RSTS PROFESSIONAL

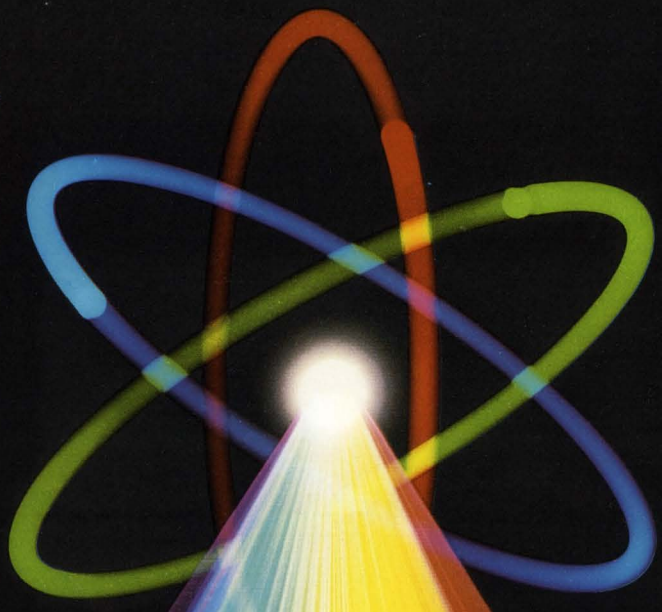
Volume 5, Number 2

April 1983
10^{00}/issue, 35^{00}/year

INSIDE:

Visit us at
DEXPO™ East 83
Kiel Auditorium
St. Louis, Missouri
May 22-24, 1983
Booth #108

NOW AVAILABLE FOR VAX



USER-11: POWERFULLY PRODUCTIVE.

People productivity. It's more important than ever. And a good database system can mean *real* productivity.

USER-11 is a high-performance database system.

It is a fact: Software designed with USER-11 is built more quickly, operates more reliably, and performs better than other software techniques.

USER-11 is unique. It's easy to install. Easy to learn. And easy to apply. Adaptive tools and a standard approach ensure that maintenance is easier than ever.

A key to USER-11's success is its powerful, dictionary-based modules. Software developers simply describe and assemble these modules to create custom business packages—at an unprecedented rate.

Naturally USER-11 is supported with excellent documentation and a variety of training options for beginner to expert. Our commitment is to your complete satisfaction.

Whether you are a software provider or a software user, we guarantee you will be delighted.

Ask us about USER-11 and our family of business software products, or better yet, ask a *productive* USER!

NCCS USER 0110

North County Computer Services, Inc.
2235 Meyers Ave.,
Escondido, California 92025
(619) 745-6006, Telex: 182773

***USER-11 is currently available for DEC computers using the RSTS and VMS operating systems.**

CIRCLE 30 ON READER CARD

What's the most efficient way to distribute financial models from DEC to desks?*

MAPS/*Host*™ & MAPS/*Pro*™

MAPS™ financial modeling software lets you offer your decision makers a single, comprehensive solution to their complex financial processing needs.

MAPS/Pro software is designed to run on DEC's powerful new Professional 350 desktop computer—to give you the full benefit of the 350's P/OS multi-tasking operating system, 5MB Winchester disk, bit-mapped graphics, special function keys and application menus. Fully-compatible MAPS/Host software runs on PDP-11 and VAX computers.

Working independently at desktop 350s, MAPS lets you and your users take advantage of state-of-the-art microprocessing hardware and software.

With 350s linked together, MAPS' common planning language and centrally-administered data base allow users to exchange data and financial models directly. Plus users can upload applications to a PDP-11 or VAX when additional computing power is needed—making *truly* distributed financial planning possible.

Designed for total financial decision support, MAPS features unlimited logic, professional report formats, sophisticated consolidation capabilities and advanced data calculations. Plus instant access to a common data base and model library that maximizes user efficiency. With MAPS there's no limit to the size or complexity of the system you can develop. Yet MAPS' online Help, Business English, and visual data editing make MAPS easy to use. And, fully-documented, it's easy to learn. What's more, Ross "Hot-line" support is as close as the phone if a problem does occur.

For more information on MAPS software, just return our coupon. Or call Ross Systems toll free at (800) 547-1000 (in California, call (415) 856-1100).

* DEC, VAX *and* PDP *are registered trademarks of Digital Equipment Corporation.*
MAPS, MAPS/*Host and* MAPS/*Pro are registered trademarks of Ross Systems Incorporated.*

OK, ROSS

... I'd like to know more about how MAPS can offer me true distributed financial planning.

Please send more information on ☐ *MAPS/Host* ☐ *MAPS/Pro*

☐ *I/We have access to a VAX or PDP-11*

☐ *Have a Ross representative call on me*

R4

Name _____

Title _____

Company _____

Address _____

City _____ *State* _____ *Zip* _____

Phone (_____) _____

CIRCLE 1 ON READER CARD

ROSS SYSTEMS

1860 Embarcadero Road
Palo Alto, CA 94303
Regional offices in San Francisco, New York, Dallas, Los Angeles.

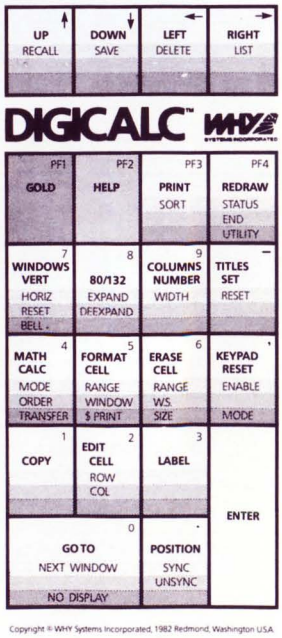
DIGICALC,™ Powerful Financial Forecasting

With incredible range and depth, DIGICALC can provide the executive, accountant or professional with a worksheet capable of multi-year forecasts, budgeting and consolidations.

DIGICALC is designed for use on DEC* systems exclusively. The significance of its design is the incomparable degree of "help" built into it. DIGICALC has been called "the most user friendly program available for DEC computers."

DIGICALC runs on VMS*, RSTS/E*. More information is available in brochure form, but to really *feel* the power of DIGICALC call today for a free dial-up demonstration.

- FINANCIAL MODELING.
- ON-LINE HELP AND SELF TEACHING MODE.
- TEN KEY NUMERIC DATA ENTRY.
- EXTERNAL FILE INTERFACE.
- "BOARDROOM QUALITY" REPORTS.
- EXTENSIVE MATH FUNCTIONS ALGEBRAIC FUNCTIONAL LOGICAL SCIENTIFIC USER DEFINED FUNCTIONS.
- SAVES AND RECALLS WORKSHEETS.
- REGRESSION FOR FOUR PERIOD WITH BI-VARIANCE AND NINE PERIOD WITH FIXED VARIANCE.
- DYNAMIC NUMERIC AND ALPHABETIC SORTING OF ENTRIES BY ROW FOR AN ENTIRE WORKSHEET.
- CONSOLIDATION OF ANY NUMBER OF WORKSHEETS INTO A SINGLE SUMMARIZED OUTPUT WORKSHEET.



SYSTEMS INCORPORATED
16902 Redmond Way
Redmond, WA 98052 U.S.A.
(206) 881-2331

*DEC, VMS, RSTS/E are registered trademarks of Digital Equipment Corporation.

CIRCLE 72 ON READER CARD

Contents

Coming . . .

- QUEDEV.B2S
- Compilation Aid for Basic Plus Two
- Memo: "A Computerized Note File"
- Tape Copy
- ACCTNG.B2S
- The History of RSTS
- "Watch & Ring Me"
- RSTS/E Disk Optimization in a Multi-User Environment
- Menu Management Program
- How to Use Shareable Data
- Act 4 — A 4th Generation System Development
- ISCAN.BAS
- Future of RSTS on Micro-11's
- More . . .

The RSTS Professional Magazine, April 1, 1983, Vol. 5, No. 2. ISSN 0745-2888 is published bi-monthly by M Systems, Inc., 161 E. Hunting Park Avenue, Philadelphia, PA 19124. Subscriptions: single copy price 10^{00}, 35^{00} per year; $50 Canada and 1st class. All other countries, air mail, $60 US. Second Class postage paid at Philadelphia, PA. POSTMASTER: Send all correspondence and address changes to: RSTS Professional, P.O. Box 361, Ft. Washington, Pa. 19034-0361, telephone (215) 542-7008.

From the Publishers . . .

THE RIGHT TOOL

Carl B. Marbach

With the correct tool some jobs that are otherwise impossible are easy. I learned that fact one summer working in a gasoline station. Changing drum brakes is a cinch with the brake tools, without them it is dangerous to fingers and other anatomical parts. Spark plugs can be changed easily if you have a long 7/8" socket and wrench, try it with an open end wrench and you'll wind up with skinned knuckles.

With computers the same rule applies. Notice how programmers run to get the best tool; TECO still lives, EDT can't be lived without and LINED, the PDP-10 original line oriented editor died. Other tools we use include cross reference programs, load maps and debuggers. We use the computer itself as a tool for programming.

What happens if you don't have a tool you need? The job can usually be done but it takes longer and has more mishaps along the way. In the early versions of RSTS we didn't have multi-language capability. Some independent software people solved that one, but it wasn't as straightforward as it is now. Want to write FORTRAN? Now you can use the best of the PDP-11 compilers on RSTS, but this was not always the case. We need the best tools available because without them we cannot make the best use of our system.

Sometimes RSTS has lagged behind the others in providing needed tools. DECNET for RSTS was last to be implemented and Phase 4 is not likely to be available at all. RSX has I and D space set up so that a program can address an additional 64K. I have been told that doing the same for RSTS is possible, but it is not planned. Needed features for BASIC-PLUS are not being added for lack of space (it is rumored to be within one or two bytes of its limit). RSTS is lagging again, but this time I think it is very serious.

DEC has put forward a new line of "personal" computers. Included in this line is the PROFESSIONAL series based on the PDP-11. Unfortunately, the operating system used on the Professionals is a derivative of RSX called POS (Professional Operating System). DEC is providing a "tool kit" for use on VAX and RSX for program development for software to be run on the Professional computers. NO RSTS TOOLKIT! Why? Is it too hard to do? I can write programs on RSTS for both VAX and RSX, why not the Professional? Is RSTS not important enough? A popular operating system for the personal computers in the marketplace today is CP/M (By Digital Research, Inc.) which most observers think was modeled after RSTS. According to DEC, they sold more RSTS systems last year than in any year prior to that. It is important. You and I know that.

We now have Version 8.0 soon to be released. It has things that I can't imagine were important to many of us. How many really wanted a new directory structure? How many needed it? I have had reports that BASIC V2 (Version 2 of BASIC-PLUS 2) has many RSTS problems, not the least of which is a faulty installation procedure. Shops currently running BASIC-PLUS 2 can not just put the new product up and have it work; one commercial installation submitted six SPR's in one day! They were told by telephone response that these might not be fixed until the next release which will ONLY run on V8.0 of RSTS.

What are OUR developers thinking about. We need tools! I have taken the position that integrating DEC's new personal computers into our existing systems is critical to keeping us up to date. But, no tools! Can I write a BASIC-PLUS 2 program and run it on a Professional 350? I think I can; but I could do it better if the people who represent us at DEC would provide us with some of the things we really need in timely fashion. Why RSX and VAX and not RSTS. Scream. Holler. Yell.

SOFTWARE — A FLOPPY FUTURE

Dave Mallery

I have been thinking about software (as a business) and some of the futures that I can see . . .

First of all, a survey:

Price Range	Market	$$ Expected
$50-$200	10000 + mass merchant mail order blister pack	$1,000,000
$500-$2500	500-2000 units national advt mail and phone	$1,000,000
$5000-$15000	250-500 units national advt mail blitz oem and rep small sales force	$1,000,000
$25,000-$75,000	20-100 units vertical markets 3-piece suits own oem set-up	$2,000,000

It seems that in every case, there is about the same amount of money to be made, and the market definition drives the price. I repeat that this is the current situation, not the future.

Whether you noticed or not, the world as we knew it has just changed. The change is not limited to steel-workers, auto assemblers and tire makers.

Just as new technologies and cheaper overhead finished off the above industries as we know them, the existence (real or imagined) of a $10,000 PDP/11 under your desk and a spread sheet in your terminal (Rainbow or Pro) has changed the way our industry will operate in the future. Check out Lisa, 1,2,3 and the like. Take a good look . . . a very good look.

I note with interest that a VAX 750, configured to perform in 11/70 dimensions, costs today what the 11/70 cost in 1975. I guess that for a few more years, DEC will be able to sell big Caddies and Lincolns to the affluent few. After all, they have to get the billions invested in VMS back somehow.

The bulletin that I have for you today is that $3000 machines don't use $10,000 software. Dollar-fifty gas does not run 8 mpg chariots. (for long) . . .

Some questions (sans answers):

1) Will Computerland and the like control all future software distribution?

2) Who will be able to afford the outlay to create software for mass markets since break-even will be at 5000 units per annum?

3) How do you do a new release or a maintenance update for an $85 package with 10,000 users?

4) Ain't it fun to be alive? ♥

RSTS PROFESSIONAL

Publishers: R.D. Mallery, Carl B. Marbach

Managing Editor: James L. Trichon

Director of Advertising: Helen B. Marbach

Business Manager: Peg Leiby

Assistant to the Editor: Hope Makransky

Editorial Assistant: Lindia DiBiasio

Subscription Fulfillment:
Kathi B. Campione, Claire Hollister
Steven Barsh, Margie F. Pitrone
Connie Mahon

United Kingdom Representative:
Pauline Noakes
RTZ Computer Services Ltd., P.O. Box 19
1 Redcliff Street, Bristol, BS99-7JS
Phone: Bristol 24181

Mid-Atlantic Representative:
Ed Shaud
P.O. Box 187, Radnor, PA 19087
Phone: 215/688-7233

N.Y. & New England Representative:
Jack Garland
P.O. Box 314 S.H.S., Duxbury, MA 02332
Phone: 617/934-6464

Contributors:
Philip G. Anthony, Robert L. Besner
Steven Edwards, M. Christopher Getting
Craig Goodrich, Michael C. Greenspon
Edward A. Heinrich, Jim Hobbs
Michael H. Koplitz, Michael Mayfield
Bob Meyer, Kevin Munday
David Patterson, Steve Roy
Kelvin Smith, John Wolnisty

Cartoons: Douglas Benoit

Design, Typesetting & Layout: Grossman Graphics

Printing & Binding: Schneider Litho Co., Inc.

Cover Photo: David Sheppard

Editorial Information: We will consider for publication all submitted manuscripts and photographs, and welcome your articles, photographs and suggestions. All material will be treated with care, although we cannot be responsible for loss or damage. (Any payment for use of material will be made only upon publication.)

BEFORE you add memory (or anything else) to increase system performance

You should add DOPTER!

DOPTER is an easy to use RSTS/E disk copying program which

INCREASES SYSTEM PERFORMANCE UP TO 50%.

DOPTER performs all of the standard functions necessary to structure a RSTS/E disk volume and automatically does the following:

- Places all files and free space in their optimum positions.
- Produces better optimized MFD/UFD's than REORDR.
- Deletes unused file attributes from source, task, and object library files saving UFD and cache accesses.
- Places and pre-extends the MFD.
- Places the most used files at the front of the UFD's.
- Places the UFD's with the most activity toward the front of the MFD.

For More Information

If you would like more information on how you can increase the performance of your RSTS/E system with DOPTER and a free copy of "RSTS/E DISK OPTIMIZATION IN A MULTI-USER ENVIRONMENT", phone or write SPH today.

RSTS/E is a registered trademark of Digital Equipment Corporation.

SPH

System Performance House, Inc.

5522 Loch More Court · Dublin, Ohio 43017 · 614-265-7788

CIRCLE 108 ON READER CARD

LETTERS to the RSTS Pro...

Send letters to: Letters to the RSTS Pro, P.O. Box 361, Ft. Washington, PA 19034-0361.

A short note that you may find suitable for the RSTS Professional letters column.

I was asked if it was possible to make the "DETACH" SYS call non-priviledged. I said "No" but started looking and seem to have found a patch that allows it.

The patch is an adaptation of feature patch 3.5.4. That patch told how to make privileged, several non-privileged SYS calls. In essence, it told how to turn on a bit for each call in a table. My patch turns OFF that bit for the appropriate table entry. Here it is:

```
RUN $ONLPAT
Command File Name? <cr>     (RETURN for manual patching)
File to Patch? <lf>         (LINEFEED to patch installed
File found in account [0,1] monitor)
Module name? RSTS
Base address? $UUOTB
Offset address? 7-UU$MIN*2
 Base   Offset  Old       New?
?????? ?????? ??????  ? Q&177776
?????? ?????? ??????  ? ^C   (Uparrow-C to exit)
```

The patch has been tried and tested on RSTS 7.0, 7.1, and 7.2. The "old" value that's changed varies from one release to another, and possibly from one version of a release to another. But the detach call does become non-privileged, so I must be doing something right. I also suspect that it will be relatively immune to changes in RSTS.

If anyone does choose to apply this patch, they may want to apply the sequence number part of the original (Base: $$0305, Offset: O, set to: Q!200) to indicate that a "fiddle" has been done. They may also want to keep the "old" value just in case they want to put it back sometime (though setting that location to Q!1 will do just as well).

Tom Britton, Sr. Systems Programmer
CBL Canterbury Limited
New Zealand

* * *

I liked Allan Woloshin's article on the Enable/34. However, there are some things that I would like to add, in the hope that it might assist some readers. We have a PDP-11/45 with 512Kw (and an Enable/34, of course).

One disadvantage not mentioned is that data space is disabled by the Enable patches. This means fewer small buffers on your system, if that is a concern. Also, not only does the CPU not go any faster, it actually slows down! Technically, the CPU itself runs at the same speed, but the Enable slows memory access down. Someone from Able Computer mentioned a 10% reduction in memory speed. I have no benchmarks to prove or disprove this. I will say, however, that even a 10% reduction in memory speed is a vast improvement over being disk-bound.

I take issue with the statement that the Enable is difficult to install. The documentation is sufficient and we were up within a half-hour. Maybe this is a function of the difference between a PDP-11/34 and a PDP-11/45.

We, too, experienced some difficulties (after the Enable was installed) having to do with Tape Controller activity and bus termination. The worst part of the entire ordeal was not having any diagnostics. Supposedly, according to our friendly Able technician, they have finally released diagnostics. I'll believe it when I see it. I heard something similar over one year ago, but we never saw the software.

Overall, I was pleased with Able Computer's assistance and the dramatic performance increase after installing the Enable. If asked — "Are the headaches worth the extra memory?" — I would have to say an *un*hesitant, "Yes!" All of our Run Time Systems are now resident. We have resident libraries for the first time, more than a token XBUF, and almost no swapping.

My advice (if anyone wants it) is to add an Enable if you have 11/45 or some other 18 bit machine. If you have an 11/34, then sell the CPU and replace it with an 11/44 and extra memory. That would be more cost-effective. Depending on where you buy your parts, it may be outright less expensive.

As of last fall, we got an 11/70 to supplement our 11/45. The 11/70 is faster, it has CACHE and ¼MB more memory (and therefore more XBUF). Yet, the 11/45 has followed right on its heels. From a user's standpoint — even with 30 jobs on the system — they cannot tell the difference between the 11/70. Compare that to one year ago, when it took five minutes to log-on without the Enable, and I think you'll understand my favorable attitude toward it.

Alan Conroy, System Manager
Seattle Pacific University, Seattle, WA

* * *

In any situation there are PROS and CONS. I'm glad to see that the ENABLE article (FEB '83) was presented with both. I'm sure as time goes on, any vendor can and does iron the bugs out of their hardware. But its important for anyone to be careful when purchasing hardware (even from DEC), since bugs are inevitable in new technology. Keep an eye out for my next article: "Buyer Beware!!"

Steve Roy
Diversified Consulting
Bloomfield, CT

* * *

Just a note to tell you how much I enjoyed your article on "Decus in Australia" in the December RSTS Pro. A very glowing report and, in contrast with many US visitors, you got (almost) all your facts right! A few points you might find of interest:

- The "bedroom communities" and "minor cities" you refer to, are indeed called "suburbs" locally.
- Canberra was actually *created* to be the Australian capital. The *site* for Canberra was chosen to be halfway between Sydney & Melbourne, to prevent rivalry.
- No grafitti in Melbourne? I've seen plenty — but then I haven't seen yours!
- One wallaby, two wallabies.
- The Koala is related to the Wombat and is definitey *NOT* a bear!

Any chance of copies of some of the photos - particularly those of Chris, yourself, Carey (in the library booth) and myself?

I'm glad you enjoyed your visit to Australia & your participation in Decus. I certainly enoyed making your acquaintence and benefiting from your (and Dave's) knowledge and experience.

Your editorial in the "RSTS Professional" of December 1982 ("If you want it you have to ask for it") was discussed at the December meeting of the Decus Australia Board. As a RSTS person and a representative from the world of commerce, I was asked to write to you on behalf of the whole Board, to express our appreciation.

We in Decus Australia have a wish to handle the "Commercialism" issue both realistically and responsibly. We are pleased that your editorial was written in that same spirit, presenting what we feel to be an accurate reflection of our attitude, to the RSTS community.

Jo Kruithof, Decus Australia

* * *

I started reading and enjoying the RSTS Professional about six months ago and I believe that there has been an article on the advantages of task-building Basic-Plus programmes against RSX using Digital's CSPCOM which is bundled into Version 7.0 and onwards.

Bearing in mind the volume of sites around the world which are still using Basic-Plus (as opposed to Basic-Plus+2) I do not think that it has been brought home to your readers the advantages which this process can bring.

We have developed an extensive accounting package known as INCA which consists of some 150 Basic-Plus programmes and recently completed the task-building of this suite. In order to obtain some measure of the gain from this process we ran two bench-mark tests, on our 11/34 running under RSTS V7.2, first under the .BAC programmes and then under the .TSK programmes. Each bench-mark consisted of a series of different operations (i.e. file merge, file update and print to spool file) and as it happened, under .BAC the elapse time was 700 minutes and 701 minutes. Under .TSK the elapsed time was 450 minutes and 429 minutes respectively.

The run time statistics were even more impressive. Under .BAC they totalled 36,750 and 36,887 CPU seconds whereas under .TSK they only logged up 16,559 and 16,962 seconds.

. . . continued on page 70

EMULEX TALKS DEC 3

TAMING THE EAGLE...

High-speed disk drives, like the Fujitsu Eagle, on your Q-bus? You bet, with Emulex's new SC03. This single-board controller supports full 22-bit addressing, boasts 14-sector buffering, and can handle 32 different combinations of drive configurations and much more—all for $2800 list.

Current emulations: single or multiple RM02, RM05, RM80 or RP06's in either standard or expanded versions. You can look for the models that support the Eagle in April.

THROUGHPUT, THROUGHPUT, AND MORE THROUGHPUT...

Why should you VAX-750 and 780 users wait in a data traffic jam on the Unibus? An Emulex-controlled subsystem incorporating a CDC, Fujitsu, or other SMD disk drive can hook you right into the high-speed CMI or SBI bus.

Q-bus and Unibus environmentalists, take heart: You can plug an Emulex-packaged subsystem into Q-bus and Unibus models and gain efficiency—like no more infuriating, software-crunching "data lates."

BUSY SIGNALS...

All you DMF-32 dependent VAXers can expand communications with the DMF-32 emulating F models of Emulex's CS11 and CS21 communications MUXes. The F's offer 16 to 48 remote modem-controlled lines, compared to two remote modem-controlled lines per DMF-32. Present CS11-21ers can upgrade to the F models via a $350 PROM change.

What's more, Emulex hears that DMF-32's are available from DEC only on certain VAX-11 models, and then only after a wait. The F's are available now for all VAX-11s.

RECIPE FOR VAX STORAGE...

Take one Fujitsu Eagle, add one CDC Keystone streamer for backup, mix well with an Emulex disk controller and an Emulex tape coupler, and pour into a 42-inch DEC-compatible cabinet. The result: The PXD51 Series, a complete line of high-speed, high-reliability mass-storage subsystems that provide combined disk and tape capability.

Storage capacities: 349 to 414 MBytes. Prices from $26,150. Compare that to the competition, byte for byte.

FROM THE EMULEX FILE...

Emulex's figures for the first half of FY '83 are in: Revenues up 98 percent; earnings up 120 percent; earnings per share up 93 percent (that reflects our second public offering, completed in October). Emulex has reduced prices on selected Q-bus and Unibus products—SC02, SC04, TC01, SC21/V, and TC11. Special invitation: Next time you're in Southern California, give us a call to schedule a visit to our new 70,000 square-foot home in Costa Mesa, and we'll talk DEC there.

3545 Harbor Blvd., P.O. Box 6725,
Costa Mesa, California 92626,
Toll Free (800) 854-7112, In Calif. (714) 662-5600.

CIRCLE 58 ON READER CARD

THE RSTS/E BACKGROUND

By Michael H. Koplitz

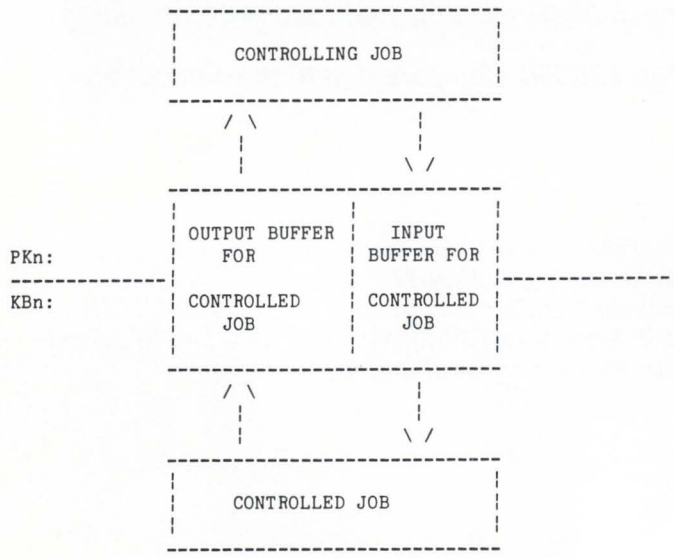
Background processing is used to allow a user to accomplish a task without tying up his terminal or his time. While a background process is running, the user can continue his efforts in other areas. RSTS/E allows a type of background processing with the BATCH system, which is part of the spooling package. APTK and ATPRO could be thought of as background processors, except that the terminal is tied up. Background, therefore, has not been available under RSTS/E until now.

THE PSEUDO KEYBOARD

To be able to create the illusion of background, the RSTS/E feature of the pseudo keyboard is used. A pseudo keyboard is a logical device which has all of the characteristics of a terminal, but there is not any physical terminal associated with it. The pseudo keyboard has both input and output buffers. Naturally, programs can access these buffers.

The number of pseudo keyboards is determined during SYSGEN time. RSTS/E assigns the device name PKn: to each pseudo keyboard and associates each with a keyboard unit number, KBn:. A physical keyboard is not allocated to the pseudo keyboards. Therefore if eight pseudo keyboards are generated during SYSGEN, the first non-pseudo keyboard would be KB9:.

Usage of a pseudo keyboard requires a controlling program. The pseudo keyboard will respond to RSTS/E commands the same as any other terminal on the system would. The actual input/output buffers for the pseudo keyboard are the input/output buffers from the controlling program. The controlling program will OPEN the pseudo keyboard on a RSTS/E channel. This causes input/output buffers to be created. These buffers are associated with the pseudo keyboard.



PSEUDO KEYBOARD OPERATIONS

PSEUDO KEYBOARD OPERATIONS

The controlling job uses the pseudo keyboard (PKn:) to perform input and output operations to the controlled job. The controlled job accepts the commands and returns information to a keyboard number (KBn:). The controlled job does not know that it is dealing with a pseudo keyboard when it accesses KBn:.

USING A PSEUDO KEYBOARD

Utilization of a pseudo keyboard is done via the BASIC-PLUS or BASIC-PLUS-2 language. An "OPEN" statement is used to access a pseudo keyboard. The pseudo keyboard is treated like a file.

```
100 OPEN "PK1:" AS FILE #1%
```

This statement will open PK1: on channel one. PK1: will translate into KB3:. If KB3: is "opened" by the controlling job, an error will be generated. As far as the controlling job knows, KB3: is disabled.

Two errors can occur on the open statement. ?NOT A VALID DEVICE (ERR = 6) can occur if the pseudo keyboard specified does not exist on the system. If the pseudo keyboard requested is in use the error ?DEVICE NOT AVAILABLE is generated.

CREATING THE JOB

The pseudo keyboard must first be opened on a channel before the job can be created. The job is then created by placing a valid sign-on command in the output buffer of the pseudo keyboard channel. PRINT or PUT statements are used to place commands into the output buffer. Remember that the output buffer for the pseudo keyboard from the controlling job becomes the input buffer for the controlled job. For example, the following PRINT statement will invoke LOGIN.

```
10 PRINT #1%, RECORD 1%, "HELLO";CHR$(13%);
```

RSTS/E will generate a <CR> and <LF> for every record sent to the pseudo keyboard. Therefore, the PRINT statement MUST have the semicolon following it if a CHR$(13%) is appended to the statement. A PUT statement would therefore require the CHR$(13%).

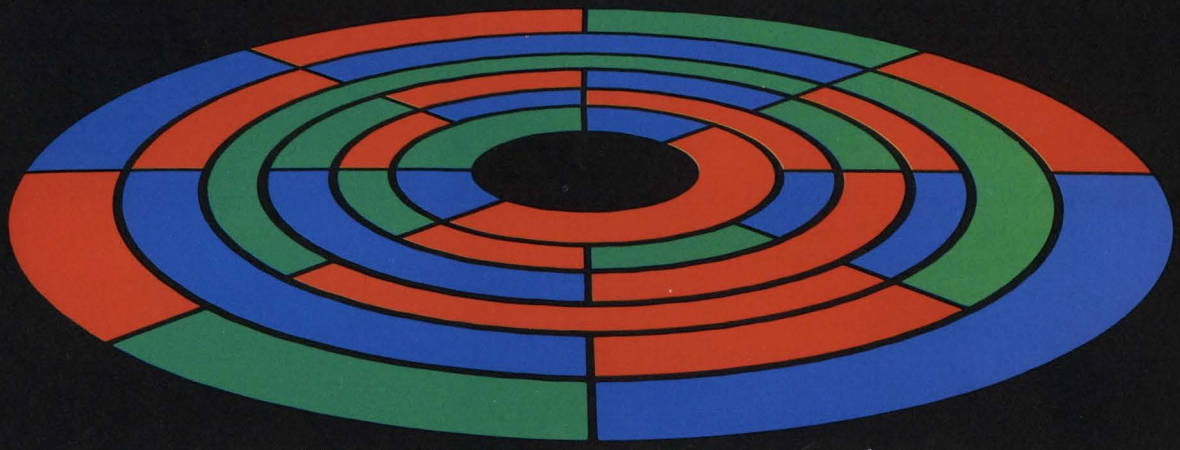
A GET statement, discussed later, is used to retrieve the "#" from LOGIN. Then the account number and the password are transmitted to the pseudo keyboard. The entire login procedure can be condensed into one statement:

```
10 PRINT #1%, RECORD 1%, "HELLO 100,100;DEMO";CHR$(13%)
```

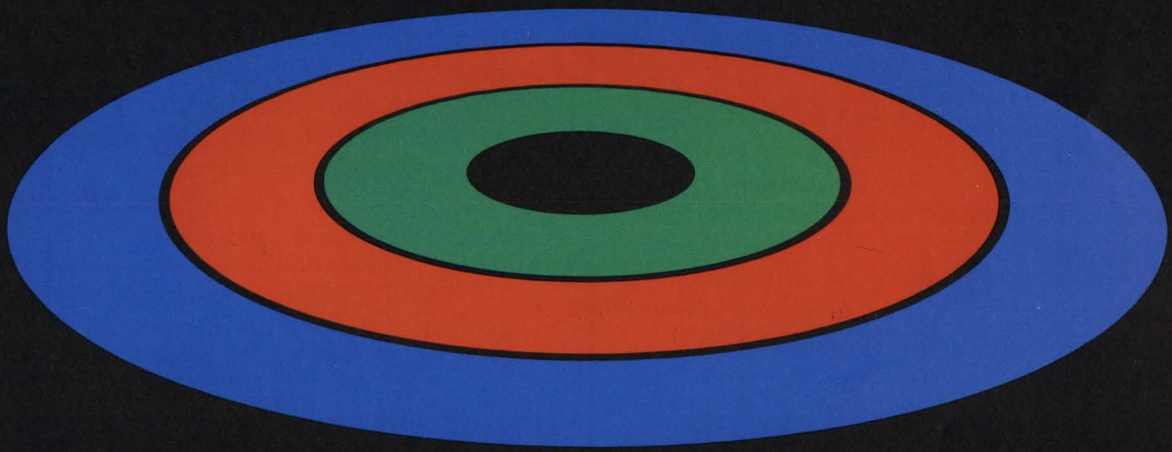
SENDING COMMANDS TO THE CONTROLLED JOB

The PRINT and PUT statements are used to send com-

WHAT YOU DON'T KNOW ABOUT YOUR DISKS IS COSTING YOU MONEY



If your disk looks like this, you're wasting system performance.



If your disk looks like this, you're using DISKIT.

When the job you're running requires reading the "red" file, it naturally happens faster on a well-ordered disk. Disks become "fragmented" as you use your computer. The system slows down. And that costs you money.

Now, you can restructure your disks and get back that lost performance (up to 50%) without spending a dime on new hardware. DISKIT is the original software system that makes this possible.

But don't confuse DISKIT with other system utilities, DISKIT is a complete "software tool kit" that optimizes your RSTS/E system.

DISKIT is:

- DSU — The utility which restructures the information on your disk, making data fast and easy to access.
- DIR — The incredible directory tool that finds files at the rate of 400 per second.
- RDR — Reorders disk directories 30 times faster than ever before possible.
- OPEN — Displays complete job statistics and file activity so you can see what your system is doing.
- DUS — The set of CALLable subroutines which pre-extend file directories, reducing fragmentation.

In today's tight economy, it's more important than ever to get the most out of your hardware investment. Call or write today and start getting your money's worth from your computer.

Software Techniques Incorporated

5242 Katella Avenue
Los Alamitos, CA 90720
United States
Phone: (714) 995-0533

287 London Road
Newbury, Berkshire RG13 2QJ
United Kingdom
Phone: 44 (0) 635-30840

CIRCLE 65 ON READER CARD

mands to the controlled job. The controlled job must be in ↑C or KB state for the command to be sent. The controlling job can force a command to the terminal even if the job is not in ↑C or KB state. The RECORD clause in the PRINT and PUT statements informs RSTS/E how to respond to the command. The values and meanings of the RECORD clause are:

Value	Meaning
1	The system does not check the job status before sending data to the pseudo keyboard.
2	The system tests to determine whether the pseudo keyboard is at ↑C or KB state. If not then ERR = 3, ?DEVICE IN USE, is generated.
4	The system does not send data to the pseudo keyboard but instead errs if the pseudo terminal is not ready to accept data. If no error occurs then control proceeds to the next statement in the controlling job's program.
8	The system waits for small buffers to become available for input if this bit is set. An error is generated if there are no small buffers available.

RECORD 6% option will ensure that the controlled job is at command level. If the controlled job is at KB state the error, ?PROGRAMMABLE ↑C TRAP (ERR = 28), is generated. A ↑C can be forced to the controlled job and then command level would be reached.

RECORD 16% option allows the controlling job to kill the controlled job. Note that when a logical CLOSE or END is encountered the controlled job will also be killed.

The following is an example of how a command would be sent to the pseudo keyboard:

```
1020 ON ERROR GOTO 32000
1030 PRINT #1%,RECORD 4%
1040 PRINT #1%,"SY/A-";CHR$(13%);
1050 GOTO 100
32000 IF ERL = 1030 THEN PRINT "?Pseudo keyboard not at";
            " ↑C or KB status"
\            RESUME 100
```

RECEIVING INFORMATION FROM THE CONTROLLED JOB

Information from the controlled job is obtained by using the GET statement in combination with the FIELD statement. The channel number for the GET and FIELD statement is the channel number that the pseudo keyboard is OPENed on.

```
GET #1%
```

This statement will retrieve data from the pseudo keyboard on channel one. The system will not stall the controlling program if there is not any data waiting to be returned. Instead the error, ?END OF FILE ON DEVICE (ERR = 11), is produced. If the controlled job produces output faster than the controlling job accepts it, then the controlled job goes into a TT (output wait) state.

The FIELD statement performed after a GET statement would be as follows:

```
FIELD #1%,RECOUNT AS A$
```

The output produced by the controlled job is stored in the variable A$. It must be remembered that A$ contains the <CR> and <LF> characters. Also A$ may not be full lines of output. Therefore when the output from the controlled job is displayed the carriage return must be omitted. For example:

```
PRINT A$
```

This would print the output from the controlled job correctly.

PSEUDO.BAS — THE RSTS/E BACKGROUND PROGRAM

Now that the method to utilize pseudo keyboards is understood, the RSTS/E background environment can be established. The program PSEUDO.BAS will perform the necessary tasks to create the RSTS/E background. PSEUDO.BAS is a combination of ATPK, BATCH, and ATPRO. PSEUDO.BAS understands command/control files from each of these processors.

To achieve a true background processor, PSEUDO.BAS must be able to operate in detached mode. PSEUDO.BAS can indeed operate either while connected to a terminal or detached. PSEUDO.BAS is even smart enough to log the user back in at his terminal after it has detached. PSEUDO.BAS will also kill itself when it is finished with the background processing. This ensures that there are no detached hibernating jobs left on the system.

PSEUDO.BAS has two prompts, the first being for the command file name. An extension must be given; PSEUDO.BAS has no extension defaults. The command file can be a CMD or CTL or any other type of file. Again for the true background appearance, PSEUDO.BAS will allow the user to enter the commands at the terminal which are to be processed in background. To do this enter "*" to the command file question.

Two switches exist to this question. If the command file name is followed by a /DET, PSEUDO.BAS will detach. The user can then do whatever is desired. The other switch is /RET. With this switch, PSEUDO.BAS will log the user back in. Not using a switch indicates that PSEUDO.BAS is to run at the terminal. By not using /DET or /RET, PSEUDO.BAS will resemble ATPK.

The second prompt deals with the log file that PSEUDO.BAS will print the processing information and errors to. If a carriage return is entered in response to this question, the log file is the KB:. If KB: is specified then the job can not detach. Any disk file or device can be entered to this question. One switch exists to this prompt, and that is /TIME:xxx. A default of 120 minutes has been developed for the maximum amount of processing time allowed to the background job. The /TIME switch will override that number. The maximum amount of time to give the job is entered in minutes.

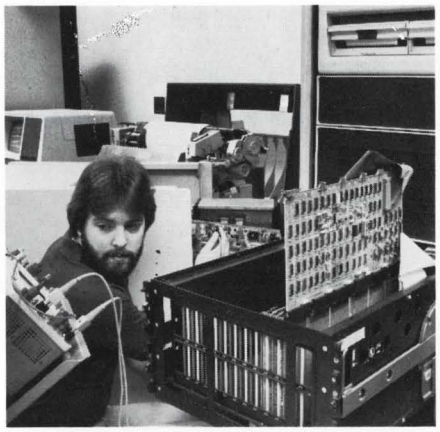
If PSEUDO.BAS is detaching, it will inform the user of this status. If PSEUDO.BAS is logging the user back in, then the user will be informed about this process.

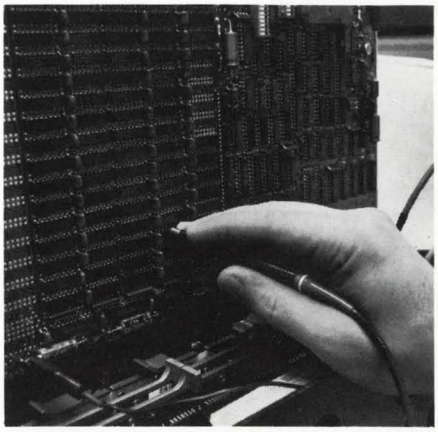
```
001     EXTEND
\       ON ERROR GOTO 32000
\       X$ = SYS(CHR$(6%)+CHR$(-7%))
\       OPEN "KB:" AS FILE #12%
\       RUN.TIME = 120
\       TIME.START = INT(TIME(0%)/60)           !EXTEND MODE, SET ^C TRAPPING,
                                                ! OPEN KB: ON CHANNEL 12,
                                                ! INIT DEFAULT AMOUNT OF RUN
                                                ! TIME, GET TIME STARTED


005 !*****************************************************
\   !*
\   !* THIS PROGRAM WAS WRITTEN BY M H KOPLITZ,
\   !*
\   !* THIS PROGRAM IS A COMBINATION OF BATCH/ATPRO/ATPK
\   !*  PROCESSORS. THIS PROGRAM WILL DO COMMAND FILE
\   !*  PROCESSING THE WAY ATPRO DOES, NO SPECIAL
\   !*  COMMANDS NEEDED. IT IS LIKE ATPK BECAUSE IT DOES
\   !*  THE WORK ON A PSEUDO KEYBOARD. IT IS LIKE BATCH
\   !*  BECAUSE THE USER CAN DETACH THE CONTROLLING JOB,
\   !*  ALSO THE USER CAN BE RE-LOGGED IN IF REQUESTED.
\   !*
\   !* THE USER MAY ALSO CREATE THE COMMAND FILE TO
\   !*  PROCESS WHEN PSEUDO IS INVOKED, NO NEED TO CREATE
\   !*  THE COMMAND FILE ANY MORE.
\   !*
\   !*****************************************************************
\   !*
\   !* RUNNING PSEUDO:
\   !*
\   !* RUN PSEUDO
\   !* PSUEDO V1.0      Command file processor
\   !*
\   !* Command file name?
\   !*
\   !*          To this input the command filename with extension is given.
\   !*          A .CTL file from batch may be used as the command file.
\   !*          If a * is entered the commands to process will be inquired
\   !*            for from the KB:.
\   !*          Two switches exist: DET - causes pseudo to detach.
\   !*                              RET - causes pseudo to detach then
\   !*                                    re-log the user back in.
\   !*
\   !* Log file name?
\   !*
\   !*          The default to this question is KB:. If KB: is used then
\   !*            PSEUDO can not detach.
\   !*          Extension must be given.
\   !*          One switch: TIME:xxx, where xxx is the time limit of the
\   !*             process (in minutes), 120 minutes default.
\   !*
\   !*****************************************************************


010     PRINT "PSEUDO  V1.0    ";
                "Command file processor"
\       PRINT
\       PRINT "Command file name";
\       INPUT LINE COMMAND.FILE$
\       COMMAND.FILE$ = CVT$$(COMMAND.FILE$,-1%)
\       GOTO 32767 IF COMMAND.FILE$ = ""
\       SWITCH% = INSTR(1%,COMMAND.FILE$,"/")
\       GOTO 015 IF SWITCH% = 0%
\       SWITCH$ = RIGHT(COMMAND.FILE$,SWITCH%+1%)
\       COMMAND.FILE$
                = LEFT(COMMAND.FILE$,SWITCH%-1%)
\       GOTO 015
                IF SWITCH$ = "DET"
                OR SWITCH$ = "RET"
\       PRINT "?Illegal switch"
\       GOTO 32767                              !PRINT THE BANNER, THEN GET
                                                ! THE COMMAND FILE NAME, CHECK
                                                ! FOR A DET OR RET SWITCH.


015     PRINT
\       PRINT "Log file name";
\       INPUT LINE LOG.FILE.NAME$
\       LOG.FILE.NAME$
                = CVT$$(LOG.FILE.NAME$,4%)
\       SWITCH1% = INSTR(1%,LOG.FILE.NAME$,"/")
\       GOTO 020 IF SWITCH1% = 0%
\       SWITCH1$
                = RIGHT(LOG.FILE.NAME$
                        ,SWITCH1% + 1%)
\       LOG.FILE.NAME$
                = LEFT(LOG.FILE.NAME$
                        ,SWITCH1%-1%)
\       GOTO 017 IF LEFT(SWITCH1$,4%) = "TIME"
\       PRINT "?Illegal switch"
\       GOTO 32767                              !GET THE LOG FILE NAME AND
                                                ! LOOK FOR A SWITCH


017     T% = INSTR(1%,SWITCH1$,":")
\       RUN.TIME = VAL(RIGHT(SWITCH1$,T%+1%))   !GET THE NEW RUN TIME LIMIT


020     LOG.FILE.NAME$ = "KB:"
                IF LOG.FILE.NAME$ = ""          !IF NO LOG FILE NAME THEN
                                                ! IT IS KB:


025     GOTO 050 IF COMMAND.FILE$ <> "*"
\       COMMAND.FILE$ = NUM1$(TIME(0%))+".TMP"
\       OPEN COMMAND.FILE$ FOR OUTPUT AS FILE #2%
\       PRINT
\       PRINT "Enter commands to process, ";
                "end with ^Z"
\       PRINT                                   !IF COMMAND FILE = * THEN
                                                ! USER TO INPUT COMMANDS


030     INPUT LINE #12%,INPUT.LINE$
\       INPUT.LINE$ = CVT$$(INPUT.LINE$,4%)
\       PRINT #2%,INPUT.LINE$
\       GOTO 030                                !INPUT THE VALUES


050     OPEN COMMAND.FILE$ FOR INPUT AS FILE #2%
\       OPEN LOG.FILE.NAME$
                FOR OUTPUT AS FILE #3%
\       PRINT #3%,STRING$(70%,42%)
\       PRINT #3%
\       PRINT #3%,"PSEUDO command file ";
                  "processor starting at: ";
                  DATE$(0%);"  ";TIME$(0%)
\       PRINT #3%,"Processing file ";
                  COMMAND.FILE$
\       PRINT #3%
\       PRINT #3%,STRING$(70%,42%)
\       X% = -1%                                !OPEN THE COMMAND FILE AND LOG
                                                ! FILE, PRINT FIRST BANNERS.


060     GOTO 070 IF SWITCH$ <> "DET"
\       GOTO 100 IF LOG.FILE.NAME$ = "KB:"
\       PRINT "%PSEUDO will be detaching in";
                " job slot";PEEK(518)/2
\       CLOSE #12%
\       DETACHING% = 1%
\       X$ = SYS(CHR$(6%)+CHR$(7%))             !IF DET SWITCH THEN DETACH


070     GOTO 100 IF SWITCH$ <> "RET"
\       CLOSE #12%
\       PRINT "%PSEUDO will be detaching in ";
                "job slot";PEEK(518)/2
\       PRINT
\       PRINT "%PSEUDO will re-log in ";
                "within one minute"
\       PRINT
\       X$ = SYS(CHR$(6%)+CHR$(7%))
\       DETACHING% = 1%
\       JDB% = PEEK(520%)
\       IOB% = PEEK(JDB%)
\       DDB% = PEEK(IOB%)
\       TERMINAL.NUMBER%
                = SWAP%(PEEK(DDB%+2%)) AND 127%
\       RETURNING% = 1%                         !IF RET SWITCH THEN GET
                                                ! TERMINAL NUMBER AND DETACH


100     X% = X% + 1%
\       PRINT #3%,"?No pseudo keyboards ";
                "available"
                IF X% > 8%
```

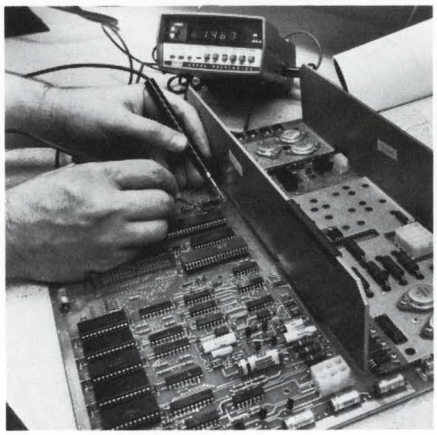
```
\       GOTO 32767
                IF X% = 8%
\       OPEN "PK"+NUM1$(X%)+":" AS FILE #1%      !TRY TO GET A PSEUDO KEYBOARD

105 !*******************************************
\   !*
\   !* ROUTINE ABOVE IS SET FOR 8 PSEUDO KEYBOARDS
\   !*  ADJUST THE EIGHT TO THE NUMBER OF PK:
\   !*  ON THE SYSTEM
\   !*
\   !*******************************************

150     INPUT LINE #2%,COMMAND.LINE$
\       COMMAND.LINE$ = CVT$$(COMMAND.LINE$,4%)
\       GOTO 160
                IF LEFT(COMMAND.LINE$,4%)
                        = "$JOB"
\       GOTO 160
                IF LEFT(COMMAND.LINE$,3%)
                        = "LOG"
\       NO.LOG% = 1%                             !READ IN FIRST COMMAND, IF LOG
                                                 ! OR $JOB THE INDICATE THAT
                                                 ! FIRST COMMAND WAS PROCESSED.
                                                 ! A JOB/LOG LINE IS NOT NEEDED


160     A% = PEEK(PEEK(PEEK(520%)+8%)+24%)
\       PROJ% = SWAP%(A%) AND 255%
\       PROG% = A% AND 255%
\       X$ = SYS(CHR$(6%)+CHR$(14%)
                +CHR$(0%)+CHR$(0%)
                +CHR$(0%)+CHR$(0%)
                +CHR$(PROG%)+CHR$(PROJ%))        !PROJECT/PROGRAMMER/PASSWORD

170     PASSWORD$ = MID(X$,9%,4%)
\       PASSWD$ = RAD$(SWAP%(CVT$%
                  (LEFT(PASSWORD$,2%))))
                + RAD$(SWAP%(CVT$%
                  (RIGHT(PASSWORD$,3%))))
\       GOTO 175 IF RETURNING% = 0%
\       X$ = SYS(CHR$(6%)+CHR$(-4%)
                +CHR$(TERMINAL.NUMBER%)
                + "HELLO "
                + NUM1$(PROJ%)
                + ","
                + NUM1$(PROG%)
                + CHR$(13%))
\       SLEEP 3%
\       X$ = SYS(CHR$(6%)+CHR$(-4%)
                + CHR$(TERMINAL.NUMBER%)
                + PASSWD$+CHR$(13%))
\       X$ = SYS(CHR$(6%)+CHR$(-4%)
                + CHR$(TERMINAL.NUMBER%)
                + CHR$(13%))                     !IF RET SWITCH USED THEN RELOG
                                                 ! USER IN.

175     PRINT #1%, RECORD 1%,"HELLO ";
             PROJ%;",";PROG%;CHR$(13%);          !LOG IN ON PSEUDO KEYBOARD

180     SLEEP 2%
\       GET #1%
\       FIELD #1%, RECOUNT AS RETURNED.LINE$
\       PRINT #3%,RETURNED.LINE$;                !WAIT FOR MESSAGE BACK

190     PRINT #1%, RECORD 1%,PASSWD$;CHR$(13%); !SEND OUT PASSWORD

200     SLEEP 1%
\       GET #1%
\       FIELD #1%, RECOUNT AS RETURNED.LINE$
\       PRINT #3%,RETURNED.LINE$;                !GET INFO FROM PSEUDO KEYBOARD

210     PRINT #1%, RECORD 1%,CHR$(13%);          !SEND RETURN IN CASE OF DETACH
                                                 ! QUESTION IN LOGIN

250     SLEEP 1%
\       GOSUB 1000
\       GET #1%
\       FIELD #1%, RECOUNT AS RETURNED.LINE$     !GET INFO FROM PSEUDO KEYBOARD


260     PRINT #3%,RETURNED.LINE$;
\       GOTO 250                                 !PRINT LINE FROM PSEUDO
                                                 ! KEYBOARD TO LOG FILE

300     PRINT #1%, RECORD 4%                     !IS PSEUDO KEYBOARD AT ^C
                                                 ! STATE?

305     GOTO 310 IF NO.LOG% = 0%
\       MESSAGE.BACK$ = COMMAND.LINE$
\       NO.LOG% = 0%
\       GOTO 320                                 !IF NO LOG/JOB LINE THEN
                                                 ! THIS AREA IS VERY IMPORTANT

310     INPUT LINE #2%,MESSAGE.BACK$
\       MESSAGE.BACK$ = CVT$$(MESSAGE.BACK$,4%)
\       GOTO 350 IF LEFT(MESSAGE.BACK$,3%)
                = "BYE"
\       GOTO 350 IF LEFT(MESSAGE.BACK$,4%)
                = "$EOJ"
\       GOTO 400 IF LEFT(MESSAGE.BACK$,4%)
                = "$EOD"
\       GOTO 450 IF LEFT(MESSAGE.BACK$,5%)
                = "$DATA"
\       GOTO 400 IF LEFT(MESSAGE.BACK$,2%)
                = "^Z"                           !INPUT A LINE FROM THE COMMAND
                                                 ! INPUT FILE AND SEE IF IT IS
                                                 ! SPECIAL

320     PRINT #1%, MESSAGE.BACK$;CHR$(13%);
\       GOTO 250                                 !SHIP OUT COMMAND TO PSEUDO
                                                 ! KEYBOARD

350     PRINT #1%,"BYE F";CHR$(13%);
\       SLEEP 2%                                 !SIGN OFF PSEUDO KEYBOARD
                                                 ! JOB

360     GET #1%
\       FIELD #1%,RECOUNT AS DATA.RETURNED$
\       PRINT #3%,DATA.RETURNED$
\       GOTO 360                                 !GET INFO FROM PSEUDO KEYBOARD

400     PRINT #1%,CHR$(26%);
\       GOTO 250                                 !SEND A ^Z TO PSEUDO KEYBOARD

450     GOTO 310                                 !IGNORE THE $DATA LINE

1000 !*****************************************************
\    !*
\    !* THIS SECTION CHECKS THE LENGTH OF RUN-TIME
\    !*
\    !*****************************************************

1010    AMOUNT.RUN.TIME = INT(TIME(0%)/60%)
                        - TIME.START
\       AMOUNT.RUN.TIME = INT(TIME(0%)/60%)
                        + 1439 - TIME.START
                IF AMOUNT.RUN.TIME < 0
\       GOTO 1100
                  IF AMOUNT.RUN.TIME <= RUN.TIME
\       PRINT #3%
\       PRINT #3%,"***TIME LIMIT EXCEEDED"
\       PRINT #1%,RECORD 1,CHR$(3%);
\       GOTO 350                                 !CHECK TIME LIMITS

1100    RETURN                                   !END OF FUNCTION

32000 !*************************************************
\     !*
\     !* ERROR CONTROL
\     !*
\     !*************************************************

32005   IF ERL = 250
                THEN RESUME 300

32010   IF ERL = 300 THEN RESUME 250

32020   IF ERL = 100 THEN RESUME 100

32030   IF ERL = 180 THEN RESUME 180

32040   IF ERL = 220 THEN RESUME 220

32050   IF ERL = 360 THEN  32700

32060   IF ERL = 310 THEN RESUME 360

32070   IF ERR = 28 AND ERL > 210
                THEN RESUME 350

32080   IF ERL = 030 THEN CLOSE #2%
\               RESUME 050

32090   IF ERL = 010 THEN RESUME 32767

32100   IF ERL = 050 AND ERR = 5
                THEN PRINT "?Can not find file or account"
\               RESUME 32767

32110   IF ERR = 28 THEN RESUME 32767
32120   IF ERL = 050
                THEN PRINT "?Device not available"
\               RESUME 32767

32130   IF ERL = 320
                THEN SLEEP 5%
\               RESUME 320

32190   PRINT #3%
\       PRINT #3%,"Error = ";ERR;" on line ";ERL
32200   GOTO 32700


32700   PRINT #3% FOR X% = 1% TO 3%
\       PRINT #3%,STRING$(70%,42%)
\       PRINT #3%
\       PRINT #3%,"PSEUDO command file ";
                "processing complete ";
                DATE$(0%);"  ";TIME$(0%)
\       PRINT #3%
\       PRINT #3%,STRING$(70%,42%)
\       CLOSE #X% FOR X% = 1% TO 12%
\       GOTO 32767 IF DETACHING% = 0%
\       X$ = SYS(CHR$(6%)+CHR$(8%)
                + CHR$(PEEK(518)/2)
                + STRING$(23%,0%)
                + CHR$(0%)
                + CHR$(255%))                    !END THE PROCESSING

32767   END
```

The Single Source for Digital Products Service

DEC Module Repair, Exchange or Replacement







Photos show how we're fully equipped to repair and test DEC,* PDP-11* and TERMINAL MODULES. All modules we repair are 100% inspected.

Visit us at Booth No. 505

DEXPO™ East 83
The 3rd National DEC-Compatible Industry Exposition
Kiel Auditorium
St. Louis, Missouri
May 22-24, 1983

DIGITAL PRODUCTS REPAIR CENTER
939 Eastwind Drive
Westerville, Ohio 43081
614/890-0939

Expert, Fast and Reliable Work

Serving the computer systems market since 1973, we are now troubleshooting, repairing and testing hundreds of modules weekly.

We can turn around a critical repair job in just 24 hours! Ten-day service is normal.

The modules we return to you are guaranteed to meet or exceed the manufacturer's specifications. And we prove it by submitting an inspection and test report.

For emergency service, 7 days a week, 24 hours a day, call

614/890-0939

*DEC and PDP are registered trademarks of Digital Equipment Corporation.

H-1580

CIRCLE 140 ON READER CARD

RELIANCE ELECTRIC

THE RSTS CRYSTAL BALL

FMS V1.5 for RSTS/E

By Michael C. Greenspon, Integral Information Systems, Los Angeles, CA

Greetings. This issue of the Crystal Ball features an overview of FMS V1.5 for RSTS, some undocumented FMS features, internals information, bugs, and hacks. Also included are some undocumented monitor features in RSTS V7.1 and later.

FMS

DEC's Forms Management System (FMS) serves as a terminal I/O front-end for application programs. FMS consists of three major components: the forms editor (FED), the forms utility and librarian (FUT), and the forms driver (FDV).

FED allows a programmer to interactively design the forms that the application will display and use to retrieve input. FED uses an EDT-like keypad layout to edit and assign attributes to a form. Since the programmer sees exactly what will be displayed, there is no guesswork with a form description language (such as INDENT's) and no need to use a screen layout form. This saves an extraordinary amount of programmer time (and frustration!).

FUT functions mainly as a librarian, allowing the programmer to concatenate and librarify a group of form files into one form library. FUT also prints hardcopy descriptions of a form file, including the actual screen image and attributes.

The forms driver (FDV) contains the code which implements the FMS calls and performs the actual terminal I/O. A portion of the forms driver is implemented in the RSTS monitor (the FMS option must be selected during SYSGEN), but the remainder must be linked with the application program, optionally as a resident library. The forms driver occupies a little under 4KW of code space.

FMS V1.5 ON RSTS

I recently completed a complex mailing list system for RSTS, using RMS and FMS from MACRO-11. Without FMS, the entire project would easily have taken 50% longer to write and debug. I was surprised and quite pleased to find that FMS is efficient, very flexible, and well documented. My complements, DEC, on probably the best layered product since DTR.

Contrary to popular belief, FMS on RSTS is a relatively light system load. A number of programmers that I've talked to have heard horror stories of FMS V1.0 on RSX, and have falsely assumed that FMS on RSTS is just as piggy. The forms driver for FMS V1.0 (currently on VMS and RSX) uses single character input (the equivalent of the RSTS "one-shot" ODT option) and is contained entirely within the user task.

The user task is scheduled and run every time the operator hits a key. The RSTS implementation is newer, and quite different (hence the version number change) in that the low-level portion of the forms driver is part of the RSTS monitor. This reduces the input overhead for FMS to almost nil. It also means that the user task isn't executed until there is actually data to be processed, so it can be swapped out if needed.

FMS ENHANCEMENTS

The next release of FMS (version 2.0) is scheduled for late spring or early summer of 1983, and will include monitor support for RSX and VMS similar to RSTS's. Also, FMS V2.0 will feature a number of major enhancements, such as user-defined field validation routines and VT100 graphics support. Unfortunately, I needed this functionality six months ago. With a little hacking, kludgery, and clever use of the FMS calls, I persuaded FMS to perform as required. As far as I know, the following kludges only apply to RSTS FMS V1.5, since FMS V1.0 may use incompatible form definitions.

Problem:

FMS V1.5 does not support the VT100 line drawing character set, making it difficult or impossible to display effective boxes, borders, lines, etc.

Solution:

After a little experimentation, I determined that the forms driver (FDV) doesn't care about what is in the [displayed background] text portion of the form. I ODT'd a form file, and after a few minutes, the format of most of the file header and the text section became obvious. I manually patched in the escape sequence to set up the graphics character set, and the appropriate SIs and SOs to shift in and out of the graphics set. Miraculously, it worked the first time.

After patching a form file with ODT, it became impossible to edit the form image using FED, since I had inserted non-printing characters in the text section. FED assumes that the characters it outputs from the text section will print, and move the cursor appropriately. The non-printing

SIs and SOs completely threw off FED's cursor positioning and movement, and caused the image to be displayed incorrectly, even though the forms driver showed the form perfectly.

After doctoring a couple of forms, I was getting pretty tired of using ODT. Also, if I wanted to make any changes, I had to re-patch the file, since I couldn't FEDit the munged file. After some lossage due to incorrectly hacking a form definition, I decided it was time to invent a better way to modify the form files.

The VT100 line drawing set is mapped into some of the lower case characters, such as "q", "x", etc. The lower case characters can, of course, be typed in with FED. All I had to do was write a program to hack the form file, and provide some way for the program to distinguish between normal and graphics mode characters. The easiest way to mark text on an FMS form is with VT100 video attributes. I chose to indicate graphics mode characters by assigning them all four AVO attributes (bold, underscore, blink, and reverse) — a combination which more obscures the text than highlights it, and would never be used. With a magical program to do my forms hacking, I could type up the form with FED, assign all four video attributes to graphics text, and let the program chomp away.

The program, FI.B2S, (I'm not sure why I called it that — I think I typed "I" instead of "U" originally and never bothered changing it . . .) is listed at the end of the article.

Please note that the resulting forms (with the graphics sequences) cannot be FEDited. I recommend keeping two copies (or two libraries) of forms that use graphics — one with the graphics text highlighted and FEDitable, the other FI'd and displayable. Also, FUT may report strange error messages when trying to print descriptions of hacked forms. You should print your form descriptions from the normal, FEDitable forms. The forms driver has no problems, however, with FI'd forms.

Problem:

My mailing list application replaced an existing system, which did its terminal input using echo control. The operators were accustomed to terminating fields with the RETURN or ENTER keys, not TAB. Unfortunately, FMS uses ENTER to terminate the entire form; i.e., "Ok, I am done with EVERYTHING on the screen" instead of just the current field. This would have caused many headaches for the operators, not to mention the poor old programmer who had to debug the thing. (When was the last time YOU hit the TAB key to terminate a command line? What a crock . . .)

Solution:

This seemed like a trivial hack — just go in and patch a couple of bytes in the forms driver. Wrong. The values were in the monitor's portion of the forms driver, and I didn't have sources to that. Just as I was about to disassemble the interesting stuff, I realized that I could translate the field terminators without any patching at all! This will require a little explaining, so

first, some background information . . .

FMS has two basic calls to retrieve input from the terminal: GETALL and GET. (GETAF is not significantly different from GET for our purposes.) GETALL allows the operator to move the cursor anywhere on the form, fill in any or all of the needed information, and terminate the form. Ideally, executing a GETALL is similar to reading a record from a file — one call, and *boom!* — all of the fields are filled and returned in order. I have not yet found an (ideal . . .) application for which GETALL is suitable, because it does not allow for individual field validation, which is essential in almost all applications.

The GET call retrieves data for a single field. Additionally, FMS supplies a numeric code for the field terminator. The application program can process and validate the data in the field, and then execute a PFT (process field terminator) call to advance or backup to the appropriate field. PFT takes as an argument the terminator code to process.

My simple hack is to merely translate the terminator codes behind FMS's back. For example, the program executes a GET call, and the operator responds and terminates the field with the ENTER key. FMS returns the data, and the terminator code FT$NTR, which means "done with form." The program processes the returned data, translates the terminator to FT$NXT (advance to next field), and executes a PFT call. FMS advances its context to the next field on the form. Without patching, ENTER has successfully been translated to TAB. Unfortunately, this does create one minor problem . . .

Problem:

If ENTER is translated to TAB (next field), there is no terminator to indicate "end of form."

Problem:

FMS uses the VT100 arrow keys and PF keys for editing and control functions, and does not process control characters as field terminators. Application-defined function keys are, therefore, scarce. If the VT100 is in application keypad mode, FMS will process the keypad keys as terminators, however the operator will not be able to use the keypad keys for numeric input.

Solution:

For my application, only three function keys were needed: end of form, abort form, and a form-dependent key, usually used to jump around on the screen. The FMS release notes describe an FMS build option to enable four keys as FMS terminators. (The keys are PF4, Gold-PF4, Gold-Rightarrow, and Gold-Leftarrow.) The option can be enabled by setting the global symbol FT$USR non-zero at taskbuild time, or, if you are using the FMS resident library, when the library is built. (More on this in a moment.)

I used PF4 for end of form, Gold-PF4 for abort form (with a confirm), and Gold-Rightarrow as the form dependent key. The same routine that translates FT$NTR to FT$NXT also translates the code for PF4 to FT$NTR, and the other keys to my own internal terminator codes.

For applications which require more than four function keys, but also need the numeric keypad, there are a number of solutions. The most elegant that I could think of involves the use of a second prefix key, and a little cheating. (Note: this should work in theory, but, since I have had no use for it, I have not tested it.)

First, build the program or the FMS reslib with FT$USR set non-zero (see below), so that the unused arrows and PF keys are available. Define the PF4 key as an alternate "Gold" key, let's say, "Silver." Normally, the VT100 should be in numeric keypad mode. When PF4 is pressed, FMS will return control to your program, which should enable application keypad mode on the terminal, and then re-get the current field. If the operator hits a keypad key, FMS will again return to your program, with the numeric code for the terminator. The program should turn off application keypad mode, and perform the special function associated with the Silver-foo key. This gives the program access to 13 function keys. By defining more prefix keys, (Gold-PF4, Gold-Rightarrow, etc.) you can create up to 52 function keys.

The program must "cheat" in order to switch to application keypad mode and back. Once the program has "attached" to the terminal, (in the FMS sense, not the RSTS sense) any normal RSTS terminal input will effectively "detach" the terminal. (This follows the same rules as other special terminal modes.) However, FMS mode is a special input mode only — outputting to a terminal open in FMS mode will not cancel FMS mode, provided the output is done on the correct channel (i.e., not channel 0). Therefore, you can output the needed escape sequences directly.

Problem:

The procedure for building a task with additional function key support (FT$USR set non-zero) is described in the FMS release notes. However, if the FDV resident library is built with this procedure, the additional function keys are not enabled.

Solution:

The FDV resident library has a null root segment, so that it can be clustered. The GBLDEF taskbuilder option defines the symbol FT$USR to be non-zero in the root segment, but not in the overlay segment where the actual code is.

One possible solution is to edit the ODL file and discard the null overlay root, but this will make the library unclusterable. Since the code only references the symbol in one place, I have chosen to patch the library. The following patch will enable additional function key support for the FMS reslib:

For the FDV reslib without debug support:

```
File to patch? FDVRES.LIB
Base address? $FDV
Offset address? 7012
 Base   Offset  Old     New?
??????  007012  000000  ? 1     ; non-zero to enable
??????  007014  001402  ? ^C    ; Uparrow C to exit
```

For the FDV reslib with debug support:

```
File to patch? FDV RDB.LIB
Base address? $FDV
Offset address? 7224
 Base   Offset  Old     New?
??????  007224  000000  ? 1     ; non-zero to enable
??????  007226  001402  ? ^C    ; Uparrow C to exit
```

Problem:

Since the GETALL call does not allow for individual field validation, it is fairly useless. Executing individual GETs and PFTs to retrieve a form quickly becomes cumbersome in an application with many forms or fields.

Solution:

Version 2.0 of FMS will provide user-defined field validation routines, so that GETALL can be used effectively. It is a simple task to write a subroutine to simulate this under V1.5. The subroutine I used in my application is called GETFRM, and is listed at the end of the article, along with some support macros. If you are using FMS from a high-level language, you'll have to write your own subroutine. Since "high-level" languages are highly restrictive, highly piggy, etc., you might have some problems dispatching to the validation routines (i.e., variable subroutine calls). (Moral: write it in MACRO next time.)

As suggested in the FMS documentation, it is possible to use named data to describe the required field validation. For my purposes, this would have been overkill, and less efficient (of space and time). This method is, however, particularly effective, especially for complex applications or high-level languages.

I wasn't satisfied with the bagbiting macros that DEC supplied, so I hacked up some of my own (CALFMS, DOFMS, etc.). GETFRM uses these for its FMS calls. Unfortunately, they are rather long and can't be printed here, but they are available on the monthly goodies tape.

FMS V1.5 BUGS

I have only found one serious FMS bug so far: overlaid forms do not clear the correct area on the screen. Instead of erasing the specified lines, FMS erases the top line several times. I had been using a kludge to get around the problem for some time, but a patch was recently made available to me. Special thanks to Mark Hartman for passing it on to me, and to the DEC FMS people, for their lightning-fast response to Mark's SPR. The patch to the FMS phase of the monitor is listed below. Mark said that the patch is for V7.2 only, however it will probably work for V7.1. (I have not tested it — the offset address from $FMSSE may be different, and a different portion of patch space may need to be used.)

Monitor patch to fix FMS overlaid forms:

```
File to patch? <LF>
Module name? FMS
Base address? $FMSSE
Offset address? 4230
Base    Offset  Old     New?
??????  004230  ??????  ? FMSPAT+60
??????  004232  ??????  ? ^Z          ; Control/Z for new base...
Offset address? ^Z
Base address? FMSPAT
Offset address? 60
Base    Offset  Old     New?
??????  000060  000000  ? 33633       ; This all looks like data...
??????  000062  000000  ? 55633
??????  000064  000000  ? 115562
??????  000066  000000  ? 115470
??????  000070  000000  ? 32443
??????  000072  000000  ? 55633
??????  000074  000000  ? 115513
??????  000076  000000  ? 41133
??????  000100  000000  ? ^C          ; Uparrow C to exit
```

RANDOM

As usual, I would appreciate any feedback you have on FMS, RSTS, or whatever. Although FMS has some minor problems and bugs, I found it to be very well suited to my application, and many other screen-based interactive systems. It is relatively low CPU overhead, not too piggy on space, and can save an immense amount of programmer time. FMS is highly flexible, and, with a little imagination, can help make an applications package consistent and good looking. FMS V2.0 will offer even more functionality and flexibility than the current release, and should be seriously considered for any new interactive applications. On a scale of 1 to 10, 10 being best, I give FMS a 9.2.

I'd like to thank the FMS people for the FMS magic session at the Anaheim DECUS. Although there were only about 150 people present (hidden in a small room behind the exhibit hall), it was one of the most lively and informative sessions of the week. Also, thanks for the FDV sources — they came in handy.

HIDDEN UU.MNT BITS

There are a number of undocumented mode bits for the

UU.MNT call (mount sub-function) under RSTS V7.1 and later. For more information on the UU.MNT call, consult the System Directives manual or the RSTS/E Programming Manual. The recognized mode bits are as follows: (undocumented bits are indicated with an asterisk)

Bit	Octal	Meaning if set
* 6	100	Write check all FIP writes (UO.WCF in UNTOWN)
* 7	200	Write check all writes (UO.WCU in UNTOWN)
* 10.	2000	Automatically lookup pack ID
* 11.	4000	Use named directories regardless of PRI/PUB
* 12.	10000	Don't use named directories
13.	20000	Mount the pack read only
14.	40000	Mount the pack private
15.	100000	Mode bits are "real" - use them

Bit 10. is particularly useful — it allows mounting the pack even if the pack ID is unknown. Now you can ditch all of those stupid programs to print a NFS disk pack's ID. Bits 11. and 12. control whether or not the NAME.SYS file will be used (if named directories have been 'genned into the monitor). Normally, NAME.SYS is not used on public disks other than the system disk, and is used on all private disks. You can override this by setting bits 11. and 12. appropriately.

Bits 6 and 7 control whether or not FIP writes or all writes will be checked. These bits correspond to UO.WCF and UO.WCU in UNTOWN. Also, there are two additional undocumented bits in UNTOWN which control data and directory caching for a particular disk unit. If set correctly, data and directory caching can be enabled or disabled on a per-unit basis. (I remember this being one of the wishes at a past DECUS . . . it is already implemented to some extent.) Unfortunately, there is no call to modify these bits, so the only way to change them is by poking monitor memory.

CONCLUSION

If you aren't up to keying in the programs, send $25.00 to IISI (Attn:MCG) and we'll send you a tape of all of the Crystal Ball goodies. Please specify 300 or 1600bpi.

Much of the information printed in this column is the direct result of reader inquiries and comments. If you have a topic that you would like to see explored in a future issue, communicate it to me, preferably in writing. I will attempt to investigate/document any RSTS subject, provided there is sufficient user interest (i.e., RESPONSE).

I hope you have enjoyed this installment of the RSTS Crystal Ball. I will continue to try to present information which is interesting and useful. If you have any questions, gripes, or suggestions, feel free to call or write to the address below. Until next time, happy hacking!

Michael C. Greenspon
C/O Integral Information Systems
9832 Vicar Street, Suite 100
Los Angeles, California 90034
(213) 558-0732

```
!------------------------------------------------------------------------&
! [MCG]                                                                  &
!                                                                        &
!                                                                        &
!             Program:        FI                                         &
!             Description:    Hacks FMS form files                       &
!             Version:        X1.0                                       &
!             OS Version:     RSTS 7.1-11 [and later]                    &
!             Package:        Internal hacks                             &
!             Author:         Michael C. Greenspon                       &
!             Edit date:      14 November 1982                           &
!                                                                        &
!                                                                        &
!------------------------------------------------------------------------&
!                                                                        &
!                                                                        &
```

```
!       Title to and ownership of this software shall at all times re-   &
!       main with Integral Information Systems.                          &
!                                                                        &
!       The information in this software is subject to change without    &
!       notice and should not be construed as a commitment by Integral   &
!       Information Systems.                                             &
!                                                                        &
!       This software is unreleased and Integral Information Systems     &
!       has no obligation to support it at this time, unless stated      &
!       elsewhere in writing.                                            &
!                                                                        &
!                                                                        &
!------------------------------------------------------------------------&
!                                                                        &
!                                                                        &
!       Modification history:                                            &
!                                                                        &
!                                                                        &
!       Date            Ver/Ed  By      Reason                           &
!       ----------      ------  ----    ----------------------           &
!                                                                        &
!       14 November 82  1.0/01  MCG     Initial creation                 &
!                                                                        &
!                                                                        &
!------------------------------------------------------------------------&
!                                                                        &
!                                                                        &
!       Description:                                                     &
!                                                                        &
!       FI is a kludge which hacks VT100 graphics and line drawing       &
!       characters into FMS forms. The input form file should be         &
!       created with FED as usual, and all graphics mode characters      &
!       should be given all four AVO attributes (bold, reverse, blink,   &
!       underscore). The output form file will be displayed by FDV       &
!       with graphics.                                                   &
!                                                                        &
!       Note: Output form files can't be edited with FED or printed      &
!       with FUT.                                                        &
!                                                                        &
!                                                                        &
!------------------------------------------------------------------------&
!                                                                        &
!                                                                        &
!       Compilation instructions:                                        &
!                                                                        &
!       Simple ... FI is self-contained, so enter BP2 and                &
!                                                                        &
!       OLD FI                                                           &
!       COM/OBJ                                                          &
!       BUI                                                              &
!       RUN $TKB                                                         &
!       @FI                                                              &
!                                                                        &
!                                                                        &
!------------------------------------------------------------------------&

600     !                                                                &
        !                                                                &
        !       Compile time constants:                                  &
        !                                                                &
        !                                                                &
                                                                         &
        .DEFINE .IDENT$         =       "FI" ! Who we are                &
\       .DEFINE .VERSION$       =       "V7.2-69" ! What version we are  &
                                                                         &
\       .DEFINE .ESC%   =       155%    ! Parity-high escape             &
\       .DEFINE .SO%    =       142%    ! Parity-high SO                 &
\       .DEFINE .SI%    =       143%    ! Parity-high SI                 &

800     !                                                                &
        !                                                                &
        !       MAPs and COMMONs:                                        &
        !                                                                &
        !                                                                &
                                                                         &
        MAP (INP) INP$ = 512%           ! Map entire input buffer        &
\       MAP (INP) INP$(511%) = 1%       !  by bytes                      &
\       MAP (INP) INP%(255%)            !  and words                     &
\       MAP (OUT) OUT$ = 512%           ! Same story for output buffer   &
\       MAP (OUT) OUT$(511%) = 1%                                        &
\       MAP (OUT) OUT%(255%)                                             &
                                                                         &
1000    !                                                                &
        !                                                                &
        !       Yes, there is actually code here somewhere...            &
        !                                                                &
        !                                                                &
                                                                         &
        PRINT IF CCPOS(0%)              ! Reset to margin if needed      &
\       PRINT .IDENT$ + HT + .VERSION$ + HT + HT                         &
              + "Integral Information Systems" + CR + LF                 &
              + "FMS forms hacker" + CR + LF                             &
                                        ! Print our header               &
                                                                         &
1010    LINPUT "Input filename";a$      ! Get filename                   &
\       A$ = A$ + ".FRM" UNLESS INSTR(1%,A$,".")                         &
                                        ! Default the extension the quick and &
                                        !  dirty way                     &
\       PRINT IF CCPOS(0%)              ! Reset margin                   &
\       OPEN A$ FOR INPUT AS FILE #1%, MODE 8192%, MAP INP               &
\       LINPUT "Output to";B$           ! Open file for input, get the   &
                                        ! output file name               &
\       B$ = B$ + ".GRF" UNLESS INSTR(1%,B$,".")                         &
\       PRINT IF CCPOS(0%)              ! Do bagbitting extension defaulting &
\       OPEN B$ FOR OUTPUT AS FILE #2%, MAP OUT                          &
                                        ! Open output file               &
                                                                         &
\       GRAPH$ = CHR$(.ESC%) + ")0" + CHR$(.SI%)                         &
                                        ! Set graphics mode as G1 set, and &
                                        ! switch to G0 set just in case... &
                                                                         &
\       GET #1%, RECORD 1%              ! Get the header block           &
\       OLD.TLEN% = INP%(4%)            ! Get old text length and        &
\       OLD.FLEN% = INP%(12%)           !  old form length from header   &
                                                                         &
```

Intersil Systems

MEETS THE DEC® DH-11 CHALLENGE

The challenge; increased communications capacity, increased reliability, and improved technology - at decreased cost.

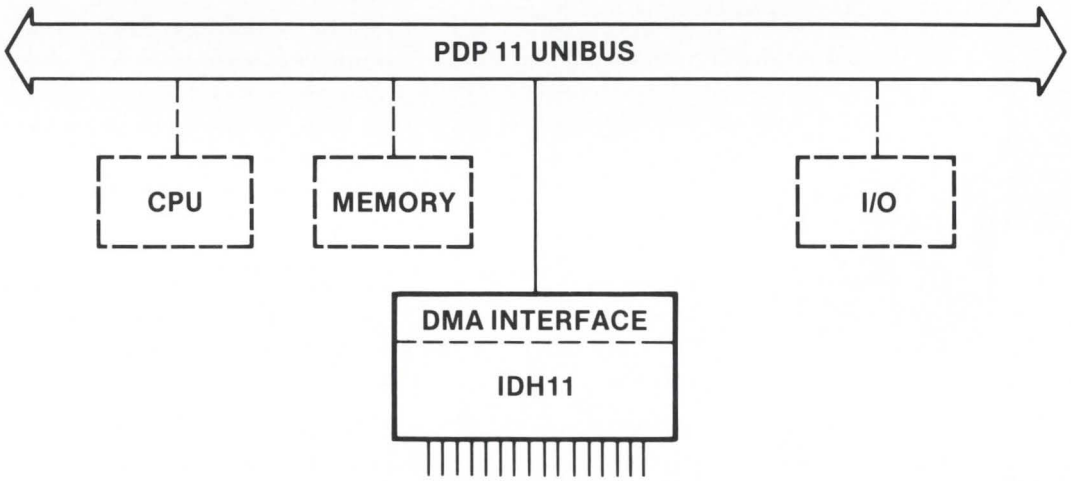
That's not a challenge for Intersil Systems.

- **Increased Capacity –** One Intersil DH-11 replaces nine DEC card slots
- **Increased Reliability –** Only one chance for a failure instead of nine.
- **Improved Technology –** The latest microprocessor based technology is employed.
- **Decreased Cost –** 66% less! One Intersil DH-11 costs 66% less than one DEC DH-11.

In addition the Intersil DH-11 offers these features:

- 16 Asynchronous local or remote channels on the UNIBUS® .
- DMA output to free the CPU from Interrupt handling.
- On-board diagnostics.
- Complete software compatibility.

IDH11



16 ASYNCHRONOUS CHANNELS TO LOCAL OR REMOTE TERMINALS

Get all the challenging facts. Call (408) 743-4300, TWX: 910-339-9369, or write Intersil Systems, Inc., 1275 Hammerwood Avenue, Sunnyvale, CA 94086

® Trademarks of Digital Equipment Corporation

CIRCLE 171 ON READER CARD

```
\       UNLESS RAD$(FNGET.WORD$(0%))+RAD$(FNGET.WORD$(0%)) = "FRMLBR"          &
        THEN    PRINT "?Not an FMS form file, cretin"                          &
                                        ! Insure this is an FMS form file      &
        \       CLOSE 1%, 2%            ! Croak if it isn't...                 &
        \       GOTO 1010                                                      &
                                                                               &
        ELSE                                                                   &
                                                                               &
        \       PRINT "Form: " + RAD$(INP%(2%))+RAD$(INP%(3%))                 &
                                        ! Identify form for the user           &
        \       J% = FNPUT.WORD%(INP%(0%)) + FNPUT.WORD%(INP%(1%))             &
                        + FNCOPY.WORD%(16%) + FNPUT.FIELD%(1%,1%,0%,GRAPH$)    &
                                        ! Copy form header to output file,     &
                                        ! and insert graphics sequence at      &
                                        ! start of form text                   &
        \       J% = FNGET.WORD%(0%)    ! Get first word of text section       &
                                                                               &
        \       PRINT "[Blank form]" IF J% = -1%                               &
                                        ! Say this form is blank if the text   &
                                        ! section is null                      &
        \       UNTIL   J% = -1%        ! Until we hit the end of the text,    &
                                                                               &
                \       Y% = J% AND 255%                                       &
                \       X% = SWAP%(J%) AND 255%                                &
                                        ! Get the [x,y] screen coordinates for &
                                        !  the text                            &
                \       ATR% = FNGET.WORD%(0%)                                 &
                \       CNT% = FNGET.BYTE%(0%)                                 &
                                        ! Get video attributes, and text length &
                                                                               &
                \       TEXT$ = ""                                             &
                \       TEXT$ = TEXT$ + CHR$(FNGET.BYTE%(0%))                  &
                                FOR FOO% = 1% TO CNT%                          &
                                        ! Get the text string                  &
                                                                               &
                \       Z% = FNGET.BYTE%(0%) IF (CNT% AND 1%) = 0%             &
                                        ! The text is padded to an even word   &
                                        ! boundary, so make sure we get the    &
                                        ! trailing null if this is the case     &
                                                                               &
                \       ST$, EN$ = ""   ! Zero out prefix and suffix strings   &
                \       ST$ = CHR$(.SO%) IF ATR% = 15%                         &
                \       EN$ = CHR$(.SI%) IF ATR% = 15%                         &
                                        ! Put SO/SI pair around the text if    &
                                        ! all four AVO attributes are set      &
                                                                               &
                \       ATR% = 0% IF ATR% = 15%                                &
                                        ! Clear all video attributes if grafix &
                                                                               &
                \       Z% = FNPUT.FIELD%(X%,Y%,ATR%,ST$ + TEXT$ + EN$)        &
                                        ! Write the hacked text to the output  &
                                        ! file                                 &
                \       J% = FNGET.WORD%(0%)                                   &
                                        ! Get the next word from the input file &
        \       NEXT                                                           &
        \       J% = FNPUT.WORD%(J%)    ! Write out the end-of-text-section    &
                                        ! marker                               &
        \       ON ERROR GOTO 1020      ! Set error trap for EOF error         &
        \       UNTIL 0%                                                       &
                \       J% = FNCOPY.WORD%(1%)                                  &
                                        ! (Slowly) copy the rest of the form   &
                                        !  to the output file                  &
        \       NEXT                                                           &

1020    RESUME 1030 IF ERR = 11%        ! Resume if this is the expected EOF   &
\       ON ERROR GOTO 0                                                        &

1030    PUT #2% IF OUT.BUFPTR%          ! Punt the last block if we need to    &
\       GET #2%, RECORD 1%              ! Get the header block again           &
\       DLEN% = FLEN% - OLD.FLEN%       ! Calculate the new form length        &
\       OUT%(4%) = OLD.TLEN% + DLEN% + 4%                                      &
\       OUT%(12%) = FLEN% + 4%                                                 &
                                        ! Update the lengths in the header     &
\       PUT #2%, RECORD 1%                                                     &
\       CLOSE 1%,2%                                                            &
\       GOTO 32767                                                             &
                                        ! Punt it back to disk, close the files &
                                        !  and exit...                         &
                                                                               &


15000   !                                                                      &
        !                                                                      &
        !       Random wizard-defined functions                                &
        !                                                                      &
        !                                                                      &
                                                                               &
        DEF* FNGET.WORD$(N%)            ! Skip N% words and return a word from &
                                        !  the input file                      &
\       IN.BUFPTR% = FNADV.PTR%(2%)     ! Advance pointer by two as much as we &
                FOR JUNK% = 1% TO N%    !  need to                             &
\       FNGET.WORD$ = INP%(IN.BUFPTR%/2%)                                      &
                                        ! Return the word                      &
\       IN.BUFPTR% = FNADV.PTR%(2%)     !  and advance pointer to next word    &
\       FNEND !FNGET.WORD$!                                                    &

15010   DEF* FNGET.BYTE%(N%)            ! Skip N% bytes and return a byte from &
                                        !  the input file                      &
\       IN.BUFPTR% = FNADV.PTR%(1%)     ! Advance pointer as much as needed    &
                FOR JUNK% = 1% TO N%                                           &
\       FNGET.BYTE% = ASCII(INP$(IN.BUFPTR%))                                  &
                                        ! Return the byte                      &
\       IN.BUFPTR% = FNADV.PTR%(1%)     !  and advance pointer to next byte    &
\       FNEND !FNGET.BYTE%!                                                    &

15020   DEF* FNADV.PTR%(INC%)           ! Advance pointer by INC%              &
\       IN.BUFPTR% = IN.BUFPTR% + INC%  ! Simple                               &
\       IF      IN.BUFPTR% > 511%       ! If off end of buffer, then           &
        THEN    GET #1%                 !  get next block and reset pointer    &
        \       IN.BUFPTR% = 0%                                                &

15030   FNADV.PTR% = IN.BUFPTR%         ! Return the buffer pointer            &
\       FNEND !FNADV.PTR%!                                                     &

15040   DEF* FNPUT.WORD$(ARG%)          ! Write a word to the output file      &
\       OUT%(OUT.BUFPTR%/2%) = ARG%     ! Buffer the word                      &
\       OUT.BUFPTR% = OUT.BUFPTR% + 2%  ! Increment the buffer pointer         &
\       IF      OUT.BUFPTR% > 511%      ! If off end of buffer,                &
        THEN    OUT.BUFPTR% = 0%        !  then reset the pointer              &
                \       PUT #2%         !  and punt the block to disk          &
                \       OUT$ = STRING$(512%,0%) ! Zero the output buffer       &

15050   FNPUT.WORD% = ARG%              ! Return the word written              &
\       FNEND !FNPUT.WORD%!                                                    &

15060   DEF* FNPUT.BYTE%(N%)            ! Write a byte to the output file      &
\       OUT$(OUT.BUFPTR%) = CHR$(N%)    ! Buffer character                     &
\       OUT.BUFPTR% = OUT.BUFPTR% + 1%  ! Increment buffer pointer             &
\       IF      OUT.BUFPTR% > 511%                                             &
        THEN    OUT.BUFPTR% = 0%                                               &
                \       PUT #2%         ! Punt block and zero buffer if we are &
                \       OUT$ = STRING$(512%,0%) !  done with the block         &

15070   FNPUT.BYTE% = N%                ! Return the written character         &
\       FNEND !FNPUT.BYTE%!                                                    &

15080   DEF* FNCOPY.WORD%(BLETCH%)      ! Copy BLETCH% words from input to     &
                                        !  output file                         &
\       FNCOPY.WORD% = FNPUT.WORD%(FNGET.WORD%(0%))                            &
                FOR LOCAL% = 1% TO BLETCH%                                     &
                                        ! Just write read words...             &
\       FNEND !FNCOPY.WORD%!                                                   &

15090   DEF* FNPUT.FIELD%(XCOOR%,YCOOR%,ATR.WORD%,DATA$)                       &
                                        ! Write a text field to output file    &
\       FNPUT.FIELD% = FNPUT.BYTE%(YCOOR%) + FNPUT.BYTE%(XCOOR%) +             &
                FNPUT.WORD%(ATR.WORD%) + FNPUT.BYTE%(LEN(DATA$))               &
\       FNPUT.FIELD% = FNPUT.BYTE%(ASCII(MID$(DATA$,FOO%,1%)))                 &
                FOR FOO% = 1% TO LEN(DATA$)                                    &
                                        ! Write header and data                &
\       FNPUT.FIELD% = FNPUT.BYTE%(0%)  ! Pad data with a null if it is odd    &
                UNLESS LEN(DATA$) AND 1% ! (that's odd...)                     &
\       FLEN% = FLEN% + LEN(DATA$) + 5% ! Update global form lengths           &
\       FLEN% = FLEN% + 1%                                                     &
                UNLESS LEN(DATA$) AND 1%                                       &
\       FNEND !FNPUT.FIELD%!                                                   &

32767   END                             ! All good things must come to an end
```

```
;
; Support macros for GETFRM
;
; Usage is:
;
; -display form, set first field to get in call blocks-
;
;       GETFORM [ABORT=aboaddr,SPEC=pfx,UPCASE=YES/NO]
;
;       aboaddr is a symbol to jump to if the form is aborted
;
;       pfx is a three character prefix for the generated PSECTs and
;       symbols. It should be used with all FIELD and TERM macros for
;       this GETFORM, and must not be used again in another GETFORM.
;
;       UPCASE=YES indicates that all data fields should be CVT$$'d to
;       upper case before calling the field processing routine. UPCASE=NO
;       is the reverse.
;
; If the form is terminated normally (with FORM), control will return
; in-line, to the instruction after the GETFORM.
;
; In order to perform individual field validation, use the FIELD macro:
;
;       FIELD   name,pfx
;
;       name is the name of the FMS field on the form to perform special
;       validation for.
;
;       pfx is the same three character prefix used with the GETFORM macro.
;
;       Whenever a terminator is entered in the named field, control will
;       be passed to the instruction following the FIELD macro. (See GETFRM
;       routine for more details)
;
; To perform special terminator checking, use the TERM macro:
;
;       TERM    term,pfx
;
;       term is the numeric code for the FMS terminator to perform special
;       checking on.
;
;       pfx is the same as above...
;
;       Whenever the "term" terminator is entered on the form, first the
;       special field validation, if any, will be performed, and then if
;       terminator processing is still requested, control will pass to the
;       instruction following the TERM macro. (See GETFRM for more details...)
;
; You should use the ENDFORM macro to terminate the table definitions.
;
;       ENDFORM pfx
;
;       pfx is the same three character prefix used with the GETFORM that
;       you are ENDing...
;
; Have fun; hack, hack ~[MCG]~
;

.MACRO  GETFORM ABORT=0,SPEC,UPCASE=YES
.NLIST
$$$     =       0
.IF     DIF     <UPCASE>,<YES>
$$$     =       1
.ENDC
.IF     NB      <SPEC>
.LIST
        CALLX   GETFRM,R5,<ABORT,FRM'SPEC,$$$>
.NLIST
.PSECT  FRM'SPEC,D,RO,GBL,CON
FRM'SPEC:       .WORD   0,0
        UNORG
.IFF
.LIST
        CALLX   GETFRM,R5,<ABORT,0,$$$>
```

```
.NLIST
.ENDC
.LIST
.ENDM   GETFORM

.MACRO  FIELD   NAME,TABLE
.NLIST
$$1     =       .
.PSECT  FLD'TABLE,D,RO,GBL,CON
.IF     NDF     FLD'TABLE
FLD'TABLE:
.ENDC
$$$     =       .
        .ASCII  "NAME"
$$0     =       .
.IF     GT      6-<$$0-$$$>
.REPT   6-<$$0-$$$>
        .BYTE   40
.ENDR
.ENDC
        .WORD   $$1
.PSECT  FRM'TABLE,D,RO,GBL,CON
.       =       0+FRM'TABLE
        .WORD   FLD'TABLE
        UNORG
.LIST
.ENDM   FIELD

.MACRO  TERM    TERM,TABLE
.NLIST
$$1     =       .
.PSECT  TRM'TABLE,D,RO,GBL,CON
.IF     NDF     TRM'TABLE
TRM'TABLE:
.ENDC
        .WORD   TERM,$$1
.PSECT  FRM'TABLE,D,RO,GBL,CON
.IF     NDF     FRM'TABLE
FRM'TABLE:
.ENDC
.       =       2+FRM'TABLE
        .WORD   TRM'TABLE
        UNORG
.LIST
.ENDM   TERM

.MACRO  ENDFORM NAME
.NLIST
.IF     DF      FLD'NAME
.PSECT  FLD'NAME,D,RO,GBL,CON
        .WORD   0
.ENDC
.IF     DF      TRM'NAME
.PSECT  TRM'NAME,D,RO,GBL,CON
        .WORD   0
.ENDC
        UNORG
.LIST
.ENDM   ENDFORM
```

```
.SBTTL  GETFRM - Get an FMS form

;+
;
; GETFRM - Get an FMS form
;
; Call: FMSARG+F$NAM -> current field name
;       FMSARG+F$NUM =  current field index value, if indexed
;       Form context must be established, i.e. form displayed
;
;       CALL    GETFRM,R5,<ABORT,SPEC,UPCASEFLAG>
;
;       ABORT is the address to exit through if ABORT is entered and confirmed,
;       or zero if ABORT is invalid.
;
;       SPEC is the address of an exception processing table, as follows:
;
;               .WORD   address of fields table
;               .WORD   address of terminators table
;
;       The fields table contains an entry for each field that needs special
;       processing.  An entry consists of the six character field name in ASCII
;       followed by the one word address of the routine to call when a term-
;       inator is processed in this field.
;
;       The terminators table contains an entry for each terminator that needs
;       special processing. An entry consists of one word which is the numeric
;       code returned by FMS for the terminator, and the one word address of
;       the routine to call when this terminator is hit.
;
; Back: Return when FORM is hit, or exit through ABORT address if selected.
;
; Note: Field processing is done before terminator processing.
;
;       When a field processing routine is called, all registers are saved.
;       The return address specifies terminator processing. If no terminator
;       processing is desired (i.e. re-get current field if invalid) then
;       return to the next word, i.e. ADD #2,(SP) before returning.
;
;       Normally the GETFORM, FIELD, TERM and ENDFORM macros are used to
;       call this routine and set up table entries for exception processing.
;
; Also: TE$NTR, TE$ABO, and TE$FRM should be equated to the value of the
;       appropriate FMS terminators, i.e. ENTER, ABORT (Gold-PF4), and FORM
;       (PF4). (You can, of course, use whatever keys you want...)
;
;       XLATRM should translate the returned FMS terminator codes into your
;       special terminator codes (TE$xxx, etc.) if there is an equivalent,
;       otherwise, it should leave the FMS terminator as-is. (GETFRM expects
;       the translated terminator returned in R1.)
;
;       The FMSDEF and FMSIMP macros define the FMS impure area and call
;       blocks.  The symbol FMSARG is defined to be the starting address
;       of the FMS call block.
;
```

```
;       STRBF2 is a fixed-length scratch string buffer.
;
;       CVT$$ is a fairly standard string-trim routine.
;
;-

.ENABL  LSB

GETFRM::REGSAV                  ; Save everything
        BR      30$             ; Get the first field and process it

10$:    CALL    XLATRM          ; Translate the terminator again
        CMP     #TE$FRM,R1      ; End of form?
        BNE     20$             ;  no, go on
        REGRES                  ; Restore all
        ADD     #6,R5           ; Skip arguments
        RETURN  R5              ;  and get out

20$:    CALFMS  DOTERM,ARG=#FMSARG ; Process the terminator
30$:    CALFMS  GET,ARG=#FMSARG,ERR=FMSER2 ; Get us a field
        CALL    XLATRM          ; Translate the terminator
        CMP     #TE$ABO,R1      ; ABORT form?
        BNE     40$             ;  no, go on
        TST     @TOS.R5(SP)     ; Any abort routine?
        BEQ     40$             ;  no, go on

        FMSG    <Hit ENTER to abort form, or PF4 to keep editing.>,ARGS=#FMSARG
        PUSH    FMSARG+F$NAM    ; Save field name
        PUSH    FMSARG+F$NUM    ;  and index
        CALFMS  PAUSE           ; Get a confirm
        POP     FMSARG+F$NUM    ; Restore index
        POP     FMSARG+F$NAM    ;  and name
        CALL    XLATRM          ; Translate the terminator
        CMP     #TE$NTR,R1      ; Enter?
        BNE     30$             ;  no, keep editing

        REGRES                  ; Restore everything
        PUSH    R5              ; Save argument pointer
        MOV     2(SP),R5        ; Get calling R5 value
        MOV     @(SP)+,(SP)     ; Put abort address on top of stack
        JMP     @(SP)+          ; Now go to the abort routine

FMSER2: JMP     FMSFAT          ; FMS fatal error

40$:    MOV     F$LEN(R0),R1    ; Get its length
        MOVMEM  F$VAL(R0),#STRBF2 ; Move the data into a string buffer
        ADD     R5,R1           ; Point to end of string
        CLRB    (R1)            ; Make it ASCIZ
        MOV     #^C<402>,R0     ; Set CVT mask
        MOV     TOS.R5(SP),R2   ; Get argument pointer
        TST     4(R2)           ; Wants upper case?
        BEQ     50$             ;  yep, go on
        MOV     #^C<442>,R0     ; Use different mask

50$:    CALLX   CVT$$           ; Trim up the string
        MOV     R5,R1           ; Copy pointer
60$:    TSTB    (R1)+           ; Find end of string
        BNE     60$
        MOV     2(R2),R2        ; Get table pointer
        BEQ     10$             ;  no table pointer, just process terminator
        MOV     (R2),R3         ; Get pointer to field name table
        BEQ     110$            ; No field processing, go on
70$:    TST     (R3)            ; End of table?
        BEQ     110$            ;  yep, try terminator processing
        MOV     FMSARG+F$NAM,R4 ; Get pointer to field name
        CMP     (R4)+,(R3)      ; Is this the field name?
        BNE     80$             ;  no, go on
        CMP     (R4)+,2(R3)     ; Really the field name?
        BNE     80$             ;  nah, not really
        CMP     (R4)+,4(R3)     ; This is getting "yawn" boring...
        BEQ     90$             ; Hey, MIKEY! We found one...

80$:    ADD     #10,R3          ; Try next entry
        BR      70$

90$:    REGSAV                  ; Save everything
        PUSH    FMSARG+F$TRM    ; Save the terminator code
        PUSH    FMSARG+F$NAM    ;  and the field name pointer
        PUSH    FMSARG+F$NUM    ;  and the index of the field
        CLR     -(SP)           ; Clear a word
        CALL    @6(R3)          ; Call the field processing routine
         INC    (SP)            ; Don't do terminator processing
        MOV     (SP)+,FIRQB     ; Stash this away for a minute
        POP     FMSARG+F$NUM    ; Restore the index of the field
        POP     FMSARG+F$NAM    ;  and the name pointer
        POP     FMSARG+F$TRM    ; Restore the terminator
        MOV     R5,TOS.R5(SP)   ; Paste up string pointer
        REGRES                  ; Restore everything
        MOV     FIRQB,R3        ; Pick this up again
        TST     R5              ; Any string pointer now?
        BEQ     100$            ;  no, don't display anything in field
        CALFMS  PUT,VAL=R5,LEN=-1,ARG=#FMSARG,ERR=FMSER2 ; Redisplay the field

100$:   TST     R3              ; Want terminator processing?
        BEQ     160$            ;  no, never, get field again

110$:   MOV     2(R2),R3        ; Get pointer to terminator table
        BEQ     140$            ;  none, process existing terminator
120$:   TST     (R3)            ; End of table?
        BEQ     140$            ;  yep, process terminator
        CMP     (R3)+,FMSARG+F$TRM ; Is this the terminator?
        BEQ     130$            ;  yes, dispatch to service routines
        TST     (R3)+           ; Skip service routine address
        BR      120$            ;  and keep looking

130$:   REGSAV                  ; Save everything
        CALL    @(R3)+          ; Call the service routine
         BR     150$            ; Don't do terminator processing
        REGRES                  ; Restore all
140$:   JMP     10$             ;  and do terminator processing

150$:   REGRES                  ; Restore all
160$:   JMP     30$             ;  and don't do terminator processing

.DSABL  LSB
```

♥

ENABLE/34 ANOTHER POINT OF VIEW

By Kevin Munday
System Analyst, ABLE Computer

In the February issue of The RSTS/PROFESSIONAL, an article pointed out what were felt to be some deficiencies with ABLE'S ENABLE. This is a response to that article.

The ENABLE is a hardware/software product which allows an 18 bit PDP-11 to address 22 bits of memory. This means that a 128kb RSTS system can now access four megabytes of memory (if one has that much physical memory in the machine). Such an expansion substantially increases system throughput by allowing all currently existing programs to remain memory resident. Disk swapping is substantially reduced. CPU speed becomes the only performance limiter.

The ENABLE is not intended to replace new or larger machines, but rather to upgrade existing systems when

budget does not support acquisition of a larger system or when the user wishes to optimize the existing investment.

Installing the ENABLE is indeed not a simple five minute job, but it can be done in a relatively short period of time with the help of the manual. In addition, ABLE'S Technical Support Department is always willing to help with the installation over the phone, even to the point of talking a non-technical user through restrapping of memory boards for new addresses. ABLE has a toll-free number (800-854-9471) to facilitate this kind of help.

If one buys the ENABLE, there are other hardware commitments which must be made, such as expansion space, 22 bit memory, etc. ABLE offers an additional solution, the MEGABOX, which provides an ENABLE and 22 bit memory in a separate backplane (this subsystem requires very little installation).

ENABLE does require patches to the operating system. When ABLE receives update listings from DEC (e.g., from V7.0 to V7.1), a new ENABLE patch is written within two weeks. The patch is field tested and then is available to ENABLE users. Total lag time is 45 days. ENABLE users who upgrade their operating system must contact ABLE to order the new patch.

The ENABLE has fairly tight specifications as to which devices are compatible with it on a UNIBUS. As a result, it is not a device which can go on every machine and this tends to limit the number of users who will find the ENABLE option viable.

If you do have problems with your system, ABLE is always glad to help coordinate any support efforts with DEC. But if they never receive a call from the customer, they can't do anything about it. In addition, it is easy to put the ENABLE in a mode where it is transparent to the system (18 bit as opposed to 22 bit) for testing to determine whether or not the problem lies with the ENABLE product. In any case, the customer should always feel free to call ABLE'S Technical Support Department before calling DEC.

As far as problems with the ENABLE hardware, there was at one time a problem with RSTS/E not handling parity memory properly due to the optional cache available with the ENABLE. This problem no longer exists.

The ENABLE patches are written specifically to avoid conflict with DEC patches. They do not, to ABLE'S knowledge, conflict with any DEC patch for RSTS/E. If a customer does find a conflict with DEC'S patches to RSTS/E, he should inform ABLE immediately so that the problem can be investigated and resolved.

Overall, the ENABLE performs better than originally designed. Users are discovering new applications, and the number of ENABLE/MEGABOX installations is rapidly expanding. ♥

RESIDENT LIBRARY TIPS FOR DIBOL USERS

By John Wolnisty, Beta Data Systems, Inc., Tucson, AZ

When it comes to utilizing the elegant facilities RSTS has available, DIBOL programmers have traditionally been left with the short end of the stick. Never being one to leave well enough alone, I became intrigued with the use of read/write resident libraries to share data among several concurrently running programs. Furthermore, the programs were in DIBOL. Version 4.5 of DIBOL supports resident library use, and in fact, the run time support is a resident library (previous versions were a run-time system) itself. Resident library support was added to allow use of the RMS resident library and little else. DIBOL does not use a standard subroutine linkage convention, so use of a resident library for user-written routines was not a prime consideration. Why a read/write resident library? The uses for such a structure are only limited to the programmer's imagination. I have implemented them in three different applications: a "real-time" file access program, a multi-user game and a mail-box program. To describe how this was implemented some DIBOL internals need to be explored first.

COMMON SECTIONS.

A COMMON section in DIBOL is similar to COMMON found in FORTRAN, in that variables defined in the COMMON are available to all the subroutines of the programs. One difference is that the names MUST be the same throughout the main program and subroutines and must be five characters or less. COMMON defined in the main program generates a global symbol consisting of the five characters of the name and a '$' appended to it. COMMON defined in a subroutine generates a global reference to the symbol of the same name. These references are all resolved by the task builder at link time. The following program, which serves no useful purpose except illustration, shows use of COMMON in a DIBOL program.

```
START   ;This is a conventional DIBOL program using COMMON
        ;
COMMON  DATA1
        CUST1,  A400
        LOCKF,  D1
        ITEMS,  A50
        COMPA,  A200
        RECNM,  D5
RECORD
        INFILE, A26,'[45,55]CUSMAS.DD5/MODE:256'
PROC
        OPEN(1,U,INFILE)
        READ(1,CUST1,1)
        XCALL PASS1
        XCALL PASS2
        WRITE(1,CUST1,1)
        STOP

SUBROUTINE      PASS1

        ;The common section here will generate global references
        ;to the names within the common section. The presence of
        ;the common in the root resolves these references.

COMMON  DATA1
        CUST1,  A400
        LOCKF,  D1
        ITEMS,  A50
        COMPA,  A200
        RECNM,  D5
PROC
        READ(1,CUST1,1)
        CUST1=
        RETURN

SUBROUTINE      PASS2
COMMON  DATA1
        CUST1,  A400
        LOCKF,  D1
        ITEMS,  A50
        COMPA,  A200
        RECNM,  D5
PROC
        WRITE(1,CUST1,1)
        RETURN
```

A portion of the task builder map reveals what happened to the COMMON section.

```
GLOBAL CROSS REFERENCE
SYMBOL  VALUE      REFERENCES...

COMPA$  003075-R    PASS1   # RPRO1  <-- A peice of our COMMON
CUST1$  002172-R    PASS1   # RPRO1  <-- And another
DATA1$  002172-R      PASS1   # RPRO1  <-- Notice the address: same as
ITEMS$  003013-R    PASS1   # RPRO1       above
LOCKF$  003012-R    PASS1     PASS1   # RPRO1
PASS1   002002-R  # PASS1     RPRO1
PASS2   002114-R  # PASS2     RPRO1
RECNM$  003405-R    PASS1   # RPRO1
RPRO1   003446-R  # RPRO1.
RSDDT   002000-R  # RSDDTX    RSDIRT
$AC     000156      RSDDTX    RSDIRT
$ACEND  000222      RSDDTX    RSDIRT
$ACEXT  000110      RSDDTX    RSDIRT
```

Note in the map that the individual components of DATA1 are allocated in the same order as in the program. This may or may not be important. The name of the COMMON (if it was named) always is of zero length.

TRICKING DIBOL INTO USING A RESIDENT LIBRARY.

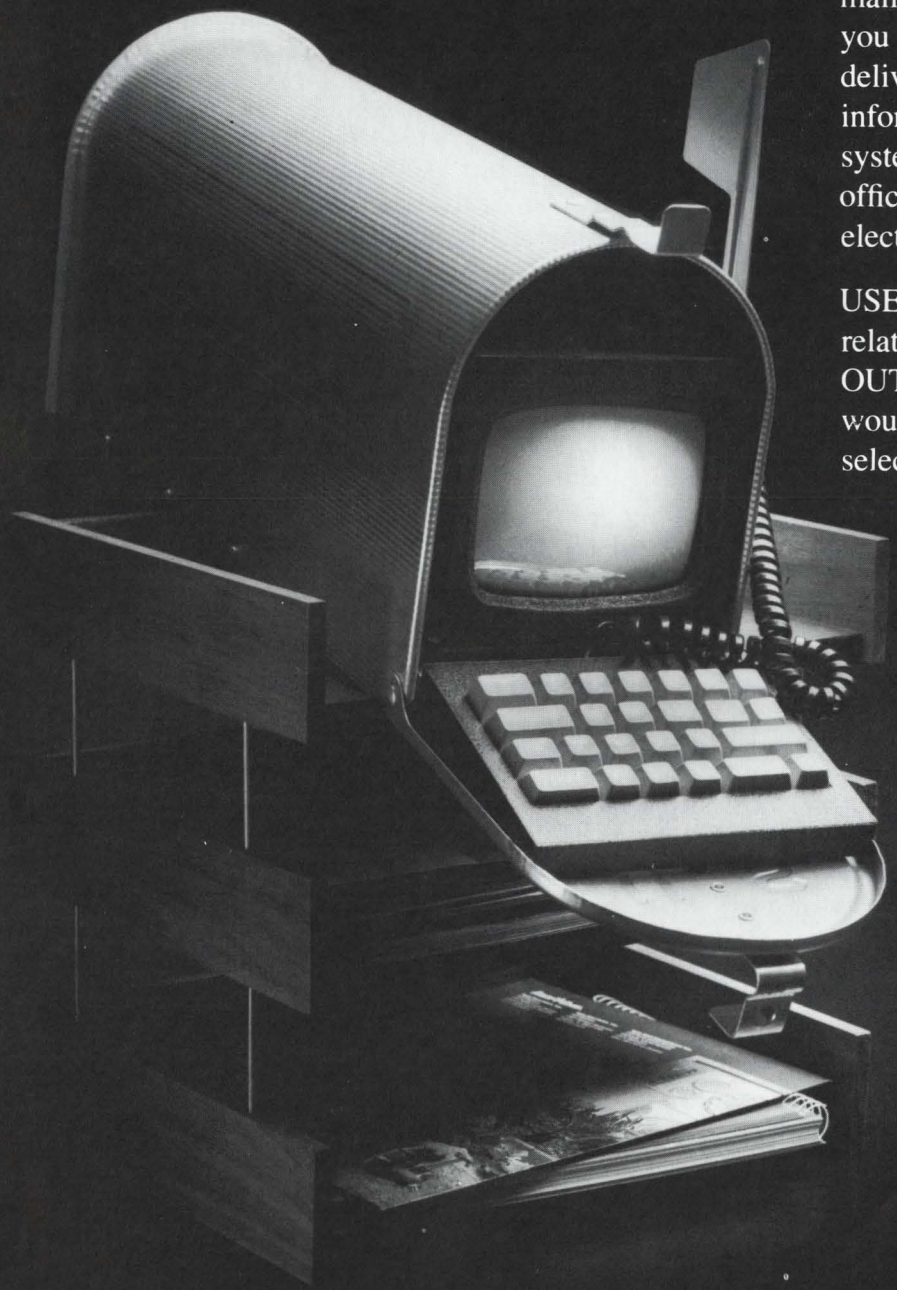
If all the subroutines do for a COMMON section is generate a global reference, it seems to reason that ANY global section with the proper name would resolve the reference. This is indeed true! Just remove the COMMON section from the main program, and insert a section of code with the proper global symbols in its place. Lets do that now.

Our original DIBOL program now becomes:

```
START   ;This is the DIBOL program with COMMON removed
        ;
RECORD
        INFILE, A26,'[45,55]CUSMAS.DD5/MODE:256'
PROC
        OPEN(1,U,INFILE)
        XCALL PASS1
        XCALL PASS2
        STOP
```

Now no CUST1$ is generated along with the other items. If we task built this, we would get unresolved reference errors for every item in the COMMON. It is now necessary to con-

ELECTRONIC MAIL. PRACTICALLY SPEAKING.



Sooner or later you will be using electronic mail. It just makes good sense. When you do, you will want a system that is complete—a delivery system, a scheduling system, and an information manager. Your electronic mail system will become an essential part of your office environment. USER-MAIL is such an electronic mail system.*

USER-MAIL's power is easy to control. It relates to the way you work. Electronic IN, OUT, and HOLD baskets are just what you would expect. You can scan your IN basket, selecting only those message subjects you wish to read. Or, you can place a message into your HOLD basket for a number of days to have it automatically reappear in your IN basket on the appointed day. You can even have USER-MAIL recall specific messages by providing your own selection criteria. Replying, forwarding, and sending to groups are as easy as can be. And these are just a few of the features in store for you.

You owe yourself a closer look. Write for a brochure or give us a call direct.

Visit us at
DEXPO™ East 83 Kiel Auditorium St. Louis, Missouri May 22-24, 1983
Booth #108

NCCS USER 0110

North County Computer Services, Inc.
2235 Meyers Ave.
Escondido, California 92025
(619) 745-6006, Telex: 182773

*USER-MAIL is currently available for DEC computers using the RSTS/E and VMS operating systems.
RSTS is a registered trademark of Digital Equipment Corporation.
USER-MAIL is a trademark of Logic eXtension Resources.

CIRCLE 76 ON READER CARD

struct a new COMMON. This must be done in MACRO, but is easy, even for non-MACRO types. It would look something like:

```
        .PSECT  DATA1$
DATA1$::
CUST1$::        .BLKB   400.
LOCKF$::        .BLKB   1.
ITEMS$::        .BLKB   50.
COMPA$::        .BLKB   200.
RECNM$::        .BLKB   5.
        .END
```

MAC this file, then task build as follows:

```
TKB>DATA/-HD,DATA,DATA=DATA
TKB>/
Enter Options:
TKB>STACK=0
TKB>PAR=DATA:160000:20000
TKB>//
```

Note: You can make it position independent if you want. DBLRES is position independent and will move where necessary.

Now make it into a resident library using MAKSIL and answering the questions (mainly taking defaults after specifying the library name). You may now ADD the resident library with UTILTY with the following precautions: ADD with the RW attribute, set protection code as needed to allow the proper users write access (<0> if you're not sure or don't care), ADD with STAY attribute.

USING THE RESIDENT LIBRARY.

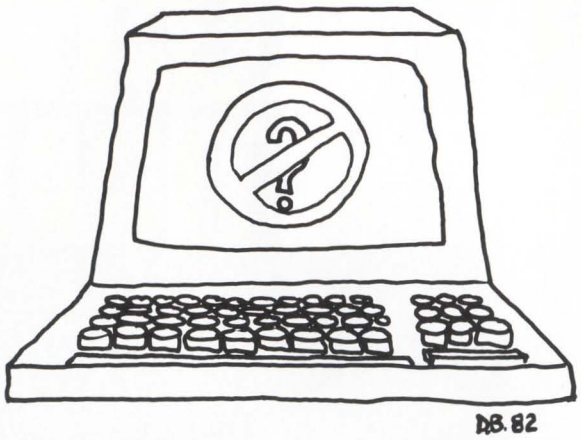
We now take our previous program and re-task build as follows:

```
TKB>PROG=PROG,PASS1,PASS2
TKB>LB:DBLRES/LB
TKB>/
Enter Options:
TKB>RESLIB=LB:DBLRES/RO
TKB>RESLIB=[1,4]DATA/RW
TKB>//
```

Notice the lack of unresolved reference errors, as they were resolved to the resident library. The DIBOL program uses the COMMON area in the resident library exactly the same as it did before. There are several new features now. Any other DIBOL programs may specify this resident library during their task builds. When data is changed by one program accessing any variable in the resident library, that change becomes immediate in all programs using the library. This fact is extremely useful for sharing data among different programs, and only maintaining a single copy of the data. It also poses a problem: you may have to install a lock variable to flag that the data is currently being updated to synchronize the use of the library. Since an entire APR is used to map the library, you can use up to 8192 bytes of data in the resident library. Notice also that the bulk of your program, and in fact all code referencing the resident library, must be in a subroutine. You can code the entire program as a subroutine and have a dummy main routine to call it, if needed.

SUMMARY.

There are a variety of tricks to be done under RSTS DIBOL, but like the preceeding, they are not always apparent to the casual observer. With a little digging, the DIBOL programmer can easily refute the smart-aleck BASIC-PLUS-2 programmer. STOP. ♥



Problem: RSTS does not support fruit. I can connect my ORANGE and my GRAPE to the Unibus, but RSTS refuses to acknowledge it. This fruit is essential for feeding the hamster (the one who runs real fast on his treadmill, to keep the computer running) whenever the hamster runs out of hamster food. As a result, whenever I am running low on hamster food, the hamster slows down or stops running, instead of eating the fruit. Then my customers call and complain.

Isn't there some way to support fruit on RSTS? Or possibly allow two hamsters to fit into the computer, taking shifts. RSTS could ring a bell every half hour to trade hamsters, so that each would have to take his fair turn and we wouldn't have any goof-off hamsters.

Problem: RSTS does not boot on my TRS-80. I called DEC support about this, but they said that RSTS is only capable of running on a PDP-11 and could not be made to run on a TRS-80 without massive patches. Is this true? They said something about RSTS being a "Sales Aid," and that the main reason RSTS was created is to help sales of PDP-11's. I told them that I didn't appreciate them making jokes with me, how can anything that costly be a Sales Aid?

That's when they hung up on me.

Can't you offer a feature in RSTS so that those of us without PDP-11s can use RSTS? Especially us with TRS-80's and APPLEs. Thank you. ♥

UNITRONIX means DIGITAL

PDP 11/23 SYSTEMS
with 256kB*
memory, 160 Mbyte
winchester, std. 45 ips
9-track tape backup, VT101
CRT, LA120 printer.
Runs RSX11, RSTS,
TSX OR UNIX.
$32,000
*up to 4MB avail.

TERMINALS
VT100 LA12
VT101 LA34
VT102 LA50
VT125 LA100
VT131 LA120

CPUs
PDP 11/23,
11/44 & VAX

INTERFACES
MODEMS

COMPLETE DATA SYSTEMS
with standard
and
custom
software

SPECIAL
LSI-11 MODULES, PERSONAL COMPUTERS

WHY YOU SHOULD CALL UNITRONIX FIRST

- **Immediate Delivery from $6 mm Warehouse Inventory**
- **Complete Equipment Service Capability**
- **Excellent Pricing/Discount Structures**

In just over 6 years, Unitronix Corporation has emerged as a leading supplier of Digital Equipment Corporation (DEC) microcomputers, terminal products, printers, accessories and supplies on both a domestic and international level. Our new, ultra-modern 20,000 sq. ft. facility in Somerville, New Jersey, houses executive offices, in-house programming, customer service, sales and marketing, component testing and complete warehouse facilities.

UNITRONIX
CORPORATION
197 Meister Ave., Somerville, NJ 08876
TELEX: 833184

AUTHORIZED
digital®
TERMINALS DISTRIBUTOR

CALL FOR LATEST PRICES
(201) 231-9400

CIRCLE 25 ON READER CARD

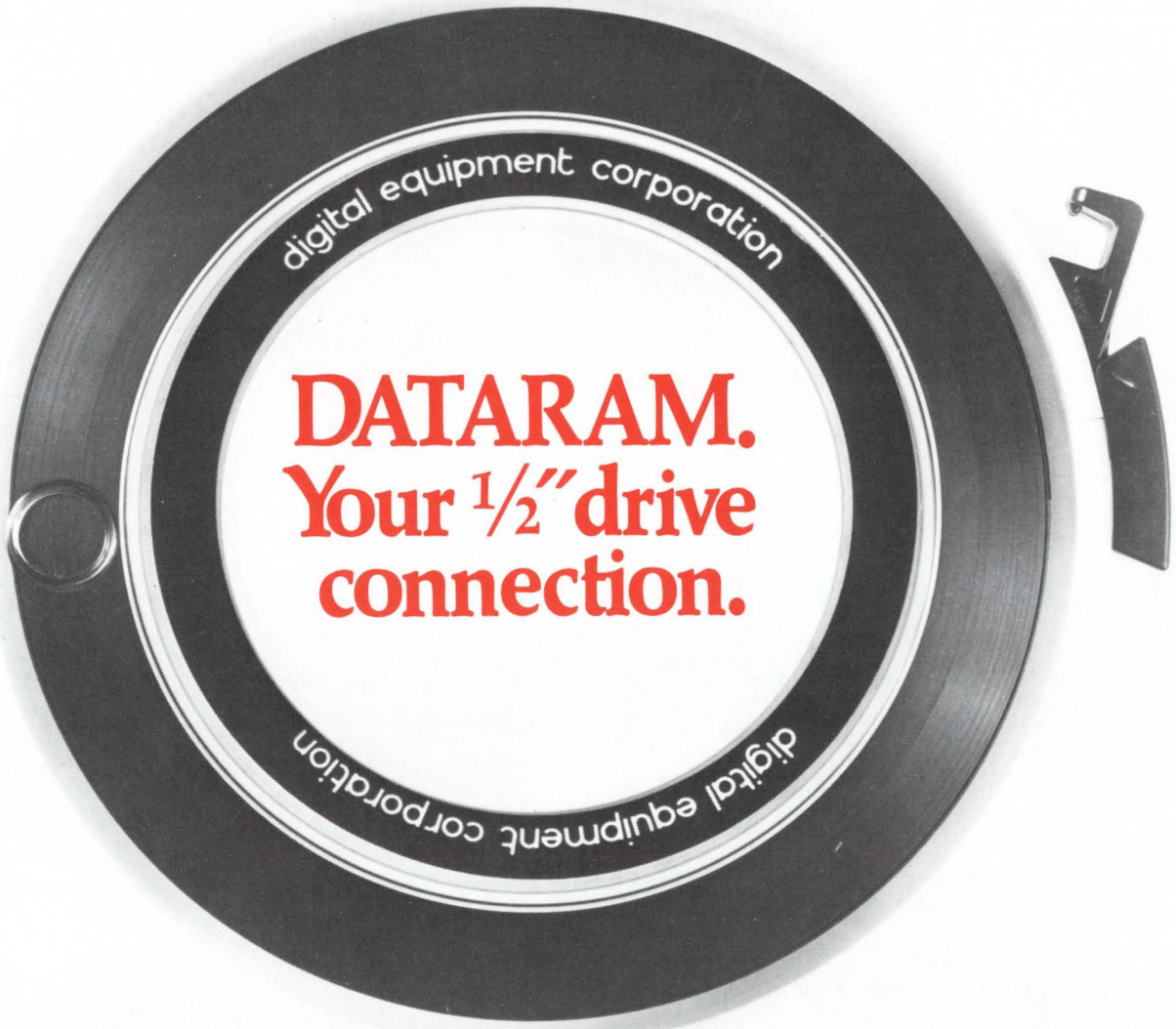
LSTLOG.MAC

By David Patterson, Sivall's, Inc., Odessa, TX

DEC has given us the ability to assign a number of things to a job that are not physical devices. These consist of a private library (the '@' character), a default protection code, and up to four logical assignments, (three if any of them include PPN's). Besides the fact that more than three or four logicals would occasionally be handy in a development situation, the only real problem is that DEC gave us no way to find out what a job currently has assigned. In a development situation where you are using copies of your production data files on your development account for testing purposes this can be annoying, since the only way to verify the assignments is to deassign everything and then reassign them.

When I started using the private logicals for development on a regular basis, I decided to write a program to display the current settings for the current job. This is really not a terribly difficult task since the information is all kept in the job's low-core area (Systems Directives Manual, Section 2.4). All I had to do was format and print it. The following program does just that. I hope others will find is as useful as I have.

```
        .nlist  bin
        .nlist  bex
        .nlist  me
        .list   ttm
        .enabl  lc

        title   LSTLOG,<List user logical table>,04,22-Jun-82,<DMP>

        .sbttl  Comments and Edit History.

;
;       Program name:           LSTLOG
;       Date written:           9-Sep-81
;       Author:                 David Patterson
;       Installation:           Sivalls, Inc.
;
;       Remarks:
;
;               This is a stand-alone program which will print
;               a list of the users current logical assignments
;               on his terminal.
;
;               The information in this document is subject to change
;               without notice and should not be construed as a commitment
;               by either the author or Sivalls, Inc.
;
;       Linking Information:
;
;               MAC LSTLOG,LSTLOG=COMMON,LSTLOG
;               TKB LSTLOG=LSTLOG
;
;       Modification History:
;
;               Ver/Edit        Date            Modification
;
;               V7.0-00         9-Sep-81        Initial conception (DMP).
;
;               V7.0-01         14-Sep-81       General cleanup (DMP).
;
;               V7.0-02          2-Oct-81       Fix problem in check for
;                                               end of logical table (DMP).
;
;               V7.0-03         12-May-82       Fix problem with premature
;                                               exit if the first unused
;                                               entry is not the last entry.
;
;               V7.0-04         22-Jun-82       Fix problem with the PPN
;                                               check.  If the RAD-50 repre-
;                                               sentation of the fourth
;                                               logical assignment happened
;                                               to set the sign bit the
;                                               program was interpreting it
;                                               as PPN's for the second and
;                                               third logicals.
;

        .page
        .sbttl  Global symbols

        global  lstlog                  ; Starting address.
        global  $c5ta                   ; Radix-50 to ASCII conversion.
        global  $cbdsg                  ; Number conversion.
        .mcall  exit$s                  ; Exit macro.


        .page
        .sbttl  Macros

.macro  num$    val,stradr                      ; Convert a value to a
        .nlist                                  ; string.
        push    <r1,r2>                         ; Save these.
        mov     stradr,r0                       ; Where to put the string.
        mov     val,r1                          ; The value to convert.
        clr     r2                              ; Zero spppression.
        call    $cbdsg                          ; (In SYSLIB).
        pop     <r2,r1>                         ; Get them back.
        .list
.endm   num$

.macro  radtoa  val,stradr                      ; Convert a rad-50 number
        .nlist
        push    <r1,r2>                         ;  to ascii.
        mov     stradr,r0                       ; Where to put it.
        mov     val,r1                          ; The value.
        call    $c5ta                           ; Convert it.
        pop     <r2,r1>                         ;
        .list
.endm   radtoa


        .page
        .sbttl  Main program

        .psect  lstlog,rw,i,lcl,rel,con

lstlog: tst     usrprt                          ; Is a default pro set?
        beq     20$                             ; No. skip to the PPN.
        mov     #defpro,r0                      ; Address of protection
        bic     #377,usrprt                     ; Clear the flag.
        swab    usrprt                          ; Get the prot in the low byte
        num$    usrprt,r0                       ; <nnn
        swab    usrprt                          ; Put it back
        bis     #377,usrprt                     ;    the way it was.
        movb    #'>,(r0)+                       ; <nnn>
10$:    cmpb    #15,(r0)                        ; Zap the rest of the
        beq     20$                             ;  default message.
        clrb    (r0)+                           ;
        br      10$                             ;
20$:    tst     usrppn                          ; Is a PPN assigned?
        beq     40$                             ; No. Skip to the logicals.
        mov     #defppn,r0                      ; PPN address.
        mov     usrppn,r1                       ; Get the PPN.
        call    fmtppn                          ; Format it into the buffer.
30$:    cmpb    #15,(r0)                        ; Zap the rest
        beq     40$                             ;  of the default
        clrb    (r0)+                           ;  message.
        br      30$                             ;
40$:    mov     #4,r4                           ; Four logoicals in table.
        cmp     #177777,usrlog+30               ; Is there a PPN in use?
        bne     50$                             ; Nope.
        dec     r4                              ; Yep. That means only 3.
50$:    mov     #usrlog,r2                      ; Start of logical table.
        mov     #usrlog+32,r3                   ; Start of PPN's.
        mov     #logicl,r0                      ; Start of output.
60$:    tst     (r2)                            ; Is the current one used?
        beq     70$                             ; Nope.
        call    fmtlog                          ; Yes so format it.
70$:    add     #8.,r2                          ; No. Bump the pointer.
        tst     (r3)+                           ; And the PPN pointer.
        sob     r4,60$                          ; Next one.
        cmp     #logicl,r0                      ; Did we do any logicals?
        bne     80$                             ; Nope.
        mov     #defend,r0                      ; Yep.
80$:    movb    #12,(r0)+                       ; Final <LF>.
        call    clrxrb                          ; Clear the XRB.
        mov     #headng,xrb+xrloc               ; Where it starts.
        sub     #headng,r0                      ; Calculate the length.
        mov     r0,xrb+xrlen                    ; Number of bytes to
        mov     r0,xrb+xrbc                     ;  write (twice).
        .write                                  ; Write it.

        exit$s                                  ; That's all folks


        .page
        .sbttl  Local Subroutines

;
;       Clear the XRB.
;

clrxrb:                                         ; Clear the XRB.
        push    r0                              ; We may need
```



It's easy to interface your ½" drive to a DEC computer. When you have connections.

Dataram provides tape drive connections to your host LSI-11, PDP-11, or VAX computer, with a family of couplers/controllers that operate in NRZI, PE, or GCR modes. Dataram's couplers/controllers operate with ½" tape drives from all major manufacturers. As slow as 25 ips — or as fast as 125 ips. 200 BPI to 6250 BPI. With TM11 and TS11 emulations.

Start-stop or streaming. Efficient streaming is supported by a unique RSX-11M utility, FASTSAVE-11M, which provides optional backup and save capability for Dataram's streamer coupler. A full one-year warranty is standard.

For more information about ½" drive connections, call (609) 799-0071. We'll help you make the connection you need!

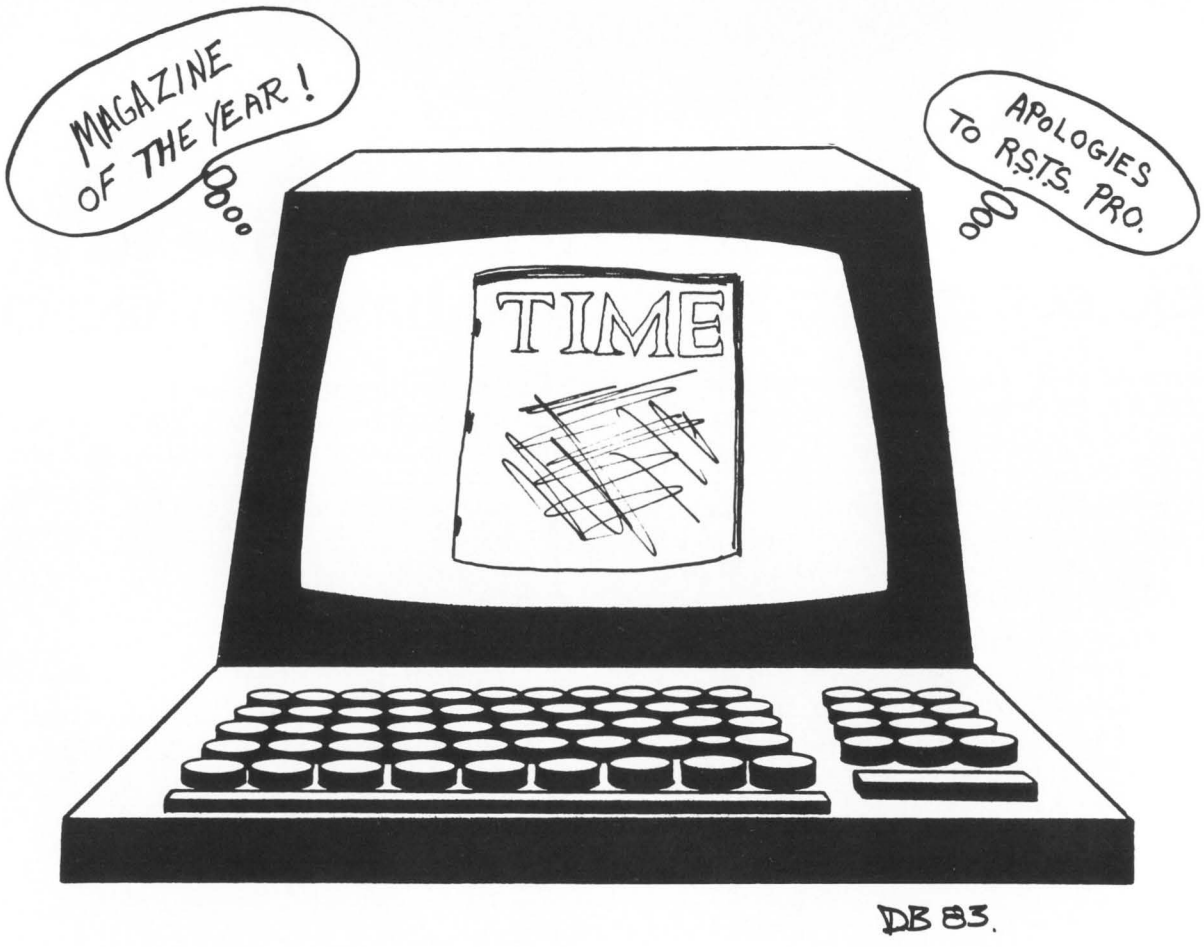
STANDARD AND STREAMER		GCR
AMPEX	KENNEDY	STC
CIPHER	PERTEC	TELEX
CDC	S. E. LABS	
DATUM	TANDBERG	
DIGI-DATA	TDX	

Dataram Corporation. Princeton Road. Cranbury, NJ 08512. (609) 799-0071.

DATARAM

CIRCLE 55 ON READER CARD

```
        push    r1                      ;  these later.
        mov     #xrb,r0                 ; Address of the XRB.
        mov     #xrbsiz,r1              ; The length in bytes.
10$:    clrb    (r0)+                   ; Clear this byte.
        sob     r1,10$                  ; Until we're done.
        pop     r1                      ; Get them back.
        pop     r0                      ;
        return                          ; Return to the caller.

;
;       Format the current entry in the logical table
;       (pointed to by r2) in to the outout buffer (pointed to
;       by r0).
;
fmtlog: push    r2                      ;Save R2.
        push    r0                      ; And R0.
        mov     #32.,r1                 ; Insert 32
10$:    movb    #40,(r0)+               ;  spaces
        sob     r1,10$                  ;  here.
        sub     #11.,r0                 ; Back up 12 spaces.
        mov     (r2)+,r1                ; First half of name.
        radtoa  r1,r0                   ; Convert to ASCII.
        mov     (r2)+,r1                ; Second half of name.
        radtoa  r1,r0                   ; Convert to ASCII.
20$:    cmpb    #40,-(r0)               ; Back up past the spaces
        beq     20$                     ;  (to float the colon).
        inc     r0                      ;
        movb    #':,(r0)                ; Insert the colon.
        pop     r0                      ; Restore R0.
        add     #32.,r0                 ; Re-adjust the pointer.
        tst     (r2)                    ; Is there a physical name?
        bne     40$                     ; Yep.
30$:    cmp     (r2)+,(r2)+             ; Nope. Bump the pointer.
        br      60$                     ;
40$:    movb    (r2)+,(r0)+             ; First character.
        movb    (r2)+,(r0)+             ; Second character.
        bit     #177400,(r2)            ; Is there a unit number?
        beq     50$                     ; No.
        bic     #177400,(r2)            ; Zap the flag.
        num$    (r2),r0                 ; Stick it in the buffer.
        bis     #177400,(r2)            ; Un-zap the flag.
50$:    tst     (r2)+                   ; Bump it.
        movb    #':,(r0)+               ; Insert a colon.
60$:    cmp     #177777,usrlog+30       ; PPN's present?
        bne     70$                     ; Nope.
        mov     (r3),r1                 ; Yep.
        beq     70$                     ; Not this one though.
        call    fmtppn                  ; Format it.
70$:    movb    #15,(r0)+               ; <CR>.
        movb    #12,(r0)+               ; <LF>.
        pop     r2                      ; Get R2 back.
        return                          ; Back to the caller.

;
;       Format the word in R1 as a PPN ([nnn,nnn]) starting
;       where R0 is pointing.
;
fmtppn:                                 ; Format the number in R1 as a
        push    r1                      ;  PPN where r0 points.
        movb    #'[,(r0)+               ; [
        swab    r1                      ; Clear the
        bic     #177400,r1              ;       project number.
        num$    r1,r0                   ; [nnn
        movb    #',,(r0)+               ; [nnn,
        pop     r1                      ; Get the original back.
        bic     #177400,r1              ; Clear the programmer number.
        num$    r1,r0                   ; [nnn,nnn
        movb    #'],(r0)+               ; [nnn,nnn]
        return                          ; That's it.


        .page
        .sbttl  Data Area.

headng: .byte   15,12,12
        .ascii  /LSTLOG V/
        .byte   sysvel/400&377, sysvee&377
        .byte   sysvee/400&377, '-
        .byte   $$$ver&377, $$$ver/400&377
        .ascii  / User assignments listing/
        .byte   15,12,12

        .ascii  /Default protection:  </
defpro: .ascii  /none>/
        .byte   15,12

        .asciz  /User assignable PPN: /
defppn: .ascii  /<none>   /
        .byte   15,12,12

        .ascii  /User logical table:  Logical    Physical/
        .byte   15,12                                       ; CRLF.
        .ascii  /                     -------    --------/
        .byte   15,12                                       ; CRLF.

logicl: .ascii  /                          <none>/
        .byte   15,12                                       ; CRLF.

defend: .blkb0  < 512. - < . - headng >>        ; The rest of 512. bytes.

        size    = . - lstlog                    ; Total size of code.

        .end    lstlog
```



DN11 AUTODIALER 'ENABLE' PATCH FOR INIT.SYS

By Jim Hobbs, MIS #410, Adolph Coors Company, Golden, CO

This patch enables the DN11 auto dialer. Without modification of INIT.SYS an error of 14 (device hung write locked) is given when trying to print to channel that DN was opened upon.

After editing (use TECO, or control file becomes useless) SYSGEN.CTL to include DNDRV.OBJ, DNTBL.MAC and DNCFG.MAC as specified, run SYSBAT as usual. After patching monitor (if selected) INSTALL your newly created SIL. Ignore the 'Warning — no set up for device DN'. Use DEFAULT to establish your normal defaults.

Now at OPTION specify PATCH. The following is how we have 'enabled' the DN11 for versions 7.0 and 7.2 (we never tried to implement 7.1).

```
Option: PATCH
File to patch? INIT.SYS
Base address? XN.PKT
Offset address? 10
 Base   Offset  Old     New?
046676  000010  072406  ? ^Z
Offset address? ^Z
Base address? 072406
Offset address? 42
 Base   Offset  Old     New?
072406  000042  073446  ? ^Z
Offset address? 22
 Base   Offset  Old     New?
072406  000022  012722  ? 012602
072406  000024  000004  ? 012712
072406  000026  012602  ? 000114
072406  000030  012712  ? 052712
072406  000032  000114  ? 002001
072406  000034  052712  ? 004536
072406  000036  002001  ? 073446
072406  000040  004536  ? 005012
072406  000042  073446  ? 000207
072406  000044  000207  ? <LF>
072406  000046  012762  ? <LF>
072406  000050  177777  ? ^Z
Offset address? ^Z
Base address? ^Z
File to patch? ^Z
```

The critical information used for both version's modification is:

FILE of INIT.SYS
BASE of XN.PKT
OFFSET of 10
"OLD" of BASE and OFFSET
OFFSET of 22
REPLACEMENT of OFFSET 36 with value of OFFSET 42

After patch is installed, boot system normally — auto dialer should now work.

If patch is installed incorrectly (or perhaps even as stated although doubtful), system may not boot due to 'initialization error'. Mount SYSGEN distribution, boot from it and COPY using SUPERCEDE YES for INIT.SYS file only. If other prior modifications of INIT.SYS have been made you may want to copy the existing file to another disk or tape.

♥

STRTUP/SPAWN

By Steve Roy, Bloomfield, CT

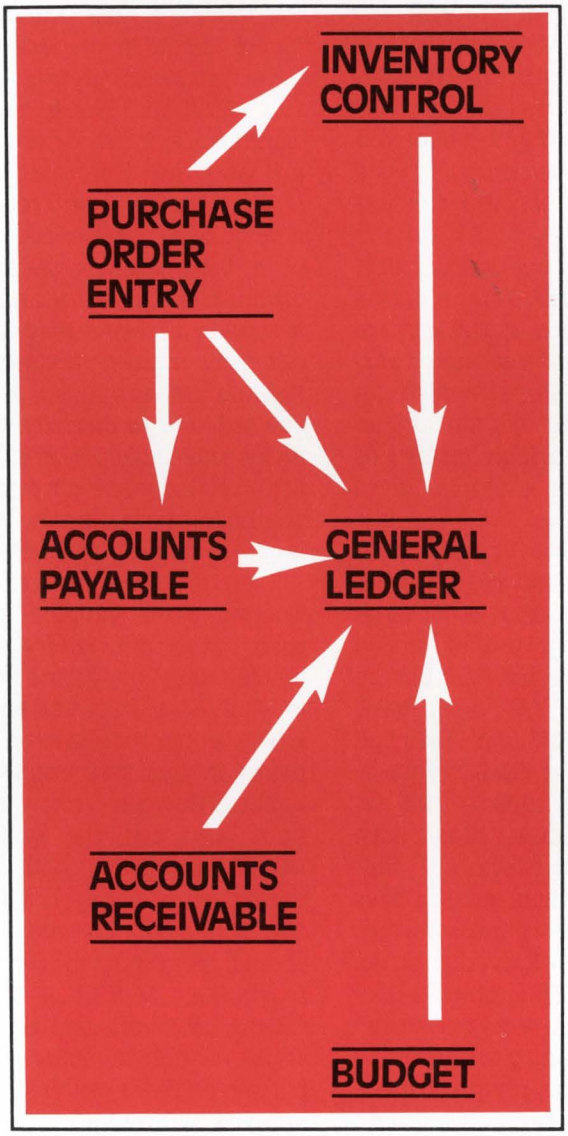
```
10      EXTEND
100     ! THIS PROGRAM IS THE CONTROLLING HALF OF A JOB STARTUP TEAM.
        ! IT CAN BE INVOKED "INLINE" OR "CHAINED-TO" BY ANY USER
        ! (DEPENDING ON ITS LOCATION AND PROTECTION CODE). THIS
        ! PROGRAM CREATES A JOB, AND THEN UTILIZES THE "SPAWN" PROGRAM
        ! TO FINISH THE CREATION PROCESS, AND CHAIN TO A USER PROGRAM.
        ! TYPICALLY, THIS IS USEFUL FOR STARTING JOBS THAT DETACH, AND
        ! SAVES HAVING TO ALLOW NON-PRIV USERS ACCESS TO PRIV ACCOUNTS.
        ! ATPK CAN PROVIDE THIS STARTUP CAPABILITY FROM PRIV ACCOUNTS,
        ! BUT SINCE THE "$LOGIN" COMMAND TO ATPK IS DISABLED FOR
        ! NON-PRIV USERS, THEY CANNOT STARTUP JOBS OTHER THAN IN THEIR
        ! OWN ACCOUNT.  FURTHERMORE, IF THE STARTUP OF A SPOOLER, DATA-
        ! BASE MANAGER, ETC. IS TO BE A MENU-DRIVEN CAPABILITY FOR ONE
        ! OR MORE NON-PRIV USERS, THE MENU MUST BE RE-ENTERED AFTER
        ! COMPLETION OF A STARTUP.  THIS IS IMPOSSIBLE WITH ATPK, SINCE
        ! IT DIES WITH A "?JOB LOGGED OUT" ERROR WHEN THE CONTROLLED
        ! JOB DETACHES FOR NON-PRIV USERS.

1000    ON ERROR GOTO 19000%
\       TRAP$=CHR$(6%)+CHR$(-7%)                ! CONTROL/C TRAP
\       T$=SYS(TRAP$)
\       PK%=0%                                  ! INITIAL PK NUMBER
\       PRINT "Database Manager Startup & Security Module"
\       PRINT
\       PRINT " ***** Whatever messages you want go here ******"
\       PRINT
\       PRINT "[RETURN] to continue, [^C] to goto MENU...";
\       INPUT T$
        !
        ! TYPICALLY, YOU WOULD HAVE SOME CODE HERE TO VERIFY THAT THE
        ! USER INVOKING THIS PROGRAM IS, IN FACT, THE USER YOU WANT TO
        ! BE DOING SO.  PERHAPS PRINT A RANDOM SEED, AND REQUIRE A
        ! RESPONSE BASED ON THE SEED (NUMBER, LETTERS, ETC).
        ! FAILED RESPONSES SHOULD BE LOGGED IN A FILE, ALONG WITH KB#,
        ! PPN, TIME, AND DATE. THEN, A SIMPLE GOTO 32000% WILL RETURN
        ! TO THE MENU.
\       GOSUB 10000%                            ! GET A PSEUDO-KEYBOARD
        !
        ! IDENTIFY THE KB THAT IS ATTACHED TO THE PK THAT WE GOT...
\       K%=ASCII(MID(SYS(CHR$(6%)+CHR$(-8%)+CHR$(3%)),11%,1%))/2%
        !
        ! THE NEXT TWO FILENAME-STRING-SCANS REFER TO THE "SPAWN"
        ! PROGRAM, AND THE PROGRAM TO WHICH "SPAWN" MUST CHAIN.
\       T1$=SYS(CHR$(6%)+CHR$(-23%)+"PROG:SPAWN")
\       T2$=SYS(CHR$(6%)+CHR$(-23%)+"PROG:MANAGE")
        !
        ! NEXT, CREATE A JOB (WHICH WILL RUN "SPAWN") AND TELL IT WHAT
        ! IT NEEDS TO KNOW TO START AND CHAIN.
\       T$=SYS(CHR$(6%)+CHR$(24%)+CVT%$(0%)+    ! CREATE A JOB
               MID(T1$,5%,6%)+                  ! AND START "SPAWN"
               CVT%$(-1%)+                      ! ANY EXTENSION
               MID(T2$,5%,6%)+                  ! SPAWN CHAINS "MANAGE"
               CHR$(111%)+CHR$(1%)+             ! PPN TO LOGIN UNDER
               CHR$(K%)+                        ! KB (PK) TO ATTACH
               CHR$(0%) )                       ! SPARE BYTE

1020    ! THIS IS WHERE WE WAIT FOR "SPAWN" TO CREATE A JOB.
        ! ADDITIONALLY, THIS WILL ECHO ANYTHING PRINTED BY THE CHAINED
        ! PROGRAM, ALERTING US TO ANY ERRORS ON ITS PART.
        SLEEP 2%                                ! WAIT FOR PK TO WORK
\       GET #3%                                 ! GET SOME STUFF
\       R%=RECOUNT                              ! SAVE SIZE
\       FIELD #3%, R% AS R$                     ! MAP IT
\       PRINT R$;                               ! AND ECHO LOCALLY
\       GOTO 1020%

1030    ! HERE WE CHECK TO SEE IF THE CONTROLLED JOB HAS DETACHED,
        ! IS BUSY WORKING, OR IF IT HAS RETURNED TO A "^C" STATE.
        PUT #3%, RECORD 4%, COUNT 1%
        ! IF NO ERROR IS GENERATED, THE JOB HAS GONE TO "^C" STATE...

1040    ! UPON CLOSING THE PK, ANY CONTROLLED JOB THAT HASN'T DETACHED
        ! GETS KILLED...
        CLOSE 3%                                ! ALL DONE W/PK
\       PRINT
\       PRINT "Returning to MENU..."
\       SLEEP 2%                                ! WAIT FOR THEM TO READ
\       GOTO 32000%

10000   ! HERE WE OPEN A PSEUDO-KEYBOARD.
        OPEN "PK"+NUM1$(PK%)+":" AS FILE 3%
\       RETURN                                  ! SUCCESSFUL OPEN

19000   GOTO 32000% IF ERR=28%                  ! CTRL/C
              OR ERR=11% AND ERL=1000%          ! CTRL/Z FROM KEYBOARD
\       RESUME 1030% IF ERR=11%                 ! PK: IS TOO SLOW
\       GOTO 19020% UNLESS ERR=8% AND ERL=10000%        ! PK IN-USE ?
\       PK%=PK%+1%                              ! STEP TO NEXT PK
\       RESUME 10000%
19020   RESUME 19030% IF ERR=6% OR ERR=4%       ! NO SPACE ON SYSTEM
\       RESUME 1040% IF ERL=1030% AND ERR=5%    ! CONTROLLED JOB GONE
\       RESUME 1020% IF ERL=1030% AND ERR=3%    ! CONTROLLED JOB BUSY
\       PRINT "Unexpected Error #";ERR;"at line";ERL;'in "STRTUP".'
\       PRINT "Returning to menu..."
\       SLEEP 4% \ GOTO 32000%
19030   PRINT "No room on system for STRTUP - Try again later..."
\       SLEEP 4%

32000   CHAIN "PROG:MENU"

32767   END
```

```
10      EXTEND
100     ! THIS PROGRAM IS USED IN CONJUNCTION WITH A CONTROLLING        &
        ! JOB THAT IS ATTEMPTING TO SPAWN A JOB AT A PSEUDO-KEYBOARD.   &
        ! TO SAVE THE CONTROLLING JOB FROM HAVING TO FORCE A LOGIN      &
        ! STRING (WITH ACCOUNT,PASSWORD,AND PROGRAM TO HOOK TO), THIS   &
        ! PROGRAM PERFORMS 1)ATTACH TO PSEUDO KEYBOARD, 2)LOGIN TO      &
        ! AN ACCOUNT (RATHER THAN RUNNING LOGGED-OUT), 3)CHAIN TO A     &
        ! PROGRAM TO BE RUN.  THIS PROGRAM SHOULD BE INVOKED THROUGH    &
        ! THE "CREATE-A-JOB" SYS CALL.  THE "ATPK" PROGRAM USES THE     &
        ! SPAWNING ENTRY (29000) OF "LOGIN" TO PERFORM THESE FUNCTIONS.

1000    ON ERROR GOTO 19000%                                           &
\       FIP$=CHR$(6%) \ ZO$=CHR$(0%) \ TWO.0$=CVT%$(0%) \ M1$=CHR$(-1%) &
\       CORE$=SYS(CHR$(7%))             ! GET CORE COMMON              &
        ! CORE-COMMON CONTAINS:                                        &
        !       1-2     PPN CONTAINING PROGRAM TO CHAIN TO             &
        !       3-6     PROGRAM TO CHAIN TO (RAD-50 FORMAT)            &
        !       7-8     PPN TO LOGIN UNDER                             &
        !       9       KB TO ATTACH TO                                &
        !       10      SPARE (ONLY SPARE BYTE)                        &
\       I%=ASCII(SYS(FIP$+CHR$(9%)))/2%         ! GET OUR JOB NUMBER  &
        !                                                              &
        ! BEING NOTHING MORE THAN A TWINKLE IN THE EYES OF RSTS, WE     &
        ! MUST MAKE OURSELVES HEARD... FIRST, WE ATTACH TO THE TERMINAL &
        ! SIDE OF THE PSEUDO-KEYBOARD THAT THE CONTROLLING JOB OWNS...  &
\       T$=SYS(FIP$+FIP$+CHR$(I%)+MID(CORE$,9%,1%))     ! (RE)ATTACH  &
        !                                                              &
        ! NOW WE HAVE A KEYBOARD, WE CAN (OPTIONALLY) SAY SOMETHING...  &
\       PRINT "Commencing Startup Sequence..."                         &
        !                                                              &
        ! NOW, WE HAVE TO SETUP OUR SCHEDULING PARAMETERS...            &
\       T$=SYS(FIP$+CHR$(-13%)+M1$+                                    &
                   M1$+CHR$(-8%)+           ! PRIORITY                 &
                   M1$+CHR$(6%)+            ! RUNBURST                 &
                   M1$+CHR$(31%))           ! SIZE-MAX                 &
        !                                                              &
        ! NEXT... DETERMINE WHICH ACCOUNT WE ARE SUPPOSED TO LOGIN,     &
        ! AND THEN PERFORM THE LOGIN (GIVING THE PASSWORD TO DO SO)...  &
\       PPN$=MID(CORE$,7%,2%)                                          &
\       T$=SYS(FIP$+CHR$(4%)+TWO.0$+             ! LOGIN:              &
               MID(SYS(FIP$+CHR$(14%)+TWO.0$+          ! GET PASSWORD  &
               TWO.0$+PPN$+CHR$(1%)),7%,6%))           ! GIVE IT OVER  &
        !                                                              &
        ! NEXT, BUILD THE NAME OF THE PROGRAM TO CHAIN...               &
\       T$="("+NUM1$(ASCII(MID(CORE$,2%,1%)))+          ! PROJECT #    &
               ","+NUM1$(ASCII(CORE$))+")"+             ! PROGRAMMER # &
               RAD$(SWAP%(CVT$%(MID(CORE$,3%,2%))))+    ! PROGRAM 1-3  &
               RAD$(SWAP%(CVT$%(MID(CORE$,5%,2%))))     ! PROGRAM 4-6  &
        !                                                              &
        ! WELL, IT HAS BEEN NICE, BUT WE'VE GOT TO GO... BUSY, BUSY...  &
\       CHAIN T$                                 ! SPAWN COMPLETE

19000   PRINT "Startup error";ERR;"at line";ERL;'in "SPAWN".'          &
        ! ERRORS WILL RETURN US TO A "^C" STATE, AND WE'LL GET KILLED   &
        ! WHEN THE CONTROLLING JOB CLOSES THE PSEUDO-KEYBOARD.

32767   END
```

VACS

VAX ACCOUNTING-COBOL SYSTEM

HAMILTON'S COMPLETE FINANCIAL ACCOUNTING PACKAGE



GENERAL LEDGER is a powerful and comprehensive accounting tool. It is completely flexible in design, lending itself to a wide variety of applications.

ACCOUNTS PAYABLE is a sophisticated online interactive subsystem designed to meet management's need for complete control over vendor payments, discounts and cash commitments.

PURCHASE ORDER ENTRY encompasses all the necessary functions, including encumbrance accounting to create, maintain, issue and properly track purchase orders.

INVENTORY CONTROL is designed to maintain the inventory data base and generate various management and control reports, reflecting product status, replenishment requirements, transfers and costing / pricing information.

ACCOUNTS RECEIVABLE insures proper and timely collections of funds, leading to improved and more responsible fiscal management. Detail customer information is readily available online and cash may be applied to customer accounts daily or as needed.

BUDGET is designed to complement General Ledger processing. Budget transactions using a global transaction code have the capability of affecting a vast number of accounts. The global concept enables the user to create, maintain and modify budget data that would otherwise be difficult and time consuming.

RENTAL, SALE OR TIMESHARING of VACS system, available complete or by module, with or without VAX hardware.

VAX is a registered trademark of the Digital Equipment Corporation.

HAMILTON THE HOUSE OF VAX

HGL Software Hamilton Rentals

6 Pearl Court, Allendale, NJ 07401
TOLL FREE 800-631-0298
In New Jersey 201-327-1444

415 Horner Ave, Toronto, M8W4W3
TOLL FREE Ont. & Que. 800-268-2106
All other prov. 800-268-0317
In Canada 416-251-1166

NEW YORK • DALLAS • MONTREAL • CALGARY • LONDON • PARIS • DUSSELDORF

CIRCLE 10 ON READER CARD

BASIC-PLUS-2 PROGRAMS

By Kelvin Smith, Lawrence University (class of 1984), Appleton, WI

Lawrence University is in the process of putting all administrative functions onto RSTS. Programming is done almost entirely in BASIC-PLUS-2, to make use of RMS. We have recently replaced our venerable PDP 11/45 with an 11/70, which should, for a while at least, eliminate swapping (512K words memory instead of 128K), but while that restraint on size has been taken care of, the magic 31K user job maximum when running under the RSX emulator remains fully in effect. Over the past few years, we have learned quite a bit about how space is used in BP2 programs. Particularly when using the BP2COM or disappearing RSX run-time systems, everything counts to the utmost. By getting maps from the task-builder, you can see just how much particular language elements are saving or costing, in terms of program size.

Based on what we've found, here are some suggestions on how to put your programs on a reducing plan. Note: this article is based on the state of BASIC-PLUS-2 under language version 1.6. Version 2.0 is scheduled to come out soon, which may change some of the details given, although the general principles should remain the same.

(1) Eliminate wasteful functions. By "functions" I mean not user-defined functions, although those too should be used with care (GO-SUBs operate faster and use less space), but standard features of the BP2 language. Many features of the language are normally unnecessary and are very costly in terms of size. Perhaps the best example is string arithmetic. This prehistoric monster calls in almost 1800 bytes worth of code, so making it disappear would save nearly 1K of memory (and the function operates so slowly that you'll probably speed up the program as well by avoiding it). Another function is the SYS call package. Any SYS call brings in the entire package, some 1600 bytes worth. It's much better to use the functions CTRLC (CTRL/C trap), RCTRLO (reset CTRL/O), NOECHO and ECHO (disable and enable terminal echoing), FSS$ (file string scan), and ERT$ (error text), if those are all that you need, since they pull in only the code necessary for their operation. The set consisting of READ, INPUT, LINPUT, and INPUT LINE calls in 1002 bytes of code, which might induce you to put your input statements all in one subprogram overlay (or you can use GET and MOVE to avoid it). All the other functions have size costs, too; it might be a good idea, for programs that are tight on space, to run a cross-reference including references to functions, to see if there are any which can be easily eliminated (using an intermediate integer variable instead of the INT function, for example, saves almost 200 bytes).

(2) Use RMS intelligently. The Record Management Services system is a fantastic help for operation of large data bases, but poor use of it can waste time and space for everyone on the system. One of the most basic guidelines is to use RMS only when you really need it: unless RMS transportability is necessary, replace RMS sequential and relative files with terminal-format and block I/O files (this applies even when indexed files are used). For the more adventurous, writing your own RMS overlay structure is a possibility. This can save considerable space if you aren't using all the features of RMS and you don't have (or don't have enough program space to use) the RMSRES resident library—one overlay structure which I wrote, leaving in just the read code for indexed files, not only saved about 1K of memory, but also cut 70 blocks off the task file size. Note that you will have to change the ODL file for each program to call in your new RMS overlay descriptor file (DEC passes out a prototype, called RMS11.ODL, which allows you to go through step by step, picking out what tidbits of RMS you want). Note also that for unexplained reasons, you may get diagnostic "undefined segment" warning messages during your taskbuild. If you didn't get them during taskbuilds with the standard RMS overlay descriptor, ignore them: in my work, they have never produced run-time problems.

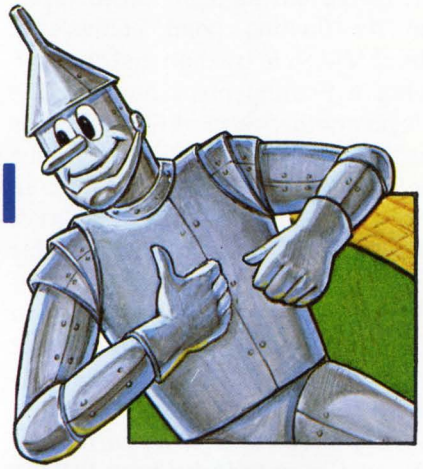
(3) All of these pale into insignificance, for those not blessed with a floating point processor, in the light of the floating point routines (value: almost 2000 bytes of code). Getting rid of all floating point operations from a program is both easier and harder than it sounds: it seems easy until you find out how many functions call in the floating point routines, at which time you're likely to despair. Therefore, I'll divide this section into two parts, one listing functions which refer to floating point operations, and the second detailing how to get around them.

(a) Of course, if you actually have a floating point variable in your program, either get rid of it or forget the whole thing — one variable, depending on your use of it, can be enough for the whole set of routines to be called in and placed in your task image by the task builder. However, many other operations also call the set: FORMAT$, NUM$ and NUM1$, ABS, and VAL are ones that you may not have known about. Also, the ERR and ERL variables should be compared with explicit integers (i.e., "IF ERR = 28%," not "IF ERR = 28"), because the compiler isn't smart enough to realize that any comparison on them would have to be with integers. (However, for other line-oriented operations—GOTO, GOSUB, RESUME — a percent sign is unnecessary, since it is compiled as referring to a statement label.)

(b) Fortunately, help is available. ABS and VAL are the easiest to deal with — it isn't documented anywhere except on the "reserved keywords list," but we have discovered that integer versions of the two functions exist under the names of ABS% and VAL%.

PERSONAL and PROFESSIONAL

Produced for the Digital Personal and Professional Computer User



Stop and See Us At Booth E

Each issue of PERSONAL and PROFESSIONAL is packed with the latest information on the world of personal computers in a high-quality, modern, attractive and easy-to-read format. The finest editors, writers, art directors, illustrators, cartoonists and photographers have been assembled to establish a new level of excellence and objectivity in personal computer magazines.

Simply stated, PERSONAL and PROFESSIONAL is the most substantial source on the Digital personal computer user's regular reading list. It's as if a team of computer experts came to your office or home every issue.

PERSONAL and PROFESSIONAL is an independent magazine, not sponsored or approved by or connected in any way with Digital Equipment Corporation.

Charter Subscription Offer

Save More Than 50% Off Cover Price

You can now become a Charter Subscriber to PERSONAL and PROFESSIONAL at absolutely No Risk.

Simply enter your subscription with payment enclosed and Save Over 50% Off The Cover Price. If not satisfied after receiving the first issue, merely return your mailing label within 15 days for a Full Refund. Offer good until July 1, 1983, and only applies to the US and Canada.

SO ACT NOW. Become a PERSONAL and PROFESSIONAL Charter Subscriber today!

NAME _____
(please print full name)

ADDRESS _____ APT. _____

CITY _____ STATE _____ ZIPCODE _____

☐ Bill me $28 for 12 issues, a 33% savings off the cover price. Offer good until July 1, 1983. Rates are double outside the US and Canada.

☐ Payment enclosed. YES, I want to take advantage of your special Charter Subscription Offer. Send me the first 12 issues of PERSONAL and PROFESSIONAL for only $20 (please remit in US currency), saving me over 50% off the cover price. Offer good until July 1, 1983. Charter Rate only applies to the US and Canada.

PERSONAL and PROFESSIONAL

P.O. BOX 114, SPRINGHOUSE, PA 19477-0114
215/542-7008

CIRCLE 177 ON READER CARD

(Note: these exist only in BP2, not in BASIC-PLUS.) Using these will avoid floating point operations, saving you both space and time (on our machine without a floating point processor, each tested out as eight times faster than its floating point equivalent). Note: if VAL% is used on a string containing a floating point number, the "?Illegal number" error (ERR = 52%) is returned. NUM$ (and NUM1$, which calls in identical routines) is harder to avoid, but again there are possibilities; a straight PRINT of an integer variable doesn't call in floating point, and if NUM$ is the only thing keeping you from getting rid of floating point operations, it isn't hard to write a user-defined function to replace them (and you may even be able to write one which will operate faster than NUM$). FORMAT$ sometimes may be unavoidable, but at other times a direct print with PRINT USING will do the trick (PRINT USING does not call the floating point routines, except for the basic move operator when a floating point number is printed FORMAT$ uses everything—add, subtract, multiply, divide). Or, you could simply (or not-so-simply) recode the program to avoid formatted output completely, which would cut out about 1800 bytes. With both NUM$ and FORMAT$, you'll have to decide whether the space saved is worth the time spent in recoding your program, although you can probably keep the user-defined function as a library function, for use in any program which needs it. Of course, the elements of the language are there to be used, and there will certainly be times where ease of coding will take precedence over size of the program. In addition, if you use the BASIC 2 runtime system, everything is already included, so there's no reason to go to the above-mentioned hassles (with the possible exception of ABS% and VAL%, which there is really no reason not to use). However, if you're trying to get the last few hundred bytes cut out to sneak a program under the 31K wire, these ideas, I think, can be helpful.

1. I want to thank Prof. James Evans, Gary Van Den Boom, and Tom Schmitz for their aid in the preliminary research for this article. ♥

DEAR RSTS MAN:

Send questions to: DEAR RSTS MAN, P.O. Box 361, Fort Washington, PA 19034-0361.

DEAR RSTS MAN:

We are having a problem putting together a workable disaster recovery plan. We have a central data center in Columbus, Ohio that contains, among other things, one VAX 11/780, seven PDP 11/70's with three RP06's and one PDP 11/04 HASP Box per 11/70, and ten PDP 11/34's that are being used as communication nodes. Our central data center is linked to four remote centers around the country through our network of about 25,000 miles of communication lines. All of the machines are now, or will soon be, at full capacity. This includes computers that are located at our remote sites.

The problem that we are faced with is that of locating a cost effective backup facility that could be used in the event that we get hit with the "BIG DISASTER." Any suggestions?

W. Chisholm, Manager,
Systems Assurance

Any suggestions?

DEAR RSTS MAN:

We here at ABS are quite proud of the fact that we use antique hardware (we have 4 PDP-11/40s) and software. We really get the mileage out of this stuff, but every so often we do find value in upgrading to newer technology.

For this reason, I felt it was time to investigate DEC's SORT-11 package as a replacement for the "XQWIK" family. True, SORT-11 does not run as fast as some commercially available products, but it sure beats "XQWIK, et al", and it does have a rather complete record selection process. Bundled with the fact that it comes "free with the purchase of each RSTS" (our favorite feature), it appeared to be the logical step for us.

I have managed to make our "Type 2" block I/O files look like RMS sequential fixed-length record files by writing proper attributes. I've even managed to make the sort and selection process work on ASCII (Key type "C") fields. Here is my problem:

Apparently, SORT-11's Binary (Key type "B") fields are not equivalent to XQWIK's Integer (Key type "I") fields. Also, SORT-11's Floating-Point (Key type "F") does not match XQWIK's Floating-Point (Key type "F") format. Worse than that, DEC provided no upward compatibility to SORT-11 for data stored in CVT%$ or CVTF$ format.

I would like to know if there is any way to get SORT-11 to properly recognize data in CVT%$ and CVTF$ format? Are there undocumented key types? Has anybody devised a SORT-11 patch? Will DEC upgrade SORT-11 for these field types?

I hope there is a cure for this that can be made directly to SORT-11, in that pre and post file processors will defeat the purpose of using SORT-11 in the first place. Thanx for your help, and for the help of all the RSTS pros who are reading!

Bob Ashcraft
Applied Business Services

Dear Bob: *I believe there was a DECUS library contribution that connected older file formats to SORT-11.*

P.S. We're busy setting up our fourth & fifth '40's.

DEAR RSTS MAN:

Here at our company several of us have entertained a heated debate as to the effects of 9600 baud terminals on a RSTS/E system. Several articles and authorities on optimizing RSTS/E performance have stated that terminals running at high baud rates can seriously degrade overall system performance. The explanation given is that a terminal running at high speeds demands a very large share of total CPU time just to create the characters and output them to the terminal and also that the associated job requires a greater number of

. . . continued on page 70

SORT BENCHMARKS ON DEC 11/34 WITH RSTS/E

By Robert L. Besner, Dept. of National Defense, Ottawa, Ontario, Canada

Introduction

The benchmark of sorting algorithms was conducted to evaluate the performance of these sorts on a PDP 11/34 using the RSTS/E operating system. In the study, performance is defined as a minimum of computer facilities to produce a measured amount of throughput.

Three types of benchmarks were:

a. integers with BASIC-PLUS code,
b. integers with BASIC-PLUS-2 code,
c. string fields with BASIC-PLUS-2 code.

Five algorithms used were:

1. Bubble
2. Shell
3. Heap
4. Quick
5. Hart

Note that all benchmark jobs were executed on a quiet system (single user, under Batch).

Appendix A contains integer sort algorithms; Appendix B contains string data sort code.

2. Algorithms coded in BASIC-PLUS sorting integers

This section is a benchmark of all algorithms coded in BASIC-PLUS. The unsorted array contains integers varying from 0 to 99 randomly generated.

2.1 Bubble sort

```
RUN DM1:NBUBBL.BAC

This is a benchmark of the BUBBLE sort     24-Dec-81      02:31

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 142 KCT, 4.7 CPU SEC. AND  5 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 2652 KCT, 66.3 CPU SEC. AND  67 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 9632 KCT, 240.8 CPU SEC. AND  242 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY
```

2.2 Shell sort

```
RUN DM1:NSHELL.BAC

This is a benchmark of the SHELL sort      24-Dec-81      02:36

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 44 KCT, 1.1 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 272 KCT, 6.8 CPU SEC. AND  7 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 696 KCT, 17.4 CPU SEC. AND  18 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 2430 KCT, 48.6 CPU SEC. AND  49 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 5718 KCT, 95.3 CPU SEC. AND  96 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 10367 KCT, 148.1 CPU SEC. AND  151 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 13587 KCT, 194.1 CPU SEC. AND  195 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 24424 KCT, 305.3 CPU SEC. AND  306 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY
```

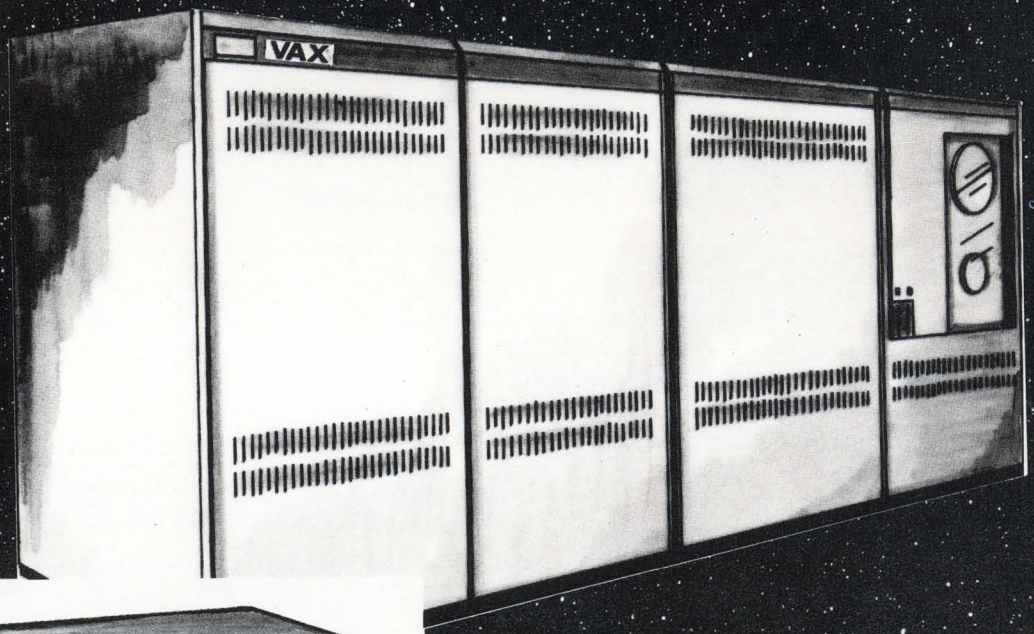
2.3 Heap sort

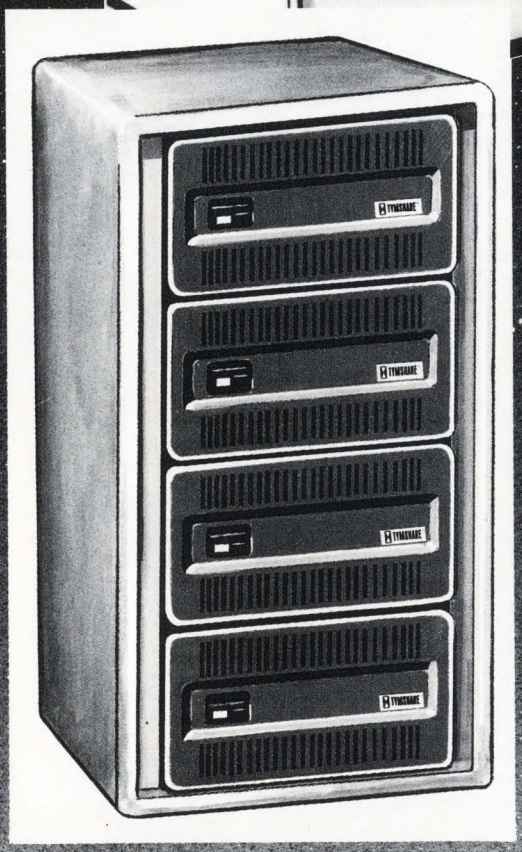
```
RUN DM1:NHEAP.BAC

This is a benchmark of the HEAP sort       24-Dec-81      02:50

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 40 KCT, 1.3 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 260 KCT, 6.5 CPU SEC. AND  7 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 576 KCT, 14.4 CPU SEC. AND  15 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 1600 KCT, 32 CPU SEC. AND  32 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 3096 KCT, 51.6 CPU SEC. AND  53 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 4921 KCT, 70.3 CPU SEC. AND  71 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 6321 KCT, 90.3 CPU SEC. AND  91 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 8936 KCT, 111.8 CPU SEC. AND  112 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY
```

2.4 Quick sort

```
RUN DM1:NQUICK.BAC

This is a benchmark of the QUICK sort      24-Dec-81      02:57

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 28 KCT, .7 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 140 KCT, 3.5 CPU SEC. AND  4 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 320 KCT, 8 CPU SEC. AND  8 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 1015 KCT, 20.3 CPU SEC. AND  21 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 2070 KCT, 34.5 CPU SEC. AND  35 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY
```

Voyager™ I from Tymshare
Your FUTURE Is NOW !





The future has a way of becoming the present. Tymshare has made the future NOW, in announcing the first in a family of DEC* compatible disk subsystems.

Tymshare Voyager Series disk subsystems offer the DEC computer user, increased disk storage capacity, reduced space and power requirements and attractive pricing alternatives.

Have you evaluated your present disk subsystem costs?

When it comes to experience and reputation Tymshare is second to none. When it comes to Price/Performance and Mixed Vendor Support, Tymshare is one step beyond . . .

TYMSHARE®

10100 Liberty St. · Tremont, Ca. 94538
(415) 794-2538

*DEC, DIGITAL and VAX are registered trademarks of Digital Equipment Corporation

CIRCLE 36 ON READER CARD

```
f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 3451 KCT, 49.3 CPU SEC. AND  50 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 5720 KCT, 71.5 CPU SEC. AND  72 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 7136 KCT, 89.2 CPU SEC. AND  90 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY
```

2.5 Hart sort

```
RUN DM1:NHART.BAC

This is a benchmark of the HART sort      24-Dec-81      03:02

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 28 KCT, .7 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 185 KCT, 3.7 CPU SEC. AND  4 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 410 KCT, 8.2 CPU SEC. AND  9 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 1239 KCT, 17.7 CPU SEC. AND  18 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 2296 KCT, 28.7 CPU SEC. AND  29 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 3820 KCT, 38.2 CPU SEC. AND  39 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 10 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 5736 KCT, 47.8 CPU SEC. AND  48 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 12 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 7982 KCT, 61.4 CPU SEC. AND  62 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 13 K OF MAIN MEMORY
```

2.6 Comparison of algorithms

	Bubble	Shell	Heap	Quick	Hart
number of statements	8	15	31	44	76
100 numbers	5 sec. (4k)	2 sec. (4k)	2 sec. (4k)	1 sec. (4k)	1 sec. (4k)
400 numbers	67 sec. (4k)	7 sec. (4k)	7 sec. (4k)	4 sec. (4k)	4 sec. (5k)
800 numbers	242 sec. (4k)	18 sec. (4k)	15 sec. (4k)	8 sec. (4k)	9 sec. (5k)
1600 numbers	-	49 sec. (5k)	32 sec. (5k)	21 sec. (5k)	18 sec. (7k)
2400 numbers	-	96 sec. (6k)	53 sec. (6k)	35 sec. (6k)	29 sec. (8k)
3200 numbers	-	151 sec. (7k)	71 sec. (7k)	50 sec. (7k)	39 sec. (10k)
4000 numbers	-	195 sec. (7k)	91 sec. (7k)	72 sec. (8k)	48 sec. (12k)
4800 numbers	-	306 sec. (8k)	112 sec. (8k)	90 sec. (8k)	62 sec. (13k)

3. Algorithms coded in BASIC-PLUS-2 sorting integers

This section is a benchmark of all algorithms coded in BASIC-PLUS-2. The unsorted array contains integers varying from 0 to 99 randomly generated.

3.1 Bubble sort

```
RUN DM1:NBUBBL

This is a benchmark of the BUBBLE sort     15-Dec-81      03:24

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 44 KCT, 1.1 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 725 KCT, 14.5 CPU SEC. AND  15 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 2865 KCT, 57.3 CPU SEC. AND  61 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY
```

3.2 Shell sort

```
RUN DM1:NSHELL
This is a benchmark of the SHELL sort      15-Dec-81      03:26

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 8 KCT, .2 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 65 KCT, 1.3 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 145 KCT, 2.9 CPU SEC. AND  3 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 510 KCT, 8.5 CPU SEC. AND  9 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 936 KCT, 15.6 CPU SEC. AND  16 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 1715 KCT, 24.5 CPU SEC. AND  25 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 2608 KCT, 32.6 CPU SEC. AND  34 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 5058 KCT, 56.2 CPU SEC. AND  61 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 9 K OF MAIN MEMORY
```

3.3 Heap sort

```
RUN DM1:NHEAP
This is a benchmark of the HEAP sort       15-Dec-81      03:28

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 12 KCT, .3 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 4 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 70 KCT, 1.4 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 160 KCT, 3.2 CPU SEC. AND  4 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 402 KCT, 6.7 CPU SEC. AND  7 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 728 KCT, 10.4 CPU SEC. AND  11 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 1085 KCT, 15.5 CPU SEC. AND  17 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 1464 KCT, 18.3 CPU SEC. AND  20 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 2088 KCT, 23.2 CPU SEC. AND  24 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 9 K OF MAIN MEMORY
```

3.4 Quick sort

```
RUN DM1:NQUICK
This is a benchmark of the QUICK sort      15-Dec-81      03:30

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 5 KCT, .1 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 35 KCT, .7 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY
```

```
c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 85 KCT, 1.7 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 258 KCT, 4.3 CPU SEC. AND  5 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 483 KCT, 6.9 CPU SEC. AND  7 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 707 KCT, 10.1 CPU SEC. AND  11 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 1072 KCT, 13.4 CPU SEC. AND  14 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 1422 KCT, 15.8 CPU SEC. AND  16 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 9 K OF MAIN MEMORY
```

3.5 Hart sort

```
RUN DM1:NHART

This is a benchmark of the HART sort      15-Dec-81     03:31

a) NUMBER OF SORTED ITEMS =  100 INTEGERS
   SORT USED 5 KCT, .1 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 INTEGERS
   SORT USED 40 KCT, .8 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 INTEGERS
   SORT USED 102 KCT, 1.7 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 INTEGERS
   SORT USED 280 KCT, 3.5 CPU SEC. AND  4 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 8 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 INTEGERS
   SORT USED 531 KCT, 5.9 CPU SEC. AND  6 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 9 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 INTEGERS
   SORT USED 825 KCT, 7.5 CPU SEC. AND  8 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 11 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 INTEGERS
   SORT USED 1092 KCT, 9.1 CPU SEC. AND  10 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 12 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 INTEGERS
   SORT USED 1792 KCT, 12.8 CPU SEC. AND  13 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 14 K OF MAIN MEMORY
```

3.6 Comparison of algorithms

	Bubble	Shell	Heap	Quick	Hart
number of statements	8	15	31	44	76
100 numbers	2 sec. (4k)	1 sec. (4k)	1 sec. (4k)	1 sec. (5k)	1 sec. (5k)
400 numbers	15 sec. (5k)	2 sec. (5k)	2 sec. (5k)	1 sec. (5k)	1 sec. (5k)
800 numbers	61 sec. (5k)	3 sec. (5k)	4 sec. (5k)	2 sec. (5k)	2 sec. (6k)
1600 numbers	-	9 sec. (6k)	7 sec. (6k)	5 sec. (6k)	4 sec. (8k)
2400 numbers	-	16 sec. (6k)	11 sec. (7k)	7 sec. (7k)	6 sec. (8k)
3200 numbers	-	25 sec. (7k)	17 sec. (7k)	11 sec. (7k)	8 sec. (11k)
4000 numbers	-	34 sec. (8k)	20 sec. (8k)	14 sec. (8k)	10 sec. (12k)
4800 numbers	-	61 sec. (9k)	24 sec. (9k)	16 sec. (9k)	13 sec. (14k)

Table 1 shows the elapsed time in seconds required by each algorithm to sort "X" number of integers.

Table 2 shows the amount of KCTs required by each algorithm to sort "X" number of integers. KCT (Kilo-core-ticks) is the memory usage factor per tenth of CPU seconds. One KCT is equal to the usage of 1K of memory for one tenth of a second. For example, a job that requires 10.5 CPU seconds and 15K of memory is using 1575 KCTs.

TABLE 1

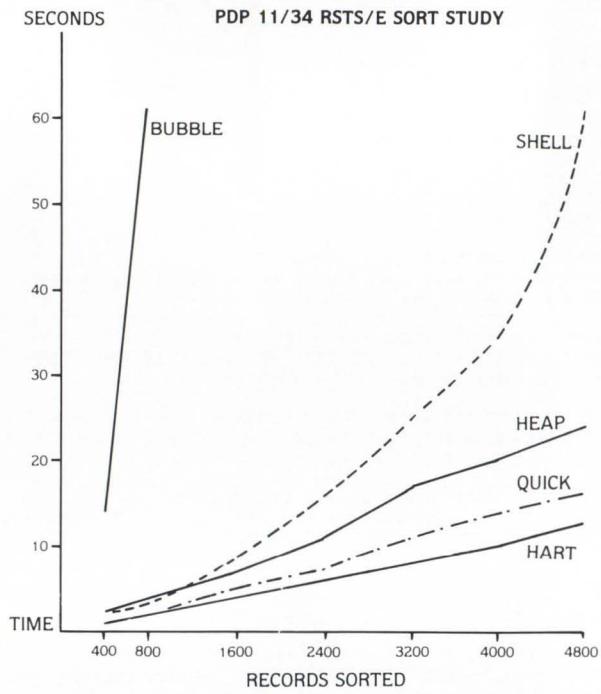
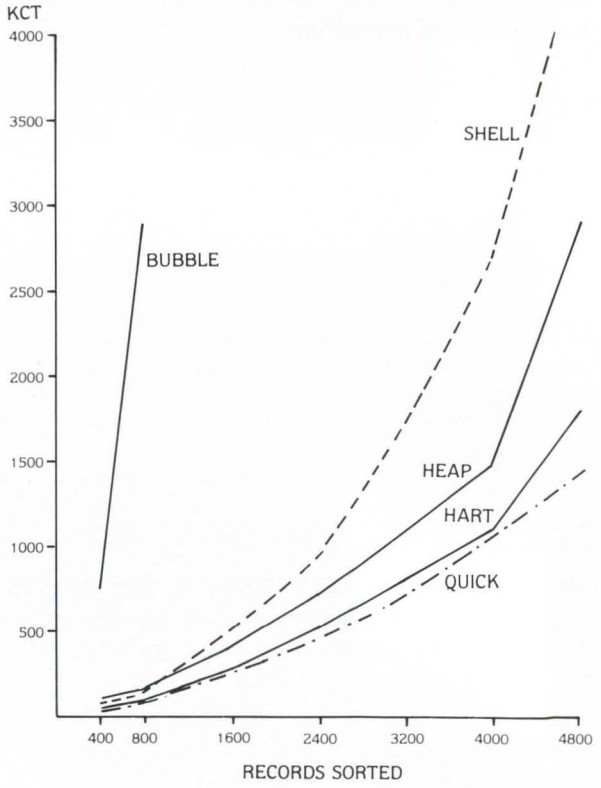


TABLE 2



4. Algorithms coded in BASIC-PLUS-2 sorting string fields

This section is a benchmark of all algorithms coded in BASIC-PLUS-2. Each string field contains three characters

and each character contains a letter varying from 'A' to 'J' randomly generated.

No benchmark for the Hart algorithm is included in this section. The complexity of the algorithm and the large amount of memory space it requires make this algorithm not suited to sort string fields.

4.1 Bubble sort

```
RUN DM1:SBUBBL

This is a benchmark of the BUBBLE sort     23-Dec-81      02:31

a) NUMBER OF SORTED ITEMS =  100 STRING FIELDS (3 CHAR)
   SORT USED 210 KCT, 4.2 CPU SEC. AND  5 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 STRING FIELDS (3 CHAR)
   SORT USED 4236 KCT, 70.6 CPU SEC. AND  71 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 STRING FIELDS (3 CHAR)
   SORT USED 18865 KCT, 269.5 CPU SEC. AND  271 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY
```

4.2 Shell sort

```
RUN DM1:SSHELL

This is a benchmark of the SHELL sort      23-Dec-81      02:37

a) NUMBER OF SORTED ITEMS =  100 STRING FIELDS (3 CHAR)
   SORT USED 40 KCT, .8 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 STRING FIELDS (3 CHAR)
   SORT USED 306 KCT, 5.1 CPU SEC. AND  6 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 STRING FIELDS (3 CHAR)
   SORT USED 875 KCT, 12.5 CPU SEC. AND  13 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 STRING FIELDS (3 CHAR)
   SORT USED 3190 KCT, 31.9 CPU SEC. AND  32 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 10 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 STRING FIELDS (3 CHAR)
   SORT USED 5868 KCT, 49 CPU SEC. AND  50 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 12 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 STRING FIELDS (3 CHAR)
   SORT USED 12420 KCT, 82.8 CPU SEC. AND  83 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 15 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 STRING FIELDS (3 CHAR)
   SORT USED 15930 KCT, 88.5 CPU SEC. AND  89 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 18 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 STRING FIELDS (3 CHAR)
   SORT USED 25368 KCT, 120.8 CPU SEC. AND  121 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 21 K OF MAIN MEMORY
```

4.3 Heap sort

```
RUN DM1:SHEAP

This is a benchmark of the HEAP sort       23-Dec-81      02:44

a) NUMBER OF SORTED ITEMS =  100 STRING FIELDS (3 CHAR)
   SORT USED 55 KCT, 1.1 CPU SEC. AND  2 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 STRING FIELDS (3 CHAR)
   SORT USED 348 KCT, 5.8 CPU SEC. AND  6 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 STRING FIELDS (3 CHAR)
   SORT USED 1050 KCT, 15 CPU SEC. AND  15 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 STRING FIELDS (3 CHAR)
   SORT USED 2950 KCT, 29.5 CPU SEC. AND  30 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 10 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 STRING FIELDS (3 CHAR)
   SORT USED 6133 KCT, 51.1 CPU SEC. AND  52 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 13 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 STRING FIELDS (3 CHAR)
   SORT USED 9735 KCT, 64.9 CPU SEC. AND  65 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 15 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 STRING FIELDS (3 CHAR)
   SORT USED 16542 KCT, 91.9 CPU SEC. AND  93 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 18 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 STRING FIELDS (3 CHAR)
   SORT USED 23604 KCT, 112.4 CPU SEC. AND  113 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 21 K OF MAIN MEMORY
```

4.4 Quick sort

```
RUN DM1:SQUICK

This is a benchmark of the QUICK sort      23-Dec-81      02:50

a) NUMBER OF SORTED ITEMS =  100 STRING FIELDS (3 CHAR)
   SORT USED 20 KCT, .4 CPU SEC. AND  1 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 5 K OF MAIN MEMORY

b) NUMBER OF SORTED ITEMS =  400 STRING FIELDS (3 CHAR)
   SORT USED 144 KCT, 2.4 CPU SEC. AND  3 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 6 K OF MAIN MEMORY

c) NUMBER OF SORTED ITEMS =  800 STRING FIELDS (3 CHAR)
   SORT USED 343 KCT, 4.9 CPU SEC. AND  5 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 7 K OF MAIN MEMORY

d) NUMBER OF SORTED ITEMS =  1600 STRING FIELDS (3 CHAR)
   SORT USED 1110 KCT, 11.1 CPU SEC. AND  12 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 10 K OF MAIN MEMORY

e) NUMBER OF SORTED ITEMS =  2400 STRING FIELDS (3 CHAR)
   SORT USED 2418 KCT, 18.6 CPU SEC. AND  19 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 13 K OF MAIN MEMORY

f) NUMBER OF SORTED ITEMS =  3200 STRING FIELDS (3 CHAR)
   SORT USED 3570 KCT, 23.8 CPU SEC. AND  25 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 15 K OF MAIN MEMORY

g) NUMBER OF SORTED ITEMS =  4000 STRING FIELDS (3 CHAR)
   SORT USED 5724 KCT, 31.8 CPU SEC. AND  32 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 18 K OF MAIN MEMORY

h) NUMBER OF SORTED ITEMS =  4800 STRING FIELDS (3 CHAR)
   SORT USED 8127 KCT, 38.7 CPU SEC. AND  39 SEC. (ELAPSED TIME)
   PROGRAM EXECUTED IN 21 K OF MAIN MEMORY
```

For DEC* Computers, 100% Compatible.

For DEC Owners and Users, 100% Essential.

Thousands of New Products. 100% DEC-Compatible.

All the latest hardware, software, services and supplies designed to run on your DEC computer. All the DEC-compatibles you've read about, heard about, but have never seen demonstrated. Plus, thousands more. Newer and better than anything on the market today. More DEC-compatibles than at any other show in the world!

Over 250 Vendors. 100% DEC-Friendly.

Meet the vendors who can help your DEC system reach a new standard of performance. Because you never have to ask, "Is it DEC-Compatible?" you get fast answers to the really important questions. It's the one Show for everyone who owns, manages, or uses a DEC computer. So bring the entire decision-making team: top management, financial management, DP management and senior staff.

Especially for DECUS* Conference Registrants. 100% Free.

It's easy to attend DEXPO East 83. And hard to miss. Especially for DECUS members attending the St. Louis conference. Free shuttle buses will take you from the Show to the conference in just five minutes. And conference attendees will be able to use their DECUS badges to enter the Show without paying a registration fee.

Group Registration Discounts Available

Bring Your Boss for FREE!

Use Registration Form

*DEC and DECUS are registered trademarks of Digital Equipment Corporation.

DEXPO™ East 83

The Third National DEC-Compatible Industry Exposition

Kiel Auditorium, St. Louis

Sunday-Tuesday • May 22-24, 1983

DEXPO East 83 Registration Form

Kiel Auditorium, St. Louis
Sunday-Tuesday, May 22-24, 1983

CHECK REGISTRATION PLAN (For Additional Registrations Duplicate Form on a Copying Machine):

I. **Advance Registration** saves you time and money. Because you get your credentials in the mail, there's no waiting in long registration lines.

- ☐ **Half-Price Advance Registration.** Complete form and mail with payment ($5.00) before April 25, 1983.
- ☐ **Free Admission to DECUS Conference Registrants.** Complete form and mail before May 2, 1983. No payment required.

Send to: **Expoconsul International Inc.**, 55 Princeton-Hightstown Rd., Princeton Junction, NJ 08550

II. **To Register at the Show,** bring completed form with you.

- ☐ **Free Admission to DECUS Conference Registrants.** Present completed form and DECUS badge. No payment required.
- ☐ **"Bring Your Boss for Free" Team Registration Discount.** Present completed forms and business cards. Registration fee is $5.00 for each visitor—in groups of two only.
- ☐ **Regular Registration.** Present completed form and payment ($10.00).

For information, call or write:
Expoconsul International, Inc.
55 Princeton-Hightstown Rd.
Princeton Junction, NJ 08550
TEL: 609-799-1661

Please Type • Use Capital Letters Only

Name ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Title ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Company ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Address ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

City ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

State ☐☐ **Zip** ☐☐☐☐☐ **Telephone** (☐☐☐) ☐☐☐-☐☐☐☐

NO ONE UNDER AGE OF 18 WILL BE ADMITTED

1. Title/Function
Check One:
- A ☐ Owner/Pres./CEO/Gen. Mgr.
- B ☐ Vice President
- C ☐ Controller/Treas./CFO
- D ☐ Financial/Acct. Mgmt.
- E ☐ Marketing Mgmt.
- F ☐ Administrative Mgmt.
- G ☐ MIS Director/DP Mgr./WP Mgr.
- H ☐ DP Mgmt. (Supervisory)
- I ☐ DP Staff
- J ☐ Engineering/Mfg. Mgmt.
- K ☐ Consultant
- L ☐ Other _____ (specify)

2. Business or Industry
Check One:
- A ☐ Banking/Financial
- B ☐ Manufacturing
- C ☐ Retailing/Distribution
- D ☐ Professional Services
- E ☐ Government
- F ☐ OEM/Computer Consultant
- G ☐ Education
- H ☐ Health Care Institutions
- I ☐ Other _____ (specify)

3. Computer Applications
Check All That Apply:
- A ☐ Accounting/Office
- B ☐ Banking/Financial
- C ☐ Research/Scientific
- D ☐ Industrial/Engineering
- E ☐ Resale
- F ☐ Other _____ (specify)

4. Type of Computer Used
Check All That Apply:
- A ☐ *PDP 8 Family
- B ☐ *Q Bus (11/03, 11/23, 11/33)
- C ☐ *Unibus (11/24, 11/44, 11/70, 11/34, 11/40)
- D ☐ *Vax
- E ☐ *10 or 20
- F ☐ Non-DEC* _____ (specify)

☐ **I am a DECUS conference attendee.**

5. Operating System
Check All That Apply:
- A ☐ *OS/8
- B ☐ *RT 11
- C ☐ *RSX, *IAS
- D ☐ *RSTS
- E ☐ *VMS
- F ☐ *Tops 10, 20
- G ☐ ** Unix
- H ☐ Other _____ (specify)

6. Language Used
Check All That Apply:
- A ☐ Fortran(s)
- B ☐ Basic(s)
- C ☐ PL/1
- D ☐ Cobol
- E ☐ Pascal
- F ☐ DIBOL/DBL
- G ☐ Other _____ (specify)

*Registered trademarks of Digital Equipment Corporation.
**Registered trademark of Bell Labs.

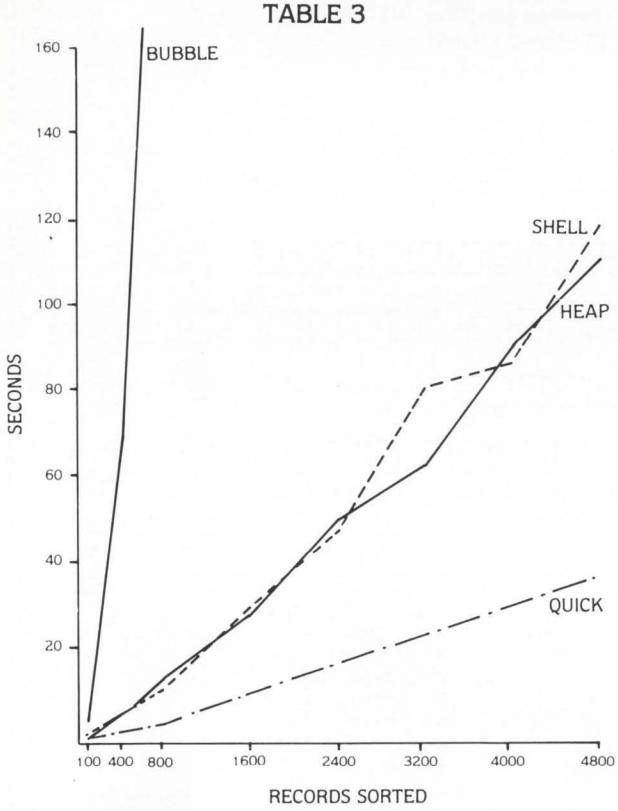
A B C D

4.5 Comparison of algorithms

	Bubble	Shell	Heap	Quick
number of statements	8	15	31	44
100 string fields	5 sec. (5k)	2 sec. (5k)	2 sec. (5k)	1 sec. (5k)
400 string fields	71 sec. (7k)	6 sec. (7k)	6 sec. (7k)	3 sec. (7k)
800 string fields	271 sec. (7k)	13 sec. (7k)	15 sec. (7k)	5 sec. (7k)
1600 string fields	-	32 sec. (10k)	30 sec. (10k)	12 sec. (10k)
2400 string fields	-	50 sec. (12k)	52 sec. (13k)	19 sec. (13k)
3200 string fields	-	83 sec. (15k)	65 sec. (15k)	25 sec. (15k)
4000 string fields	-	89 sec. (18k)	93 sec. (18k)	32 sec. (18k)
4800 string fields	-	121 sec. (21k)	113 sec. (21k)	39 sec. (21k)

Table 3 shows the elapsed time in seconds required by each algorithm to sort "X" number of string fields.

TABLE 3



5. Conclusion

5.1 Numeric sorts

To sort large arrays (over 1500 records) the Hart algorithm is the fastest and if the response time is a very important criteria for the success of your application then it can be useful. But the Quick algorithm is the most efficient in all situations. For sorts varying from 800 to 4800 numbers, Quick requires only from 1 to 4 additional seconds on a quiet system and uses from 1k to 5K less memory.

On the other hand, if the size of your program is reaching its limit and if you can not afford the Quick sort then the Shell algorithm might be the solution to your problem. It is reasonably fast when used to sort small arrays and is only 15 statements long.

The Bubble sort should never be used.

5.2 String field sorts

It is very obvious that the Quick algorithm is by far the best one. The trade off between the length of the algorithm (Quick: 44 statements, Heap: 33 statements, Shell: 15 statements) is outclassed by the response time gain. Compared to the Heap algorithm, the Quick requires from 3 to 74 seconds less to sort from 400 to 4800 string fields on a quiet system.

Again the Bubble sort should never be used.

5.3 Programming languages

BASIC-PLUS-2 sorts identical to BASIC-PLUS demonstrated at least four times the speed. This must be attributed to the system time used in converting the BASIC-PLUS interpretive language to executable code. This observation concerns numeric array sorts only. No benchmarks were done on string fields sorts coded in BASIC-PLUS because of the large amount of memory space usually required by string fields sorts. BASIC-PLUS program can only grow to a maximum of 16K in comparison to 24K in BASIC-PLUS-2.

5.4 Miscellaneous observations

This study also demonstrated that it is 4 to 6 times faster to sort string field arrays loaded in a channel memory buffer than identical arrays loaded in any other memory portion of a program. The following examples show how it's done.

a) loading an array using the channel memory buffer.

```
10      SORT.ITEM%=100 &
      \ REC.LEN%=3%
20      DIM A.RRAY$(100%)
30      OPEN "FILE.SRT" FOR INPUT AS FILE 1%, &
              RECORDSIZE SORT.ITEM%*REC.LEN% &
      \ GET #1% &
      \ FOR I%=1% TO SORT.ITEM% &
      \ FIELD #1%, (I%-1%)*REC.LEN% AS A.RRAY$(0%), &
                   REC.LEN% AS A.RRAY$(I%) &
      \ NEXT I% &
```

b) loading an array in any other memory portion of a program.

```
10      SORT.ITEM%=100% &
      \ REC.LEN%=3%
20      DIM A.RRAY$(100%)
30      FOR I%=1% TO SORT.ITEM% &
      \ FOR J%=1% TO REC.LEN% &
      \ A.RRAY$(I%)=A.RRAY$(I%)+STRING$(1%,INT(10%*RND)+65%) &
      \ NEXT J% &
      \ NEXT I% &
```

Sorting process is probably faster because while the complete string array elements are being physically and repeatedly moved around in memory for arrays loaded as example (b) only the address pointers of string array elements are changed in arrays loaded as example (a).

APPENDIX A

Algorithm 1 - Bubble Sort

```
1000    !*************************************************************&
        !***                                                       ***&
        !***                    INITIALISATION                     ***&
        !***                                                       ***&
        !*************************************************************&
          SORT.ITEM%=100% &
```

```
1010      DIM A.RRAY$(100%) &

17000     !*****************************************************************&
          !***                                                           ***&
          !***                      BUBBLE SORT                          ***&
          !***                                                           ***&
          !*****************************************************************&
            FOR I.NDEX1%=1% TO SORT.ITEM%-1% &

17010       FOR I.NDEX2%=I.NDEX1%+1% TO SORT.ITEM% &

17020       IF A.RRAY$(I.NDEX1%)>A.RRAY$(I.NDEX2%) &
            THEN W.ORK1%=A.RRAY$(I.NDEX1%) &
                 \ A.RRAY$(I.NDEX1%)=A.RRAY$(I.NDEX2%) &
                 \ A.RRAY$(I.NDEX2%)=W.ORK1% &

17030       NEXT I.NDEX2% &

17040       NEXT I.NDEX1% &
```

The most obvious algorithm is the Bubble Sort: for each item we replace is the least value in the remaining items, interchanging so the least value "bubbles up" to the front.

Algorithm 2 - Shell Sort

```
1000      !*****************************************************************&
          !***                                                           ***&
          !***                    INITIALISATION                         ***&
          !***                                                           ***&
          !*****************************************************************&
            SORT.ITEM%=100% &

1010        DIM A.RRAY$(100%) &

17000     !*****************************************************************&
          !***                                                           ***&
          !***                      SHELL SORT                           ***&
          !***                                                           ***&
          !*****************************************************************&
            CENTER.OF.ARRAY%=SORT.ITEM% &

17010       CENTER.OF.ARRAY%=INT(CENTER.OF.ARRAY%/2) &
          \ CENTER.ELEMENT%=CENTER.OF.ARRAY% &

17020       CENTER.ELEMENT%=CENTER.ELEMENT%+1% &
          \ CONTENT.OF.CENTER.ELEMENT%=A.RRAY%(CENTER.ELEMENT%) &
          \ WORK.CENTER.ELEMENT%=CENTER.ELEMENT% &

17030       KEEP.CENTER.ELEMENT%=WORK.CENTER.ELEMENT% &
          \ WORK.CENTER.ELEMENT%=WORK.CENTER.ELEMENT%-CENTER.OF.ARRAY% &
          \ GO TO 17040 IF WORK.CENTER.ELEMENT%<1% &
          \ GO TO 17040 IF CONTENT.OF.CENTER.ELEMENT% > &
                           A.RRAY%(WORK.CENTER.ELEMENT%) &
          \ A.RRAY%(KEEP.CENTER.ELEMENT%)=A.RRAY%(WORK.CENTER.ELEMENT%) &
          \ GO TO 17030 &

17040       A.RRAY%(KEEP.CENTER.ELEMENT%)=CONTENT.OF.CENTER.ELEMENT% &
          \ GO TO 17020 IF CENTER.ELEMENT% < SORT.ITEM% &
          \ GO TO 17010 IF CENTER.OF.ARRAY% > 1% &
```

A slight increase in program length brings a radical improvement in execution speed. The Shell Sort works like a Bubble Sort except that the comparison intervals decrease by powers of 2, so that initial exchanges tend to move items farther, much quicker than in the Bubble Sort.

Algorithm 3 - Heap Sort

```
1000      !*****************************************************************&
          !***                                                           ***&
          !***                    INITIALISATION                         ***&
          !***                                                           ***&
          !*****************************************************************&
            SORT.ITEM%=100% &

1010        DIM SORT.ARRAY%(100%) &

17000     !*****************************************************************&
          !***                                                           ***&
          !***                      HEAP SORT                            ***&
          !***                                                           ***&
          !*****************************************************************&
             CENTER.ELEMENT%=(SORT.ITEM%/2%)+1% &
          \  TO.SORT%=SORT.ITEM% &

17010       GO TO 17100 IF TO.SORT% < 2% &
          \ IF CENTER.ELEMENT% > 1% &
            THEN CENTER.ELEMENT% = CENTER.ELEMENT% - 1% &
                 \ CONTENT.WORK.ELEMENT% = SORT.ARRAY%(CENTER.ELEMENT%) &
                 \ GO TO 17030 &

17020       CONTENT.WORK.ELEMENT% = SORT.ARRAY%(TO.SORT%) &
          \ SORT.ARRAY%(TO.SORT%) = SORT.ARRAY%(1%) &
          \ TO.SORT% = TO.SORT% - 1% &
          \ IF TO.SORT% = 1% &
            THEN SORT.ARRAY%(1%) = CONTENT.WORK.ELEMENT% &
                 \ GO TO 17090 &

17030       SORTED.ELEMENT% = CENTER.ELEMENT% &

17040       WORK.ELEMENT% = SORTED.ELEMENT% &
          \ SORTED.ELEMENT% = 2% * SORTED.ELEMENT% &
          \ IF SORTED.ELEMENT% > TO.SORT% &
            THEN GO TO 17080 &
            ELSE IF SORTED.ELEMENT% = TO.SORT% &
                 THEN GO TO 17060 &
```

```
17050     IF SORT.ARRAY%(SORTED.ELEMENT%) < SORT.ARRAY%(SORTED.ELEMENT% + 1%) &
          THEN SORTED.ELEMENT% = SORTED.ELEMENT% + 1% &

17060     IF CONTENT.WORK.ELEMENT% < SORT.ARRAY%(SORTED.ELEMENT%) &
          THEN GO TO 17070 &
          ELSE GO TO 17080 &

17070     W.ORK1%=SORT.ARRAY%(SORTED.ELEMENT%) &
        \ SORT.ARRAY%(SORTED.ELEMENT%) = SORT.ARRAY%(WORK.ELEMENT%) &
        \ SORT.ARRAY%(WORK.ELEMENT%) = W.ORK1% &
        \ GO TO 17040 &

17080     SORT.ARRAY%(WORK.ELEMENT%) = CONTENT.WORK.ELEMENT% &

17090     GO TO 17010 &

17100   !***    END OF SORT     ***&
```

Algorithm 4 - Quick Sort

```
1000    !*************************************************************&
        !***                                                       ***&
        !***                    INITIALISATION                     ***&
        !***                                                       ***&
        !*************************************************************&
          SORT.ITEM%=100% &

1010      DIM SORT.ARRAY1%(100%) &
        ! &
        ! NUMBER OF ELEMENTS IN SORT.ARRAY2% EQUALS 2*LOG2(NUMBER OF RECORDS) &
        ! ================================================================ &
        \ DIM SORT.ARRAY2%(10%) &

17000   !*************************************************************&
        !***                                                       ***&
        !***                     QUICK SORT                        ***&
        !***                                                       ***&
        !*************************************************************&
          ARRAY2.INDEX%=0% &
        \ FIRST.ELEMENT%=1% &
        \ LAST.ELEMENT%=SORT.ITEM% &

17010     GO TO 17080 IF LAST.ELEMENT%<FIRST.ELEMENT%+1% &
        \ GO TO 17020 IF LAST.ELEMENT%>FIRST.ELEMENT%+1% &
        \ GO TO 17080 IF SORT.ARRAY1%(FIRST.ELEMENT%) <= &
                         SORT.ARRAY1%(LAST.ELEMENT%) &
        \ W.ORK1%=SORT.ARRAY1%(FIRST.ELEMENT%) &
        \ SORT.ARRAY1%(FIRST.ELEMENT%)=SORT.ARRAY1%(LAST.ELEMENT%) &
        \ SORT.ARRAY1%(LAST.ELEMENT%)=W.ORK1% &
        \ GO TO 17080 &

17020     CENTER.ELEMENT%=INT((FIRST.ELEMENT%+LAST.ELEMENT%)/2) &
        \ CONTENT.OF.CENTER.ELEMENT%=SORT.ARRAY1%(CENTER.ELEMENT%) &
        \ SORT.ARRAY1%(CENTER.ELEMENT%)=SORT.ARRAY1%(FIRST.ELEMENT%) &
        \ HIGH.ELEMENT%=LAST.ELEMENT% &
        \ LOW.ELEMENT%=FIRST.ELEMENT% &

17030     LOW.ELEMENT%=LOW.ELEMENT%+1% &
        \ GO TO 17060 IF LOW.ELEMENT%>HIGH.ELEMENT% &
        \ GO TO 17030 IF SORT.ARRAY1%(LOW.ELEMENT%) <= &
                         CONTENT.OF.CENTER.ELEMENT% &

17040     GO TO 17060 IF HIGH.ELEMENT%<LOW.ELEMENT% &
        \ GO TO 17050 IF SORT.ARRAY1%(HIGH.ELEMENT%) < &
                         CONTENT.OF.CENTER.ELEMENT% &
        \ HIGH.ELEMENT%=HIGH.ELEMENT%-1% &
        \ GO TO 17040 &

17050     W.ORK1%=SORT.ARRAY1%(LOW.ELEMENT%) &
        \ SORT.ARRAY1%(LOW.ELEMENT%)=SORT.ARRAY1%(HIGH.ELEMENT%) &
        \ SORT.ARRAY1%(HIGH.ELEMENT%)=W.ORK1% &
        \ HIGH.ELEMENT%=HIGH.ELEMENT%-1% &
        \ GO TO 17030 &

17060     SORT.ARRAY1%(FIRST.ELEMENT%)=SORT.ARRAY1%(HIGH.ELEMENT%) &
        \ SORT.ARRAY1%(HIGH.ELEMENT%)=CONTENT.OF.CENTER.ELEMENT% &
        \ ARRAY2.INDEX%=ARRAY2.INDEX%+2% &
        \ GO TO 17070 IF HIGH.ELEMENT%+HIGH.ELEMENT% <= &
                         FIRST.ELEMENT%+LAST.ELEMENT% &
        \ SORT.ARRAY2%(ARRAY2.INDEX%-1%)=FIRST.ELEMENT% &
        \ SORT.ARRAY2%(ARRAY2.INDEX%)=HIGH.ELEMENT%-1% &
        \ FIRST.ELEMENT%=HIGH.ELEMENT%+1% &
        \ GO TO 17010 &

17070     SORT.ARRAY2%(ARRAY2.INDEX%-1%)=HIGH.ELEMENT%+1% &
        \ SORT.ARRAY2%(ARRAY2.INDEX%)=LAST.ELEMENT% &
        \ LAST.ELEMENT%=HIGH.ELEMENT%-1% &
        \ GO TO 17010 &

17080     GO TO 17110 IF ARRAY2.INDEX%<=0% &
        \ FIRST.ELEMENT%=SORT.ARRAY2%(ARRAY2.INDEX%-1%) &
        \ LAST.ELEMENT%=SORT.ARRAY2%(ARRAY2.INDEX%) &
        \ ARRAY2.INDEX%=ARRAY2.INDEX%-2% &
        \ GO TO 17010 &

17090   !***    END OF SORT     ***&
```

By introducing an auxiliary array SORT.ARRAY2 as a pushdown stack, a further speedup is possible. The Quick Sort recursively breaks the array SORT.ARRAY1 into pairs of subarrays such that all items in one subarray are less than or equal to all items in the other subarray.

Algorithm 5 - Hart Sort

```
1000    !*************************************************************&
        !***                                                       ***&
        !***                    INITIALISATION                     ***&
        !***                                                       ***&
        !*************************************************************&
          SORT.ITEM%=100% &

1010      DIM SORT.ARRAY1%(100%) &
        ! &
        ! NUMBER OF ELEMENTS IN SORT.ARRAY2% EQUALS  &
        !       NUMBER OF RECORDS TO SORT +          &
        !       LOG2(NUMBER OF RECORDS TO SORT) + 2  &
        ! ========================================== &
        \ DIM SORT.ARRAY2%(112%) &

17000   !*************************************************************&
        !***                                                       ***&
        !***                     HART SORT                         ***&
        !***                                                       ***&
        !*************************************************************&

          K1=0 &
        \ N%=SORT.ITEM% &
        \ I%=0%&
        \ M1%=0% &
        \ T2=0 &
        \ T4=0 &
        \ J%=N%+1% &
        \ SORT.ARRAY2%(1%)=1% &
        \ SORT.ARRAY2%(J%)=1% &
        \ K2=1 &
        \ GO TO 17170 IF N%<=1% &
        \ S1=N% &

17010   ! CLIMB TREE TO TWIGS &
        ! ==================== &
          GO TO 17020 IF S1<4 &
        \ K2=K2*2 &
        \ B2=S1/2 &
        \ S1=INT(B2) &
        \ T4 = T4+(B2-S1)*K2 &
        \ GO TO 17010 &

17020   ! INITIAL CALCULATIONS &
        ! ==================== &
          T4 = K2-T4 &
        \ B2=K2/2 &

17030   ! NEXT TWIG. &
        ! ========== &
          GO TO 17170 IF K1=K2 &
        \ K1=K1+1 &
        \ T1=K1 &
        \ B1=B2 &
        \ T3=T2 &

17040   ! INC COUNTER AND CARRY &
        ! ===================== &
          T1=T1/2 &
        \ GO TO 17050 IF INT(T1)<T1 &
        \ M1%=M1%+1% &
        \ T2=T2-B1 &
        \ B1=B1/2 &
        \ GO TO 17040 &

17050   ! TWIG CALCULATIONS &
        ! ================= &
          T2=T2+B1 &
        \ GO TO 17060 IF S1=2 &
        !                   &
        ! 3-TWIGS AND 4-TWIGS &
        ! ================== &
        \ GO TO 17070 IF T3<T4 &
        !   &
        ! 4-TWIG &
        ! ====== &
        \ M1%=-M1% &
        \ GO TO 17090 &

17060     GO TO 17080 IF T3<T4 &

17070   ! 3-TWIG &
        ! ====== &
          M1%=M1%+1% &
        \ I%=I%+1% &
        \ SORT.ARRAY2%(I%)=I% &
        \ SORT.ARRAY2%(J%)=I% &
        \ J%=J%+1% &

17080   ! 2-TWIGS &
        ! ======= &
          M1%=M1%+1% &

17090     I%=I%+1% &
        \ L1%=I% &
        \ SORT.ARRAY2%(I%)=I% &
        \ SORT.ARRAY2%(J%)=I% &
        \ L0%=J% &
        \ J%=J%+1% &
        \ I%=I%+1% &
        \ L2%=I% &
        \ SORT.ARRAY2%(I%)=I% &
        \ SORT.ARRAY2%(J%)=I% &
        \ GO TO 17110 &

17100   ! MERGE TWIGS AND BRANCHES &
        ! ======================== &
         ( J%=J%-1% &
        \ L0%=J%-1% &
        \ L1%=SORT.ARRAY2%(L0%) &
        \ L2%=SORT.ARRAY2%(J%) &

17110     GO TO 17130 IF SORT.ARRAY1%(L1%)<=SORT.ARRAY1%(L2%) &
        \ SORT.ARRAY2%(L0%)=L2% &

17120     L0%=L2% &
        \ L2%=SORT.ARRAY2%(L0%) &
        \ GO TO 17140 IF L2%=L0% &
        \ GO TO 17120 IF SORT.ARRAY1%(L1%)>SORT.ARRAY1%(L2%) &
        \ SORT.ARRAY2%(L0%)=L1% &
```

```
17130     LO%=L1% &
        \ L1%=SORT.ARRAY2%(LO%) &
        \ GO TO 17110 IF L1% <> LO% &
        \ SORT.ARRAY2%(LO%) = L2% &
        \ GO TO 17150 &

17140     SORT.ARRAY2%(LO%) = L1% &

17150     M1% = M1% - 1% &
        \ GO TO 17100 IF M1% > 0% &
        \ GO TO 17030 IF M1% = 0% &

17160   ! 2ND HALF OF 4-TWIG &
        ! ================== &
          M1% = 1% - M1% &
        \ GO TO 17090 &

17170   ! OUTPUT &
        ! ====== &
          LO% = SORT.ARRAY2%(N%+1) &
        \ FOR I% = 1 TO N% &
        \ PRINT I%,SORT.ARRAY1%(LO%) &
        \ LO% = SORT.ARRAY2%(LO%) &
        \ NEXT I% &
```

Besides an auxiliary pushdown stack, Hart uses an array SORT.ARRAY2 of "links" which point to items in SORT.ARRAY1. Using an implicit balanced binary tree, Hart's algorithm sorts by doing a minimal number of comparisons. Hart includes the pushdown at the end of array SORT.ARRAY2.

APPENDIX B

Same as algorithms 1 through 5 of Appendix A except A.RRAY$ replaces A.RRAY% giving alpha sorts instead of numeric.

♥

RSTS System Manager's Workshop Tips for Maintaining Terminal Characteristics

Marlene Jackson
Cooperative Services, Inc.
P.O. Box 5171
Santa Fe, NM 87502

Here are two suggestions for helping you keep track of terminal characteristics:

1. Use a wonderful but poorly documented feature of RSTS, $TTYSET.MCM, a file in the format:

```
CONSOLE,'RESUME ANY;LA36;NOLC INPUT;SPEED 300;NOBREAK'
CAROL,'RESUME ANY;VT50;NOTAB;SPEED 4800;NOBREAK'
BOB,'RESUME ANY;VT100;NOTAB;SPEED 4800;WIDTH 132;NOBREAK'
BOARDRM,'RESUME ANY;VT52;NOTAB;SPEED 2400;NOBREAK'
ALICE,'RESUME ANY;VT100;NOTAB;SPEED 4800;WIDTH 132;NOBREAK'
DIALUP, 'RESUME ANY;VT50;NOTAB;SPEED 300;NOBREAK'
TED,'VT100;WIDTH 132;SPEED 1200;NOBREAK'
```

Your $TTYSET.CMD file can then look like this:

```
RUN $UTILTY
SEND KB0: SETTING TERMINAL CHARACTERISTICS
EXIT
RUN $TTYSET
KB0:;CONSOLE
KB1:/RING;PK
KB2:/RING;PK
KB3:;BOB
KB4:;CAROL
KB5:;TED
KB6:;ALICE
KB7:;BOARDRM
KB8:/RING;DIALUP
.
.
.
```

2. In one of your system management accounts, have a database which describes each port. Key it on keyboard number and include in it:

```
$TTYSET.MCM macroname
Terminal type (hardcopy, VT100, etc.)
Location descriptors
.
.
.
```

You may then:

Write a program for updating the database which recreates $TTYSET.CMD each time the $TTYSET.MCM macroname field is changed, making your startup command file correct at all times.

Refer to that database for terminal type in programs which want to do screen formatting. Your users will think your programs are very clever because they behave differently when run on a hardcopy than when run on their klunky old terminal and still another way when run on the boss's VT100.

♥

TIPS & TECHNIQUES

A Column For The Advanced RSTS/E User

Steven T. Edwards, Software Techniques, Inc.

BEEP-BEEP FOR GARBAGE COLLECTION

On RSTS, most programming is done in either BASIC-PLUS or BASIC-PLUS-2. Both of these programming languages support dynamic character strings. A dynamic string is a string whose space is allocated dynamically at run time. Each time a string assignment statement is executed, the language object time system (OTS) allocates space from the program's free space for the destination string. The space previously allocated to the destination string is not available for re-use until all of the free space has been exhausted and the OTS does some form of space reclamation or garbage collection. The resources expended to perform this garbage collection are resources that could be put to better use, like playing MTREK.

This is not news to most of us. Most of us are aware that the OTS is doing something but we don't know why or when. This article will give a brief and simplistic explanation of why the garbage collector is executed, and give you the tools you need to tell when the garbage collector is executed. This article will also address techniques to reduce the need for garbage collection.

Why

The in-memory layout of a BASIC program (minus run-time systems and resident libraries) is shown in figure A.

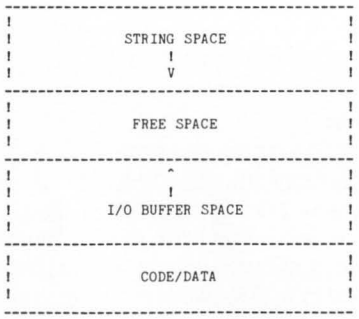


FIGURE A

What this figure shows is that the dynamic string space expands down from the top of the program (from high memory addresses towards low memory addresses), and the I/O buffer space expands up. The free space is the amount of space not currently allocated to either I/O buffers or string space.

When a dynamic string assignment statement is executed, the OTS checks to see if there is enough free space to contain the destination string. If there is not enough free space, the OTS will call the garbage collector to reclaim deallocated space. After the garbage collection, if there still is not enough free space to contain the destination string, the OTS will request the operating system (RSTS) to increase the amount of memory allocated to this job. If the operating system cannot expand the program, the OTS will abort the program with a "?Maximum memory exceeded" error. If the operating system can expand the program, the OTS will move the dynamic strings to the new top of memory, creating a larger free space. (See figures B — E.)

```
Program executes:        A$ = "HELLO THERE"
-----------------------------------------
!HELLO THERE                            !
-----------------------------------------
```

FIGURE B

```
Program executes:        B$ = "THIS IS A TEST"
-----------------------------------------
!HELLO THERETHIS IS A TEST              !
-----------------------------------------
```

FIGURE C

```
Program executes:        A$ = ""
(OTS marks A$ as deallocated)
-----------------------------------------
!**********THIS IS A TEST               !
-----------------------------------------
```

FIGURE D

```
Program executes:        C$ = STRING$(50,64)
(C$ is too large for free space, OTS
repacks dynamic strings, still not enough
free space, OTS expands program, moves
dynamic strings to new top of memory,
allocates C$)
-----------------------------------------
!**********THIS IS A TEST               !
-----------------------------------------
-----------------------------------------
!THIS IS A TEST                         !
-----------------------------------------
-----------------------------------------
!                                       !
!THIS IS A TEST                         !
-----------------------------------------
-----------------------------------------
!THIS IS A TEST@@@@@@@@@@@@@@@@@@@@@@@@@!
!@@@@@@@@@@@@@@@@@@@@@@@@               !
-----------------------------------------
```

FIGURE E

When

The patches below (one for BASIC-PLUS and one for BASIC-PLUS-2) will tell us when the OTS does a garbage collection operation. The patches will cause the OTS to ring the terminal bell by printing a CONTROL-G each time the garbage collector is executed.

BASIC-PLUS

To install the patch to the BASIC-PLUS run-time system, copy your existing run-time system to a new file called BASBEL. This is the file that we will be patching.

```
       PIP [0,1]BASBEL.RTS/MO:16=[0,1]BASIC.RTS
       RUN [1,2]ONLPAT
          .
          .
          .

!
!       PATCH TO CAUSE THE BASIC-PLUS RUN-TIME SYSTEM TO 'BEEP' WHEN THE
!       GARBAGE COLLECTOR IS CALLED.
!
!       COPYRIGHT (C) 1982, 1983 BY SOFTWARE TECHNIQUES, INC.
!       LOS ALAMITOS, CA 90720
!
!       NOTE THAT IT IS UP TO YOU TO FIND 16 WORDS OF UNUSED PATCH SPACE.
!
File to patch? [0,1]BASBEL.RTS
Base address? ECONOM
Offset address? 250
 Base   Offset  Old     New?
??????  000250  ??????  ? 4737          ;       JSR @#BASBEL
??????  000252  ??????  ? PA+offset     ; (USE YOUR OWN OFFSET.)
??????  000254  060001  ? ^Z
Offset address? ^Z
Base address? PA+offset                 ; (USE YOUR OWN OFFSET.)
Offset address? 0

 Base   Offset  Old     New?
??????  ??????  ??????  ? 012701        ; BASBEL:MOV #442,R1
??????  ??????  ??????  ? 442
??????  ??????  ??????  ? 012711        ;       MOV #1,(R1)
??????  ??????  ??????  ? 1
??????  ??????  ??????  ? 012121        ;       MOV (R1)+,(R1)+
??????  ??????  ??????  ? 012721        ;       MOV #BEL,(R1)+
??????  ??????  ??????  ? .+22
??????  ??????  ??????  ? 005021        ;       CLR (R1)+
??????  ??????  ??????  ? 005021        ;       CLR (R1)+
??????  ??????  ??????  ? 005021        ;       CLR (R1)+
??????  ??????  ??????  ? 005021        ;       CLR (R1)+
??????  ??????  ??????  ? .WRITE        ;       .WRITE
??????  ??????  ??????  ? 016001        ;       MOV 2(R0),R1
??????  ??????  ??????  ? 2             ; (CODE THAT WE CLOBBERED WITH OUR JSR.)
??????  ??????  ??????  ? 207           ;       RETURN
??????  ??????  ??????  ? 7             ; BEL:  .WORD 7
??????  ??????  ??????  ? ^Z
Offset address? ^Z
Base address? ^Z
File to patch? ^Z
```

To use this new run-time system, add the run-time system to the monitor tables, and then either switch into BASBEL to compile a program, or name an existing compiled file to BASBEL.

BASIC-PLUS-2

To install the patch to the BASIC-PLUS-2 OTS, extract the "$STMSC" module from the OTS library, patch the object module, and DO NOT replace it in the OTS library.

```
                BP2BEL.MAC
;       PATCH TO CAUSE THE BASIC-PLUS-2 OBJECT-TIME SYSTEM TO 'BEEP'
;       WHEN THE GARBAGE COLLECTOR IS CALLED.

;       COPYRIGHT (C) 1982, 1983 BY SOFTWARE TECHNIQUES, INC.
;       LOS ALAMITOS, CA 90720

        .TITLE  $STMSC

        .IDENT  /SOFTEC/

        .PSECT  BP2OTS,RW,I,LCL,REL,CON

$$$$$$  =       .                      ; SAVE THE START OF THE PSECT.
.       =       $$$$$$+46              ; OFFSET TO THE CALL FOR THE
                                       ;   STRING COMPACTOR.
        CALL    BP2BEL                 ; CALL THE BEEPER.
.       =       $$$$$$+610             ; OFFSET TO THE ROUTINE WE
                                       ;   REPLACED.
COMPAC:                                ; (MAKE UP A LABEL.)

        .PSECT  BP2BEL,RW,I,LCL,REL,CON

BEL:    .WORD   7                      ; BEEP.

BP2BEL:
        MOV     R1,-(SP)               ; SAVE R1.
        MOV     #442,R1                ; ADDRESS OF XRB.
        MOV     #1,(R1)                ; LENGTH OF BEL.
        MOV     (R1)+,(R1)+            ;   "    "   "
        MOV     #BEL,(R1)+             ; ADDRESS OF BEL.
.REPT   4
        CLR     (R1)+                  ; ZAP A WORD.
.ENDR
        EMT     4                      ; .WRITE (ASSUME NO ERRORS).
        MOV     (SP)+,R1               ; RESTORE R1.
        JMP     COMPAC                 ; COMPACT THE STRINGS.

        .END
```

To use this new OTS module, you must link it into your task image. You must reference the BP2BEL module in your task builder CMD/ODL files. Do not link the task to any BASIC resident libraries because the task builder will not resolve the reference to the $STMSC module correctly. You also must reference BP2BEL before you reference the OTS library.

TKB task = object,BP2BEL,LB:BP2COM/LB

So what

Now that we can tell when a program is garbage collecting, what can we do about it? The first step is to identify the programs that are constantly calling the garbage collector. It is interesting to note that it's not just the 16KW BASIC-PLUS or 31KW BASIC-PLUS-2 programs that constantly call the garbage collector. All programs that constantly allocate/deallocate dynamic strings will incur the wrath of the garbage collector. The frequency of calling the garbage collector is a function of the size of the free space and the frequency of string allocation. You can reduce the frequency of calling the garbage collector by either increasing the size of the free space, or reducing your program's demands upon it.

Increasing the size of the free space can be accomplished by allocating a large string (several thousand bytes long) to force the program expansion, and then deallocating the string by assigning it to a null string (""). In BASIC-PLUS-2 this can also be accomplished by using the task builder "EXTTSK" directive.

You can reduce your program's demands upon free space by:

1. Using fixed length strings in BASIC-PLUS-2 by declaring the strings in a MAP/COMMON statement.

2. Pre-allocating strings and then using LSET/RSET to assign values.

3. Deallocating strings as soon as they are no longer needed. BASIC-PLUS-2 automatically deallocates all local dynamic strings when a sub-program is exited.

4. Coding programs so that they don't build strings piece by piece.

Now, the next time you find yourself sitting in front of your terminal wondering why your BASIC program takes so long to run, you have the tools to see if dynamic string garbage collection is the cause.

Visit us at
DEXPO™ East 83
Kiel Auditorium
St. Louis, Missouri
May 22-24, 1983
Booth
430

DBL FOR PDP-11 AND VAX BASED COMPUTERS

DBL, a structured superset of DEC's DIBOL-11 Business Programming Language, is available in two new releases: **DBL/VMS** is available for VAX/VMS in native mode, emits in-line code, and is source code compatible with DIBOL-11 code. **DBL Runtimes** are available for most DEC PDP-11 operating systems as a runtime-only license. Call for minimum quantities and prices. DBL is available for RT-11, TSX/TSX-Plus, RSTS, RSX-11M/M-Plus, and VAX/VMS native mode.

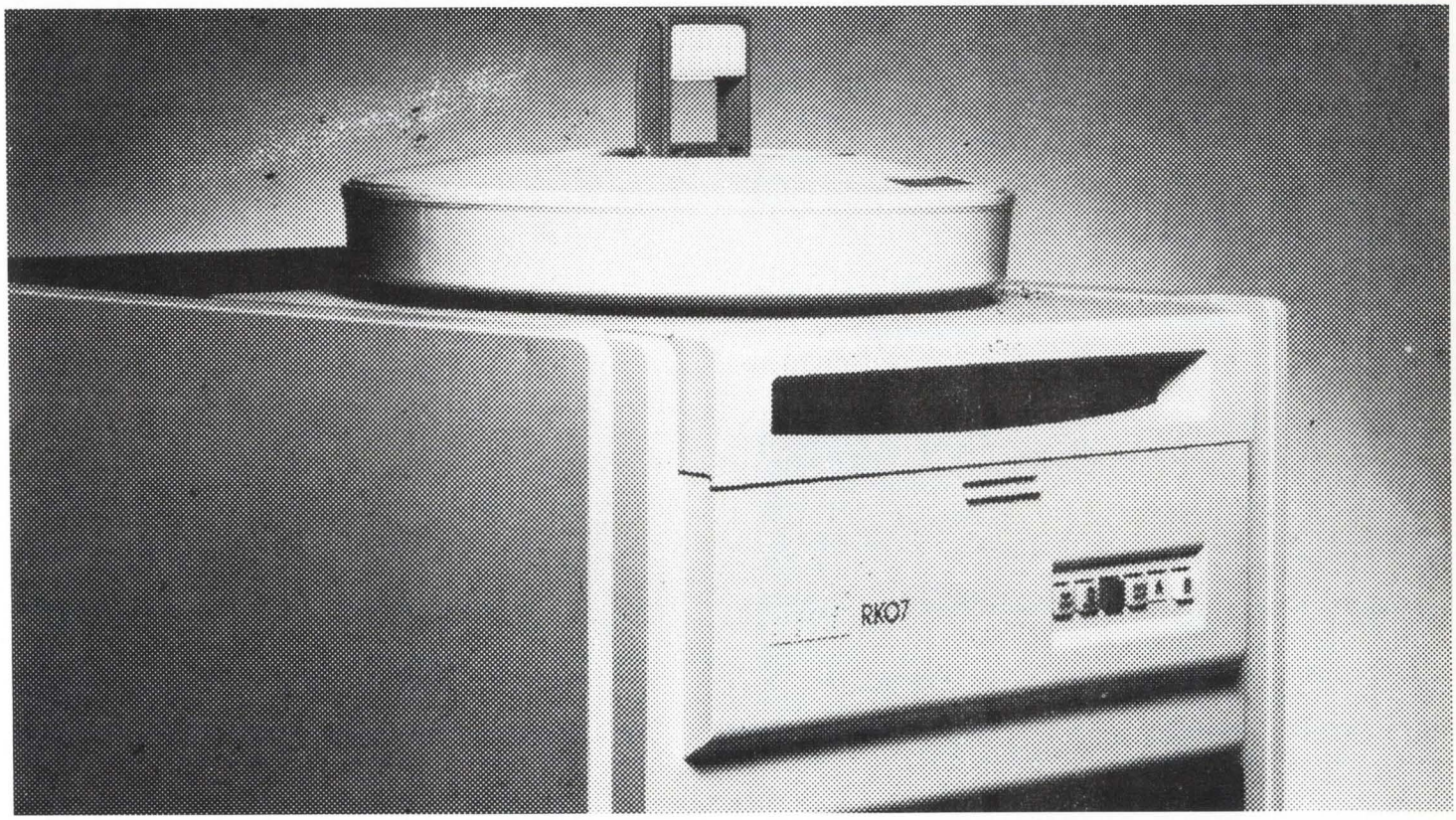
DISC

3336 BRADSHAW ROAD • SUITE 340 • SACRAMENTO, CA 95827 • 916/363•7385 • TWX 910/367•3701
DIGITAL INFORMATION SYSTEMS CORPORATION

CIRCLE 32 ON READER CARD

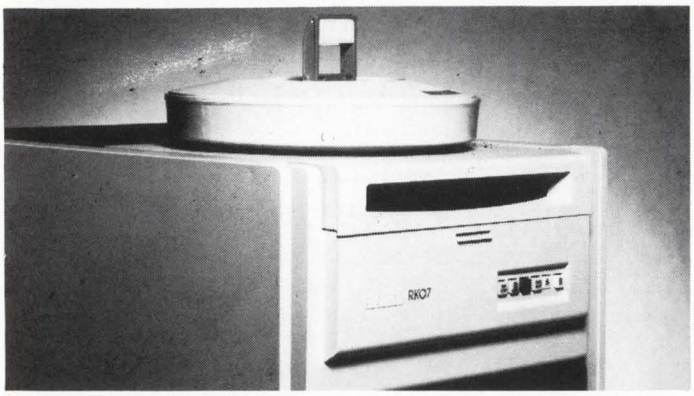
The VAX-SCENE

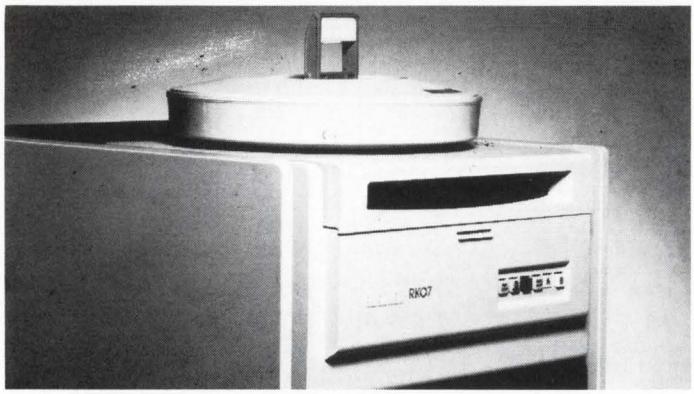
Number 13 (RSTS PROFESSIONAL, Vol. 5, No. 2) **April 1983**

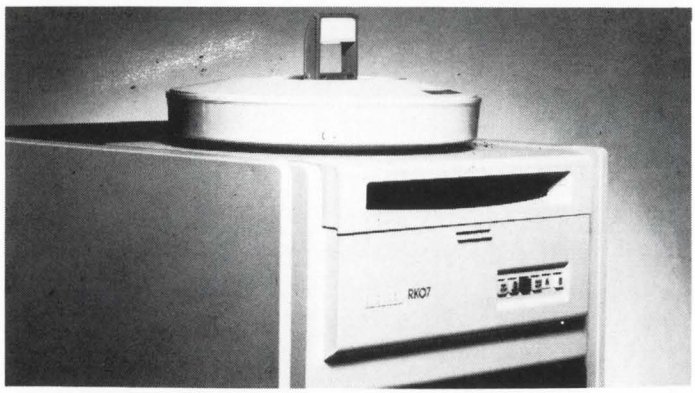


INSIDE:

Learning to Use the VAX Debugger







LEARNING TO USE THE VAX DEBUGGER (for those of you who make mistakes . . .)

By Bob Meyer

The VAX debugger is probably one of my favorite pieces of software on the whole system. (Yes, I'm one of those weirdos who LIKES to write software.) If any of you have used RSXODT on RSTS, RSX or IAS, you're in for quite a treat. This debugger is fully symbolic (meaning it knows about variables, psects, labels, etc., in your program) and understands the VAX instruction set; a feature which lets you examine instructions in assembly language format, as well as type in mnemonics to be assembled (on the spot) by the debugger. (Neat!)

In this article we'll talk about some of the basic debugger commands and (as always) give some examples in MACRO (is there anything else??)

First let's consider a simple program (you may want to key this in and try the examples that follow):

```
.TITLE  DEBUG
.IDENT  /0.0/

.PSECT  IMPURE
NUM1:   .WORD   1
NUM2:   .WORD   2

.PSECT  CODE
BOB:    .WORD   0
        MOVZWL  NUM1,R0
10$:    INCW    R0
        CMPW    R0,#10.
        BLSS    10$
        RET
        .END    BOB
```

(by the way, this 'program' isn't meant to do anything useful . . .)

Assemble this little beauty with:

```
$ MAC BOB/ENABLE=DEBUG
```

assuming you called it BOB.MAC (how flattering!). The /ENABLE=DEBUG causes the assembler to place debugger records in the object file containing names of symbols in your program (and probably other things I know very little about).

Then link:

```
$ LINK BOB/DEBUG
```

This links your image with the symbolic debugger.

When you run the program, the debugger will take control:

```
$ RUN BOB
                VAX-11 DEBUG Version 2.3-5
%DEBUG-I-INITIAL, language is MACRO, module set to 'DEBUG' DBG>
```

Let's look at the EXAMINE command first. The EXAMINE command allows us to look at anything in our virtual workspace (this also includes various system routines if you reference them.) EXAMINE has several modes in which you can look at your program or data; the default mode is hex:

```
DBG>E    NUM1
DEBUG\NUM1:     00020001
DBG>
```

In this example I've asked to 'Examine NUM1', which contains a constant 1. (Note the EXAMINE command can be abbreviated to 'E'.) The 'DEBUG' in the debugger's response is the .TITLE of the current module. The NUM1 is the nearest symbol to the location we're examining; the number that follows is the value of NUM1. (Actually, since we're examining in LONGWORD mode by default, we're seeing NUM1 and NUM2 back-to-back.)

Some options of the EXAMINE command are:

/OCTAL, /HEX, /DECIMAL

which cause the debugger to ACCEPT and DISPLAY all numbers in the requested radix.

The options:

/BYTE, /WORD, /LONGWORD

specify the format to display the data in; for example:

```
DBG>E/WORD NUM1
DEBUG\NUM1:   0001

DBG>E/BYTE NUM1
DEBUG\NUM1:   01

DBG>E/LONG NUM1
DEBUG\NUM1:   00020001
```

The qualifiers OCTAL, DECIMAL, & HEX can be combined with BYTE, WORD, & LONGWORD as needed.

Perhaps the most useful form of the EXAMINE command is the /INSTRUCTION option:

```
DBG>E/IN BOB
DEBUG\BOB:   HALT
```

Issuing subsequent EXAMINE commands will cause the debugger to list sequential memory locations:

```
DBG>E/I
DEBUG\CODE+01:   HALT
DBG>E/I
DEBUG\CODE+02:   MOVZWL   L↑DEBUG\NUM1,R0
DBG>E/I
DEBUG\CODE+09:   INCW     R0
```

or to list a block of memory, specify the memory range separated by a colon ':' —

```
DBG>E/I BOB:BOB+10
DEBUG/BOB:       HALT
DEBUG\CODE+01:   MOVZWL   L↑DEBUG\NUM1,R0
DEBUG\CODE+09:   INCW     R0
DEBUG\CODE+0B:   CMPW     R0,#0A
DEBUG\CODE+0E:   BLSS     DEBUG\CODE+09
DEBUG\CODE+10:   RET
DBG>
```

(Some of the above numbers (CODE+01, etc.) may be inaccurate here.)

Other EXAMINE switches are:

/ASCII:n
Lists memory in ASCII format; n is the number of bytes to list (default is 16.) Assuming you have some ASCII text defined similar to this:

```
MSG:   .ASCII   /Macro men do it Faster!/
```

Typing the command:

```
DBG>E/AS:5
```

would print:

```
DEBUG\MSG: Macro
```

/SYMBOL
Lists symbol names along with memory contents. This is the default.

/NOSYMBOL
Suppresses listing of symbol names, and prints the absolute (virtual) values instead:

```
DBG>E/I/NOSYM BOB:BOB+10
00000201:   HALT
00000202:   MOVZWL   L↑DEBUG\NUM1,R0
00000209:   INCW     R0
0000020B:   CMPW     R0,#0A
0000020E:   BLSS     DEBUG\CODE+09
00000210:   RET
DBG>
```

Also, the command:

```
DBG>E  PSW
```

will format and display the bits in the program status word.

The defaults in the EXAMINE (as well as DEPOSIT) commands can be changed using the SET MODE command. Some of the SET MODE commands are:

SET MODE INSTRUCTION	Attempts to disassemble memory contents and display mnemonics.
SET MODE NOINSTRUCTION	Displays memory in the current numeric mode (hex by default).
SET MODE OCTAL [or DECIMAL or HEX]	Establishes the default radix for input & output of non-symbolic data.
SET MODE SYMBOL	Displays symbol names from the symbol table in the image instead of memory addresses.
SET MODE NOSYMBOL	Displays (virtual) memory address of examined locations.
SET MODE WORD [or BYTE, or LONGWORD]	Displays all memory contents as words, bytes, or longwords, as specified.

The DEPOSIT command can be used to change the contents of memory in the user workspace. The DEPOSIT command takes most of the same switches as the EXAMINE command. Some examples of DEPOSIT follow:

```
DBG>D NUM1=5
```

Places the number 5 into location NUM1. (Remember, the default is hex unless you change it.)

```
DBG>D/OCT NUM1=10
```

Deposits an octal 10 (8.) into NUM1.

```
DBG>D/HEX/WORD NUM2=0E
```

Places the hex value E (14.) in the location NUM2. Note that when depositing hex values A-F a leading zero is required so the debugger doesn't try to interpret the letter as a symbol.

```
DBG>D/ASCII:6 MSG='Howdy!'
```

Places the specified string in memory starting at the

location 'MSG'. If the length of the string is greater than the number specified with the /ASCII:n switch, the string is padded to the right with spaces. The default length is 4 bytes.

```
DBG>D/I BOB='CLRW R0'
```

This command (DEPOSIT/INSTRUCTION) will 'assemble' the instruction (and operands) and place the result in the specified memory location (far out!). Keep in mind that when modifying instructions, you have to keep track of the instruction length(s). For example, replacing a three word instruction with a one word instruction leaves you with two words of trash (they wouldn't let me say 'crap' . . .) which can cause some pretty interesting results when the processor attempts to execute them. So, we NOP them: first we'll put the new instruction in:

```
DBG>D/I BOB='CLRW R0'
```

Then the NOPs:

```
DBG>D/I 'NOP'
DBG>D/I 'NOP'
```

To begin execution of your program, use the GO command:

```
DBG>GO
```

The debugger tells you where it's starting from:

```
Start at DEBUG\CODE+00
```

and turns control over to the program. (Remember, the sample program doesn't actually do anything . . .)

The STEP command is useful for single stepping through the program and examining or changing values as you go:

```
DBG>STEP
```

The debugger prints:

```
Start at DEBUG\CODE+02
Stepped to DEBUG\CODE+09: INCW R0
```

telling you the address of the instruction to be executed, then the address and the assembly mnemonic of the instruction to be executed by the next STEP command. STEP can be shortened to just 'S'.

It might be advisable to get a feel for the debugger at a time when you don't really need it, rather than waiting until you have a real problem on your hands.

Well, I'll leave you with the words of my great, great, grandfather, Oswald Meyer, who said "That's not a bug, that's a feature!"

Enjoy.

OPTIMIZING BACKGROUND USAGE

Michael Mayfield, Northwest Digital Software, Box 2-743, Newport, WA 99156

Last issue I introduced a patch to the RSTS monitor that gave RSTS high priority, realtime response. This article will describe a patch for optimizing the use of low priority, background tasks.

The priority structure of RSTS allows you to specify that certain tasks are more important than others and should be allowed to run more often. Conversely, you can specify that certain programs are less important and should run less often.

By making it low enough priority, a program will only run if no other program on the system is runnable. This allows you to make use of computer time that would normally be wasted, supposedly without degrading the performance of other more important programs.

Unfortunately, it doesn't really work that way. Once the low priority program starts to execute it is allowed to continue for its entire run burst, usually 1/10th of a second.

If the higher priority tasks are performing a lot of I/O, the system can have lots of little inactive times. The background task will run for its entire run burst during each of these inactive times. This can significantly degrade the performance of the higher priority tasks.

The following patch allows a task running below priority -64 to immediately stop running when a higher priority task becomes runnable. Average response degradation is less than one millisecond.

This patch uses patch space that is allocated for possible monitor patches. Future monitor patches may require the same patch addresses. If this occurs, the realtime patch will have to be removed or moved to a different location in patch space. The patch is position independent and can be installed in any other area in patch space that is not in use.

The comments following the semi-colons are for information only and can be ignored while entering the patch, although they will not cause any problems if entered. <LF> is used to signify a linefeed. ?????? is used to signify that any value is acceptable for this field.

As with all patches, be sure that the offset and old values are correct for each line before making any changes. If any of the old values is incorrect, abort the patch by typing ↑C.

```
RUN [1,2]ONLPAT
Command file name? LOWPRI.LOG=
File to patch? <LF>
Module name? <LF>
Base address? PATCH+340
Offset address? 0
 Base    Offset  Old      New?
??????  000000  000000   ? 105737  ;Is a job currently running
??????  000002  000000   ? JOB
??????  000004  000000   ? 1414    ;No
??????  000006  000000   ? 13746   ;Point to JOB for current job
??????  000010  000000   ? JOBDA
??????  000012  000000   ? 62716   ;Point to JOB+JOPRI (Job's priority)
??????  000014  000000   ? 34
??????  000016  000000   ? 123627  ;Is current job low priority
??????  000020  000000   ? -64.    ;  (priority threshold)
??????  000022  000000   ? 2003    ;No
??????  000024  000000   ? 52737   ;Schedule the I/O job immediately
??????  000026  000000   ? 20000
??????  000030  000000   ? L3QUE2
??????  000032  000000   ? 105737  ;Replace the patched instruction
??????  000034  000000   ? JOB
??????  000036  000000   ? 207     ;Return from patch
??????  000040  000000   ? ^Z
Offset address? ^Z
Base address? IOFIN4
Offset address? 62
 Base    Offset  Old      New?
??????  000062  105767   ? 4737    ;Enter patch
??????? 000064  ??????   ? PATCH+340  ;NOTE: New value must match patch base
??????  000066  001003   ? ^C      ;End of patch
```

"*The Bridge*™ is software that creates a virtual microcomputer at every terminal connected to my mini. I have all the functions of a micro, but without micro limitations.

"The *z-Board*™ has four z-80a® microprocessors per board to execute programs at high speed. Faster than many dedicated micros. And it has 256K bytes of RAM, plus a bit slice state machine. That's the guts of four micros for less than you might pay for one.

"With *The Bridge*, I can run CP/M® based programs. I like that. And micro programs like Supercalc® are easy to use, and inexpensive. I like that, too.

"But the best thing about *The Bridge* is systems integration. Now everyone in the office uses the same system — no more problems with disk formats, incompatible languages or programs. *The Bridge* provides each user with a virtual microcomputer with the advantages of a mini's high-speed printers, hard disks, and communications.

"*The Bridge* with a *z-Board* gives me the performance of four microcomputers — at a fraction of the cost."

The Bridge and z-Board are trademarks of Virtual Microsystems, Inc.
z-80a is a registered trademark of Zilog, Inc.
CP/M is a registered trademark of Digital Research.
Supercalc is a registered trademark of Sorcim, Inc.

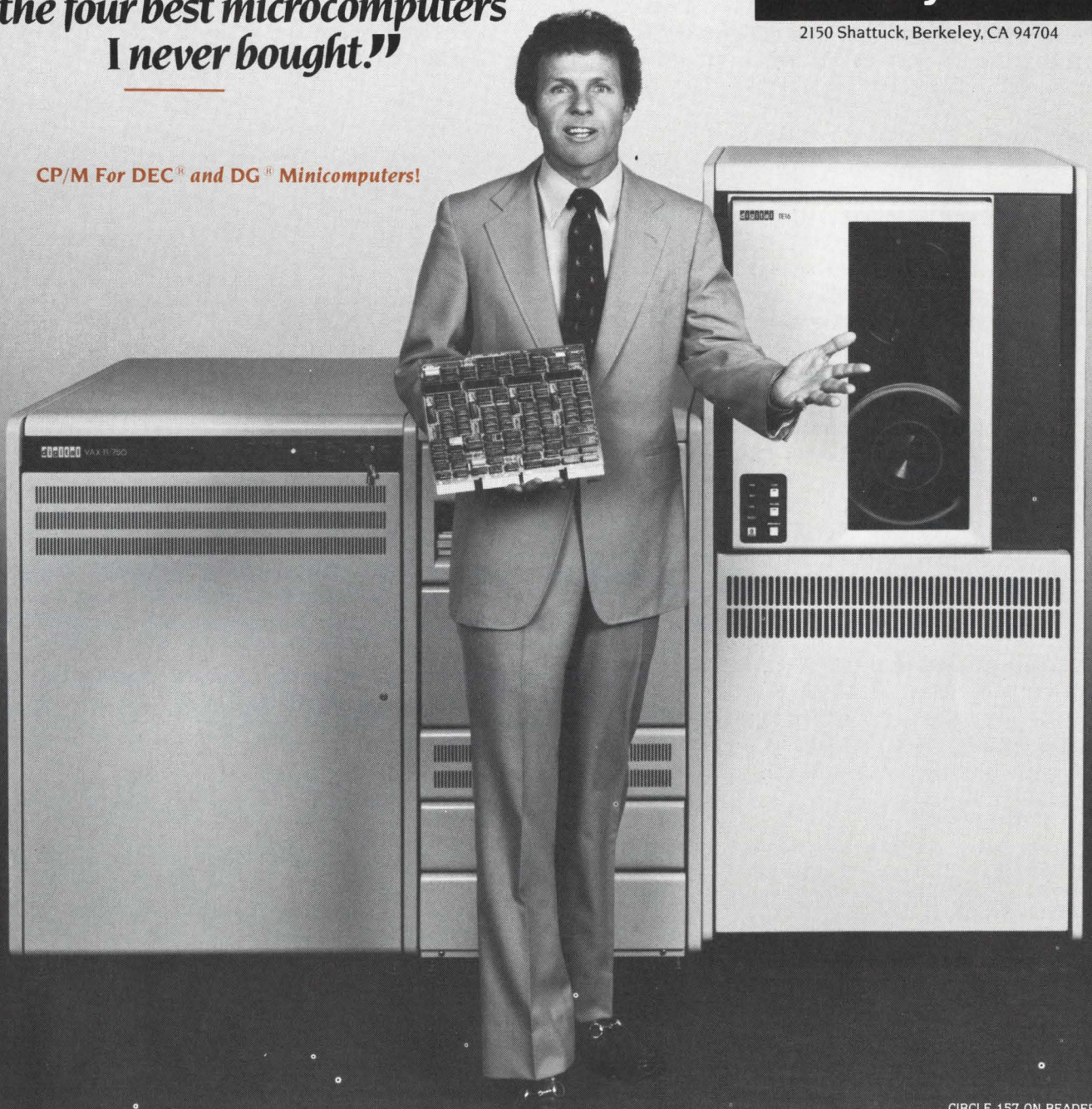
For information, call Jim Swanson (415) 841-9594.

virtual
microsystems

2150 Shattuck, Berkeley, CA 94704

"The Bridge and a z-Board — the four best microcomputers I never bought."

CP/M For DEC® and DG® Minicomputers!



CIRCLE 157 ON READER CARD

USE[ful,less]
BASIC PLUS 2 SOFTWARE

By Edward A. Heinrich

This article contains some routines for the Basic Plus 2 applications programmer. Although I myself prefer to play with the OS instead of applications (actually I'd rather be skiing, but . . .), in most companies applications programs are the bread and butter that pay the salaries. I have developed some routines coded in MACRO-11 and callable from BASIC PLUS 2. This article will provide a few that have been found useful and suggest some alternative ways of obtaining information normally only obtainable by doing SYS() calls. If you are not a MACRO programmer, please don't get scared and turn the page — all the routines are fairly simple and can be used just as if they were BASIC PLUS 2 subroutines.

XCONTG.MAC

Here is a favorite contradiction—'Good Disk Management' dictates that files be created contiguously. In real life though, dynamic files eventually become full, and if created contiguously, RSTS will generate a ?Protection Violation error message when it attempts to extend the file. This occurs because a bit is set in the file's directory entry (Bit 4 in USTAT), indicating that the file is contiguous. It used to be that when this occurred a user would call one of the programmers who would then use UTILTY to FLAG the file /NOCTG. While workable, this is not a desirable solution. Eventually the programmmers become tired of spending their time flagging files so they no longer create them contiguous, and violate the 'Good Disk Management' rule. To get around this problem, I have written a MACRO routine that will do the dirty work of unsetting the contiguous bit in the directory entry if a Protection Violation error occurs on a PUT or UPDATE, and keeps the programming staff programming.

To use XCONTG, trap for an ERR = 10%, resume at a line number external to the error handler, and CALL XCONTG(CHANNEL%) where CHANNEL% is the channel number of the file you desire to flag as non-contiguous. Note that no errors are expected in XCONTG, but if any do occur, the error code will be returned in CHANNEL%.

```
23100     IF    ERR = 10%
          THEN  RESUME 29000

29000     PRINT #12%, 'File is Full - Extending ... Please Wait'
      \   CHANNEL% = 1%
      \   CHANNEL% = 2% IF    ERL = 1020
      \   CALL XCONTG(CHANNEL%)
      \   IF   CHANNEL%
          THEN  PRINT #12%, '?XCONTG Error '; ERT$(CHANNEL%)
          \     GOTO 32767
          ELSE  GOTO 1020 IF   ERL = 1020
          \     GOTO 2000
          ! Print an informational message to the user
          ! Set up the channel number depending on where the
          ! error occurred
          ! Call XCONTG to unset contiguous bit
          ! If any errors returned then complain
          ! Else branch back into code
```

CAP.MAC

This is MACRO version of a routine developed by James F. Shaughnessy, Jr. (See 'Input Loop Programming Technique' in RSTS Professional Vol. 4, No. 1, February 1982). In data entry programs it is often desirable to allow the operator to enter data in lower case. It is also desirable to have the first letter of each word capitalized. Jim coded a routine that allows the operator to enter all the data in lower case and capitalizes the first letter in each word. I personally find this a very useful feature and the data entry operators seem to like it too, so I decided to convert it to MACRO.

```
    Programming example -

2000      CALL USERIN(10%,14%,INPUT$)
          ! Take in user input and do any edits for <CR> or whatever

2100      CALL CAP(INPUT$)
      \   PRINT #12%, INPUT$
          ! Call CAP to capitalize 1st letter in every word
          ! Reprint it for the user to see

If the operator precedes a character with a slash, '\', the character
is not capitalized and the slash is removed from the string.
```

KBNUM.MAC

Do you ever have the need to obtain either the current keyboard number and/or the installation name? You can do it by using the SYS() CALL to return error messages, but that requires a thread, $IESYS, that takes up 1194 bytes of address space. You can often write little MACRO routines that accomplish the same purpose as SYS() functions but require substantially less space. KBNUM.MAC is an example of such a routine. You may CALL KBNUM(KB%,I$) to return the current keyboard number and installation name, or CALL KBNUM(KB%) if you only want the keyboard number. Note that since MACRO programs cannot create strings, I$ must be defined in the calling routine prior to calling KBNUM.

TTECHO.MAC

There are occasions when you may wish to disable the echo control on a keyboard, for example, to take in a password. Once again I have gotten around the use of the SYS() call routine by a MACRO program that will enable or disable echo control on the keyboard opened on channel #12. TTECHO contains two entry points so that you may just do a call to either entry point without the need to pass any flags. You may modify it to operate on another channel by changing the

MOVB #24.,@#XRB+6 to the desired channel number * 2.
To use TTECHO and disable echo control—CALL NOECHO.
To re-enable echo control—CALL ECHO.
In the .ODL file add the subroutine name TTECHO.

LOOK MA, 'No SYS() function'

Another way to eliminate some SYS() calls is to access virtual sections of your program and read what RSTS has put there. This, if you are not a MACRO programmer or have never used the VSECT directive, needs some explanation.

FIRQB — File Request Queue Block — is the main area used to exchange information between the monitor and the job for operations that involve file or device operations.

VSECT — a task builder directive that allows a user job to have direct access to a Virtual SECTion, or address, in the user's job area. It is currently undocumented in the RSTS/E Task Builder Manual, but it is documented in the RSX version. The format of the VSECT directive is as follows,

VSECT = MAPNAM:STARTLOC:LENGTH

where MAPNAM is the name of a buffer, i.e., a mapped buffer, STARTLOC is the starting virtual address, i.e., the FIRQB is at location 402 in a job using the . . . RSX runtime system, and LENGTH is the length to map. To access the FIRQB we would place in our BP2 program a map similar to this one:

```
700     MAP (FIRQB)     FIRQB%,
                        FQFUN%,
                        FQFIL%,
                        FQPPN%,
                        FQNAM1%,
                        FQNAM2%,
                        FQEXT%,
                        FQSIZ%,
                        FQBUFL%,
                        FQMODE%,
                        FQFLAG%,
                        FQPROT%,
                        FQDEV$ = 2%,
                        FQDEVN%,
                        FQCLUS%,
                        FQNENT%
```

(For a full description of the above naming scheme refer to Chapter 2 of the RSTS/E Systems Directives manual.)

Then in the .CMD file we add the following line:

VSECT = FIRQB:402:40

We can now access the FIRQB from BP2 just the same as if we were writing in MACRO. Now that you are familiar with the VSECT command, I'll give a few examples of how to use it. If you require the job number all you need to do is map the FIRQB, include a VSECT for it, and the job number is at your finger tips. RSTS always returns the current job number at FIRQB+2. Even if your program does not do any monitor calls, the job number will be available since it is returned when the task is loaded from the disk on a RUN command.

```
700     MAP (FIRQB)     FILL$ = 2%,
                        FQJOB$ = 1%             ! Current job number

710     JOB.NUM% = ASCII(FQJOB$) / 2%           ! Get our job number
```

(Note that if this is all of the FIRQB you will need, you should modify the VSECT length to 3.)

We can access core common in the same manner. Core common is used to exchange information between user programs and the monitor and the job. Core common is located at location 460(8), and is 200(8) bytes long. The first byte contains the length of the valid data in core common. The next 177(8) bytes contain the data. You can do some rather interesting things with core common. One is to use core common as an intermodule common area. You map it the way you would any shared data area but since you have 177(8) bytes of space always in the low section of the task you might as well take advantage of it. Consider the following BP2 example:

```
700       MAP (CORCMN)    CORCMN$          =   1%,    ! Length of CORCMN
                          CORE_COMMON$     = 127%     ! Rest of buffer
      \   MAP (CORCMN)    FILL$            =   1%,    ! Fill for CORCMN
                          FILE_LOCATION$   =  13%,    ! Location of files
                          USER_ID_NO%,                ! User's id number
      \   MAP (CORCMN)    E_RROR_FLAG%,               ! Intermodule errors
                          COMMON_FLAG%,               ! Bit map flags
                          LAST_COMPANY$    =   6%,    ! Last company ID #
                          LAST_RFA%                   ! Records' RFA

      ! The first map is a generalized map of core common
      ! The second map contains description of info passed from a
      ! chaining program.
      ! The third map is data passed between different subroutines.
      ! Be sure to include VSECT=CORCMN:460:200 in the .CMD file.
```

Another location in the low section of a task that is useful is the KEYWORD at location 400 (8). The keyword defines the job's privilege. By testing the appropriate bits of this word, we can ascertain the job's privilege.

```
700       MAP (KEYWRD)    KEYWRD%                     ! Map the Keyword
          ! Include in .CMD  VSECT = KEYWRD:400:2

710       GOTO 32767 IF  (KEYWRD% AND 1024%) = 0%
          ! Kick them out unless permanently privileged, i.e., [1,*]
```

One word of warning concerning the use of the VSECT directive. It is available under V7.0—7.2 of the RSTS task builder. Since it is NOT documented, there is no guarantee that DEC will not remove it from a future implementation of the task builder. Also, although the above uses do, in fact, work and are implemented at other RSTS sites, I have never seen a DEC program use these techniques. (Then again I have seen very few DEC programs coded in BP2). Futhermore, the enclosed program listings are believed to be correct, (I hope I have submitted the correct versions), but the author takes no responsibility for any programming errors. I hope that you find these routines helpful.

```
        .TITLE  KBNUM
        .IDENT  /V1.0/
        .ENABL  LC
        .PSECT  KBNUM,RO,OVR

;+
;       Project     : In-House
;       Program     : KBNUM.MAC
;       Author      : Ed Heinrich
;       Function    : Return keyboard number and, optionally, Installation name
;       Edit Date   : 16-Jun-82
;       Edit Level  : V1.0
;-
KBNUM::
        MOV     #FIRQB,R4       ; Address of FIRQB
        MOV     #FQBSIZ/2,R3    ; Length of FIRQB
1$:     CLR     (R4)+           ; Clear a word
        SOB     R3,1$           ; While some there

        MOVB    #16,@#FIRQB+3   ; Function code for return error
        MOVB    #0,@#FIRQB+4    ; Error # 0 (Installation name)
        CALFIP                  ; Call RSTS
        CLR     @2(R5)          ; Clear out integer
        MOVB    @#FIRQB+3,R3    ; Get the keyboard number into R3
        BIC     #^C377,R3       ; Only low order byte
        ASR     R3              ; Divide by 2
        MOV     R3,@2(R5)       ; Put it in linkage register
        CMPB    #1,(R5)         ; How many parameters passed
        BEQ     3$              ; Only 1 then boogie

        MOV     #34,R2          ; Maximum bytes of data
        MOV     #FIRQB+4,R3     ; Location of data
        MOV     4(R5),R4        ; Address of string header
        MOV     (R4),R4         ; Address of string

2$:     MOVB    (R3)+,(R4)+     ; Get some work done
        SOB     R2,2$           ; While something to do

3$:     RTS     PC              ; Snag return address off PC
        .END
```

```
        .TITLE  CAP
        .IDENT  /V1.00/
        .ENABL  LC
        .PSECT  CAP,RO,OVR
;+
;       Project         : In-House
;       Program         : CAP.MAC
;       Author          : Ed Heinrich
;       Function        : Capitalize 1st letter in word
;       Edit Date       : 19-Jun-82
;       Edit Level      : V1.0
;-

CAP::   MOV     2(R5),R0        ; Address of string descriptor
        MOV     2(R0),R1        ; Length of string
        MOV     (R0),R2         ; First character in string
        MOV     R5,-(SP)        ; Save R5

        CLR     R0              ; Scratch register
        CLR     R5              ; Scratch register
        MOV     R2,R4           ; Address of output string
        MOVB    (R2)+,R3        ; Load up the address of 1st byte
        BR      3$              ; And branch to check for '\'

1$:     MOVB    (R2)+,R3        ; Put chars into a scratch register
        TST     R0              ; Just found a space?
        BNE     2$              ; Yes so skip this check
        CLR     R0              ; Zap the flag
        CMPB    R3,#40          ; Look for a space
        BNE     2$              ; None there so branch
        MOV     #177777,R0      ; Set flag to -1
        BR      5$              ; And jump

2$:     TST     R0              ; Last Char a space
        BEQ     5$              ; No then skip convert

3$:     CMPB    R3,#134         ; 1st letter a '\' ?
        BNE     4$              ; No so continue
        CLR     R0              ; Else clear the flag
        TSTB    (R3)+           ; Pop a character
        INC     R5              ; Number of chars to pop
        BR      6$              ; And complete the loop

4$:     CLR     R0              ; Unset the flag
        CMPB    R3,#90.+32.     ; Look for lower case z
        BHI     5$              ; Too high then branch
        CMPB    R3,#65.+32.     ; How about lower case a
        BLO     5$              ; Nope, then branch
        BICB    #32.,R3         ; Yep so change it

5$:     MOVB    R3,(R4)+        ; Write out new string

6$:     SOB     R1,1$           ; For full length
        TST     R5              ; Anything there
        BEQ     END             ; Nope then nothing to do

8$:     MOVB    #40,(R4)+       ; Get rid of the junk
        SOB     R5,8$           ; While some

END:    MOV     (SP)+,R5        ; Restore R5
        RTS     PC              ; Exit subroutine

        .END
```

```
        .TITLE  XCONTG
        .IDENT  /V1.0/
        .ENABL  LC

;+
;       Project         : In-House Development
;       Program         : XCONTG.MAC
;       Author          : Ed Heinrich
;       Function        : Make a full contiguous file non-contiguous
;       Edit Date       : 02-Aug-82
;       Edit Level      : V1.0
;       Call Format     : CALL XCONTG(CHANNEL%)
;       Assembly Format : MAC XCONTG=[1,2]COMMON,XCONTG
;
;       Program operation: XCONTG is called from a Basic Plus 2 module which
;       passes the channel number on which the file to unset the contiguous bit
;       is opened.  XCONTG, reads through the job tables to obtain the FCB
;       for the channel number, follows the FCB links to obtain the needed
;       information on the file, and finally does a .UUO to unset the bit.
;       If the .UUO was successful, CHANNEL% is returned as zero.  If any
;       errors were encountered CHANNEL% contains the error code.
;       Note that the calling program must be privileged to execute the
;       PEEKs of XCONTG.
;-

        .GLOBL  XCONTG

        .PSECT  XCONTG,RO,OVR

;
;               Begin by obtaining monitor tables
;

XCONTG: JSR     PC,CLNXRB               ; Go zero the XRB and FIRQB
        MOVB    #UU.TB2,FIRQB+FQFUN     ; Set monitor tables part II function
        .UUO                            ; And call RSTS
        MOV     FIRQB+6,R0              ; Address of device name table
        MOV     FIRQB+12,R1             ; Address of DEVOKB
        ASR     R1                      ; Divide it by 2
        MOVB    #UU.TB1,FIRQB+FQFUN     ; Set monitor tables part I function
        .UUO                            ; And do it
```

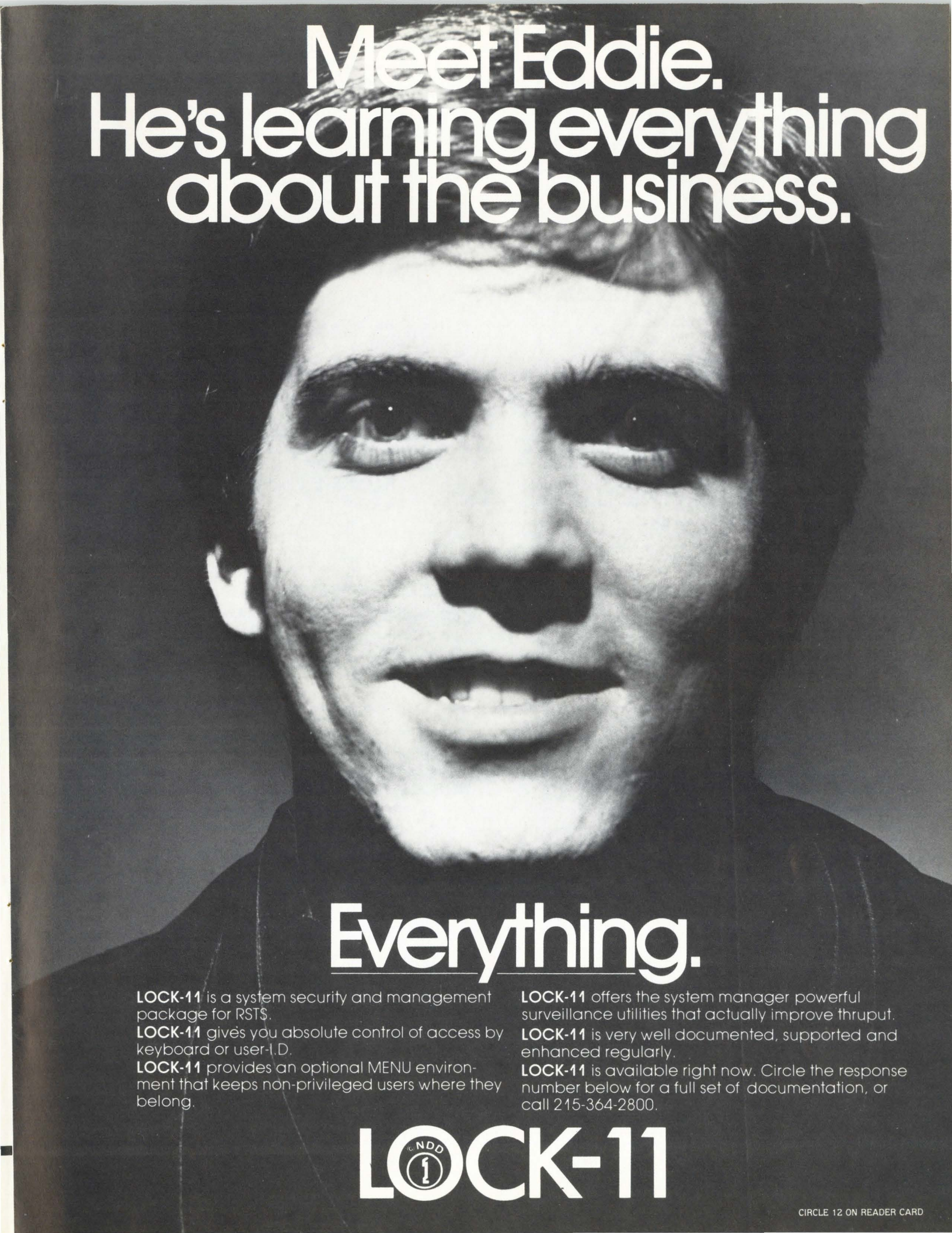
```
        MOV     FIRQB+6,R2              ; Address of device count table
        MOV     #DISK,R3                ; Address of storage area
        CLR     R4                      ; A scratch register
        PUSH    R5                      ; Push linkage register on the stack
        CLR     R5                      ; And zero it

;
;           Save all configured disk devices in location 'DISK'
;

1$:     MOV     R2,XRB                  ; Put address of devcnt in XRB
        .PEEK                           ; And peek at it
        TST     XRB                     ; What is returned by .PEEK
        BLT     3$                      ; Negative means there "Ain't none"
        MOV     XRB,R5                  ; Save the count
        MOV     R0,XRB                  ; If 'some' get devnam base address
        .PEEK                           ; And peek at it to get name

2$:     MOV     XRB,(R3)+               ; Save disk mnemonic
        MOVB    R5,(R3)+                ; And Unit number
        CLRB    (R3)+                   ; Leave next byte blank
        INC     R4                      ; Count total number saved
        DEC     R5                      ; Decrement count
        BGE     2$                      ; Do it for all units

3$:     ADD     #2,R2                   ; Increment devcnt pointer
        ADD     #2,R0                   ; And devnam for next lookup
        SOB     R1,1$                   ; Do it for all possible disk devices

;
;           G e t   t o   F i l e   C o n t r o l   B l o c k
;

        JSR     PC,CLNFQB               ; Go zap the FIRQB
        POP     R5                      ; Get back R5 from the stack
        MOV     @2(R5),R3               ; Channel file was opened on
        ASL     R3                      ; Multiple by 2
        MOV     #1010,XRB               ; Address of current job's JDB
        .PEEK                           ; Point to it
        .PEEK                           ; Now point at IOB for current job
        ADD     R3,XRB                  ; Add the channel number * 2
        .PEEK                           ; And get the WCB for this channel
        MOV     XRB,R1                  ; Save address of WCB in R1
        ADD     #10,XRB                 ; Offset to the FCB is 8
        .PEEK                           ; To get FCB for this file
        MOV     XRB,R2                  ; Better save this address or else

;
;                   L o a d   t h e   F I R Q B
;

        SUB     #26,XRB                 ; Point to filename word 1
        .PEEK                           ; And get it
        MOV     XRB,FIRQB+FQNAM1        ; Put it in FIRQB at offset 10
        MOV     R2,XRB                  ; Point XRB to FCB address
        SUB     #24,XRB                 ; Point to file name word 2
        .PEEK                           ; See what's there
        MOV     XRB,FIRQB+FQNAM2        ; Save word 2 in FIRQB
        MOV     R2,XRB                  ; Again restore the FCB address
        SUB     #22,XRB                 ; Point to extension
        .PEEK                           ; Peek at it
        MOV     XRB,FIRQB+FQEXT         ; Move the file type to the FIRQB
        MOV     R2,XRB                  ; Put back the FCB once again
        SUB     #30,XRB                 ; Now point to PPN
        .PEEK                           ; Peeking
        MOV     XRB,FIRQB+FQPPN         ; And load it in at FIRQB + 6

        MOVB    #UU.FIL,FIRQB+FQFUN     ; Set Function code
        MOVB    @2(R5),FIRQB+4          ; Channel number file opened on
        MOVB    #32.,FIRQB+5            ; And subfunction to unset contiguous
        MOV     R2,XRB                  ; Address of the  FCB
        SUB     #4,XRB                  ; Point to FIP unit number
        .PEEK                           ; And get it

;
;           D e t e r m i n e   D i s k   U n i t

        SUB     XRB,R4                  ; Point to location in disk table
        DEC     R4                      ; Take FIP unit zero into account
        ASL     R4                      ; Point to offset - Since I used two
        ASL     R4                      ; words we need to shift 2 bits
        ADD     #DISK,R4                ; Point to device mnemonic
        MOV     (R4)+,FIRQB+FQDEV       ; Move it to the FIRQB
        MOVB    (R4),FIRQB+FQDEVN       ; Move unit number in
        MOVB    #377,FIRQB+FQDEVN+1     ; Specify use unit number
        .UUO                            ; The moment we've been waiting for ...

        CLR     @2(R5)                  ; Clear the error flag
        TSTB    FIRQB                   ; Did we do it?
        BEQ     END                     ; Yes, then return to whence we came
        MOV     FIRQB,@2(R5)            ; Nope, #*@%$!, tell 'em why not
        JMP     END                     ; And return to give bad news

;
;           I n t e r n a l   S u b r o u t i n e s
;

CLNXRB: MOV     #XRB,R0                 ; Address of the transfer block
        MOV     #14/2,R1                ; Length of XRB
1$:     CLR     (R0)+                   ; Zero a word
        SOB     R1,1$                   ; Until all done

CLNFQB: MOV     #FIRQB,R0               ; Location of FIRQB into a register
        MOV     #FQBSIZ/2,R1            ; Length of the FIRQB
1$:     CLR     (R0)+                   ; Clear a word at a time
        SOB     R1,1$                   ; Check if we're done

END:    RTS     PC                      ; Universal return

DISK:   .BLKB   20                      ; A little storage

        .END                            ; That's all he wrote
```

```
        .TITLE  TTECHO
        .IDENT  /V1.0/
        .ENABL  LC
        .PSECT  TTECHO,RO,OVR
;+
;       Project         : In-House
;       Program         : TTECHO.MAC
;       Author          : Ed Heinrich
;       Function        : Enables / Disables terminal echo
;       Edit Date       : 26-May-82
;       Edit Level      : V1.0
;       Assembly Format : MAC TTECHO = COMMON$,[P,PN]TTECHO
;
;       Call Format     : FROM BP2 - CALL NOECHO for no echo
;                                    CALL ECHO for echo
;-

NOECHO::
        MOV     #3,@#XRB+0              ; Disable echo for caller
        BR      LOAD                    ; And do it up

ECHO::
        MOV     #2,@#XRB+0              ; Re-enable echo

LOAD:   CLR     @#XRB+2                 ; Only on callers terminal
        CLR     @#XRB+4                 ; No bytes to send or force
        MOVB    #24.,@#XRB+6            ; Channel 12. only
        MOVB    #2,@#XRB+7              ; Handler index for terminal
        CLR     @#XRB+10                ; Zap the rest of the XRB
        CLR     @#XRB+12                ; As good programming would dictate
        CLR     @#XRB+14                ; Until all done
        .SPEC                           ; And process our request

        RTS     PC                      ; Return to whence we came
        .END                            ; So simple yet useful
```

CALC.BAS

By Craig Goodrich, Bechtel Power Corp., Gaithersburg, MD

```
1       EXTEND
10!                                                                     &
!                                                                       &
!               T E M P L T                                             &
!                                                                       &
!                                                                       &
!               FILE NAME:      CALC.BAS                (SYSTEM NAME)   &
!               REVISION: 1.1   21-Jan-82       CG                      &
!               CREATION:       21-Jan-82                               &
!               AUTHOR:         CGoodrich       Bechtel Power           &
!                                                                       &
!                                                                       &
!                                                                       &
!               M O D I F I C A T I O N   H I S T O R Y                 &
!                                                                       &
!               V/E     DATE    PRGMR           PURPOSE                 &
!                                                                       &
!                                                                       &
!                                                                       &
!               P R O G R A M   D E S C R I P T I O N                   &
!                                                                       &
!                                                                       &
!               Allows use of terminal as desk calculator w/12 memories &
!               Parsing & evaluation algorithms based on ch.3 of        &
!                       McKeeman, Horning, & Wortman, "A Compiler Generator"  &
!                               (Prentice-Hall 1970)                    &
!                                                                       &
!                                                                       &
!               Requires B+ or B+2 RTS with LOG10(), SIN(), ATN() functions.  &
!                                                                       &
!               CCL entry at line 30000 with optional /n switch, where n is   &
!                       the number of decimal places to display.        &
!                                                                       &
!               This software is copyrighted by Bechtel Power Corp.     &
!                       and is provided to the RSTS users' community    &
!                       for its use without charge, provided that       &
!                       its source is noted in any copying hereof.      &
!                                                                       &
!               No commitment whatever to the support or reliability    &
!                       of this software is hereby made.                &
!                                                                       &
!               I / O   C H A N N E L S                                 &
!                                                                       &
!               CHANNEL         USED FOR:                               &
!                                                                       &
!               **      N O N E         **                              &
!                                                                       &
!                                                                       &
!               V A R I A B L E   D E F I N I T I O N S                 &
!                                                                       &
!               VARIABLE        DEFINITION                              &
!                                                                       &
!                                                                       &
!                                                                       &
!               F U N.C T I O N S   A N D   S U B R O U T I N E S       &
!                                                                       &
!                                                                       &
!                                                                       &
!       LINE    FNCTN NAME      DESC                                    &
!       ----    ----------      ----                                    &
!
900!                                                                    &
!                                                                       &
!                                                                       &
!               D i m e n s i o n     S t a t e m e n t s               &
!                                                                       &
        DIM     REG.(32%)                                               &
        ,       REFRESH$(16%)   !       Cursor ctl strings              &
!                                       Storage registers               &
\       DIM WEIGHT.%(2%,128%)   !       Operator precidence/jump table  &
\       DIM STACK.%(48%),IN.%(48%),OUT.%(48%)                           &
                !               For reverse-polish conversion routine   &
\       DIM PROC.STK.(16%)      !       Evaluation stack                &
\       DIM FETCHED%(16%)       !       Address-Value flag              &
!                                                                       &
999!                                                                    &
!                                                                       &
!                                                                       &
!               S t a r t   o f   P r o g r a m                         &
!                                                                       &
!
1000    PROGID$="CALC   1.1"    ! version for printing;                 &
\       PRINT IF CCPOS(0%)              ! clear cursor if left in middle &
\       Z$ = SYS(CHR$(6%)+CHR$(-7%))    ! Ctrl c trap                   &
!                                                                       &
\       ON ERROR GOTO 19000                     ! error trap.           &
\       PRINT PROGID$, TIME$(0%), DATE$(0%) UNLESS CHAIN.IN%            &
                        ! identify program.                            &
\       START.T = TIME(1%)                                              &
!                                                                       &
\       PLACES.% = 4% UNLESS PLACES.%                                   &
!                                                                       &
\       GOSUB 18070                                                     &
\       P.U$ = "###,###,###,###.########"                               &
!                                                                       &
\       Z$ = LEFT(P.U$, INSTR(1%, P.U$,".")) + STRING$(PLACES.%, ASCII("#"))  &
\       LSET P.U$ = Z$                                                  &
!               Pick how many places                                    &
\       GOSUB 10100     !       Init parser table                       &
\       GOSUB 18050 IF SCOPE.%  !       Init cursor ctl                 &
\       IN.POS$ = FNCURSOR.$(20%,4%) + ER.EOL$ IF SCOPE.%               &
\       DSP.STG$ = FNCURSOR.$(24%,2%)+ER.EOL$ IF SCOPE.%                &
\       GOSUB 15000     !       Init display                            &
\       PRINT SCR.INIT$ IF SCOPE.%                                      &
\       PRINT IN.POS$;                                                  &
!                                                                       &
1100!                                                                   &
!                                                                       &
!               O p e n   F i l e s                                     &
```


Meet Eddie.
He's learning everything about the business.
Everything.
LOCK-11 is a system security and management package for RSTS.
LOCK-11 gives you absolute control of access by keyboard or user-I.D.
LOCK-11 provides an optional MENU environment that keeps non-privileged users where they belong.
LOCK-11 offers the system manager powerful surveillance utilities that actually improve thruput.
LOCK-11 is very well documented, supported and enhanced regularly.
LOCK-11 is available right now. Circle the response number below for a full set of documentation, or call 215-364-2800.
LOCK-11
CIRCLE 12 ON READER CARD

```
!
2000!   M a i n l i n e    P r o c e s s i n g                          &
!                                                                       &
        UNTIL 0%                                                        &
\               PRINT IN.POS$;                                          &
\               INPUT Z$                                                &
\               IN.$ = CVT$$(Z$,-1)                                     &
\               PRINT DSP.STG$+IN.$; IF SCOPE.%                         &
\               GOSUB 12000!    Fix multi-character operators           &
\               GOSUB 10300!    Prune & store constants in input string &
\               GOSUB 10200!    Convert input expression to polish notation &
\               GOSUB 11600!    Evaluate the postfix string             &
\               Z1% = 0%                                                &
\               GOSUB 13000!    Display                                 &
\       NEXT                                                            &
9000!                                                                   &
!                                                                       &
!               E N D    O F    P R O G R A M                           &
!                                                                       &
!
9099    GOTO 32750                                                      &
!       to end                                                          &
!
10000!                                                                  &
!                                                                       &
!         P r o g r a m m e r    D e f i n e d    F u n c t i o n s     &
!                                                                       &
!                        a n d    S u b r o u t i n e s                 &
!                                                                       &
!
10100!  Initialize Tables to determine precidence                       &
!                                                                       &
                IPT.% = 0%                                              &
\               STK.% = 1%                                              &
\               JMP.% = 2%                                              &
!                                                                       &
\       WEIGHT.%(STK.%, Z%) = 16% FOR Z% = ASCII("A") TO ASCII("Z")     &
\       WEIGHT.%(IPT.%, Z%) = 15% FOR Z% = ASCII("A") TO ASCII("Z")     &
\       WEIGHT.%(STK.%, Z%) = 16% FOR Z% = ASCII("0") TO ASCII("9")     &
\       WEIGHT.%(IPT.%, Z%) = 15% FOR Z% = ASCII("0") TO ASCII("9")     &
\       READ Z$ UNTIL Z$ = "**TBLDAT**"                                 &
\       READ Z$                                                         &
\       UNTIL Z$ = "**ENDTBL**"                                         &
\               READ VS%,VI%,JP%        ! Stack val, Inptstr val, Jump tbl &
\               WEIGHT.%(STK.%,ASCII(Z$)) = VS%                         &
\               WEIGHT.%(IPT.%,ASCII(Z$)) = VI%                         &
\               WEIGHT.%(JMP.%,ASCII(Z$)) = JP%                         &
\               READ Z$                                                 &
\       NEXT                                                            &
!       Operators have usual meaning. % = unary LOG10(), @=arctan,$=sin &
!
10131!  Operator, stack weight, input str weight, jump table            &
!                                                                       &
        DATA    "**TBLDAT**"                                            &
,       .,16,15,0                                                       &
,       +,4,3,1                                                         &
,       -,4,3,2                                                         &
,       *,6,5,3                                                         &
,       /,6,5,4                                                         &
,       "^",8,7,5                                                       &
,       (,-7,32,0                                                       &
,       ),-7,-7,0                                                       &
,       =,2,1,6                                                         &
,       %,14,13,7                                                       &
,       @,14,13,8                                                       &
,       $,14,13,9                                                       &
,       "~",-32767,-32767,0                                             &
,               "**ENDTBL**"
10199   RETURN                                                          &
!
10200!  Convert input expression to polish notation                     &
!                                                                       &
!                                                                       &
        STACK.%(0%) = 1%        \       STACK.%(1%) = ASCII("~")        &
\       IN.$ = IN.$ + "~"                                               &
\       OUT.%(0%) = 0%                                                  &
\       SP% = 1%                                                        &
\       OP% = 0%                                                        &
\       CHANGE IN.$ TO IN.%                                             &
\       FOR IP% = 1% TO IN.%(0%)                                        &
\               IF WEIGHT.%(STK.%,STACK.%(SP%)) < WEIGHT.%(IPT.%,IN.%(IP%)) &
                        THEN GOSUB 10210                                &
                        ELSE                                            &
                IF WEIGHT.%(STK.%,STACK.%(SP%)) > WEIGHT.%(IPT.%,IN.%(IP%)) &
                        THEN GOSUB 10220                                &
                        ELSE                                            &
                IF WEIGHT.%(STK.%,STACK.%(SP%)) = WEIGHT.%(IPT.%,IN.%(IP%)) &
                        THEN GOSUB 10230
10209   NEXT IP%                                                        &
\       CHANGE OUT.% TO OUT.$                                           &
!                                                                       &
\       PRINT "ERROR! SP=";SP% IF SP% <> 0%                             &
\       RETURN                                                          &
!
10210!  Stk < Ipt                                                       &
!                                                                       &
        SP% = SP% + 1%                                                  &
\       STACK.%(SP%) = IN.%(IP%)                                        &
\       RETURN                                                          &
!
10220!  Stk > Ipt                                                       &
!                                                                       &
        OP% = OP% + 1%                                                  &
\       OUT.%(0%) = OUT.%(0%) + 1%                                      &
\       OUT.%(OP%) = STACK.%(SP%)                                       &
\       SP% = SP% - 1%                                                  &
\       IP% = IP% - 1%                                                  &
\       RETURN                                                          &
!
10230!  Stk = Ipt                                                       &
!                                                                       &
        SP% = SP% - 1%                                                  &
\       RETURN                                                          &
!
10300!                                                                  &
!               Prune & store constants in input string                 &
!       Starts at end of REG.() array & moves down                      &
!                                                                       &
!       Note: the "registers" are referred to as                        &
!                       REG.(ASCII("A ... Z") AND 31%)                  &
!       rP is always set to PI.                                         &
!       Numerics (constants) in the input expression are loaded starting &
!               in rZ and working downward, so it is possible that a    &
!               sufficiently complex expression might impinge into the  &
!               "user" area rA--rM.  No check for this is performed.    &
!                                                                       &
        C.PTR% = 27%    !       Init constant pointer                   &
\       REG.(ASCII("P") AND 31%) = PI                                   &
!                                                                       &
\       TMP.N$, IN.2$ = ""      !       Init tmp strings                &
!                                                                       &
\       FOR I% = 1% TO LEN(IN.$)                                        &
\               CH.$ = MID(IN.$,I%,1%)                                  &
\               IF FNNUMERIC.%(CH.$)                                    &
                        THEN TMP.N$ = TMP.N$ + CH.$                     &
                        ELSE                                            &
                        IF LEN(TMP.N$)                                  &
                                THEN                                    &
                                GOSUB 11320                             &
                                ELSE                                    &
                                IN.2$ = IN.2$ + CH.$
10305   NEXT I%                                                         &
\       CH.$ = ""                                                       &
\       GOSUB 11320 IF LEN(TMP.N$)                                      &
\       IN.$ = IN.2$                                                    &
\       RETURN                                                          &
!
11320!  Store the numeric                                               &
!                                                                       &
        C.PTR% = C.PTR% - 1%                                            &
\       REG.(C.PTR%) = VAL(TMP.N$)                                      &
\       IN.2$ = IN.2$ + CHR$(C.PTR% OR 64%) + CH.$                      &
\       TMP.N$ = ""                                                     &
\       RETURN                                                          &
!                                                                       &
\       DEF FNNUMERIC.%(Z$) = (INSTR(1%,"0123456789.",Z$) <> 0%)        &
!
11600!                                                                  &
!               Evaluate the postfix string                             &
!                                                                       &
!                                                                       &
        PSP% = -1%      ! Processing stack pointer                      &
\       MAT FETCHED% = ZER      !       Zero fetched flags              &
\       FETCHED%(0%) = 0%                                               &
\       CHANGE OUT.$ TO IN.%                                            &
!                                                                       &
\       FOR IP% = 1% TO IN.%(0%)                                        &
\               Z% = WEIGHT.%(JMP.%,IN.%(IP%))                          &
\               IF Z% = 0% THEN GOSUB 11650                             &
                        ELSE                                            &
                ON Z% GOSUB                                             &
                        11710                                           &
,               11720                                                   &
,               11730                                                   &
,               11740                                                   &
,               11750                                                   &
,               11760                                                   &
,               11770                                                   &
,               11780                                                   &
,               11790
11605   NEXT IP%                                                        &
!                                                                       &
\       RESULT., REG.(ASCII("T") AND 31%) = FNFETCH.(0%)                &
!                                                                       &
\       RETURN                                                          &
!                                                                       &
\       DEF* FNFETCH.(Z%)                                               &
\               IF FETCHED%(Z%)                                         &
                        THEN FNFETCH. = PROC.STK.(Z%)                   &
                        ELSE                                            &
                Z1% = PROC.STK.(Z%)                                     &
\               FNFETCH.,PROC.STK.(Z%) = REG.(Z1% AND 31%)              &
\               FETCHED%(Z%) = -1%
11616   FNEND                                                           &
!
11650!  Put ascii val of pointer into processing stack                  &
!                                                                       &
        PSP% = PSP% + 1%                                                &
\       PROC.STK.(PSP%) = IN.%(IP%)                                     &
\       FETCHED%(PSP%) = 0%                                             &
\       RETURN                                                          &
!
11700!  Fetch operands -- 11700 for 2 operands, 11705 for 1             &
!                                                                       &
        OP.2. = FNFETCH.(PSP%)                                          &
\       PSP% = PSP% - 1%                                                &
!
11705   OP.1. = FNFETCH.(PSP%)                                          &
!                                                                       &
\       RETURN                                                          &
!
11710!  +       Add                                                     &
        GOSUB 11700                                                     &
\       PROC.STK.(PSP%) = OP.2.+OP.1.                                   &
\       RETURN                                                          &
!
11720!  -       Subtract                                                &
        GOSUB 11700                                                     &
\       PROC.STK.(PSP%) = OP.1.- OP.2.                                  &
\       RETURN                                                          &
!
11730!  *       Mul                                                     &
        GOSUB 11700                                                     &
\       PROC.STK.(PSP%) = OP.2.*OP.1.                                   &
\       RETURN                                                          &
!
11740!  /       Div                                                     &
        GOSUB 11700                                                     &
\       PROC.STK.(PSP%) = OP.1./ OP.2.                                  &
\       RETURN                                                          &
!
11750!  ^       Exponentiation                                          &
        GOSUB 11700                                                     &
\       PROC.STK.(PSP%) = OP.1.^ OP.2.                                  &
\       RETURN                                                          &
!
11760!  =       Assignment                                              &
        GOSUB 11705                                                     &
\       PSP% = PSP% - 1%                                                &
\       Z1% = PROC.STK.(PSP%)                                           &
\       REG.(Z1% AND 31%), PROC.STK.(PSP%) = OP.1.                      &
\       FETCHED%(PSP%) = -1%                                            &
```

```
\       Z1% = 0% IF Z1% > (13% OR 64%)  !       Only show 1st 13 registers      &
\       GOSUB 13000     !       Display value                                   &
\       RETURN                                                                  &
!               A (perhaps unexpected) consequence of the algorithms            &
!               used in this program is that assignment statements may          &
!               be embedded -- e.g A=P*(C=D/2) is perfectly legal.  It          &
!               will assign rD/2 to rC and rP*(rD/2) to rA.  BLISS also          &
!               has this facility, according to rumor.                          &
!
11770!  %       Log 10                                                          &
        GOSUB 11705                                                             &
\       PROC.STK.(PSP%) = LOG10(OP.1.)                                          &
\       RETURN                                                                  &
!
11780!  @       Atn (deg)                                                       &
        GOSUB 11705                                                             &
\       PROC.STK.(PSP%) = FNDEG.(ATN(OP.1.))                                    &
\       RETURN                                                                  &
!
11790!  $       Sin (deg)                                                       &
        GOSUB 11705                                                             &
\       PROC.STK.(PSP%) = SIN(FNRAD.(OP.1.))                                    &
\       RETURN                                                                  &
!                                                                               &
\       DEF FNRAD.(Z.) = (Z./360.)*2.*PI                                        &
\       DEF FNDEG.(Z.) = (Z./(2*PI))*360.                                       &

12000!  Fix multi-character operators                                           &
!                                                                               &
        Z% = INSTR(1%, IN.$, "**")                                              &
\       WHILE Z%                                                                &
\               IN.$ = LEFT(IN.$, Z% -1%) + "^" + RIGHT(IN.$,Z% + 2%)           &
\               Z% = INSTR(1%, IN.$, "**")                                      &
\       NEXT                                                                    &
!                                                                               &
\       RETURN                                                                  &
!
13000!  Display                                                                 &
        IF NOT SCOPE.% THEN                                                     &
                PRINT USING SPACE$(20%)+"! ="+P.U$, CHR$(Z1%),PROC.STK.(PSP%)   &
                ELSE                                                            &
                PRINT USING REFRESH$(Z1% AND 31%), PROC.STK.(PSP%)              &
\               PRINT IN.POS$;
13099   RETURN                                                                  &
!
15000!  Display                                                                 &
!                                                                               &
        RETURN UNLESS SCOPE.%                                                   &
\       SCR.INIT$ = CLR.SCRN$                                                   &
 +"  +-*/ () Operators have usual meaning. % = unary LOG10(), @=arctan,$=sin "  &
 +      CHR$(13) + CHR$(10) + "         Registers A-L may be set & used."       &
 +              "  ^Z to exit."                                                 &
 +      CHR$(13) + CHR$(10) + " rP = pi"                                        &
 +      CHR$(13) + CHR$(10) + " Latest result "                                 &
 +      CHR$(13) + CHR$(10) + " is always stored"                               &
 +      CHR$(13) + CHR$(10) + " in rT."                                         &
\       FOR I% = ASCII("A") TO ASCII("M")                                       &
\               PT% = I% AND 31%                                                &
\               COL.% = 40% + (25%*-(I%>70%))                                   &
\               LIN.% = (PT%)*3%                                                &
\               LIN.% = (PT%-6%)*3% IF PT% > 6%                                 &
\               SCR.INIT$ = SCR.INIT$ + FNCURSOR.$(LIN.%,COL.%)+CHR$(I%)        &
\               LIN.% = LIN.% + 1%                                              &
\               REFRESH$(PT%) = FNCURSOR.$(LIN.%,COL.%-14%)                     &
                        + P.U$                                                  &
\       NEXT I%                                                                 &
\               REFRESH$(0%) = IN.POS$ + CHR$(10%)+SPACE$(15%) + P.U$           &
\       RETURN                                                                  &
!
18000!          Standard subroutines and functions go here                      &
!                       in NRTRV?-based programs because                        &
!                       of DB/2's bunnybrained use of high                      &
!                       line numbers                                            &
!                                                                               &
!
18050!  Useful constant strings                                                 &
!                                                                               &
!       For VT52 screen control                                                 &
!                                                                               &
        ESC.$ = CHR$(155%)      !       Escape                                  &
!                                                                               &
\       DEF FNCURSOR.$(LN%,COL%)=ESC.$+"Y"+CHR$(128% OR 31%+LN%)                &
                                +CHR$(128% OR 31%+COL%)                         &
!               Direct cursor addressing                                        &
!               PRINT FNCURSOR.$(LINE%,COL%); to position cursor                &
!                       128% bit set to avoid problems with cursor              &
!                               esc seq in print-using strings, since           &
!                               chr$(31% + 4%) = "#"    v/e 1.2 15-Jan-82 cg    &
!                                                                               &
\       C.HOME.$ = ESC.$ + "H"  !       Home cursor                             &
\       ER.EOS$ = ESC.$ + "J"   !       Erase to end of screen                  &
\       ER.EOL$ = ESC.$ + "K"   !       Erase to end of line                    &
\       CLR.SCRN$ = C.HOME.$ + ER.EOS$  !       Home + Erase screen = Clear     &
!                                                                               &
\       FIP.$ = CHR$(6%)+CHR$(16%)      !       FIP Terminal call               &
\       ESC.SEQ$ = FIP.$ + STRING$(2%,255%)+STRING$(17%,0%)+CHR$(255%)          &
\       NO.ESC.SEQ$ = LEFT(ESC.SEQ$,21%)+CHR$(128%)                             &
!               To set ESC SEQ characteristic, "Z$ = SYS(ESC.SEQ$)"             &
!                       likewise NO...                                          &
!                                                                               &
\       RETURN                                                                  &
!
18070!  S Y S J O B . B A S                                                     &
!                                                                               &
!       v/e     1.3     Get TODAY.% and KBM.$                   CG Sep 82       &
!               1.2     Enhance to get terminal characteristics CG Aug 81       &
!               1.1     Author  CGoodrich       Bechtel             Jul 81      &
!                                                                               &
!       Utility subroutine to return data about current job:                    &
!                                                                               &
!               DET.% = Boolean (job is detached)                               &
!               JOB.% = Job #                                                   &
!               KB.% = job kb#                                                  &
!               KB.WIDTH.% = Current width setting of job kb                    &
!               KBM.$ = Job KBM name                                            &
!               PK.% = Boolean (job kb is pseudokeyboard)                       &
!               PRJ.% = Project                                                 &
!               PRG.% = Programmer #                                            &
!               RTS.$ = Run-time system name                                    &
!               SCOPE.% = Boolean (job kb is a tube)                            &
!               TODAY.% = Today in RSTS format                                  &
```

. . . continued on page 69

DIBOL and MACRO: Oh, Yes, You Can!

By M. Christopher Getting, JBM Group,
King of Prussia, Pa., and
Philip G. Anthony, Fidelity Bank, Philadelphia, Pa.

Q: What language is used at more PDP-11 installations than any other?

A: DIBOL, according to DEC's published figures.

Q: What language has the least access to PDP-11 systems and gets the least support of any language distributed by DEC?

A: Here's a hint. Its name starts with "D" and ends with "L," and it's used at more PDP-11 installations than any other.

The fact is that DIBOL has major advantages for commercial applications over any other language DEC supports. Its predefinition of fields—and the ability to preload those fields—reduces CPU utilization at run time to well below that of BASIC-PLUS or BASIC-PLUS-2. Its record handling gives wide flexibility that other languages achieve only by using RMS, with its high CPU overhead, waste of disk space, and gigantic program size or complicated overlay structure. Program development with DIBOL takes less time than with BASIC, BP2, and DEC COBOL in DEC's own tests with experienced programmers.

Yet DIBOL has been the Ugly Duckling so far as DEC has been concerned. For years, DEC's listings of languages available on PDP-11s have excluded DIBOL. Most local DEC support centers don't have even one software specialist who speaks DIBOL. And perhaps most annoying to DIBOL users, there is no way from within DIBOL to perform functions that are taken for granted by BASIC and BP2 users, such as directory lookups, assigning and reassigning devices, reattaching to a terminal after a detach, even getting the job's own CPU time, device time, and KCTs for accounting purposes. (Admittedly, DEC is getting better about this, especially with V4.5. But they've got a way to go before they provide anything near the range of BASIC-PLUS's SYS() calls.)

And DEC doesn't want DIBOL users to do it for themselves, using MACRO-11 subroutines to enhance the language and provide hooks into the system. There's no equivalent to the section in the BASIC-PLUS-2 RSTS Users Guide that details BP2's calling sequences and internal structure. One knowledgeable Deccie said frankly that Digital just doesn't want to be put in the position of supporting a "nonstandard" language interface.

Well, it really isn't any more difficult under DIBOL than it is under BP2. In fact, it's simpler, since all DIBOL fields and literals are just alpha strings. Decimals are distinguished solely by the fact that the only characters DIBOL permits in them are the ASCII characters "0" through "9" and "p" through "y" the latter only in the rightmost byte to indicate that the number represented is negative. More about literals in a little bit.

Here's how it's done. Arguments are pushed onto a stack referenced by R5, just the way they are in BP2 and FORTRAN. Unlike those languages, though, DIBOL reverses the argument stack — information about the last argument is found lowest on the stack, then the next to last, and so on until the first argument is reached (see Figure 1).

In detail, the lowest word on the R5 argument stack, the one R5 points to, is the number of arguments times two (there are two words of information for each DIBOL argument). Next is the address of the first byte of the last argument. Then comes the length of the last argument. If the argument is an array name, the passed length is the length of one element of the array; number of elements is not available. The high-order bit, bit 15, of this word is turned on to indicate a literal rather than a field or record handy to make sure you're not writing into a literal, which could give some surprising results. This convention explains the length limitation on DIBOL alpha fields. The sequence of address-length is repeated for each argument back to the first.

With this information, it's relatively easy to write a macro to retrieve any desired argument. The one we use is shown in Figure 2. It places the address of the first byte of the DIBOL argument in R0 by default and the length in R1, but this can be changed if those registers are in use by merely specifying two other registers in the macro call. The length is moved into R1 last to provide easy error checking against writing into a literal: The instruction

```
BMI     LABEL
```

immediately following the macro transfers program execution to LABEL if a literal has been passed.

If one of the passed arguments is decimal, a further step may be necessary: conversion from ASCII to a binary value that MACRO-11 can handle on input, or from binary to ASCII on the way out. Such a conversion routine would be necessary, for instance, in a MACRO-11 subroutine to return the size of the last opened file. Fortunately for those of you who don't want to write your own, the system provides a series of conversion routines in SYSLIB.OLB, which is included on the distribution. They're detailed in the IAS/RSX-11 System Library Routines Reference Manual. One and two-word binary values are converted to ASCII via $CBDMG and $CDDMG, while an ASCII string can be transformed to one or two-word binary using $CDTB and .DD2CT. All of these require the programmer to keep track of negatives; they were designed for terminal input and output, not to help DIBOL users.

With all this in mind, we're ready to start interfacing MACRO-11 and DIBOL. GLEN in Figure 3 is a quick subroutine that emulates the BASIC/BP2 LEN function. It takes two arguments: the field to be examined, which may be alpha, decimal, or literal; and the return field, which must be decimal, five bytes long because of the demands of $CBDMG. Such a subroutine can be called from within a DIBOL subroutine to determine whether a passed field is long enough to contain the information to be returned. All it does, of course, is examine the data on the R5 stack, convert the length word to ASCII, and pop it into the second-argument field.

Error handling may look confusing at first, but it simply traps the routine to the appropriate DIBOL error number — in this case Error 6, "Incorrect number of arguments," or Error 8, "Writing into a literal" — and lets the DIBOL RTS or RESLIB issue its message. It then bombs the program with EMTs 50, to switch to the user's default RTS, and 46, to exit to the system default RTS if 50 fails. Care should be taken with DIBOL error handling, though. Some errors can't be handled this simply; specifically, we've had trouble with Error 31, "Argument wrong size." It may be worthwhile to write your own "fatal error" handler that puts out your own error message (using the next subroutine) and issues EMTs 46 and 50.

Hooking into RSTS isn't very much more difficult than this. The RSTS System Directives Manual provides full information on calling sequences for RSTS EMTs and .UUO subfunctions, along with a cross reference to the corresponding BASIC-PLUS function or SYS() call. In most cases, all that's necessary is to load the appropriate bytes or words of FIRQB or XRB, as shown in the manual, and then issue a MACRO-11 EMT instruction. Figure 4 shows a slightly abbreviated version of one of our favorites, PRINT, a simple subroutine to permit output on Channel 0 — always open but normally forbidden to DIBOL users. We've used this one to decrease program size by not requiring an I/O buffer in memory for terminal output when the calling program is only issuing informational messages at various points in processing.

Finally, a couple of caveats. While PRINT is a perfectly usable subroutine, it has a limited purpose. In general, MACRO-11 subroutines should not be used for I/O, since there may be interference from DIBOL's own I/O mechanisms — which are admirably suited to their purpose in any case. Specifically, we've found that a subroutine to read on Channel 0 produces somewhat undesirable results.

Second, DIBOL itself uses the first 1000-octal bytes of an executing program for its own purposes. Thus, a .UUO function that returns information in FIRQB or XRB will operate properly, but the information may no longer be there if a DIBOL statement intervenes between the MACRO subroutine's asking RSTS for the data and the information's being returned.

Third, users should be very careful about calling DIBOL subroutines from a MACRO subroutine. It's completely possible to set up a stack within a MACRO program that would appear valid to the called DIBOL subroutine, but other structures within DIBOL may become corrupted in the process.

Last, a general-purpose warning: the DIBOL argument stack structure we've described is not supported by DEC and may never be. It's unlikely that Digital will change things around at this late date — they'd have to rewrite a lot of code to do it — but they've done flaky things before now and may just decide at some point to switch this too. In line with their current policy of making everything look like

VAX — even RSTS (what is DCL good for, anyway?) — it's barely possible that they might make up their minds to "unify" calling procedures. If they do, they certainly won't alter the BP2 calling sequence — and that leaves them only one option. So while these routines have been checked out through DIBOL V4.5, there's no guarantee that V4.6 or V5.0 will work the same way. Check out new releases for yourself before assuming that any user-provided MACRO subroutines will still work.

For all these warnings, we've found that MACRO-11 subroutines enhance DIBOL's capabilities immensely. They provide the user with options that can simplify the hardest job in the business — commercial programming — and permit the fullest use of RSTS's capabilities to get the work done. Enjoy!

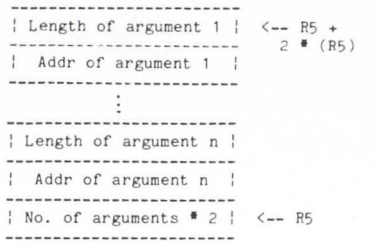


FIGURE 1. Layout of the DIBOL Argument Stack

```
        .NLIST
;
;ARGD.MAC
;
;MACRO-11 routine to extract the address
;and length of a DIBOL argument from the
;R5 argument stack
;
;ARGD takes one required argument and two
;optional arguments:
;
;       ARGN    = Number of argument to return
;       RG0     = Register for address of the
;                 argument (default R0)
;       RG1     = Register for length of the
;                 argument (default R1)
;
;Since the length move is done last, the in-
;struction
;
;       BMI     LABEL
;
;transfers execution to LABEL if the argument
;is a literal (to permit error trapping).
;
;Copyright (C) 1982 JBM Group
;
;This routine is made available free of charge
;to the RSTS community, with no warranty,
;express or implied, with inclusion of the
;above copyright notice.  The authors make no
;commitment that the information contained herein
;is now or will in the future be correct.  How-
;ever, it works on two PDP-11s running three
;versions of DIBOL-11 under RSTS.
;
;RSTS, PDP-11, MACRO-11, and DIBOL-11 are
;proprietary trade names of Digital Equipment
;Corporation.
;
        .MACRO  ARGD    ARGN,RG0=R0,RG1=R1
        MOV     ARGN,RG0
        MOV     (R5),RG1        ;No. of arguments
        INC     RG1             ;No. arguments + 1
        ASL     RG1             ;Times 2 for word
        ADD     R5,RG1          ;End of list + 2
        ASL     RG0             ;Arg no. * 2 for word
        ASL     RG0             ;2 words per argument
        SUB     RG0,RG1         ;RG0 contains a
                                ;negative offset
        MOV     (RG1)+,RG0      ;Address of argument
        MOV     (RG1),RG1       ;Length of argument
        .ENDM   ARGD

        .LIST
```

FIGURE 2. Macro to Return a DIBOL Argument

```
        .LIST   TTM
        .NLIST  TOC
        .ENABL  LC
        .TITLE  GLEN
        .IDENT  /M1.01E/

;Subroutine to return the length of a DIBOL
;field.  To use, XCALL GLEN(FLD,LEN), where
;FLD is the field to be checked and LEN is
;a five-byte decimal field (size is required
;by $CBDMG) for returning the length.
;
;Assemble with ARGD.MAC

        .PSECT  SUBRI   RO,I,LCL,CON,REL

GLEN::  MOV     R0,-(SP)        ;Save registers R0-R2
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        CMP     (R5),#4         ;Two arguments?
        BNE     ERRARG
        ARGD    #2              ;Get return field data
        BMI     ERRLIT          ;Literal?
        MOV     10(R5),R1       ;Length of passed argument
        MOV     #-1,R2          ;Zero-fill flag for $CBDMG
        JSR     PC,$CBDMG       ;Convert to ASCII
        MOV     (SP)+,R2        ;Restore registers
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        RTS     PC              ;That's it!
ERRARG: TRAP    6.              ;"Illegal number of arguments"
        BR      ERRBYE
ERRLIT: TRAP    8.              ;"Writing into a literal"
ERRBYE: MOV     #402,R0         ;Address of FIRQB
        MOV     #20,R1          ;No. of words in FIRQB
10$:    CLR     (R0)+           ;Clear a word
        SOB     R1,10$          ;Loop back for next
        EMT     50              ;.RTS
        EMT     46              ;.EXIT (if .RTS fails)
        .END
```

FIGURE 3. Subroutine GLEN

```
        .LIST   TTM
        .NLIST  TOC
        .ENABL  LC
        .TITLE  PRINT
        .IDENT  /M1.01E/

;This subroutine prints a message using Channel 0,
;thus saving I/O buffer space in a program when the
;only interaction with the terminal involves print-
;ing informational messages.  To use, XCALL PRINT(MSG),
;where MSG is the message to print.
;
;This example uses no error trapping.  Assemble with
;ARGD.MAC.

        .PSECT  SUBRI   RO,I,LCL,CON,REL

XRB     =       442             ;Address of transfer
                                ;request block
CRLF:   .BYTE   15,12           ;Carr. ret., line feed

PRINT:: MOV     R0,-(SP)        ;Save registers R0-R2
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        ARGD    #1              ;Output string information
        BIC     #100000,R1      ;Clear literal flag bit
        MOV     #XRB,R2         ;Address of XRB
        MOV     R1,(R2)+        ;Length of output
        MOV     R1,(R2)+        ;Again
        MOV     R0,(R2)+        ;Address of output
        JSR     PC,WRITIT       ;Write it out
        MOV     #XRB,R2         ;Now do the same with CRLF
        MOV     #2,(R2)+
        MOV     #2,(R2)+
        MOV     #CRLF,(R2)+
        JSR     PC,WRITIT
        MOV     (SP)+,R2        ;Restore registers
        MOV     (SP)+,R1
        MOV     (SP)+,R0
        RTS     PC              ;Tha-tha-that's all, folks!
WRITIT: .REPT   4               ;Clear remainder of XRB
        CLR     (R2)+
        .ENDM
        EMT     4               ;RSTS .WRITE (see manual)
        RTS     PC              ;Back to PRINT
        .END
```

FIGURE 4. Subroutine PRINT

CALC.BAS . . . continued from page 65

```
!                                                                       &
!                                                                       &
!                                                                       &
        TODAY.$ = SWAP$(CVT$$(MID(SYS(CHR$(6%)+CHR$(-3%)), 27%,2%)))    &
!                                                                       &
\       Z$ = CHR$(6%)+CHR$(26%)+STRING$(28%,0%) ! Set up FIP call string &
\       Z$ = SYS(Z$)                                                    &
!                                                                       &
\       JOB.$ = ASCII(Z$)/2%                                            &
\       PRG.$ = ASCII(MID(Z$,21%,1%))                                   &
\       PRJ.$ = ASCII(MID(Z$,22%,1%))                                   &
!                                                                       &
\       RTS.$ = RAD$(SWAP$(CVT$$(RIGHT(Z$,27%))))                       &
              + RAD$(SWAP$(CVT$$(RIGHT(Z$,29%))))                       &
                                                                        &
\       KBM.$ = RAD$(SWAP$(CVT$$(RIGHT(Z$,23%))))                       &
              + RAD$(SWAP$(CVT$$(RIGHT(Z$,25%))))                       &
!                                                                       &
!               Note: The following checks for detached condition       &
!                       by looking at the 128% bit of the job kb#.      &
!                       If the job is detached, fip 26 returns the      &
!                       one's complement of the kb#; therefore if       &
!                       there are no kb#s above 127 in the system,      &
!                       the 128 bit will indicate detachedness.         &
!                       RSTS 7.0 doesn't allow kb#s above 127 anyway.   &
!                       If future releases do allow KB128: (unlikely),  &
!                       this test may not be reliable.                  &
!                       At present, anyway, the RSTS Programming Manual &
!                       is mistaken in sec. 7.3.2.2 where it asserts    &
!                       that you have to peek around (and therefore     &
!                       have priviledge) to determine detachedness.     &
!                                                                       &
!                       RSTS 7.1 -- They've fixed it, but p.7-153 is    &
!                               misleading about byte 4 (you can't ask  &
!                               BASIC if it's negative). -- CG 16-Mar-82 &
!                                                                       &
\       KB.% = ASCII(MID(Z$,4%,1%))                                     &
\       DET.% = ((KB.% AND 128%) <> 0%)                                 &
\       KB.% = 255% XOR KB.% IF DET.%   ! One-byte complement           &
!                                                                       &
\       Z$ = "  "       \       RSET Z$ = "0" + NUM1$(JOB.%)            &
\       TMP.FIL$ = Z$ + ".TMP"                                          &
!       LOGIN will delete XXXnn.TMP files where nn is the job number -- &
!               set up filenames as "cccc"+TMP.FIL$ after calling this  &
!               routine                                                 &
!                                                                       &
!       Terminal characteristics                                        &
\       IF NOT DET.%                                                    &
                THEN                                                    &
        Z$ = CHR$(6%)+CHR$(16%)+CHR$(0%)+CHR$(255%)+STRING$(28%,0%)     &
\       Z$ = SYS(Z$)                                                    &
\       SCOPE.% = (ASCII(MID(Z$,11%,1%)) = 255%)                        &
\       KB.WIDTH.% = ASCII(MID(Z$,5%,1%)) - 1%                          &
\       PK.% = (ASCII(MID(Z$,19%,1%)) = 8%)                             &
!                                                                       &
18079   RETURN                                                          &
!                                                                       &
\       DEF* FNTMP.FIL$(F$)                                             &
\               GOSUB 18070 UNLESS LEN(TMP.FIL$)                        &
\               Z$ = "*****"                                            &
\               LSET Z$ = F$ + "0000"   !       Make sure len = 4%      &
\               FNTMP.FIL$ = Z$ + TMP.FIL$                              &
\       FNEND                                                           &
```

```
!                       This function call lets you set up a temp filename      &
!                   by simply calling it -- e.g. SCRATCH$ = FNTMP.FIL$("SCR")   &
!                           OPEN SCRATCH$ FOR OUTPUT AS . . .                   &
!                                                                               &
19000!                                                                          &
!                                                                               &
!       E R R O R   T R A P   H A N D L I N G   R O U T I N E                   &
!                                                                               &
!                                                                               &
19005   E%=ERR\ E1%=ERL\        E1%=LINE IF E%=28%                              &
!        RECORD ERROR NUMBER, LINE IN E%,E1%                                    &
!                                                                               &
19010   RESUME 32750 IF E% = 11%                                                &
19020   IF E% = 28% THEN                                                        &
                PRINT "^C isn't very nice."                                     &
\       Z$ = SYS(CHR$(6%)+CHR$(-7%))    ! Ctrl c trap                           &
\               SLEEP 5%                                                        &
\               RESUME 32750                                                    &
!                                                                               &
19025   IF E% = 30000%                                                          &
                THEN                                                            &
                INPUT "Bad switch -- Decimal places"; PLACES.%                  &
\               RESUME 30099                                                    &
!               Generated in VAL at chain entry                                 &
19030   PRINT "**SYNTAX ERROR**"; E%;                                           &
\       SLEEP 5%                                                                &
\       PRINT CHR$(13%)+SPACE$(48%);                                            &
\       RESUME 2000                                                             &
19099   PRINT\  PRINT "** ERROR ** !"                                           &
\       PRINT "???"; IF PK.%    ! Signal BATRUN that an error has occurred      &
\       PRINT RIGHT(SYS(CHR$(6%)+CHR$(9%)+CHR$(E%)),3%);                        &
        "(";E%;") at line";E1%;"of '";PROGID$;"'"                               &
\       PRINT "Abnormal "; \    STOP                                            &
                                                                                &
19999!                                                                  \       &
!                                                                               &
!               S T A N D A R D   S U B R O U T I N E S   A N D                 &
!                                                                               &
!                               F U N C T I O N S                               &
!                                                                               &
!                                                                               &
30000!  C h a i n   E n t r y   P o i n t                                       &
!                                                                               &
        C.COM$ = CVT$$(SYS(CHR$(7%)), 255%)                                     &
\       CHAIN.IN% = -1%                                                         &
\       Z% = INSTR(1%, C.COM$, "/")     !       Switch?                         &
\       IF Z% = 0%                                                              &
                THEN                                                            &
                PLACES.% = 4%                                                   &
                ELSE                                                            &
                PLACES.% = VAL(RIGHT(C.COM$, Z% + 1%))                          &
\               PLACES.% = 8% IF PLACES.% > 8%                                  &
!                                                                               &
30099   GOTO 1000                                                               &
!                                                                               &
31000!  D A T A     S T A T E M E N T S                                         &
!                                                                               &
!                                                                               &
32750   PRINT CLR.SCRN$ IF SCOPE.%                                              &
\       PRINT PROGID$,TIME$(0%);                                                &
\       CLOSE Z% FOR Z% = 1% TO 12%                                             &
\       PRINT USING "       ####.# CPU seconds",  (TIME(1%)-START.T)/10.       &
!                                                                               &
32767   END                                                                     &
```

Dear RSTS Man

. . . continued on from page 36

small buffers to contain the characters being input and output. The figures cited were 500 microseconds CPU time per character output. That sounds incredibly high to me, but I do not know all that is involved with terminal I/O under RSTS.

We have a mixture of DZ's and DH's on our 3 machines (two 11/70's and one 11/45). For a while those on the programming staff who chose to do so and were also attached to a computer via a DH with its 30 character buffer, were allowed to run their terminals at 9600 baud. Now management has decided that to avoid performance degradation, we all shall run at less than 9600 baud, preferrably 2400.

Unfortunately EDT is my favorite editor and running it at low speeds is like driving a Ferrari through an endless school zone. I enjoy EDT so much that I'm using it to produce this letter, even though I have WORD11 available to me.

Our business is a small time-sharing service bureau and therefore we are understandably disk-bound. How much of an affect does running terminals (and a small number of them at that) at high baud rates actually have? Could you also please explain the reaction between the CPU and DZ/DH on terminal I/O (especially regarding handling of interrupts)?

Thank you very much.

Hans Hazelton
Confused Programmer
Juneau, AK

Dear Confused: *DZ's require the CPU for each character of both input and output. DH's on RSTS, require service for each input character, but only for every 30 characters of output. DH's could theoretically avoid single character input, but then they would have to be sensitive to delimiters. To avoid that problem, the silo limit is set to one character.*

I have two recommendations:

1) Sell your DZ's and get DH's.

2) Settle on 4800 for tubes as a good compromise.

I have seen user satisfaction increase when changing from 9600 to 4800 simply becuase the 'stuttering' was eliminated.

♥

LETTERS to the RSTS Pro...

continued from page 6

Thus overall we an elapsed-time saving of 37.25% and a run-time saving of 54.75%. I trust that you will agree that statistics of this nature are worthwhile publishing.

It would be ungracious of me not to state that I could not have accomplished this conversion operation without the active support of Software Techniques Limited and their TASKIT product which decompiles the .BAC programmes to create a .B2S programme which is then converted by CSPCOM and TKB into the RSX tasks.

Jeffrey Seymour, Director
Pegasus Computer Processing

```
RUN
TSK11M  03:19 PM          26-Jan-83
                --ELAPSED TIME--          ---RUN TIME----
                OLD  NEW SAVING           OLD NEW SAVING--

GC 5210          145  85  41.3793          5857  1998  65.887
GC 5812          202  162  19.802          6958  4180  39.8534
GC 5446          177  70  60.452           10222  3882  62.0231
GC 3518          176  133  24.4318         14713  6994  52.6435

GC 0             700  450  35.7143         36750  16559  54.9415


BL 3210          94  51  45.7447           4180  1658  60.3349
BL 3662          9  7  22.2222             1402  378  73.0485
BL 3220          94  51  45.7447           4180  1658  60.3349
BL 3712          51  41  19.6078           2840  2071  27.0775
BL 3432          87  75  13.7931           4384  3586  18.2026
BL 3632          57  30  47.3684           3155  1503  52.3613

BL 0             701  429  38.8017          36887  16962  54.0163

Ready

BYEY
Logged off with 448 blocks in use
Job 7 [1,21] KB15 26-Jan-83 03:19 PM
System RSTS V7.2-04 PCP72C
Run time : 6 seconds
Elapsed time : 1 minute
Good afternoon
```

* * *

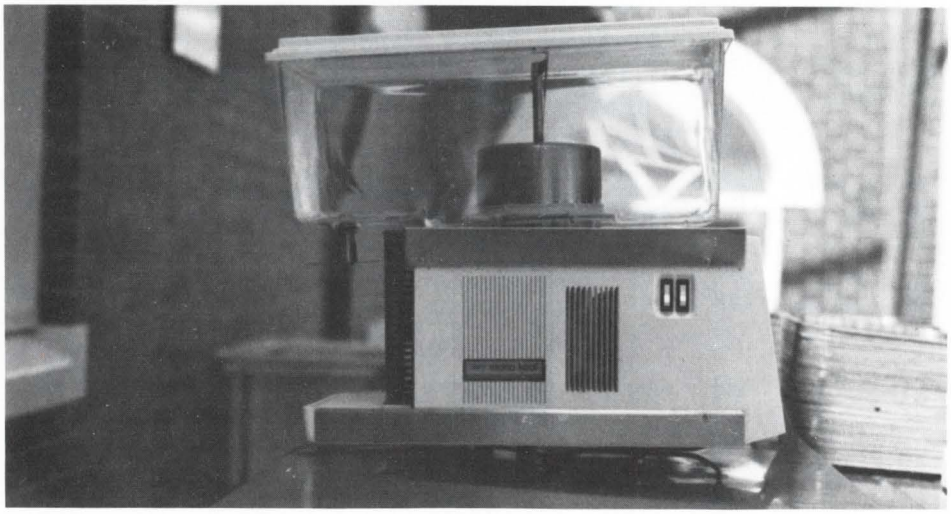
Please find enclosed another "TECO' photo — unfortunately not very clear, and probably in the U.S., anyway: "TECO MONO KOOL". All sorts of paraphrasing is possible:

"KEEP COOL WITH TECO";
"COOL TECO";
"TECO
THE ONE TO KEEP COOL WITH".

Your magazine has been an invaluable aid for our small installation where the system manager has to be a "master" of all trades. However, many ideas have been gleaned from "RSTS Pro" for courses we run with our secondary students and evening adult classes, and to first keep things going a bit better than otherwise would be possible.

Hopefully I can persuade some of my students to get round soon to dropping you a line on some of the things they are doing with the help of RSTS/E.

Bernard Halpin, Centre Manager
BCC Computing Centre
Bunbury, W.A.



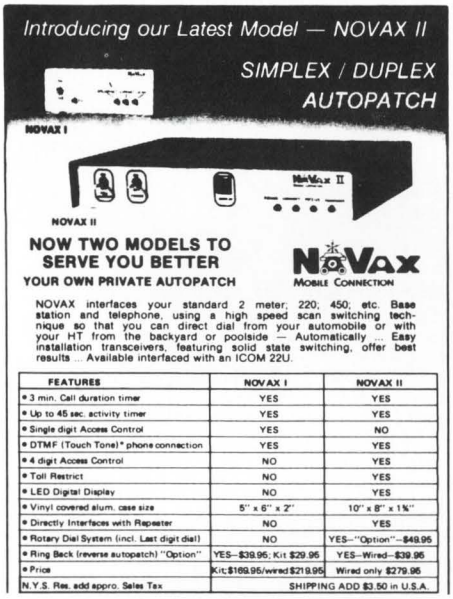
* * *

Thanks for the 'T' shirt. The "Novax II" is a device for interconnecting the switched to telephone network to a two-way radio circuit, half duplex over a single radio frequency. It has been advertised recently in amateur radio publications, and I have enclosed a copy of the ad.

Perhaps the most exciting thing is that the 'Novax II" is sold by C.D.C. (in this case, Current Development Corporation). When I first saw it, I thought maybe our friends from Minneapolis had figured out a way for me to emulate VMS on my 11/70!

I enjoy your publication, and look forward to each and every issue.

Paul E. Anderson, Vice President
GALLO SALES COMPANY, INC.



This is the answer to the question! See p. 32 of Last Issue.

* * *

I would like to add my accolades to the many others who have stated their excitement over your magazine.

In Volume 3, Number 3, September, 1981 there was a program for a Corrupt Disk which saved us untold hours of work.

We got an "Irrevocably Corrupt Disk" at approximately 3:00 p.m.; the local Digital software office was contacted. Their response was that only the boys in Massachusetts could recover the disk. RDC was contacted and they had no idea how to recover the information on our disk.

"CORRUPT" was then remembered, located, run, and within one hour the corrupted directory structure was rebuilt.

Keep up the good work and keep those programs and articles coming. As a final thought, this magazine should be standard issue to all those Digital software experts who charge customers big bucks for analyzing and recommending.

Logan A. Ragan
Manager/Computer Services
Bridge Brand Food Services Ltd.
Alberta, CAN.

* * *

The Dec. '82 article by Michael H. Koplitz ("How Do You Read RSTS/E Monitor Tables?") was just what I've been looking for.

With it, I was able to write a program that would periodically examine the job tables to see if any privileged jobs are present. Being that I am in a high school environment, no one should be listed that I don't know about (Students do NOT have privileged accounts since certain school system administrative functions are done on our PDP 11/34).

Missing from my program is the ability to list the KB: at which the particular jobs are running. I don't believe there was anything in the article and subsequent program that explained how to do that. If someone could explain how to do that, it would be greatly appreciated.

Although the program I wrote is not (and was not intended to be) a ". . . Great Piece of Art . . .", I've included it.

PEEKPR.BAS

```
5 ON ERROR GOTO 5000
10 EXTEND
20 OPEN "SPY.PRI" FOR OUTPUT AS FILE #1%
50 JOBTBL% = SWAP%(CVT$%(MID(SYS(CHR$(6%)+CHR$(-3%)),11%,2%)))
90 JOB.NUMBER% = JOB.NUMBER% + 1%
91 SLEEP 60 IF PEEK(JOBTBL% + (JOB.NUMBER% * 2%)) = -1%
92 JOB.NUMBER% = 1% IF PEEK(JOBTBL% + (JOB.NUMBER% * 2%)) = -1%
94 GOTO 90 IF PEEK(JOBTBL% + (JOB.NUMBER% * 2%)) = 0%
100 JDBPTR% = PEEK(JOBTBL% + (JOB.NUMBER% * 2%))
200 JDB2% = PEEK(JDBPTR% + 8%)
260 J2NAME1 = PEEK(JDB2%+12%)
262 J2NAME2 = PEEK(JDB2%+14%)
280 J2PPN% = PEEK(JDB2%+24%)
282 PROJ% = SWAP%(J2PPN%) AND 255%
284 PROG% = J2PPN% AND 255%
400 IF PROJ% = 1 THEN IF JOB.NUMBER% <> 1 THEN IF RAD$(J2NAME1)+RAD$(J2NAME2)<>"
PEEKPR" THEN IF RAD$(J2NAME1)+RAD$(J2NAME2)<>"VT50PY" THEN GOSUB 2000
500 IF PROJ% = 10 AND PROG% = 0 THEN GOSUB 2000
1000 GOTO 90
1999 STOP
2000 REM File for storage
2050 PRINT #1%, JOB.NUMBER%;"[";PROJ%;PROG%"]",RAD$(J2NAME1)+RAD$(J2NAME2);" ";TI
ME$(0%);TIME(0%),DATE$(0%)
2100 RETURN
5000 CLOSE 1%
9999 END
Ready
```

LINE 91 determines how long to wait between "sweeps" of the Job Tables.

Always, I add the appropriate line at the start to DETACH the program.

LINE 400 determines what JOBS are to be logged in "SPY.PRI" that was opened in LINE 20. Job #1 is excluded because it will always be "ERRCPY" on my system. "PEEKPR" (the program itself) and "VT50PY" are also excluded.

I would love to be able to add to the entries on LINE 2050 the KB: but I don't know how to do it.

The file "SPY.PRI" can be PIPed out to paper when needed, and the program restarted to erase the old file and begin anew.

Fred Gervasoni
Joel Barlow High School
West Redding, CT

Sample Print of SPY.PRI

```
5 [ 1  75 ]  GRIPE   07:43 AM 27832      17-Feb-83
5 [ 1  75 ]  GRIPE   07:44 AM 27892      17-Feb-83
5 [ 1  75 ]  UTILTY  07:46 AM 28012      17-Feb-83
5 [ 1  75 ]  ACCLST  07:47 AM 28072      17-Feb-83
4 [ 1  75 ]  NONAME  07:49 AM 28192      17-Feb-83
4 [ 1  75 ]  NONAME  07:50 AM 28252      17-Feb-83
4 [ 1  75 ]  NONAME  08:16 AM 29812      17-Feb-83
4 [ 1  75 ]  BANNER  08:17 AM 29872      17-Feb-83
4 [ 1  75 ]  BANNER  08:18 AM 29932      17-Feb-83
5 [ 1  75 ]  NONAME  10:12 AM 36772      17-Feb-83
5 [ 1  75 ]  TALK    10:13 AM 36832      17-Feb-83
9 [ 1  75 ]  NONAME  12:36 PM 45412      17-Feb-83
9 [ 1  75 ]  NONAME  12:37 PM 45472      17-Feb-83
9 [ 1  75 ]  NONAME  12:38 PM 45532      17-Feb-83
9 [ 1  75 ]  NONAME  12:39 PM 45592      17-Feb-83
9 [ 1  75 ]  NONAME  12:40 PM 45652      17-Feb-83
9 [ 1  75 ]  NONAME  12:41 PM 45712      17-Feb-83
9 [ 1  75 ]  NONAME  12:42 PM 45772      17-Feb-83
9 [ 1  75 ]  NONAME  12:43 PM 45832      17-Feb-83
9 [ 1  75 ]  NONAME  12:44 PM 45892      17-Feb-83
9 [ 1  75 ]  NONAME  12:45 PM 45952      17-Feb-83
9 [ 1  75 ]  NONAME  12:46 PM 46012      17-Feb-83
9 [ 1  75 ]  NONAME  12:47 PM 46072      17-Feb-83
9 [ 1  75 ]  NONAME  12:48 PM 46132      17-Feb-83
9 [ 1  75 ]  GRIPE   12:49 PM 46192      17-Feb-83
9 [ 1  75 ]  WALL    12:50 PM 46252      17-Feb-83
9 [ 1  75 ]  ALPH    12:51 PM 46312      17-Feb-83
9 [ 1  75 ]  HNGMAN  12:52 PM 46372      17-Feb-83
9 [ 1  75 ]  HNGMAN  12:53 PM 46432      17-Feb-83
9 [ 1  75 ]  HNGMAN  12:54 PM 46492      17-Feb-83
9 [ 1  75 ]  HMAN    12:55 PM 46552      17-Feb-83
9 [ 1  75 ]  HMAN    12:56 PM 46612      17-Feb-83
9 [ 1  75 ]  HMAN    12:57 PM 46672      17-Feb-83
9 [ 1  75 ]  HMAN    12:58 PM 46732      17-Feb-83
9 [ 1  75 ]  FREQ    12:59 PM 46792      17-Feb-83
9 [ 1  75 ]  FREQ    01:00 PM 46852      17-Feb-83
9 [ 1  75 ]  FREQ    01:01 PM 46912      17-Feb-83
6 [ 1  75 ]  NONAME  01:17 PM 47872      17-Feb-83
6 [ 1  75 ]  NONAME  01:18 PM 47932      17-Feb-83
6 [ 1  75 ]  NONAME  01:19 PM 47992      17-Feb-83
6 [ 1  75 ]  NONAME  01:20 PM 48052      17-Feb-83
6 [ 1  75 ]  NONAME  01:21 PM 48112      17-Feb-83
6 [ 1  75 ]  ...EDT  01:22 PM 48172      17-Feb-83
6 [ 1  75 ]  ...EDT  01:23 PM 48232      17-Feb-83
6 [ 1  75 ]  ACCLST  01:24 PM 48292      17-Feb-83
6 [ 1  75 ]  ACCLST  01:25 PM 48352      17-Feb-83
6 [ 1  75 ]  ACCLST  01:26 PM 48412      17-Feb-83
6 [ 1  75 ]  ACCLST  01:27 PM 48472      17-Feb-83
6 [ 1  75 ]  AC
Ready
```

DEXPO EAST '83 PREVIEW

To Accomodate Busy Attendees St. Louis Decus Meetings, DEXPO East 83 Will Be Open Sunday, May 22

St. Louis, MO — In addition to providing free admission to registrants attending the DECUS (DEC-Users Society) conference, DEXPO East 83 — The Third National DEC-Compatible Industry Exposition — will open Sunday, May 22, in order to provide DECUS registrants with unhurried access to the thousands of DEC-compatible products and services that will be on exhibit. DEXPO East 83 will be held at Kiel Auditorium in St. Louis, May 22-24.

"The show is meant to be the *one* event for everyone in the DEC-compatible community, a place where users and vendors can work together on problems and solutions. So it's up to us to make it easy for DECUS registrants to take an active role in the show and the future of our industry," observed Larry Hollander, president of Expoconsul International, Inc., organizers of the DEXPO shows.

"Once the DECUS meetings open, it will be difficult for the registrants to spend long stretches of time at the DEXPO East 83," he continued, "so we have arranged also for shuttle buses to speed them from the conference to the show and back again. Of course, we expect they'll want to attend some of the Product Forums, too." The Product Forums, which are free to all registered DEXPO visitors, provide in-depth, practical information on many of the newest DEC-compatibles to be found at the show. Approximately 60 vendor-sponsored presentations will be given during the three-day event.

"Putting the latest DEC-compatible technology to work is what the Product Forums are all about," explained Hollander. "Visitors also like the fact that they can follow-up on the most intresting presentations by visiting the speakers right in their booths." To date, the following 63 Product Forums have been scheduled:

Sunday, May 22

Time	Topic/*Sponsor*	Room
1:00 p.m.	UNIX Emulation Under VMS *Human Computing Resources*	A
1:00 p.m.	Soup-up Your VAX's and PDP-11's Terminal Handling Performance *Xyplex Inc.*	B
1:30 p.m.	The RIMS/MPG Application Program Generating System *Information & Systems Research*	A
1:30 p.m.	The 68000 Microprocessor Applied to the Q-Bus World *Ranyan Corporation*	B
2:00 p.m.	Data Analysis & Statistics For Non-Programmers *Minitab Project*	A
2:00 p.m.	High Resolution Color and Monochrome Dot Graphics *Peritek Corporation*	B
2:30 p.m.	AIDE™ — Computer Aided Software Development System *OASYS, Inc.*	A
2:30 p.m.	New Inexpensive VT100 Emulating Terminal with PLOT-10 and ReGis Graphics Add-on *Micro-Term, Inc.*	B
3:00 p.m.	ProNET Ring Architectured Network *Proteon Associates, Inc.*	A
3:00 p.m.	Series 11 BusDriver for Direct Connection of Remote Terminal Clusters *Micom Systems, Inc.*	B
3:30 p.m.	Distributed Financial Management and Reporting With DEC Equipment *Ross Systems, Inc.*	A
3:30 p.m.	New Developments in High-Capacity Storage Technology *Disc Tech One, Inc.*	B
4:30 p.m.	EasyEntry Forms Design and Data Entry System *Applied Information Systems, Inc.*	A
4:30 p.m.	DSC 200: Audio Data Conversion System *Digital Sound Corporation*	B
5:00 p.m.	Integrating Word and Data Processing *Saturn Systems, Inc.*	A
5:00 p.m.	Bubble Memory — How To Use It in SLI-11 Systems *Bubbl-Tec*	B
5:30 p.m.	Relational DBMS and its SQL, Plus User Friendly Interface *Oracle Corporation*	A
5:30 p.m.	High Speed Printers: Current Models and Future Possiblilities *American Computer Hardware Corporation*	B
6:00 p.m.	DIGICALC™, an "Industrial Strength" Electronic Spreadsheet for DEC Computers *WHY Systems Inc.*	A
6:00 p.m.	ABLE VMZ:/LP: New Intelligent Line Printer Controller *Able Computer*	B
6:30 p.m.	Sphere: A Target Resident Interactive Realtime Programming Environment *Infosphere*	A

Monday, May 23

Time	Topic/*Sponsor*	Room
11:00 a.m.	Flexible Performance from MCBA's Modular Integrated Manufacturing System *MCBA*	A
11:00 a.m.	The Future Of Intelligent Controllers In The Peripheral Marketplace *U.S. Design Corporation*	B

Monday, May 23 *(continued)*

Time	Topic/*Sponsor*	Room
11:30 a.m.	Magnum: An Integrated Relational DBMS For Production Applications *Tymshare*	A
11:30 a.m.	CP/M For Your LSI-11 or PDP-11 *Decmation*	B
12:00 noon	NET488 — File Transfer Package for DEC Computer Networks *National Instruments*	A
12:00 noon	Alternatives To Manufacturers' Service *Grumman Data Systems Corporation*	B
12:30 p.m.	Basis: A Proven Textual Application System *Battelle Software Products Center*	A
12:30 p.m.	Digital's New Rainbow -100 and Professional - 300 Computers *New England Digital Systems*	B
1:00 p.m.	NASA's Technology Transfer Program *Cosmic*	A
1:00 p.m.	Determining Your Training Requirements — Task and Skills Analysis *Essential Resources, Inc.*	B
1:30 p.m.	System 1032: A Data Base Management System for the VAX *Software House*	A
1:30 p.m.	New Supplies and Accessories for Personal Computers *Computer Parts Exchange*	B
2:30 p.m.	Optimizing RSTS Performance with RPM, the RSTS Performance Monitor *Northwest Digital Software*	A
2:30 p.m.	New 22-bit Floppy Disk Controller and More *Micro Technology, Inc.*	B
3:00 p.m.	Accent R, 4th Generation DBMS *National Information Systems, Inc.*	A
3:00 p.m.	Protector's Security Properties and its Approach to Threats *Compion Corporation*	B
3:30 p.m.	Accounting Software for VAX *McCormack & Dodge Corp.*	A
3:30 p.m.	Overtemperature Protection for DEC Computer Systems *Nassau Systems*	B
4:00 p.m.	IBM Device Attachment Control Unit (DACU) *IBM Corporation*	A
4:00 p.m.	The Manufacturing Process of Rigid and Flexible Media *Nashua Corporation — Computer Products Division*	B
4:30 p.m.	7 Ways to Justify Quality Business Graphics In Your Organization *Software Vision*	A
4:30 p.m.	Data Communications in the Local Area Network *Teltone Corporation*	B

Tuesday, May 24

Time	Topic/*Sponsor*	Room
11:00 a.m.	Data Base Concepts In Process Control And Laboratory Automation *KineticSystems Corporation*	A
11:00 a.m.	An Overview of UNIX Operating Systems *Cambridge Digital Systems Div. of Compumart*	B
11:30 a.m.	P-Stat: An Integrated Data Management, Data Display and Statistics Package *P-Stat Inc.*	A
11:30 a.m.	Full Office Automation for RT, TSX+, RSX, RSTS and VMS *Compu-Tome Inc., Mountain West Software/Discom*	B

Tuesday, May 24 *(continued)*

Time	Topic/*Sponsor*	Room
12:00 noon	SPSS-X: An Advanced Information Analysis System ..... *SPSS, Inc.*	A
12:00 noon	New Capabilities Stat MUX: New Modems ..... *ComDesign, Inc.*	B
12:30 p.m.	4th Generation Relational Database Management System and Non-Procedural Application Generator ..... *Contel Information Systems*	A
12:30 p.m.	Focus: An AI State-of-the-Art Inferencing Engine ..... *Systems Cognition Corporation*	B
1:00 p.m.	SOFTOOL*: A Complete Change and Configuration Control and Programming Environment ..... *Softool Corporation*	A
1:00 p.m.	The SAS System of Products and Services ..... *SAS Institute Inc.*	B
1:30 p.m.	Interactive, Computer-Aided Application Generation Techniques: Data Entry, Report Generation, Data Base Management ..... *Clyde Digital Systems*	A
1:30 p.m.	Bridge and Z-Board: The Best 4 Micro Computers Never Bought . *Virtual Microsystems*	B

Tuesday, May 24 *(continued)*

Time	Topic/*Sponsor*	Room
2:30 p.m.	System Management & Performance Software ..... *Gejac Inc.*	A
2:30 p.m.	Microprocessor Based Communication Nodes for DECSystem-10's and 20's ..... *Viking Computer Corporation*	B
3:00 p.m.	Software Product Advances for VMS ..... *Evans Griffiths and Hart, Inc.*	A
3:00 p.m.	Menu-Driven Business Graphics For VAX Users ..... *Dataplotting Services Inc.*	B
3:30 p.m.	Greater Productivity At Reduced Investment Through VAX/PDP 11 Productivity Tools plus DBMS ..... *Amcor Computer Corp.*	A
3:30 p.m.	Integrated Office Automation & Decision Support Tools ..... *Henco Software*	B
4:00 p.m.	CALC-11 Plus Electronic Spreadsheets ..... *Computer Systems Corp.*	A
4:00 p.m.	Bar Code Generation & Scanning Applications ..... *Dytec South Inc.*	B

Survey Finds Visitors To DEXPO East 83 Will See A Broad Range Of DEC-Compatible Hardware & Software Never Before On Exhibit

St. Louis, MO — Among the literally thousands of DEC-compatible products and services to be exhibited at DEXPO East 83 — The Third National DEC-Compatible Industry Exposition — will be several hundred hardware and software offerings that have never been shown before. According to a poll of current exhibitors, it is projected that new software offerings at the show — to be held at the Kiel Auditorium, St. Louis, May 22-24 — will actually outnumber the exhibitors showing software; for hardware, the average exhibitor will show at least two products never before on exhibit. The same survey found that visitors will have first-time access to a substantial array of new services as well.

With 200-250 vendors of DEC-compatibles expected to participate, the show is designed to be a complete information resource for owners, managers, users and dealers of DEC-based systems. This is evidenced by the growing number of "compatible-compatibles." These are DEC-compatible products created specifically for use with other DEC-compatibles. As a result, visitors to DEXPO East 83 will be able to fashion more versatile DEC systems than ever before.

New DEC-compatibles slated to debut at the show include an increasing number of software and hardware products for use with DEC's personal computers, products for use in manufacturing and engineering environments, graphics, office automation, data communications, database management, productivity tools, voice recognition devices, analysis and forecasting systems and more.

Complete information of DEXPO East 83 is available from Expoconsul International, Inc., 55 Princeton-Hightstown Road, Princeton Junction, NJ 08550 Telephone 609-799-1661.

EXHIBITOR LIST, AS OF MARCH 8, 1983.

COMPANY	PRODUCT DESCRIPTION	BOOTH
Able Computer	PDP, LSI, & VAX Enhancements	K
Absco Systems Corp.		423
Access Technology	Decision Support Software	625
ADAC Corporation	LSI-11 Boxes/Interfaces/Systems	321
Advanced Data Management	DBMS/X-Action Proc/RPT Writer	305
*Advanced Digital Office System	BACMAC-RSTS/E Supercharger	101
AGS Management Systems, Inc.	Project Management Systems	404
Air Filtration Products, Inc.	Air Filters for DEC Drives	729
Amcor	DEC RSTS and VAX Software	231
American Computer Group, Inc.	Dealer/Terminal Distributor	J
American Computer Hardware	High Speed Printers/Subsystems	804
Anadex, Inc.	Serial Matrix Impact Printers	419
Andromeda Systems, Inc.	LSI-11 Systems and Components	320
Applied Information Systems Inc.	PDP-11 & VAX Systems Software	221-223
Atlantic Research Corporation	Data Comm Diagnostic Equipment	434
Automated Information Inc.	Compatible Disc & Controllers	725
Aviv Corporation	Disk and Tape Subsystems	312
Aydin Controls	Color Graphic Display Equipment	B
Battelle Memorial Institute	Basis (CMS); Basis-DM (DBMS)	335
Braegen Corporation	Disk Subsystems/Controller	701-705
Britton-Lee, Inc.	Intelligent Database Machines	237
Bubbl-Tec	Solid-State Mass-Storage System	624
California Computer Group	Low Cost Systems & Peripherals	615
Cambex Corporation	DEC Memories; Solid State Disc	422
Cambridge Digital/Compumart	DEC PDP-11 Based Systems	708
Care Information Systems	CARE/DM SYSTEM	416
CIE Terminals Inc.	Video Terminals/Line Printers	A
Clyde Digital Systems, Inc.	Software Utility Specialists	501
Cobar	VT100/132 Emulating Terminals	124
Comdesign	TC-3 Concentrator	118
Compion	Protector-Data Security System	301
Compu-Share, Inc.	General Accounting Software	213-217
Computer Hot Line	Advertising Publications	110
Computer Parts Exchange	Spares, Repairs & Accessories	D
Computer Products, Inc.	Measurement/Control Subsystems	732
Computer Systems Corporation	Calc-11 Electronic Spreadsheet	114
Computer Systems Development		418
Computers-R-Digital	DEC-Related Magazine Monthly	515
Computome		507
Contel Information Systems	Relational 4th Generation DBMS	420
Cosmic	NASA's Software	513
Cosmos Systems, Inc.	68000 Micro - Unix - Thernet	720
Data Processing Design, Inc.	Word Processing & Bus Graphics	300
Data Systems Design	Winchester Disc Systems	704-706
Datanex, Inc.	Interconnect DEC/IBM/CDC/etc.	106
Dataplotting Services, Inc.	VAX Graphics Software	534
Dataram Corporation	Systems, Memories & Controllers	336
Datasystems Corporation		533
Dawn Computer Corporation		600
DEC Professional Magazine	DEC Related Magazines	E
Decmation	CP/M for DEC-11 Computers	724
Digital Communications Assocs.		331
Digital Design/Morgan Grampian	Engineer's "Working Journal"	617-619
Digital Engineering	Retro-graphics TM Enhancements	424
*Digital Equipment Corp.	Rainbow and PRO 350 Computers	101
Digital Information Sys. Corp.	DBL- A Superset of DEC's DIBOL	430
Digital Sound Corp.	Audio Conversion System	734
Disc Tech One	DEC Compatible 14" Disc Drives	707
Distributed Logic Corp. (DILOG)	DEC CPU Compatible Controllers	112-113
Dytec, Incorporated	Computer Peripheral/Datacomm	516
Eaton Corp.	Third Party Maintenance	614
EEC Systems	Office Automation/Sys Software	406
Emulex Corporation	Peripheral Controllers	524
Enterprise Technology	User-Friendly Report Writer	609
Essential Resources	DEC-Compatible Training/Unix	405
Evans Griffiths & Hart, Inc.	PDP-11 & VAX Software Packages	313
FASBE Group, The	Financial Software/Consulting	105
First Computer Corp.	DEC, SMS, & First Systems	325
GEJAC	Chargeback/Performance/Comm SW	317
Grant Technology Systems Corp.	Q-Bus Compatible I/O Boards	805
Grumman Data Systems	Hardware Maintenance Services	235

COMPANY	PRODUCT DESCRIPTION	BOOTH
Halcyon	Data Communications Experts	712
Hardcopy	Publication	C
HCR	Unix Operating Systems	735
Henco Software, Inc.	4th Generation Language Sys.	413
Hewlett-Packard Company		626
Highsmith, James L. & Co.	Personal Computers & Terminals	107
IBM	DACU-OEM Attachment to IBM.	737
IMSL	MATH/STAT Fortran Software	504
Information & Systems Resch, Inc.	Program Generating System	307
Infosphere, Inc.	Falcon/QBUS RealTime/DataComm	122
Intelligent Systems Corp.	Color Graphic Terminals	622
Interactive Info. Sys., Inc.	Manufacturing & Accounting Sys.	716
Interactive Management Systems	Financial and Mfg. Software	119
Interactive Systems, Inc.	VAX, DEC10/20, PC-350 Software	401
Interactive Technology, Inc.	RDM Responsive Data Management	322
Itoh, C. Electronics		A
KineticSystems Corp.	CAMAC Based Systems & Modules	316
Logicraft, Inc.		733
Marc Software Internat'l, Inc.	Muse Word Processing Software	334
*Masstor Systems Corporation	Mass Storage Facility	H
MCBA	Applications Software	102-103
McCormack & Dodge Corporation	VAX Accounting Software	527
MCG Electronics	Data & Power Transient Protect	731
MDB Systems	Computer Interface Products	713
Meridian Consulting	Software Services/Consulting	635
Micom Systems	Datacomm Products/Bus Driver	F
Micro Technology, Inc.	Hard/Soft Controller/Subsystem	637
Micro-Term, Inc.	VT100 Compatible Terminals	407
Mini-Micro Systems	Systems Integrators' Magazine	431
Minitab Project	Software for Data Analysis	711
Minntronics Corporation	Cache Memory and Com Interface	121
Monolithic Systems Corporation		604
Mountain West Software	Office Automation Software	507
Nashua	Flexible & Rigid Media	111
Nassau Systems	Hi-Temp Protection Systems	819
National Information Systems	ACCENT R - 4th Generation DBMS	631
National Instruments	IEEE-488 Interface	605
Netcom Products, Inc.	LSI-11 System Building Options	720
New England Digital Systems	Digicalc, Lex-11, Accounting SW	101
Newman Computer Exchange		201
North County Computer Svcs Inc.	USER-11 RSTS/E & VAX DBMS	108
Northern Technologies		712-714
*Northwest Digital Software	RPM-RSTS Performance Monitor	E
Nyplan, Inc.	Spread Sheet/Financial Modeling	120
Oasys Office Automation System	Software Development System	700
Omtool Corp.		835
ORACLE Corporation	ORACLE* DBMS	425
Oregon Software	Pascal and Programming Tools	613
P-Stat, Inc.	Data Mgt. and Stat. Software	219
Pass	Hardware/Software/Service	618-620
Peed Publishing		520
Pencept, Inc.	Handprint Recognition Terminal	127
Pennington Systems, Inc.	Software Tools/Utils (11/VAX)	104
Peripheral Parts Support, Inc.	Absolute Filters/Repair Serv.	525
Peritek Corp.	High Resolution Colorgraphics	623
Plessey Peripheral Systems	Q-Bus Systems & Sub-Systems	207
Polygon Associates, Inc.	Communication Software	507
Professional Software Support	V*PLAN Electronic Spreadsheet	607
Proteon Assoc., Inc.	ProNET, Local Area Network	517
Radgo Sales Co.	Computer Sparea Media Supplies	323
Ranyan Computer Enhancement	68000-Based Q-Bus Processor	630
Raxco, Inc.		319
Relational Technology, Inc.	Relational DBMS	330
Reliance Electric Company	Digital Repair Services	505
RGTI Systems Software	RFX, Screenshare, FDV/CC, 3271/DD	337
Ross Systems	Financial Management Software	205
S & H Computer Systems, Inc.	System Software for DEC PDP-11	324
SAS Institute, Inc.	Data Analysis & Reporting S.W.	632-634
Saturn Systems	Word/List Processing, S sheet	225
Sauer Computer Systems, Inc.	DEC Software on a 32 Bit IBM	834
Signal Technology, Inc.	SMARTFORM, PACS, OMNIBASE, ILS	512
Sky Computers, Inc.	Array Processor/Dual P + Memory	519
Smith, C.D. & Associates		518
Softool	Change & Configuration Control	730
Software House	10, 20 and VAX DBMS	400
Software Results	DEC IBM RJE LINK to 56KB	125-126
Software Vision	Interactive Bus Graphics Pkgs.	918
SPSS, Incorporated	Information Analysis Software	601
Standard Engineering Corp.		522
Standard Memories/Trendata	Add-In/On Memories - Terminals	436
System Industries	Disk and Tape Storage Systems	
System Industries	A.I. Software Products	H
Systems Cognition Corporation		506
Technical Magic, Inc.		723
Tektronix, Inc.	The Graphics Standard	530
Televideo Systems, Inc.	CRT Display Terminals	612
Teltone Corporation	Data Communications Equipment	531
Tymshare, Database Systems Div.	Integrated Relational DBMS	G
Tymshare, TCSS	Hardware Support; Systems/Sales	G
U. S. Design Corp., USDC	Winchester Storage Subsystem	306
*Universal Color Systems	Color Measurement Systems	122
Viking Computer Corporation	Microprocessor Comm Nodes	115
Viking Software Services, Inc.	CRT Data Entry/Screen Design	304
Virtual Microsystems	CP/M Emulator	500
Weidner Communications Corp.	Computer-Assisted Translation	627
Western Peripherals		535
Whitesmiths, Ltd.	IDRIS: C&X Compiler	621
WHY Systems, Incorporated	Spreadsheet/Financial Analysis	123
XEROX	Manufacturing Software	412
Xylogics	Controller for Disk and Tape	736
Xyplex, Inc.	Terminal Front-End & Switch	435
Zia	RT-11/TSX Software Tools	318

EDITOR'S NOTE: The following press releases are from companies who will exhibit their products or services at DEXPO East 83.

Applied Information Offers Software For 'Professional 300 Series'

Applied Information Systems, Inc., will announce software for the Digital Equipment Corporation Professional 300 series of personal computers.

EasyEntry, the full screen forms design and data entry system for the PDP-11 and VAX, will now run under Digital's P/OS operating system. EasyEntry allows Professional users to quickly develop and implement full screen forms and data entry applications. With EasyEntry, the Professional 350 desktop computer can be used as an applications development system, as well as a data entry station which can be linked to a variety of host systems.

AIS will also announce runtime support for the AIS-PL/I compiler on the Professional 350 at DEXPO. Now PDP-11 and VAX users can develop ANSI Standard Subset G PL/I programs to be run on the Professional.

Demonstrations of EasyEntry and AIS-PL/I will be available at the AIS exhibit in Booths 221-223. Other DEC-compatible software featured in the exhibit will include the BURCOM-11 PDP/Burroughs communications system, the WP Saturn word/list processing system, and the SaturnCalc electronic spread sheet.

For more information, contact Applied Information Systems, Inc., 500 Eastowne Dr., Chapel Hill, NC 27514. (919) 942-7801.

Softool Will Demonstrate Change & Configuration Control Environment

The SOFTOOL® Programming Environment, available directly from DEC as well as from Softool Corporation, will be demonstrated.

CCC™, which is only available from Softool, is the one product that offers change control as well as configuration control. CCC™ automates the management of changes, and the management of different configurations (versions) of a software product. CCC™ can handle many things: source code, object code, documents, test data, etc. CCC™ provides comprehensive support that includes: automatic reconstruction of previous versions, difference reports, management reports, access control, archiving, compression, encryption and auto-

matic recovery. CCC™ can handle programs in Fortran, Cobol, Pascal, C, Ada® Jovial, Assembly, etc.

The SOFTOOL® Programming Environment is an integrated software development facility that automates the rapid generation of quality software. Starting from a design document, it will produce, typically, between 50% and 75% of the code needed. Then, it provides the tools necessary to automate checkout and quality assurance. The available tools include code generator, structured languages, source code and interface documenters, standards and portability auditor, error detectors, tracing, testing and optimization aids.

For further details, contact Softool Corporation at 340 South Kellogg Ave., Goleta, CA 93117. Or telephone (805) 964-0560 or (213) 382-6302.

Mountain West Software Has 'Professional 300 Series' Office Automation Package

Mountain West Software/Compu-Tome, Inc. offers a complete office automation package for PDP/11's and VAX. CT*OS is now available for the new DEC Professional 300 series, including the spelling corrector. The applications include a full featured word processing system, CT*OS Word Processing, CT*OS Spelling Corrector, Electronic Spreadsheet, Executronic Mail (TM), Communications, and a System Utility Package that includes, list file sort, printing of indexes, user index migration, and a facility for document recovery following system problems.

CT*OS (Compu-Tome Office System) features full editing capability, list processing, global search and replace, cut and paste, headers, footers, boiler plate libraries. Movement through documents by word paragraph, sentence, page, etc. Status line indicates page, line, column position, in addition to mode, document number, and account. Compu-Tome, Inc. will be soon be releasing version 5.0, that will include true multi-column compatibility, summation, table of contents, auto indexing and paragraph numbering, auto footnoting, bolding and underlining on the screen. Alternate characters, such as Greek, scientific and technical, can be displayed on the screen, depending on the user's needs.

CT*OS is now available for the DEC 300 Professional Series, with much of the same capabilities as on

the larger DEC systems. CT*OS files can be transferred easily, with our communications option, between PDP's or VAX computers and Professional series. CT*OS must be installed on both systems to utilize this capability.

For more information write Gina L. Harris, Mountain West Software, 234 East Colorado Blvd., Pasadena, CA 91101.

Ross Systems Will Show New Financial Software For 'Professional 350'

Ross Systems will demonstrate MAPS/Pro, the new financial software for the DEC Professional 350.

MAPS is Ross Systems' financial modelling package designed for an array of applications for the financial executives. Financial forecasting, budgeting, strategic business planning, personnel planning, tax analysis, and consolidations are just a few of the areas in which MAPS has been repeatedly successful. A full color graphics package is included with MAPS creating a perfect tool for faster, more accurate financial decision making. Used on Digital Equipment's PDP-11 and VAX-11 computers, MAPS is a language for decision support of non-programming executives.

The version of MAPS for the new DEC Professional 350, called MAPS-/Pro, contains all of the commands and capabilities of MAPS. MAPS-/Pro also shares features and benefits of its forebear such as Interactive "friendliness," flexible reporting, integrated graphics, prompts, HELP messages, and a library of financial functions.

INTAC is Ross Systems' database management system, and is a powerful tool for organizing and reporting strategic data. Data entry, data validation, updating, reporting and inquiry are all available in easy-to-use Business English. INTAC applications include financial information databases, headcount planning, accounting systems, asset/liability management and much more.

For more information contact Ross Systems, 1860 Embarcadero Rd., Suite 210, Palo Alto, CA 94303, (415) 856-1100.

New VTR Presentation From Care Information Systems

Care Information Systems, Inc. will introduce a new VTR presentation of the CARE/DM System.

The unique presentation is among the first step-by-step video demonstrations ever produced for sales and training use. The 45-minute VTR illustrates how the CARE/DM System software manages patient information, patient billing, insurance processing, collection management, scheduling/recall, and financial transactions. Care President, John Struckhoff also explains operating instructions and documentation as well as providing background about the product and the company.

According to Ernest Lang, Vice President, the VTR enables interested parties to receive the complete product story at their convenience and provides far more information than could be covered in a typical sales call.

The CARE/DM System operates with DEC PDP/11 and VAX Processors in RSTS/E and VMS environments. It is available on a time-shared basis, as an individual software package or as a complete system with DEC hardware.

For further information, contact: Care Information Systems, 3009 South Sixth Street, Springfield, IL 62703, (217) 522-CARE.

DISC Will Introduce Combination Package

DISC will be introducing to Systems Integrators and OEM's a combination package that includes DISC's DBL and S&H Computers Inc's TSX-Plus and RTSORT. The combination package costs integrators anywhere from $1420 for 5 copies to $1154 for 100 or more.

DBL is a structured source code compatible superset of Digital Equipment Corporation's DIBOL-11 language. DBL is currently available for DEC's RT-11, RSTS, RSX-11M and VAX/VMS environments as well as for S&H's TSX and TSX-Plus Time Sharing Extension to RT-11 on DEC minicomputers. DISC recently released DBL/VMS, the latest in the DBL series of compiler and runtime systems, for use in native mode under the VAX/VMS operating system. Early users of the product cite the in-line code and /BIND facilities as the major system performance enhancement features, which, when used with the structured extensions to the DIBOL-11 language provide an efficient and easily maintainable programming environment.

Additional product and pricing information may be obtained from DISC, 3336 Bradshaw Road, Suite 340, Sacramento, CA 95827, (916) 363-7385.

Winchester System Available From Data Systems Design

The DSD 890, a DEC-compatible 31.2 megabyte (Mb) Winchester system with 1/4-inch tape backup, is now available from Data Systems Design, Inc. It replaces three Digital Equipment Corporation (DEC) RL02 cartridge disks and a TS-11 1/2-inch tape.

The DSD 890 takes up to 80 percent less rack space and utilizes a 16.25-Mb, 1/4-inch tape for lower-cost archival storage and media transportability. At $9,895 in single quantities, the DSD 890 offers full RL02 and TS-11 emulation, faster throughput and extra features for a price comparable to two RL02 units.

The DSD 890 is fully compatible with DEC LSI-11 (Q-bus) minicomputers, and operates with RT™, RSX™, TSX™, RSTS™ and UNIX™ operating software. The 890 also supports 22-bit addressing and all of the handy DEC backup utilities, including BRU, DSC and PIP.

Other features include on-board bootstrap, 32-bit error correction, documentation to convert user software to 1/4-inch tape media and simultaneous Winchester and tape operation without throughput degradation.

The 890 operates in non-interleaved mode for a transfer rate of 364 kilobytes per second or about 15 percent faster than a single DEC RL02. However, users can elect to program two- or three-way interleaved operation for applications requiring less bus usage. The DSD 890's Winchester emulates three DEC RL02s, while the 890's 1/4-inch tape emulates the 1/2 inch TS-11 tape subsystem. This emulation, achieved through the 890's intelligent controller, allows the DEC operating software to communicate to the DSD Winchester and 1/4-inch tape as if they were DEC devices.

The 890's 1/4-inch standard cartridge tape drive has start/stop capability to provide transparent DEC software compatibility for tape backup and restore functions. The 1/4-inch tape cartridge can be shipped easily and takes up much less space than an RL02 disk pack. At a fraction of the price of an RL02 disk pack, the 1/4-inch tape cartridge is also a more economical means of archiving data.

DSD has enhanced its Hyper-Diagnostics™ by providing self-diagnostics and testing activated by a push button located behind a small door on the front bezel. A seven-segment display shows an error code if a faulty component is detected.

DSD also offers its overnight module exchange and economical warranty extension services, which are available through the company's nationwide sales and support network.

The 890 Winchester/tape system is housed in a small, 5¼-inch high rack-mount package and is attractively styled to complement a PDP-11/23 minicomputer.

Quantity prices for the 890 are available. Production units will be available in March, with delivery 30-45 days after receipt of order.

For more information contact Dianna Konrad, Data Systems Design, 2241 Lundy Avenue, San Jose, CA 95131, (408) 946-5800.

MCG Electronics Will Show Line Protectors & Systems

MCG Electronics, Inc. will be exhibiting its Surge-Master Heavy Duty AC Power Line Protectors for the first time. In addition to the Surge-Master, MCG will also exhibit its other new AC power line and data/signal line protectors.

The new, low cost series of Surge-Master Heavy Duty protectors shields computers and similar installations from lightning and transient overvoltages that can damage equipment seriously enough to require hours or even days of downtime.

Surge-Masters are designed to provide 100% protection at all times, even if they themselves have been partially damaged by a very close lightning strike on a power line. They will also absorb lesser transients, microseconds in duration, which, while doing no immediately noticeable damage, can cause computers to give garbled information and have cumulative effects that can ultimately result in random, unexplained failure.

Indicator lights on the front panel of the Surge-Master protector give its exact status at all times. A green LED indicates that it is in the "normal," ready-to-protect mode; a resettable yellow LED goes on whenever the unit absorbs a transient; and a red LED indicates the presence of a fault, with internal status lights indicating the exact fault location. The Surge-Master will also record and display the number of times that transient surges have been suppressed.

Employing two stages of protection — high speed silicon suppression which reacts within 5 nanoseconds, and truly "brute force" second stage protection — the new protectors will absorb even a very close lightning strike on a power line. To further ensure that protection is available at all times, each phase of the incoming line is protected by three independent heavy duty suppression modules, so

that if any one is knocked out by lightning, two others will still be operating to provide protection. Life expectancy for the Surge-Master units is conservatively estimated at 30 years.

Because the Surge-Master Heavy Duty Power Line Protectors are connected in shunt, rather than in series, they can be installed quickly and easily, with little or no interruption of power. When installed, using a service disconnect switch, maintenance and repair can be done without producing system "downtime". Models of the Surge-Master Heavy Duty Power Line Protectors are available for service panels rated from 100 to 3000 amps and higher; for 120, 240 and 480 VAC; and for single and three-phase power systems. All models are housed in a rugged NEMA 12 enclosure.

Availability stock to 30 days. Prices from $1692 to $96,380.

Protection from transients and surges on data lines entering main frame computers from as many as 20 outlying printers, terminals, or sensors is now available in a single 19" rack-mounted card cage from MCG Electronics, Inc. The card cage, designated DLP-40, can hold up to 20 cards, each capable of accommodating five lines. It is being exhibited for the first time.

Transients and surges re imposed on data lines by lightning, by switching surges, relays, solenoids, and heavy machinery; they can be coupled into the data lines directly, or through the AC power lines. All transients and surges cause some damage to semiconductor junctions, and this damage is cumulative so that it ultimately results in failure. A direct lightning strike, even many miles away, can do very serious damage.

The MCG system interfaces between the computer and the data line, and provides a sophisticated blend of high speed (nanoseconds) voltage limiting and brute force protection. It recovers immediately after each surge, in readiness for the next.

Plug-in cards are available to protect RS-232, RS-422, RS-423, 20ma loops, and dedicated line modems. Interface options include terminal strips, RS-232 connectors (DB-25s), and Mate-n-Lok connectors.

For further information contact William J. Purcell, MCG Electronics Inc., 160 Brook Avenue, Deer Park, NY 11729, (516) 586-5125.

National Instruments Has File Transfer Package For DEC Computer Networks

By providing IEEE-488 hardware and software for DEC computers, National Instruments can furnish a complete package that will accomplish the file-transfer task using the IEEE-488 bus standard as the communications link. The computers linked together in the network may be made up of any combination of UNIBUS and Q-bus based processors. The NET488 network software is compatible with RSX-11, RT-11, VMS and UNIX operating systems.

The main advantage provided by NET488 is the ability to transfer files between different operating systems running in different DEC computers. The operator interface to the file transfer software consists of simple query responses. File transfers may be initiated from a user written application program.

A complete NET488 hardware-/software package for a network of four Q-bus computers (with 50 meter separation) running the RT-11 operating system will cost approximately $15,000.

New National Instruments products that interface the IEEE-488 bus to S-100 computers (GPIB-696) and Multibus computers (GPIB-796) will give DEC users file transfer capability to these popular microprocessor bus structures.

For further information call Frances Drury at (800) 531-5066 (outside Texas) or (512) 250-9119.

Interactive Data Analysis Software for VAX and PDP-11's

The Minitab Project will introduce its new publication, the *Minitab Reference Manual* at DEXPO East 83 in St. Louis, Missouri, May 22-24. The Minitab exhibit, Booth 711, will feature the Reference Manual and the pocket-sized Quick Reference Card, as well as on-line demonstrations of Minitab's VAX (VMS) and PDP-11 (RSTS/E) conversions.

Originally designed in 1972 for Penn State freshmen in introductory statistics courses, Minitab is now used by 1/3 of Fortune's Top 50, almost every major American university and hundreds of smaller organizations around the world. Applications such as forecasting trends, providing decision support, and cross-tabulating survey results are just a few of the ways Minitab can be useful.

Its features include flexible plotting, easy transformation and arithmetic, convenient data editing, comprehensive table-making, curve-fitting, plus a wide range of statistical and data manipulation procedures.

The VAX and PDP-11 conversions can be installed in minutes and consume no more resources than EDT. Minitab is supported on all models running under VMS, RT-11, TSX-PLUS, IAS, RSTS/E, RSX-11 and CTS-500. Tailored specifically for the DEC environment, Minitab allows full file specifications including DECnet nodenames and features an on-line direct access HELP facility. Installation is also routine on VAX-11 models running under UNIX and DEC 10's and 20's.

The new Reference Manual and Quick Reference Card are helpful additions to other available Minitab documentation: the *Minitab Student Handbook*, the *Implementation Guide* and the *ABC's of EDA*. Two newsletters are published several times a year. Subscriptions are free upon request.

For more information about the *Minitab Reference Manual* and the rest of the Minitab system, write to the Minitab Project, 215 Pond Laboratory, University Park, Pennsylvania 16802, USA.

S&H Computer Systems To Show Latest TSX-Plus

S&H Computer Systems Inc., author of systems software for PDP-11's, will be demonstrating their latest and most powerful version of TSX-Plus.

Version 3.1 of TSX-Plus has enhancements over previous versions which include the ability to take full advantage of 22 bit addressing on the Q-bus and Unibus machines, which are capable of supporting 4mb of memory. Another new enhancement is the ability to support the abort entry point for device handlers.

For installations with 5 to 20 timesharing lines, the ability to access 4mb will increase execution speed considerably by reducing job-wait time as well as reducing or even eliminating job swapping. This is accomplished by more users remaining in core, and high speed context swapping that requires no IO.

Version 3.1 also supports a real-time program support facility that allows multiple real-time programs to be run concurrently with normal

time-sharing operations. TSX-Plus also now provides a facility that allows one or more shared run-time systems or data areas to be mapped into the address space of multiple TSX-Plus time-sharing jobs.

Another recent feature of TSX-Plus is the ability to specify the maximum amount of memory TSX-Plus will be allowed to use. This enables the user to set an upper memory limit and all memory located above this limit will never be used by the operating system. This will allow the user to access the upper memory for overlaying programs via a user written handler.

In addition, TSX-Plus allows approximately 8-9kb of additional space for device handlers, system features or terminal lines, thereby increasing the flexibility of the system.

TSX-Plus also features a new real-time EMT which allows a program to set the user mode processor priority level. This can be used in a situation where it is necessary for a real-time program to block interrupts for a short period of time while it performs some critical operation.

Most RT-11 programs can be used with TSX-Plus without change or even having to be relinked, TSX-Plus interfaces with standard RT-11 device handlers (XM version) and supports RT-11 utility programs such as PIP, DIR, DUP, MACRO, TECO, KED and K52. (The FORMAT program is not supported.) Language processors available for use with TSX-Plus include COBOL-Plus, FORTRAN, BASIC, DBL and PASCAL-2. The TSX-Plus keyboard commands are an extended set of those provided by RT-11. TSX-Plus system service calls (EMT'S) are compatible with those provided by RT-11.

Other features offered by TSX-Plus include a transparent lineprinter spooling system, a real-time program support facility, data and directory caching facility, detached jobs, a log-on and usage accounting system, shared file access control, interjob message packet communication, command files with parameters, shared run-time facility, the ability for the user to define system commands, and a program performance monitor that will print a histogram showing where a running program is spending most of its time.

TSX-Plus requires a system with memory management, at least 128k bytes of memory and an RT-11 license.

The price for TSX-Plus version 3.1 is $2,000. All users in support may receive a copy of the new version for the cost of media, shipping and handling only. All supported users receive the bi-monthly S&H Software Bulletin which includes patches, reported bugs, new enhancements, future directions of TSX-Plus, and more.

Address domestic inquiries to Gary Manookian, and all other inquiries to Richard Dohrmann, at S&H Computer Systems, Inc. 1027 17th Ave. S., Nashville, TN 37212, (615) 327-3670 Telex 786577 answer back S&H NAS.

MCBA Will Highlight Product Costing Package

The MCBA (Mini-Computer Business Applications, Inc.) booth will highlight the company's newly-released Standard Product Costing (SPC) package. SPC is the latest addition to the MCBA Manufacturing System and sixteen integrated manufacturing, distribution, and accounting packages written DIBOL for PDP-11s running RT-11. MCBA's twelve new Manufacturing System packages now available under RSTS/E (Resource Sharing Time Sharing/Extended) will also be featured at the booth.

MCBA's Standard Product Costing is an interactive, comprehensive tool which accurately maintains standard (or estimated) costs for both manufacturing and accounting management. SPC maintains up-to-date costs for all products, parts, and service provided by a company. Deviations of current cost from planned objectives can be spotted immediately through the use of Standard Product Costing, MCBA claims, and problem areas can be identified for management attention. Since SPC is integrated with MCBA's Inventory Management and Standard Product Routing packages, labor and material figures are automatically kept consistent. MCBA's SPC maintains both beginning-of-year (standard) and up-to-date (current) cost projects. Current purchase prices deviating from the standard material costs are highlighted on the *Purchase Price Variance Report*. "What if?" cost analysis can be done. Critical ratios like gross margin percentages and cost variances are provided. SPC requires MCBA's Inventory Management (1/M) and Bill of Material Processor (BOMP) packages. A source Code license for MCBA's RT-11 Standard Product Costing is $3,000 for one computer. Substantial discounts are available for resellers and multiple-CPU sites.

The MCBA Manufacturing System packages in RSTS/E which will be introduced are: Customer Order Pro-

cessing, Accounts Receivable, Fixed Assets and Depreciation, Purchase Order and Receiving, General Ledger, Inventory Management, Accounts Payable, Payroll, Bill of Material Processor, Job Costing, Shop Floor Control, and Standard Product Routing. Source code licenses for each of the MCBA Manufacturing System packages in RSTS range from $2,500 to $4,000 for a single computer system. Multiple use and reseller licenses are available at substantial discounts.

The RSTS utilities used in the MCBA packages take full advantage of the power of RSTS, such as the ability to handle up to 64 terminals and to run programs in FORTRAN and BASIC concurrent with business packages in DIBOL. Yet the functions of the new RSTS versions of MCBA's packages are in most cases identical to the RT-11 versions. MCBA claims that the compatibility of the operational aspects of the RT-11 and RSTS versions of MCBA's packages makes moving from one system to the other quite simple. Whether transferring within a large company, or moving up from a timesharing service running RSTS to an independent RT-11 system, users can run the same MCBA packages with which they are familiar, with little retraining.

OEMs and resellers of MCBA products for DEC are invited to shop at the MCBA booth at DEXPO to discuss their possible participation in the company's new support program. Dubbed SPECTRUMS (Support Program for Effective Communication To Resellers Using MCBA Software), the program is a radical departure from MCBA's traditional operating basis. Although MCBA has always licensed its packages with source code through distributors, OEMs, ISOs, and other resellers, the company has never formalized its relationships with its resellers beyond the licensing of the product. SPECTRUMS has been introduced in order to provide a closer, more structured arrangement that will benefit both the company and the resellers. The program has multiple levels and is similar in some respects to Digital's and Hewlet-Packard's programs for their OEMs and software suppliers. Resellers who qualify for the second level of MCBA's SPECTRUMS will be granted "Referenced Reseller" status and will be eligible to receive end user leads from MCBA. To qualify for this status, resellers will be required to submit an application and supply MCBA with a complete financial statement and references.

On hand at DEXPO to provide consultation on the use of MCBA packages in the manufacturing marketplace will be MCBA's new Manufacturing Market Specialist, Ken Rowand. Rowand came to MCBA recently with 35 years of experience in manufacturing and computer-based manufacturing system design. He worked for Hughes Aircraft Company in a variety of plant management positions.

MCBA is headquartered at 2441 Honolulu Avenue, Montrose, California 91020; (213) 957-2900; Telex 194188.

SMC BASIC For DEC PDP-11s Available From Computer Systems Development, Inc.

Computer Systems Development, Inc. announced the immediate availability of SMC BASIC on Digital Equipment Corporation's PDP 11 system under the RSTS/E Operating System. The development of SMC BASIC on the DEC system was a joint effort of Computer Systems Development, Inc. (CSD) and SMC Software Systems, a division of Science Management Corporation.

SMC BASIC, an easy to use yet powerful BUSINESS BASIC, is fully compatible with software applications developed under Basic Four BASIC. A broad range of mature quality-tested software packages have been installed in more than 15,000 sites during the past 10 years in the Basic Four community. These packages are now available under SMC BASIC to the DEC-user community.

SMC BASIC is the only interpretive language for PDP-11s that allows users the flexibility of using files under RMS-11K, DEC's record management system. RMS files may be defined dynamically within a program, and a program may have up to ten indexed files open at the same time. Other features of the language are business math, extensive built-in screen handling functions, automatic conversion of numeric data types, shareable public programs, external calls with recursion, and all data in common, if desired.

SMC BASIC will be distributed on DEC equipment through Computer Systems Development, Inc., an eight-year-old software company specializing in application package development for Digital Equipment Corporation and Basic Four Corporation computers.

For more information contact Jon Coleman, Computer Systems Development, Inc., 140 Mayhew Way, Suite 700, Pleasant Hill, CA 94523 (415) 930-9932.

Computer Systems Corp. To Demonstrate Both CALC-11 And CALC-11 Plus

Computer Systems Corporation will be demonstrating both CALC-11 PLUS and CALC-11 and VAX operating the UNIX operating system.

The CALC products by Computer Systems Corporation have had high user acceptance. The flexibility and reliability of these products are well known in the market place. Both products utilize the same sound design concepts which were used with the original CALC-11 product.

In addition to the extensive added features, such as the very large spreadsheet and paging capabilities of the CALC-11 PLUS product, many of the "nice to have" features have been incorporated in this mature product.

CALC-11 and CALC-11 PLUS are designed to be used for both small and large projects. CALC-11 may be used on small spreadsheets (up to 3600 cells) whereas, CALC-11 may be used on extremely large spreadsheets. Spreadsheets created with the quicker CALC-11, can be easily transported to CALC-11 PLUS. This combined use allows for a very quick creation of individual sections of a spreadsheet which can be incorporated into the much larger spreadsheets with CALC-11 PLUS.

CALC-11 and CALC-11 PLUS are packaged together at the single unit price of $2,500 which includes warranty and one year updates. Educational and volume discounts (along with site, facility and corporate pricing) are available.

For additional information contact David Tortora, Computer Systems Corporation, 5540 Rock Hampton Court, Indianapolis, IN 46268, (317) 872-7200. CALC Hotline: 1-800-428-0714. Telex 27-6243.

Nassau Systems To Exhibit Overtemperature Protection Hardware Accessory For DEC

Nassau Systems will be exhibiting an Overtemperature Protection Hardware Accessory for all Digital Equipment Corporation Computer Systems.

This will be the first trade show display of an Overtemperature Protection Device specifically made for all standard DEC Computer Sys-

tems. The product is designed to assist all DEC equipment OEM's and end users in preventing equipment damage from overtemperature conditions and has been endorsed by DEC Field Service for this purpose. The accessory provides a warning alarm and total system power shutdown at preset temperature limits designed to meet the environmental specifications in DEC computers and is equipped with the DEC standard power control and distribution systems, including all PDP11, VAX, DECSYSTEM-20 and DECSYSTEM-10 equipment.

For further information contact Nassau Systems, P.O. Box 19329 Cincinnati, OH 45219, (513) 231-1283.

IMSL Releases Edition 9 Of FORTRAN Library And Edition 4 Of TWODEPEP

IMSL, Inc. has announced the release of Edition 9 of the IMSL Library for the Digital Equipment System 10/20, VAX-11 Series, and PDP-11 Series. This version of the widely used Library has an additional 40 subroutines which bring the total to 540. Used internationally, it was designed for maximum accuracy and efficiency in mathematical and statistical problem solving.

The IMSL Library is a comprehensive set of FORTRAN subroutines which serve as building blocks that are used to save costly programming time in developing scientific and engineering application programs. They are arranged in 17 chapters, covering the total field of mathematics and statistics.

Major new subroutines for Edition 9 have been added in areas of basic statistics, differentiation, differential equations, quadrature, eigensystem analysis, random number generation, interpolation, approximation, smoothing, linear algebraic equations, special functions, utility functions, optimization, sorting, and zero and extremas.

A staff of software design specialists develops and supports the IMSL Library for the Digital Equipment computers. Experts in mathematics, statistics, scientific computing, programming and computer science test each Library subroutine for accuracy, efficiency, and reliability. IMSL supplements the expertise of this staff with a Corporate Advisory Board comprised of world leaders in mathematics, statistics, and scientific computing.

For the Digital Equipment computer, the annual subscription rates for the IMSL Library are $2,000-$2,500 for initial subscriptions, and $1,500-$2,000 for renewals. For universities, the subscription rate is discounted 40%.

IMSL TWODEPEP, now in Edition 4, is an easy-to-use finite element program that solves time-dependent, steady state and eigenvalue programs in general two-dimensional regions. TWODEPEP is useful in areas such as elasticity, diffusion, minimal surfaces, potential energy, Schrodinger equations, heat conduction, fluid mechanics, and other such applications problems. By using a preprocessor to define problems in simple, readable format, TWODEPEP eliminates programming time and is accessible to those with minimal training in partial differential equations. Another feature, TWOPLOT, is a portable graphics package that draws scalar, vector, and stress fields.

TWODEPEP is available on the Digital Equipment System 10/20 and VAX-11 Series. The annual subscription rates for TWODEPEP are $2,000 for initial subscriptions, and $1,500 for renewals. For universities, the subscription rate is discounted 40%.

For additional information contact IMSL, Inc., Sixth Floor — NBC Building, 7500 Bellaire Blvd., Huston, TX 77036-5085, (713) 772-1927; outside Texas call toll free 1-800-231-9842, or telex 79-1923 IMSL INC HOU.

Newman Computer Exchange To Provide Appraisals

Newman Computer Exchange will be a prominent exhibitor. The multi-million-dollar firm is the nation's largest dealer in new and used DEC and Data General systems, processors and peripherals, including an extensive stock of PDP8 equipment.

Newman markets late-model minicomputer equipment, by direct mail and telephone, to major corporations, universities, and government and military agencies.

Personnel will staff the Newman booth to provide equipment appraisal and other firsthand information. Also available: Catalogs, literature and free signup for mailing cycle, as well as the Newman "Blue Book" on converting surplus minicomputer equipment to cash.

For more information contact Newman Computer Exchange, P.O. Box 8610, Dept. P53J-DX, Ann Arbor, MI, 48107 (313) 994-3200.

Northwest Digital Software To Unveil RPM—RSTS Performance Monitor

RPM is a new performance optimization package from Michael Mayfield and Northwest Digital Software. RPM can drastically improve total system performance by identifying problems and the problem areas within each program. RPM is the only product that can do all this.

System tuning with RPM uses a step by step "cookbook" approach. No knowledge of system tuning or monitor internals is required. Problem areas are identified using an automatic procedure which provides a report describing, in plain English, not numbers, where the system performance can be improved. It even makes suggestions for improvement.

On-line plotting, histograms and other reports can be used to further identify problem areas. Extended monitor data collection allows plotting of information not normally available, such as seek distance, disk usage, and cache, memory, file processor (FIP) and small buffer utilization.

The programs causing the problems are then identified and can be examined in extreme detail. Detailed examination of a program includes CPU usage, a count of I/O requests and disk overhead by channel and a count of monitor calls and disk overhead by call.

For more information, contact Northwest Digital Software, Inc. at Box 2-743, Spring Valley Road, Newport, WA 99156, (509) 447-2620.

California Computer Group Presents 'ULTRAVAX' System

VAX-11 users seeking higher performance and lower equipment costs can find both in the ULTRAVAX, a family of "optimized" minicomputer systems based on Digital Equipment Corporation's VAX-11 CPUs. ULTRA VAX is the most recent addition to the ULTRAMINI™ line of DEC-compatible systems recently introduced by California Computer Group, Inc. (CCG).

CCG now offers ULTRA 730 and ULTRA 750 as complete systems. In addition, VAX-compatible disk, tape and communication subsystems, marketed under the name ULTRAKIT, are available as upgrades for 11/780 and 11/782 CPUs and for 11/730 and 11/750 systems already in place.

ULTRAVAX systems (and ULTRA

KIT subsystems) substitute firmware-driven controllers on each Unibus, CMI-bus or SBI-bus for the more traditional, discrete logic-driven controllers used in DEC's standard configurations. These microprocessor-based controllers permit integration of peripherals with larger capacities, faster access and lower costs than DEC peripherals.

ULTRAVAX controllers comprise bipolar, bit-slice microprocessor designs incorporated onto single-board or single-board-extended packages that fit into available VAX backplane slots without modifications. All basic control, status and data transfer operations are implemented by microcode. ULTRAVAX controllers are software-transparent to VAX/VMS, UNIX and DEC diagnostics.

Microprocessor-based controller design provides VAX users a highly flexible means of tailoring their systems to changing applications. Functional performance modifications in the ULTRAVAX are accomplished by simple microcode changes in the PROM, instead of changing entire modules.

One ULTRA 750 CMI-bus disk controller, model CC750-X, features four on-board drive ports which allow any given drive to be set up on any port and changed at any time without reconfiguring the controller. As many as 32 different, predetermined drive combinations on the four disk ports may be operated together. Data is transferred in 32-bit parallel words via the CMI-bus, at serial rates up to 15 MHz.

In addition to providing flexibility, ULTRAVAX controllers enhance storage capacity and eliminate speed bottlenecks. For example, VAX 11/780 users with RA81 disk drives must interface them via the Unibus; the RA81's two-megabyte transfer rate is therefore constrained by the 800-kilobyte transfer rate of the Unibus. Furthermore, this scheme limits the user to 1.5 gigabytes of storage per controller.

However, CCG's CDS474-X controller/desk subsystem interfaces directly to the 11/780's fast SBI-bus (or to the CMI-bus of the 11/750) and provides the user a true 1.8 megabyte per second transfer rate; a quiet, reliable disk drive (10,000 hours MTBF); and a storage capacity of 5.5 gigabytes per controller.

ULTRAVAX controllers also eliminate considerable hardware duplication. In DEC's Massbus, for example, some control logic hardware resides in the RH750 or RH780 adapter; the balance is incorporated into individual disk drives. This creates an unnecessary redundancy of control logic in multi-drive configurations, and an unnecessary cost to the user.

In the ULTRAVAX, however, a single disk controller functionally emulates the entire Massbus subsystem, and can control multiple drives running under RM03, RM05, RM80, RP06 or RP07 emulation. The ULTRA 750's CC750-X controller handles one to four disk drives; the CC780-X (part of 11/780 ULTRAKIT) handles one to eight disk drives or a combination of four disk and four magnetic tape drives.

As with mass storage, ULTRAVAX controls also improve the efficiency and flexibility of multiplexed communications. DEC's DZ11 multiplexer, standard on all VAX systems, performs byte transfers at 9.6 kilobaud. Maximum input data buffering is 64 characters per 16 lines. ULTRAVAX controllers enhance these performance levels by transferring whole words rather than bytes on DMA output, operating at 19.6 kilobaud. Input data buffering is expanded to 256 characters per 16 lines.

Unlike DEC's multiplexers which require up to nine backplane slots per 16 communication lines, and which are limited to one type of local transmission mode, ULTRAVAX communication controllers require only a single hex SPC slot for combinations of up to 128 remote and local communication lines emulating the user's choice of DZ11, DH11 and DMF32.

While the ULTRAVAX's price-to-performance ratio is significantly better than a standard VAX-11/730 or 11/750 in a smaller system configuration, its ratio further improves as the configuration becomes more complex.

The enhanced capabilities of ULTRAVAX and ULTRAKIT are available without sacrifice of DEC warranty coverage, service or support.

For further information, contact California Computer Group, Inc., 3303 Harbor Boulevard, Suite G-10, Costa Mesa, CA 92626 (800) 854-7488; in California: (714) 966-1661; telex 183519 CCG CSMA.

Interactive Info. Offers Resource Planning System

IMCS, a manufacturing resource planning system from Interactive Information Systems, Inc., is an integrated, on-line, interactive MRP II offering. IMCS includes state-of-the-art inventory, bill of materials, routings, material requirements planning, on-line master scheduling, shop floor control, input-output reporting, purchasing and capacity requirements planning.

IMCS is available in a BASIC PLUS version for the DEC PDP 11 family. It is also available in VAX NATIVE BASIC for the VAX family, utilizing Datatrieve, RMS and VMS features.

IFAS, a companion set to IMCS, is an interactive financial accounting system.

For further information contact: Interactive Information Systems, Inc., 10 Knollcrest Drive, Cincinnati, OH 45237, (513) 761-0132 or 5757 West Century Boulevard, Los Angeles, CA 90045, (213) 670-9340.

♥

List of Advertisers

CLASSIFIED

Send Classified Ads to: RSTS Classified, P.O. Box 361, Ft. Washington, PA 19034-0361.
$1.00 per word, first 12 words free with one year's subscription. [Be sure to include a phone number or address in your message.]

RSTS/E INTERNALS MANUAL

The RSTS community has been clamoring for years for a book that details the inner workings of RSTS/E. Well, clamor no more. Michael Mayfield of Northwest Digital Software, and M Systems, the publisher of The RSTS Professional and The DEC Professional Magazines, have teamed up to produce the RSTS/E Monitor Internals Manual.

This manual describes the internal workings and data structures of the RSTS/E monitor. It also notes differences in the internal structures between version 7.1 and earlier versions of the monitor. Future updates will include changes for new versions of the monitor.

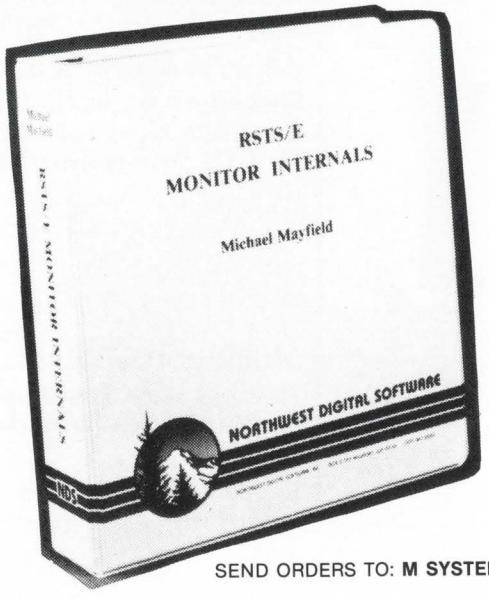
Information is available for all levels of users:

- Gain a basic understanding of the workings of the monitor for optimizing system performance.
- Information on disk structures allows recovery of data from corrupted disk packs.
- Special uses of runtime systems and resident libraries allow complex applications to be developed without degrading system performance.
- Write your own custom device drivers for that "foreign" device you need to add but thought you couldn't.

CONTENTS:

Chapter 1 describes the structures used by the monitor that are resident on disk. These include the directory structure, disk allocation tables, Save Image Library (SIL) formats, bootstrap formats and bad block mapping.

Chapter 2 describes the tables used within the monitor to control system resources and provide program services. These tables provide job, memory, file and device control, as well as program services such as interjob communication.



Chapter 3 contains information on writing and installing a custom device driver. It describes the entry points and information the driver must provide to the monitor as well as the subroutines and macros the monitor provides for the driver.

Chapter 4 contains information that enhances information already provided by Digital on writing custom resident libraries and runtime systems. It concentrates mainly on non-standard uses of resident libraries and runtime systems to increase system performance and functionality.

Appendix A provides six quick reference foldout charts:

- The directory structure.
- The monitor tables.
- Fixed memory locations and common data structures.
- Monitor subroutines.
- Device driver entry points.
- Device driver macros.

Appendix B provides examples of the peek sequences required to access most of the monitor tables. It also contains an example program that uses many of the monitor tables to display a job and open files status.

Appendix C provides an example device driver.

Appendix D provides an example runtime system that doubles as a menu system for restricting specified users to a menu of options.

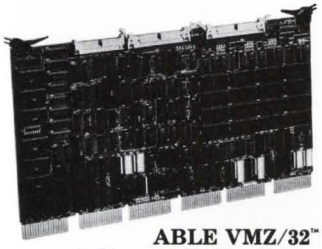
$95.00

SEND ORDERS TO: **M SYSTEMS, INC., BOX 361, FORT WASHINGTON, PA 19034-0361**

If you're in the market for communications modules, make the ABLE connection now. And join the thousands who already have.

We are known as the innovators. Most of our products are industry "firsts" which become popular quickly, then settle into a stage of steady long-term acceptance. These four DEC-compatible, communications devices fit the pattern perfectly. They are ABLE originals. They achieved instant success worldwide. They provide top performance. And they are very reliable. Read on to find the one for you.

INCREASED VAX THROUGHPUT.

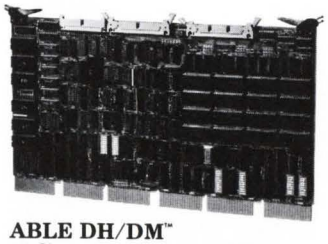


ABLE VMZ/32™
16-line DMF/32 subset

Here's an asynchronous microcontroller with programmable DMA, fully transparent to VAX/VMS as two 8-line DMF 32's and contained on a single board. Priced below the DZ11-E, it outperforms DZ or DH devices under VMS v.3, has interrupt-driven modem control on every line, and includes an output throttle which lets peripheral devices optimize their own data rate.

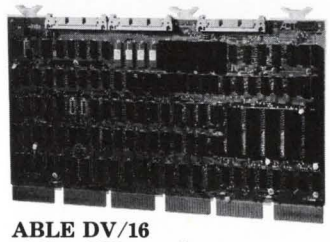
#1 UNIBUS DMA.

Then there's our DH/DM, the original multiplexer which puts 16 lines with modem control on a single board. This popular device meets UNIX VAX system needs for DMA communications requirements, serves UNIBUS systems equally well, and beats them all for MTBF, throughput and price. Other features include on-board diagnostics, modem control on all lines, superior on-board silo depth and variable prom-set.



ABLE DH/DM™
16-line combination DH11
& DM11 replacement

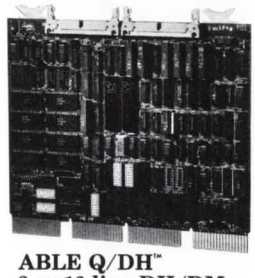
SYNC/ASYNC FLEXIBILITY.



ABLE DV/16
16-line DV11 replacement

A controller for the PDP-11 user, the DV/16 contributes microprocessor-derived flexibility, which permits mixing of sync and async lines in combinations of 4 or 8 lines with modem control and full system software compatibility. It takes less than half the space of a DV11 and uses word transfer instead of byte DMA to gain a 2 to 1 speed advantage or permit operation in half the bandwidth required for data transfers.

Q-BUS DMA.



ABLE Q/DH™
8 or 16-line DH/DM
for Q-BUS

The Q/DH is an asynchronous controller which makes DH-class performance possible on PDP-11/23 and LSI-11/23 Q-BUS systems. It connects the standard Q-BUS to as many as 16 async lines with DMA output capabilities and allows optimum Q-BUS utilization. Features include software compatibility with RSTS/E and RSX operating systems, large input silo, modem control on all lines.

Write for details on our complete line of DEC-compatible products. Be on the lookout for exciting new ABLE communications products soon to come.

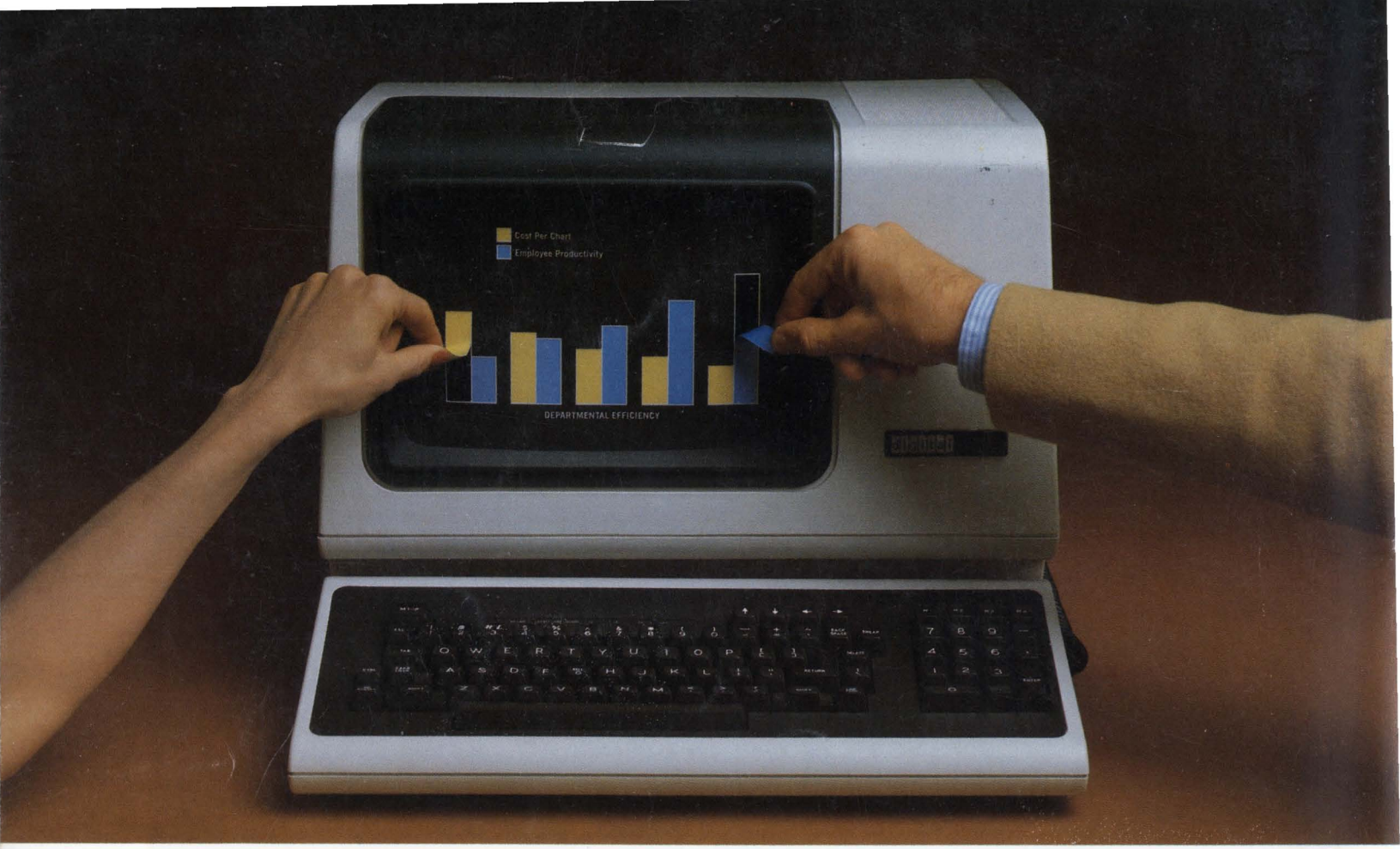
For Immediate, Toll-Free Information, Dial 800 332 ABLE.

ABLE COMPUTER

CORPORATE OFFICES
ABLE COMPUTER
1732 Reynolds Avenue
Irvine, CA 92714 • (714) 979-7030

NATIONAL OFFICES
Burlington, MA (617) 272-1330
Irvine, CA (714) 979-7030

INTERNATIONAL OFFICES
Canada (Toronto) (416) 270-8086
England (Newbury) (0635) 32125
W. Germany (Munich) 089/463080



IB Graph for DEC users. The most cost-effective way to get in touch with your data.

Meet the latest Used Software package from Data Processing Design, Inc. It's called IB Graph.™ Nothing less than the most complete, most cost-effective system for business graphics on Digital PDP-11™ and VAX™ processors.

Of course, you could buy more expensive graphics software, but you'd probably end up with more capabilities than you'd actually need—which wouldn't be cost-effective. Or you could buy *less* expensive graphics software, that simply can't do the job. Again, not very cost-effective. IB Graph is the ideal compromise, giving you literally everything you need, and no more.

IB Graph is a multi-user, interactive businss graphics system that lets you generate graphs in minutes instead of days. And get quality, full-color bar charts, line charts, or pie charts for immediate publication or presentation. With IB Graph, you can better visualize and analyze your data. You can make quicker decisions and increase your efficiency—not to mention your company's profits. Best of all, IB Graph is easy to learn and simple to use, whether you have no experience in computers at all, or are an experienced programmer.

IB Graph outputs to a variety of graphic CRTs and plotting devices. And it will continue to be compatible with future hardware announcements by DEC.

If you'd like more information, call or write to us at DPD. We'll be happy to tell you more about IB Graph.

IB Graph. You can buy a more expensive system. Or a less effective one.

You can't buy a better one.

IB Graph

Data Processing Design, Inc.
AUTHORIZED digital® COMPUTER DISTRIBUTOR

CORPORATE OFFICE • 181 W. Orangethorpe, Suite F • Placentia, CA 92670 • 714-993-4160 Telex 182-278

PDP-11, and VAX are registered trademarks of Digital Equipment Corporation. IB Graph is a registered trademark of Data Processing Design, Inc.

CIRCLE 59 ON READER CARD