
END
/*****************************************/
```

In QC

```
#include <stdio.h>
#include <graph.h>    /* graphics library    */
#include <conio.h>    /* console I/O library */

#define SCALER 10

struct videoconfig vc;    /* holds video configuration */
char error_message [] = "This video mode is not supported.";
/*-----------------------------------------------*/
```

- Continued -

```
main()
{
/* check to see if desired video mode is supported */
if (_setvideomode(_MRES4COLOR) = = 0)
{
printf("$s\n", error_message);
exit (0);
}
_getvideoconfig(&vc);/* get video configuration */
/* routine to make a graph */

make_graph();
while (!kbhit());        /* wait for keystroke */
_clearscreen(_GCLEARSCREEN);
_setvideomode(_DEFAULTMODE);/* back to default */
}
/*-------------------------------------------*/
make_graph()
{
/* declarations and initialization */

float ar;          /* aspect ratio          */
int xo, yo;        /* center screen coordinates */
char ylabel[16];   /* y axis label          */
int i;             /* looping variable      */
int row, col;      /* screen row and column */
int scaler;        /* for scaling axes      */

ar = (float) (8.5 * vc.numypixels)/ (7*vc.numxpixels);
xo = vc.numxpixels/2 - 1;
yo = vc.numypixels/2 - 1;

/* draw graph frame and x-y axes in center of screen */

_setlogorg(xo, yo);
_rectangle(_GBORDER, -100, ar * 75, 100, ar * -75);
_setcliprgn(xo - 100, yo - 75, xo + 100, yo + 75);
_moveto(-100,0);
_lineto(100,0);
_moveto(0, ar * -75);
_lineto(0, ar * 75);
```

- Continued -

```
/* label graph */

_settextposition(23,20 - 5);
printf("INPUT VOLTAGE");
row = 13 -7;
col = 20 - 17;
_settextposition(row, col);
strcpy(ylabel, " OUTPUT VOLTAGE");
for (i = 0; i< = 14; i+ +)
{
printf("%c", ylabel[i]);
_settextposition(row+ +, col);
}

/* add tic marks */

for (i = 1; i < = 19; i+ +)
{
_moveto(-100 + i * SCALER, -5 * ar);
_lineto (-100 + i * SCALER, 5 * ar);
}

for (i = 1; i < = 15; i+ +)
{
_moveto(-5, -80 * ar + i * SCALER * ar);
_lineto(5, -80 * ar + i * SCALER * ar);
}
}

/*****************************************************/
```

Notice the additional lines of code in QC to accomplish things that would be simple in HP BASIC -- about 3 times as much code. I'm sure there are ways to write this more concisely, but it does illustrate my point. You need to write a lot more code in QC than in HP BASIC, and there are a lot more places to cause errors. Furthermore, the simple errors will not be caught until you try to compile.

In QB, several lines of code are needed to duplicate HP BASIC commands.

Data Types

Using a loose definition of data types, I have included the actual names of the types supported in each language for reference.

RMB	QB	QC
I/O paths	integer	char
integer	long integer	int
real	single precision floating-point	short
complex	double precision floating-point	long
string	string	signed
array	array	unsigned
	user-defined	enum
		float
		double
		long double
		pointer
		array
		struct

Example Programs

The following examples program two instruments--an HP3325A Synthesizer/Function Generator and an HP 3456A Digital Voltmeter. First, the source is programmed to output a 2 Vrms signal, swept from 1 kHz to 10 kHz. Then, the DVM takes 20 readings from the signal and stores them in an array. Finally, the results are printed on the screen.

In QB and QC the program "includes" a file to declare variables and handle errors, which is ommitted here to save space.

This simple test program is included to give a brief overview of coding in all three languages.

Filter Test Program in QC:

```
#include <stdio.h>
#include <hpib.h>

long isc;   /* interface select code  */
long source;   /* HP3325A */
long dvm;   /* HP3456A */
short error; /* error flag on I/O calls */

#define SRQLINE 1

main ()

{
getdevs ();
initialize ();
source_setup ();
dvm_setup ();
trigger ();
wait_for_srq ();
readout ();
}

/*******************************************************/
```

- Continued -

```
error_handler (error, routine)
int error;
char *routine;

{
char *estring;
char ch;

if (error != NOERR)
{
printf ("Error in call to %s \n", routine);
printf ("    Error = %d : %s \n", error,
errstr(error));
printf ("Press  to continue: ");
scanf ("%c", &ch);
}
}
/*******************************************************/
getdevs ()
{
do
{
printf ("\n Enter HP-IB select code (usually 7) : ");
scanf ("%ld", &isc);
if ((isc<0) || (isc >16))
printf ("\n Invalid select code - must be in range 1-16 \n");
}
 while ((isc<0) || (isc >16));
 do
{
printf ("\n Enter bus address of DVM (0-29): ");
scanf ("%ld", &dvm);
if ((dvm < 0) || (dvm > 29))
printf ("\n Invalid bus address - must be in range\
0-29 \n");
}
```

- Continued -

```
while ((dvm < 0) || (dvm > 29));
do
{
printf ("\n Enter bus address of SOURCE (0-29): ");
scanf ("%ld", &source);
if ((source < 0) || (source > 29))
printf ("\n Invalid bus address - must be in\
range 0-29 \n");
else if (dvm = = source)
{
printf ("\n DVM and SOURCE may not have the\
same address \n");
source = -1;
}
}
while ((source < 0) || (source > 29));
dvm = isc * 100 + dvm;
source = isc * 100 + source;
}

initialize ()
{
error = ioreset (isc);
error_handler (error, "IORESET");
error = iotimeout (isc, 5.0);
error_handler (error, "IOTIMEOUT");
error = ioclear (isc);
error_handler (error, "IOCLEAR");
}
/*************************************************************/
```

- Continued -

```
source_setup ()
{
 char *codes;

codes = "RF2 FU1 ST1KH SP10KH MF1KH AM2VR TI5SE";
error = iooutputs (source, codes, 38);
error_handler (error, "IOOUTPUTS");
}
/*******************************************************/
dvm_setup ()

{
char *codes;

codes = "H SM004 F2 R4 FL0 Z0 4STG 20STN RS1 T4";
error = iooutputs (dvm, codes, 38);
error_handler (error, "IOOUTPUTS");
}
/*******************************************************/
trigger ()

{
char *codes;

error = iotrigger (dvm);
error_handler (error, "IOTRIGGER");
codes = "SS";
error = iooutputs (source, codes, 2);
error_handler (error, "IOOUTPUTS");
}
/*******************************************************/
wait_for_srq ()

{
int response;

do
{
do
{
error = iostatus (isc, SRQLINE, &response);
error_handler (error, "IOSTATUS");
}
```

- Continued -

```
while (response = = 0);
error = iospoll (dvm, &response);
error_handler (error, "IOSPOLL");
}
while ((response & 68) ! = 68);
}
/*******************************************************/
readout ()

{
char *codes;
float readings[20];
int i;
int numvalues;

numvalues = 20;
codes = "SO1 -20STR RER";
error = iooutputs (dvm, codes, 14);
error_handler (error, "IOOUTPUTS");
error = ioeoi (isc, 0);
error_handler (error, "IOEOI");
error = ioentera (dvm, readings, &numvalues);
error_handler (error, "IOENTERA");
printf ("\n The readings are: \n");
for (i = 0; i  numvalues; i + + )
printf ("%f \n", readings[i]);
}
```

Filter Test Program in QB:

```
DECLARE SUB INITIALIZE (ISC&, DVM&, SOURCE&)
DECLARE SUB SOURCESETUP (SOURCE&)
DECLARE SUB DVMSETUP (DVM&)
DECLARE SUB TRIGGER (DVM&, SOURCE&)
DECLARE SUB WAITFORSRQ (ISC&, DVM&)
DECLARE SUB TAKEREADINGS (ISC&, DVM&, MAX.READINGS%,
 ACT.ELEMENTS%)
DECLARE SUB PRINTREADINGS (ACTUAL%)

REM $INCLUDE: 'QB4SETUP'

OPTION BASE 1
MAX.READINGS% = 20
DIM READINGS!(20)
ACT.ELEMENTS% = 0
CALL INITIALIZE(ISC&, DVM&, SOURCE&)
CALL SOURCESETUP(SOURCE&)
CALL DVMSETUP(DVM&)
CALL TRIGGER(DVM&, SOURCE&)
CALL WAITFORSRQ(ISC&, DVM&)
CALL TAKEREADINGS(ISC&, DVM&, MAX.READINGS%,
ACT.ELEMENTS%)
CALL PRINTREADINGS(ACT.ELEMENTS%)
END

SUB DVMSETUP (DVM&) STATIC
SHARED PCIB.ERR, PCIB.BASERR, NOERR

CODES$ = "H SM004 F2 R4 FL0 Z0 4STG 20STN RS1 T4"
CALL IOOUTPUTS(DVM&, CODES$, LEN(CODES$))
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
END SUB
```

- Continued -

```
SUB INITIALIZE (ISC&, DVM&, SOURCE&) STATIC
SHARED PCIB.ERR, PCIB.BASERR, NOERR

INPUT "Enter Select Code: ", ISC&
PRINT
INPUT "Enter bus address of VOLTMETER: ", DVM&
DVM& = ISC& * 100 + DVM&
PRINT
INPUT "Enter bus address of SOURCE: ", SOURCE&
 SOURCE& = ISC& * 100 + SOURCE&
PRINT
CALL IORESET(ISC&)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
CALL IOTIMEOUT(ISC&, 5!)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
CALL IOCLEAR(ISC&)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
END SUB

SUB PRINTREADINGS (ACTUAL%) STATIC
SHARED READINGS!()

PRINT "THE READINGS ARE: "
FOR I% = 1 TO ACTUAL%
PRINT I%, READINGS!(I%)
NEXT I%
END SUB

SUB SOURCESETUP (SOURCE&) STATIC
SHARED PCIB.ERR, PCIB.BASERR, NOERR

CODES$ = "RF2 FU1 ST1KH SP10KH MF1KH AM2VR TI5SE"
CALL IOOUTPUTS(SOURCE&, CODES$, LEN(CODES$))
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
END SUB
```

- Continued -

```
SUB TAKEREADINGS (ISC&, DVM&, MAX.READINGS%,
ACT.ELEMENTS%) STATIC
SHARED PCIB.ERR, PCIB.BASERR, NOERR, READINGS!()

CODES$ = "SO1 -20STR RER"
LENGTH% = LEN(CODES$)
CALL IOOUTPUTS(DVM&, CODES$, LENGTH%)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
'
STATE% = 0
CALL IOEOI(ISC&, STATE%)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
CALL IOENTERA(DVM&, SEG READINGS!(1), MAX.READINGS%,
ACT.ELEMENTS%)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
END SUB

SUB TRIGGER (DVM&, SOURCE&) STATIC
SHARED PCIB.ERR, PCIB.BASERR, NOERR

CALL IOTRIGGER(DVM&)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
CODES$ = "SS"
CALL IOOUTPUTS(SOURCE&, CODES$, LEN(CODES$))
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
END SUB

SUB WAITFORSRQ (ISC&, DVM&) STATIC
SHARED PCIB.ERR, PCIB.BASERR, NOERR

SRQ% = 1
CHECKSTAT: CALL IOSTATUS(ISC&, SRQ%, STATUS%)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
IF STATUS% = 0 THEN GOTO CHECKSTAT
CALL IOSPOLL(DVM&, RESPONSE%)
IF PCIB.ERR  NOERR THEN ERROR PCIB.BASERR
IF (RESPONSE% AND 68)  68 THEN GOTO CHECKSTAT
END SUB
/*= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =*/
```

Notice that the QC program is about twice as long as the QB program. Also, notice IOSTATUS and IOSPOLL calls are being used instead of interrupts to check for errors. Don't forget that there is a setup file for error-handling in both cases, which is 33 lines long in QB and 26 lines long in QC.

Filter Test Program in RMB:

```
INTEGER Isc, Dvm, Source
OPTION BASE 1
DIM Readings(20)
!
GOSUB Initialize
GOSUB Sourcesetup
GOSUB Dvmsetup
GOSUB Trigger
GOSUB Wait_for_srq
GOSUB Takereadings
GOSUB Printreadings
STOP
!
Dvmsetup: !
!
OUTPUT @Dvm; "H SM004 F2 R4 FL0 Z0 4STG 20STN RS1 T4"
RETURN
!
Initialize: !
INPUT "Enter Select Code: ", Isc
INPUT "Enter bus address of VOLTMETER: ", Dvm
ASSIGN @Dvm TO (Isc * 100 + Dvm)
INPUT "Enter bus address of SOURCE: ", Source
Source = Isc * 100 + Source
ASSIGN @Source TO (Isc * 100 + Source)
ASSIGN @Isc TO Isc
RESET @Isc
CLEAR @Isc
RETURN
!
Printreadings: !
CLEAR SCREEN
PRINT "THE READINGS ARE: "
PRINT Readings(*)
RETURN
```

- Continued -