RSTS PROFESSIONAL

Volume 2, Number 3

September 1980
$7.50/issue, $20.00/year

```
START:
        SEND    <Adding Logicals>

        ADDLOG  <SY:>,<RPT>
        ADDLOG  <DB1:>,<OV>
        ADDLOG  <DB2:[1,1]>,<LB>
        ADDLOG  <SY:[1,9]>,<STEVE>
        ADDLOG  <SY:[101,101]>,<FUN>

        SEND    <Adding CCL's>

        ADDCCL  <@>,<>,<[1,2]ATPK.*>,PRIV,30000
        ADDCCL  <BAS>,<IC>,<[1,3]SWITCH.TSK>,PRIV,30050
        ADDCCL  <BP2>,<>,<LB:[1,2]BASIC2.TSK>,,30000
        ADDCCL  <IFL>,<>,<[1,2]RMSIFL.TSK>
        ADDCCL  <LI>,<>,<[1,3]DIR.TSK>,PRIV,30000
        ADDCCL  <RT>,<11>,<[1,3]SWITCH.TSK>,PRIV,30050
        ADDCCL  <UT>,<ILTY>,<[1,2]UTILTY.SAV>,,8192
        ADDCCL  <ZAP>,<>,<[1,3]ZAP.TSK>
```

INSIDE:

Two Distinguished Products for PDP-11... And now VAX Users

INTAC™ MAPS™

Interactive Data Base Management

INTAC is a new concept for data storage and retrieval that features an easy-to-use question and answer format, built-in edit rules, multi-key ISAM data access, interactive inquiry and a unique report generator.

Financial Modeling

MAPS, recognized worldwide for over five years as a leader in financial modeling and reporting, is used to construct budgets, financial forecasts, consolidations and "what if" analyses.

Ross Systems, with over seven years of proven capability, now offers these two products to current and prospective PDP-11 and VAX users. INTAC and MAPS enable business managers to produce instant reports themselves, and relieve DP managers from the pressures of special requests.

Ross Systems offers these management tools on our timesharing service, for license on existing computers and as part of a complete, in-house timesharing installation.

Call us collect for more information.

ross systems incorporated

1900 Embarcadero Road, Suite 208, Palo Alto, CA 94303 • (415) 856-1100 • Other offices in San Francisco and Los Angeles

Contents

Coming . . .

- ADD-on Memory
- Multiplexers (as in Data Comm)
- More Disk Directories
- Basic Plus Pre-Processors
- PDP-11 Basic Architecture
- **NEW! The VAX-Scene** A Special Section for VAX
- How to Make the RDC Work With You
- DECUS SanDiego Report
- Games
- How TECO?
- Why TECO?
- What to do About Small Buffer Problems
- More . . .

COVER: Part of the MACRO code used in SETUTL program. See page 35 for the rest of this program.

The RSTS Professional Magazine, September 1, 1980, Vol. 2, No. 3. Published quarterly. Single copy price $7.50, $20.00 per year. Published by M Systems, Inc., 753 Johns Lane, Ambler, Pa. 19002, telephone (215) 542-7008. Send all correspondence and change of address to: Box 361, Ft. Washington, Pa. 19034. Application to mail at second class postage rates pending at Ambler, Pa. 19002.

From the editors ...

Is the Remote Diagnostic Center Too Remote?

Carl B. Marbach, Editor

I picture the Digital Remote Diagnostic Center (RDC) located in the mountains of Colorado... my imagination has it located next to the Strategic Air Command headquarters buried deep inside one of those Rocky Mountains. For those who don't know, the RDC is DEC's answer to hardware diagnosis by remote control. On 11/70's and 11/44's the familiar front panel (with lights) has been replaced by a robot-like faceplate that hides a small intelligence (8080??) that communicates with the console via cryptic commands (↑p to enter console mode, H to halt the computer, C to continue etc...) or, if the key is in the far right position allows the computer console to be manipulated via a phone line installed for this purpose that costs about $10/month. Essentially it gives the people in Colorado the ability to do everything your field service Tech could do in person, except touch the machine. They can run diagnostics, boot tapes, examine locations, toggle? in programs, and even run as a RSTS user under timesharing if you let them.

We don't know much more about them and much of what you see above we know only from watching them work, for you see, we have NEVER had anything in writing, like a manual, to tell us what they do or how they do it. Rumor is that they have a stable of 11/70's that do the work and that the people there control their 70's that control our 70's and so on. DEC advertises that they will respond in 15 minutes to a call 24 hours a day, 7 days a week. We were given the new front panel, an 800 toll-free number, and that was it.

Even the branch is fuzzy on how it all works. "Secret", they say, proprietary to DEC. It is clearly the users responsibility to call the RDC when they have a problem, and the RDC will notify the branch when they diagnose it (later we'll discuss that!). But when the computer fails, do you call the branch or the RDC. If it's midnite and you have 8 hour coverage, call the RDC; but what if it is 1:00 PM? We call the branch.

Our policy has been to use the RDC if the computer has a problem off hours, an intermittant problem that requires long run times, or if the local field service person doesn't understand the diagnostic or what the error messages mean. The RDC has NEVER solved a problem for us in two years. We weren't disturbed by this until we came across a problem they should have solved and didn't. This is not a war story but an

... continued on page 30

Go VAX Young Man!

R. D. Mallery, Editor

DEC is telling large RSTS users who want more from their 70's to get a VAX. The decision was made on high that the 70 is the end of the 16-bit PDP 11's.

I polled the open session at Chicago — a room holding about 500 RSTS users, as to how many 70 users wanted their machines to run 40 to 50 percent faster. Fully a third of the people in the room raised their hands. I asked the DEC folks at the podium whether they saw what I saw. I then noted that DEC was creating a wonderful market for 11/70 accelerators and that if neglected, there were others in the industry who would rush to fill the gap.

I can see several ways to 'accellerate' a 70. First, raise the clock rate. This has already appeared as a product. Second, produce a faster and deeper cache as in the 11/44. There are many companies postured to produce such a product. Some people have proposed string processors but these would probably not be software transparent.

There are software transparent bulk-core swapping disk emulators available. On a large 70, where swapping is not an issue, they can be used to hold index files. This is probably not too cost effective.

The entire XBUF caching process might migrate to an intelligent disc controller thus freeing up 20% of processor time and a large chunk of XBUF.

Someone has now come up with a dual-port disk driver for RSTS. If you plan your application so that all updating is done by one processor and the other only does inquiry, you can finally connect two CPU's to one disk. But, don't expect DEC to support it. Dual porting also holds a hidden catch — the disc is not available to processor A while being used by processor B — that could be a severe slow-down.

A much better idea for closely coupling two 70's has emerged. Set up two adjacent 70's with one page of dual ported memory in common. Map this shared memory with a read-write resident library. Voila — memory speed inter-cpu communications implemented with (almost) all DEC supported techniques.

The concept of off-loading the cpu yields some fascinating ideas when we use DECNET or some other interconnect to connect adjacent cpu's with a high speed link.

One cpu can manage the data-base with several detached file handlers. Other cpu's, bristling with DH's, handle a full compliment of terminals, doing all their I/O thru the link(s) to the DP processor. This is a great solution for a dedicated application, but basically useless for an open environment general timesharing situation.

The task of speeding up a 70 breaks down to a number of areas.

CPU SPEED

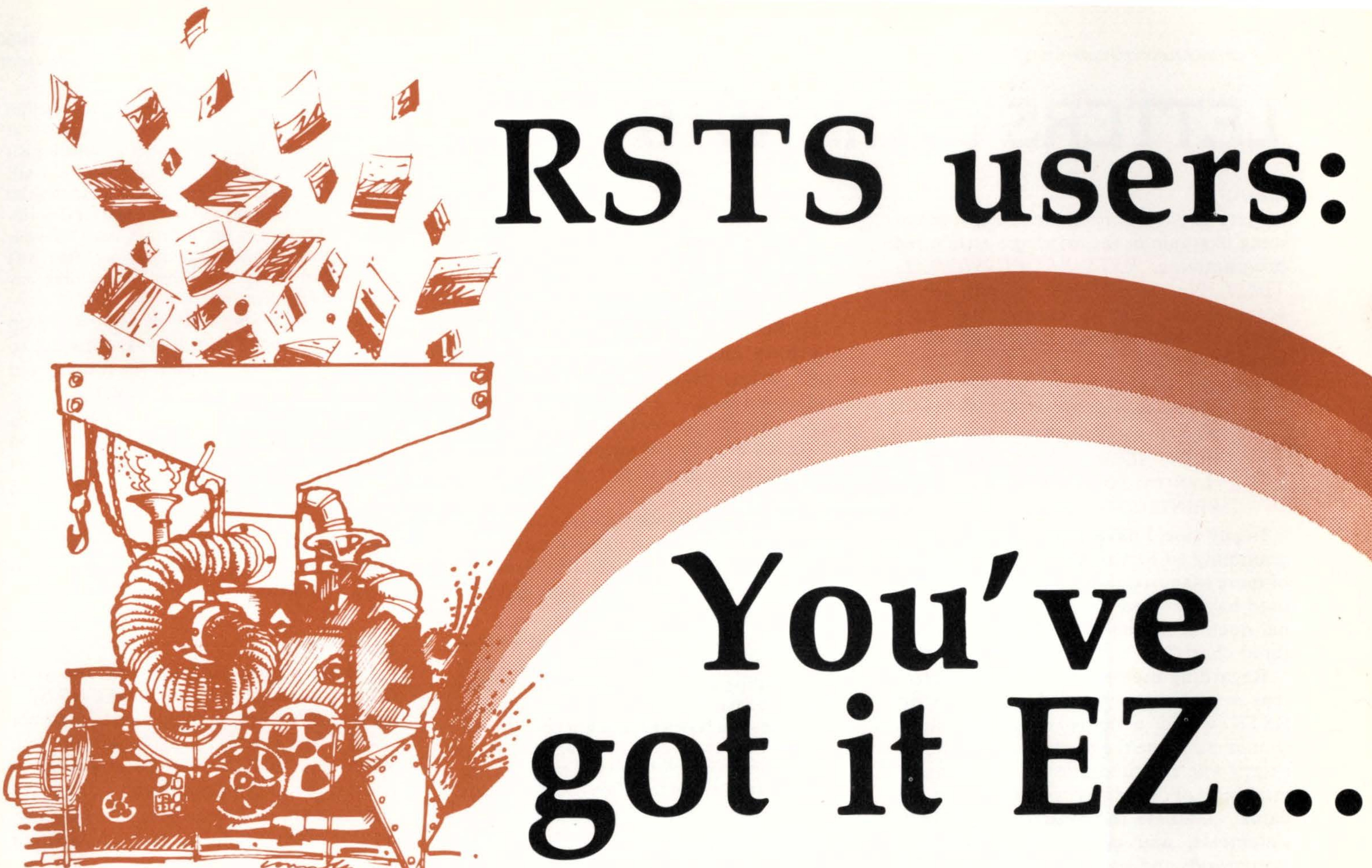
a) faster clock

b) faster and deeper cache

... continued on page 30

RSTS PROFESSIONAL*

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.)



LETTERS to the RSTS Pro . . .

Dear Sirs:

Here is a version of the first programming example in the article on structured programming, *RSTS PROFESSIONAL, Vol. 1, No. 1, p.13*. As you can see, it accomplishes almost exactly the same thing, but with one-eighth the code, and it is easier to read.

```
10 INPUT  'SERIES OF CHARACTERS,
          PLEASE';F$  &
/         PRINT 'REVERSED: ';  &
/         PRINT SEG$(F$,I%,I%); FOR I% =
           LEN(F$) TO 1% STEP —1% &
/         PRINT/GOTO 10
```

In any case, I have found structured programming to be valuable for applications of more than trivial size. The technique I've used has involved subroutines, and external documentation in the form of "structured charts".

Regarding another relevant matter, I feel that a preprocessor for the BASIC's on RSTS would simplify development. It should eliminate the need for line numbers except where desired, allow invocation and marking of modules (subroutines) by name, eliminate the need for RETURN statements, and eliminate the need for ampersands and slashes. It might also deal with nested IF's and other matters. There is no reason why such software could not generate highly readable BASIC, with appropriate comment fields for subroutines, if desired.

Thank you for your magazine. I enjoy reading it.

Sincerely,
Steven Hecht, Programmer/Analyst
Computer Department
The Architects Collaborative, Inc.

—

The following two "memos" were sent to us anonymously.

1) RSTS/E V7A CANCELLED!

We are saddened to report that the corporation has decided that there will be no RSTS/E V7A. After many years of barely surviving from one budget crisis to the next; it was finally decided that the need for compatibility among all Digital software products requires the demise of V7A.

A descendent of RSTS, V7A inherits a mighty tradition: the first PDP-11 operating system that worked (some say the only one that works), the first with error logging, the first with autopatching, the only one with emulators for RSX, RT-11 OS/8 and the IBM 1401.

Instead of V7A, the next release of RSTS/E will be known as V7.0.

We are truly gladdened in our hearts that the "/" survived.

Have a nice summer.

2) RSTS/E V7.0 IS VERY MUCH ALIVE

I would like to clarify regarding the "Death of RSTS/E V7A". This was meant to be a humorous commentary on corporate standards. Absolutely nothing has changed on the RSTS/E V7A project except the official name of the release which is now RSTS/E V7.0. In fact the 7A (7.0) project is growing in functionality and features.

RSTS/E is not now, nor has been, facing a budgetary "crisis". Major development and documentation continues on Version 7.0.

—

Dear Sirs:

I have just received my MAY/JUNE copy of your fine magazine, (the mail to and in Canada is a little slow), and was once again pleased with the content. Let me congratulate you on producing an excellent publication. I look forward to each issue eagerly.

There are a few things I wish to mention, and since I don't get time to write often, I tend to save things up until I'm ready and then let go with a gush, so stand back.

- I wish to put my name on your list if you decide to bulk purchase "A Guide to Programming in Basic-Plus". Although I have been programming, etc. with RSTS since 1973, I have never yet seen a textbook such as the one you reviewed, and it sounds interesting. Also my 14-year-old is about ready to enter the wonderful world of programming, and it sounds like this book could help there.
- I enjoyed your feature "Dear RSTS Man" and think it will prove to be very helpful in the future. Another department you might consider adding would be something called "Did You Know..." where RSTS users could submit helpful hints and little quirks they may have discovered during their wanderings through RSTS. Here is one to get you started:

  When installing BASIC PLUS 2 V1.6, with the intention of utilizing RMS-11, DO NOT install while logged into your LB: account. The reason is that one of the BASBLD dialogue questions asks if you wish to Delete/Purge Installation Files. Most users conscious of the need to conserve disk space reply YES. Section 3.26, which explains this question, fails to mention that among the files deleted are those designated ???IC?.ODL, which are the only copies of the Overlay Description files which you will need later when you attempt to Task Build RMS-11 code into a Basic Plus 2 program.
- I have recently installed V7.0 of RSTS/E, and am using the Data Cache Option (I think!). I am sure looking forward to some information which will tell me how efficient it is, or what it is doing, or if it is too big or too small, or if..., or if..., etc. Know what I mean?? I acquired a copy of LIGHTS and GSTATS as advertised by Rich Marino in the FEB/MARCH issue of RSTS PROFESSIONAL *(Vol. 2, #1)*, (why not ?, this letter is being written with WORD-11), and there is a section of GSTATS called HITS under ...DISK ACCESSES... It never shows any value, and I am not sure what it is meant to measure. (One of these days I may get around to phoning Rich and asking him). If it shows my Data Cache activity, then you can understand my puzzlement.
- I have written some software to interface with a SCHLAG MODEL 708 Door Controller system, which reads identification badges and sends employee identification information to a controller, which is connected to my 11/70. My system then verifies the identification and opens the door at which the employee is standing, relocking it or not (depending on which door he is at), after he has gone through. It was fun to write and I hope to get an article together describing the system for a future issue of your magazine, always assuming my time isn't frittered away on other things like my assigned duties, etc.

Well, that's about all for now. Keep up the good work, and I look forward to bigger and better things in the future from you and your people. Good luck!!

Yours in RSTS,
I.F. (Bud) Dawson
Supervisor, Hardware/Software Support
MacMillan Bloedel Limited, Canada

—

Thanks Bud! You can "gush" in our direction anytime. We'll let you know about "Did you know . . ."

—

Dear Editors:

We would like to make you and your readers aware of the second DECUS Specialized Symposium being held September 17-19, 1980, here at the Delta Airport Inn, Vancouver, B.C. We are putting a lot of effort into this symposium and we expect a high turnout. Contact me if you have any trouble obtaining forms through DECUS or your local DEC office.

Sincerely, Donald A. Lyall
ABACUS Computing Limited
168-10551 Shellbridge Way,
Richmond, B.C. V6X 2W9

—

RSTS Community:

Some 35 members of SENERUG — the Southeastern New England RSTS Users Group — gathered at Wheaton College in Norton, Mass. on July 24 and called for the development of a "multi-drop" terminal handling capability for RSTS, DEC's popular time-sharing operating system.

SENERUG, just under one year old, is independent of DECUS and unaffiliated with DEC. One of its goals is open communication between RSTS users and the entire DP marketplace.

The multi-drop question was prompted by a presentation by Melanson Associates and Timeplex on methods of multiplexing.

Several users saw multi-drop as an attractive capability that could reduce communications hardware costs significantly. it was felt that monitor enhancement would be required for acceptable performance, and that the level of user interest in multi-drop under RSTS should be ascertained by the organization.

The group also refined a list of user requirements for RMS-11, DEC's file-handling software now in its Version 2 development cycle; planned to investigate innovative hardware and software support arrangements; and gathered suggestions for an independent RSTS users' conference tentatively planned for the fall.

. . . continued on page 32

NEW USER'S MANUAL FOR RSTS/E

PART 2*

***Part 1, Chapters 1 and 2, appeared in the last issue of the RSTS PROFESSIONAL, Vol. 2, #2.**

By C. Galfo

PREFACE

This manual is written for system managers and system programmers new to the RSTS/E operating system and the PDP-11 family of minicomputers. The real system manuals, though considered "pocket guides" compared to non-DEC manuals, currently consist of a three foot mountain of information not cross-referenced between volumes. As a result, it is often difficult to find answers to questions posed by new users. In an effort to make the task of learning easier, I propose a more relaxed approach offering common sense, problem-solving techniques, and humor. What follows is a loose formalization of working experience compiled over several years and from many sources. The author does not wish to be held accountable for technical information appearing in this document, though praise, suggestions, and questions are encouraged.

INTRODUCTION

What is a system manager?

If you are a newcomer to DEC computer systems, then the term "system manager" will probably be unfamiliar to you. The titles of "system programmer" and "application programmer" are well-known throughout the industry, but, as I am learning in my job search, the non-DEC world does not understand my current function. Attempts have been made to pigeon-hole me as "a four year BASIC-PLUS programmer", who has, therefore, not used a "hard" language and who has, as a consequence, no value in the marketplace. I resent that description, and am fighting for the right to use those managerial skills I have had to have to make my system reliable and efficient. Here is, from my resume, a definition of a system manager's job: "responsible for the daily administration of PDP 11/70 mini-computer running twenty-four hours a day, seven days a week, year-round, ... Duties include the tailoring and maintenance of system programs, operating system conversions, data base management, and the supervision and training of software engineers, programmers, and data entry personnel. Consult with users, coordinate user activities, and produce user programs and manuals. . .". In the non-DEC world the above would represent a mythical cross between the jobs of computer operations manager and DP manager.

The typical route to the top is an abrupt promotion from programmer (taking orders) to system manager (giving orders), a switch which requires a great mental change. The purpose of this manual is to remove some of the fear of new responsibilities, by presenting lessons from collective experience. Hopefully, your work will be made more enjoyable in the end.

Chapter Three
System Definition

A lot of conversations at the conventions seem to be devoted to individuals arguing in favor of their system procedures and giving justifications for the rules imposed on the system's users. In general, it appears that many underrate the importance of a proper match between system account and file structures and system environment; with poor analysis of system needs, the system manager may find it difficult to control user activities. At the risk of fueling further debate, I will now define five account/file divisions and three data processing environments, so that there exists a criteria for ordered systems.

Account/file breakdowns — The average RSTS/E system will contain at least three elements from the following list:

1. System Libraries — All RSTS/E systems must have the accounts [0,1] (for the storage allocation table SATT.SYS and the bad block file BADB.SYS), [1,1] (for the master file directory MFD and user file directories UFD's), and [1,2] (for the system programs).
2. Production Libraries — These are accounts containing the latest working versions of user programs. They usually are non-privileged (project number   1), and, unlike system libraries, they may store the source and compiled forms of a program. Programmers may work in these accounts, but the practice is not recommended.
3. Programmer Accounts — These are either private use or development accounts, and can be privileged or non-privileged. The general rule is that private use accounts (for word processing, supported research, or personal computing projects) are non-privileged. The existence of privileged accounts in this category produces great anxiety in system managers, so much so that many systems have introduced a semi-privileged account instead.
4. Documentation Libraries — If you have production libraries with only compiled versions of user programs, documentation libraries hold the current sources and .RNO or .DOC files. However, user help files, with the extension .HLP are always kept with the compiled programs.
5. Data File Accounts — Every file that is not a program or documentation is stored in these accounts, which are built, on well-managed systems, using large disk cluster-sizes, especially when the bulk of the available disk space is used for data bases (read/write files) and dictionaries (read-only files). The biggest problems of any system can usually be traced to too many of these files in too few accounts: for example, a program creates one file for each day of the year with all of them stored in one account. Since the file processing code (FIP) in the monitor will only open one file at a time, the entire system will wait until the directory lookup on an account (in this case, through 365 files) is completed.

For systems that use the BACKUP package, and for those systems that do not want to copy all of their disks every day, all files on a disk can be classified as to the amount of work required to replace lost information. There are active files, termed "volatile", such as those of data bases and programs in heavy development, which would always be saved first, since they are the most expensive to replace. Then there are less active files, termed semi-volatile, that can be restored easily, or are changed rarely with modifications carefully noted. These files would be backed up less often. Finally, there are the read-only files, such as dictionaries and documentation, which are termed "static" files; these may be copied only once every three months. Any system can use these backup priorities by assigning files to a designated project number for any one priority level. Followed faithfully, this backup procedure will increase the life expectancy of your backup device, the system will require less operator time, and your important work will be saved.

System Environments — There are three:

1. User — This environment requires the most security, that is, protection from potential abusers. In general, this system is in an educational setting, and the number of privileged accounts is usually restricted to one or two. Elaborate monitoring systems have been developed to keep track of who uses what account on what terminal at what time of day. This type of system will have system libraries for all users, production libraries by project [*,0], and programmers by project.
2. Development — In this environment security is of less importance than control, for users are trusted not to do unthinkable things to the system when the system manager's back is turned. All accounts are privileged; thus, the greatest danger is accidental damage to someone else's work. This type of system requires a bit of tinkering with the system library programs, and will usually have only programmer accounts and data file accounts.
3. User-Development — This is the most complicated set up and a hard system to manage. Users and development personnel can share terminals, so a "big brother" program is useless; sensitive data files are often open to users and "fixers" simultaneously, and trying to get everyone's permission to take the system down is so difficult you end up hoping for crashes. This type of system requires all five account/file categories, and a set of rules for developers to follow so that users are guaranteed correctly functioning programs and uncorrupted data.

Chapter Four
System Performance Optimization

This chapter is divided into two sections, the first on BASIC-Plus program optimization, the second on system performance problems. The conclusions and numbers presented here are closely tied to hardware configuration and are therefore not ultimate solutions or absolute values. Our current configuration is made up of a PDP 11/70 with 128 K words of core memory, floating-point processor (FPP), one RP06, one RP04, two TU16's, and three DH11's. An additional 256K words of memory will soon be installed, which will alleviate almost all swapping for running jobs. As it stands now, only four 16K jobs can be resident in memory simultaneously, since the Monitor (36K), BASIC-Plus (16K), and XBUF (12K) eat up half the available memory. During peak periods in the morning and afternoon, strange things begin to happen when jobs are unable to get enough run time: for example, the SYSTAT program begins to turn out garbage or bomb, and the data entry persons have trouble transmitting long forms. The outcome of all this is that we tend to talk about our system in relative terms (zinging, chugging, flying, or loafing) and internals jargon (DB'ed, TT'ed, I/O wait, idle time, or run lock), rather than give technical explanations that sound good but mean nothing to users.

4.1 Optimizing BASIC-Plus Programs

Some time ago, while we were still running version 6C, we examined the CPU time required for frequently used operations on our system. Each operation was executed 32,766 times during the evening when the system load had been reduced from thirty or more interactive jobs to less than fifteen event-driven jobs. When the tests were repeated under version 7.0, there was no significant deviation from the pervious values. (NOTE: If you run similar benchmarks during timesharing, expect up to a ten percent variation in the CPU times: the lesson is that you can never have the system all to yourself. Furthermore, if you do not have an FPP, the operations which involve floating-point numbers will take from 8 to 10 times longer.)

Here are the times in CPU seconds corrected for 32,766 executions of the FOR..NEXT loop (3.8 seconds):

Operation	Time
String Manipulation	
ASCII("A")	3.2
CVT$%("AB")	3.1
CVT$F("ABCDEFGH")	4.3
CHR$(0%)	4.5
VAL("12")	7.0
VAL("1.2")	8.0
ASCII(RIGHT("ABCDEFGH",3%))	7.8
NUM$(1234)	43.1
NUM1$(1234)	39.8
NUM$(1.23)	113.2
NUM1$(1.23)	114.1
LEFT("ABCDEFGH",3%)	5.7
LEFT("ABCDEFGH",3)	9.7
RIGHT("ABCDEFGH",3%)	6.6
RIGHT("ABCDEFGH",3)	10.2
MID("ABCDEFGH",3%,1%)	6.4
MID("ABCDEFGH",3,1)	12.0
INSTR(1%,"12345678","1")	5.7
INSTR(1%,"12345.78",CHR$(46%))	8.9
INSTR(1%,X$,"4"), not found	
LEN(XS)=1	4.8
10	7.2
100	26.8
1000	226.1

Exit/Entry Times	
Single Line Function (%)	9.5
DEF-FNEND Function (%)	11.7
GOSUB/RETURN	5.3
GOTO/GOTO	4.2
Single Line Function (F)	12.4
Single Line Function ($)	12.4

Assignment Statements	
A% = 3%	0.8
A% = B%	1.0
A = 3	1.2
A = B	1.4
A% = 3	1.5
A% = B	1.6
A = 5%	1.7
A = B%	1.8
A% = B%(1%)	2.9
A% = B%(C%)	2.9
A% = C%(1%,3%)	3.3
A% = C%(B%,7%)	3.6
A = B%(1%)	4.1

IF Statements	
IF Y%	0.8
IF Y% = 2%	1.7
IF Y = 2	2.6
IF Y$=""	3.5
IF Y% THEN GOTO 22%	1.1
IF Y% THEN GOTO 22	1.1

GOTO Statements	
GOTO 22	0.3
GOTO 22 IF Y%	1.0
GOTO 22% IF Y%	1.0
ON Y% GOTO 22%,33%,... (ten branches)	1.0
ON Y% GOTO 22,33,... (ten branches)	1.0

These results support: 1) the value of using integer arguments wherever possible, except for line numbers; 2) the advantage of INSTR over multiple IF statements; 3) the advantage of ON-GOTO over multiple IF statements, and 4) using subroutines and GOTO/GOTO instead of functions (for frequently executed routines).

4.2 Keeping the System up to Speed

Unless you have one of those rare excess-capacity systems, the question that users will most often ask you is: why is the system so slow? Before you, the public relations manager, can answer this question honestly (other than saying you don't know), I recommend you go through the following procedure:

1. Run a SYSTAT. Maybe two, split by project number if you have a lot of jobs on the system. Look for running (RN) jobs. Check the priority. See any DB jobs?
2. Nothing unusual? Look now at KB and other jobs. Check the priority.
3. Still nothing? OK, go back to step one and check for RN jobs still RN and not swapped. Is the CPU time increasing for any job as fast as wall time? If so, try to suspend the job, or kill it, or crash the system if you haven't got high enough priority. It was probably an infinite loop.
4. Lower the priority of any diskbound jobs (analysis, file building and transfer). These appear as constant RN and DB.
5. Still slow? If you have more than one disk for swapping, look for a swap file not ADDed to the system.
6. Now it is time to check the system with ERRDIS. Any PA errors occurring in sets of two mean that the system has locked out cache memory (if you have it). Without one cache group you run about 30% slower; with both locked out, you have two choices: shutdown the machine and wait for the DEC person with a processor kit, or cut the number of jobs in half or less. This is real trouble.
7. Memory OK, no infinite loops, no disk bound jobs, no swap files missing? Only one thing left—take down the system and REORDR the directories. Works everytime.

I have a couple of suggestions on the use of data caching (new in 7.0) and public disks before I close out this manual. On the former, the manuals state that you should have a minimum of 256K words memory before trying to cache files. **Bah humbug!** One of the advantages of caching is that it can be used selectively and in small amounts. You can increase throughput for one or two users without disturbing the rest. And when the goal is to reduce user frustration, data caching is big help in that direction, even when the amount of free memory is small.

Here is the common sense rule about public disks: a system should never be SYSGEN'ed with more than one public disk (especially if those disks are large). Directory searches by default will involve all public disks — the public structure. And that takes time. Secondly, the system will attempt to split the free space equally between the disks, leaving no room for, say, development work with large files. In addition SAVRES will now require N volumes for N disks. The only advantage to multiple public disks would be the use of overlapped seeks, since, with the files split evenly over the public structure, the odds increase that two jobs will need two different disks.

ACKNOWLEDGEMENTS

I would like to thank all of the people who encouraged me on this project and I hope they will be gratified by the result. My only regret is the small amount of time I had to spend on this manual, which continually forced me to narrow my choice of topics. Barry Gershenfeld, my right hand and brain at work, did an excellent hardware overview in Chapter One — **thanks Gershe!** For your comments, etc., our address is University of Virginia, Division of Biomedical Engineering, Box 377 Medical Center, Charlottsville, Va. 22908.

BP2 (V1.6): FROM COM TO TKB

By Randy Park

NOTE

The intended audience for this article is the programmer experienced in BASIC-PLUS 2 but confused by the new features and commands of version 1.6. While certain discussions on technical topics may be incomplete they are sufficiently accurate for the intended audience. This article assumes the user is running version 1.6 of BASIC-PLUS-2, version 7.0 of RSTS/E, RSX directives are included in the monitor, and the monitor supports resident libraries.

When DEC released version 7.0 of RSTS/E, they also released version 1.6 of BASIC-PLUS-2 (BP2). This new release of BP2 has two goals. The first is to incorporate some of the enhancements that were introduced in the new release of RSTS/E. The second is to improve its reliability. The first release of BP2, version 1.04, proved to be very unreliable and quite restrictive. The next release, version 1.5, was more flexible with overlaid (multiple) maps, etc., but it also had reliability problems. The current release, version 1.6, supports the new RSTS/E features such as large files, data caching, and others. Time will tell if it's the reliable product that it is made out to be.

NOTE

If you plan to run BP2 V1.5 on RSTS/E V7.0 make sure that you are totally patched up to date and that all programs that use patched modules are re-compiled and re-linked. If you don't do this you are courting a real disaster. Hopefully no one is still running version 1.04 let alone wanting to run it under version 7.0 of RSTS/E. This new release of BP2 requires you to be running version 7.0 of RSTS/E and running version 1.8 of RMS-11. Do not try to use earlier releases.

There are several commands that can be issued after you compile your BP2 program and before you task build (TKB) it. These commands are:

HISEG	DSKLIB
RMSRES	ODLRMS
BUILD	BRLRES

Individually the purposes of these commands vary, but collectively their purpose is to insure that the TKB for the program links in the proper routines from the proper source.

Before we get into the details of each command we need some background information.

Most compilers do not generate object code that is "self contained". Instead they generate object code that makes reference to some pre-written routines. These routines are not included in your object code as a result of your compile. This is true for almost every high level language. These routines are commonly called library routines. One of the purposes of the task builder is to link these library routines with your object code. These routines are usually written in assembly language and are always assembled by themselves without any knowledge of your program.

We are concerned with two kinds of routines: common BP2 routines, and RMS-11 routines. They can be found in three places: a run-time system (HISEG), a resident library (RMSRES, BRLRES), and a disk library (DSKLIB). The disk library will usually contain all the routines your program will need while a run-time system or a resident library will usually contain only some of the routines your program will need.

The run-time system as it is defined and used under RSTS/E is a very unique animal. One of its several purposes is to share common routines. To link your BP2 program to a run-time system you will use the HISEG command. The run-time system is not directly part of your task, it is not in your ".TSK" file, but it does use up part of your program address space. Since your total program address space is 32KW, linking your task to a 4KW run-time system will reduce your maximum allowable program size to 28KW. Linking your task to a 16KW RTS will in turn reduce your maximum allowable program size to 16KW. One of the advantages of linking your program to a run-time system is that the RTS is shared between multiple programs and are in memory only once. For example: program A, which is 8KW, and program B, which is 10KW, are both linked to run-time system C, which is 16KW. Program A uses an address space of 24KW while program B uses an address space of 26KW. However, when both are running they use only 34KW of physical memory not 50KW, a savings of 16KW. This is because run-time systems are shareable (also called reentrant). Two disadvantages of using a run-time system are: it probably contains routines that you don't need or want (this is true if your RTS is also a keyboard monitor), and it always takes a multiple of 4KW of program address space (ie: 4KW, 8KW, 12KW, etc.).

A resident library's only purpose is to share common routines. It is similar to a run-time system in the following ways:

1. It is shareable between multiple programs and therefore in memory only once.

2. It always takes a multiple of 4KW of program address space.

3. It probably contains routines that your program does not need or want.

4. Taskbuilding is considerably faster if you use a run-time system or a resident library.

It is different from run-time systems in the following ways:

1. Its only purpose is the share code between different programs.

2. It can be self-relocating (like overlaid in memory but better) in that its physical memory space can be greater than its virtual address space. For example, the RMSRES resident library uses 23KW of physical memory but only BKW or your program address space.

A disk library is a file, usually with a ".OLB" extension, where common routines are kept. This file has a special format that TKB and LBR know about. If your program needs a routine that is not found in the run-time system or the resident library then it should be found in the disk library. There are normally three types of disk libraries that a BP2 program can access: a BP2 library, an RMS library, and a user library; and they should normally reside in the "LB:" account. Their differences are only in name and content.

One very important advantage of using a resident library over a disk library has to do with patching. When a module in a resident library is patched there is usually a corresponding module in the disk library that is also patched. If your program uses this patched module, and you are using the resident library version of the module, the patch will take effect as soon as the resident library is re-loaded into memory; however, if you are using the disk library version of the module the patch will take effect only when you re-link the program with TKB.

Also in the "LB:" account you will find some ".TSK" and some ".STB" files with names identical to either a run-time system or a resident library. Do not delete them. The task builder (TKB) uses them to link your program.

Of the six BP2 commands previously listed (BRLRES, RMSRES, HISEG, ODLRMS, DSKLIB, BUILD) only the BUILD command creates the command files used by TKB to link your program. These command files will have ".CMD" and ".ODL" extensions. The five other commands tell BUILD what to put into these command files. HISEG, DSKLIB, BRLRES, and RMSRES are used to decide where the common routines will come from, while ODLRMS is used to indicate how the RMS routines will be overlaid. The BUILD command has options that indicate what file organizations will be used. They are fully documented in the RSTS/E BP2 USER GUIDE.

NOTE

The SHOW command will tell you the defaults of these 6 commands dependent upon the options chosen when BP2 was installed.

— HISEG —

The HISEG command allows the user to choose the run-time system that the program will run with.

BASIC2

1. Your program will be linked with the BASIC2 run-time system.
2. It is 16KW in size.
3. Your program size max is 16KW.
4. You must use the LB:BASIC2 disk library.
5. You should not use any RMS resident library.
6. You cannot use the BP2 resident library (LB:BASICS.
7. You may use any ODL file except LB:RMSRLX and LB:RMSRLS.
8. Your program will link fast and swap fast.

BP2COM

1. Your program will be linked with the BP2COM run-time system.
2. It is 4KW in size.
3. Your program size max is 28KW.
4. You must use the LB:BP2COM disk library.
5. You should not use any RMS resident library.
6. You may use any ODL file except LB:RMSRLX and LB:RMSRLS.
7. You cannot use the BP2 resident library (LB:BASICS).

NONE

1. Your program will use the disappearing run-time system (...RSX).
2. It is OKW in size.
3. Your program size max is 31KW (not 32KW).
4. The monitor must include RSX directives (ask your system manager).
5. You may use any RMS resident library.
6. You may use any ODL file compatible with your chosen resident library.
7. You may use the BP2 resident library (LB:BASICS).

— BRLRES —

This command lets you use a BASIC-PLUS-2 resident library. It is the only BP2 resident library supplied by DEC so far. It is totally independent of the RMS resident library. Including or excluding this library does not affect the ability to include or exclude an RMS library.

LB:BASICS

1. You must specify NONE to the HISEG command.
2. Your maximum program size will be reduced by 8KW.
3. You can use any RMS resident library.
4. You can use any ODL file compatible with your chosen resident library.
5. You must use the LB:BP2COM disk library.

NONE

1. You can use any HISEG that you want.
2. You can use any RMS resident library provided the HISEG is NONE.
3. You can use any ODL file compatible with your chosen resident library.

— DSKLIB —

This command allows you to choose the appropriate BP2 disk library that will be linked with your program. You must always link your program with a BP2 disk library. There is only one RMS disk library (LB:RMSLIB). It is always linked with your program if you do any RMS I/O.

LB:BASIC2

1. This disk library is used only if you are using the BASIC2 HISEG.

LB:BP2COM

1. This disk library is always used unless you are using the BASIC2 HISEG.

— RMSRES —

This command lets you use an RMS resident library. It should only be used if you are using RMS I/O. DEC currently supplies two resident libraries: RMSRES and RMSSEQ.

LB:RMSRES

1. This resident library contains all RMS routines that you will need for RMS I/O.
2. It reduces your program size max by 8KW.
3. You must use the LB:RMSRLX ODL file.
4. You must specify NONE to the HISEG command.
5. You must use the LB:BP2COM disk library.
6. You may use the BP2 resident library.

LB:RMSSEQ

1. This resident library contains routines that support sequential file RMS I/O only.
2. You must use the LB:RMSRLS ODL file.
3. You must specify NONE to the HISEG command.
4. You must use the LB:BP2COM disk library.
5. You must use the BP2 resident library.
6. Your program size max is reduced by 4KW.

NONE

1. You may use any HISEG that you want.
2. You should use the disk library that is compatible with the chosen HISEG.
3. You may use the BP2 resident library.
4. You may use any ODL file except LB:RMSRLX and LB:RMSLRS.
5. All RMS routines will come from the RMS disk library (LB:RMSLIB) and will be included in your program address space.

— ODLRMS —

The ODLRMS command tells how the RMS routines that your program uses will be overlaid. They dictate how deeply the RMS routines will be overlaid and how much memory they will use. Un-overlaid, RMS routines can take up 22KW of program address space, leaving very little room for application code. Section 8.1 of the RMS-11 USER GUIDE describes the overlay files in good detail and gives tips on how to improve them. Although the RMS-11 manual documents the RMS16X and RMS20X ODL files, they offer no benefit if the RMS resident library is available.

LB:RMSRLX

1. This must be used if the LB:RMSRES resident library is used.
2. The LB:RMSRES resident library must be used.
3. The RMS code will not be overlaid.
4. This ODL file supports all file organizations.

LB:RMSRLS

1. This ODL must be used if the LB:RMSSEQ resident library is used.
2. The LB:RMSSEQ resident library must be used.
3. The RMS code will not be overlaid.
4. This ODL file supports sequential files only.

LB:RMS11X

1. An RMS resident library cannot be used.
2. This ODL file uses about 4.5KW of program address space.
3. There are 57 overlays.
4. Programs will execute slowly.
5. This ODL file supports all file organizations.
6. Linking will be slow.

LB:RMS11S

1. An RMS resident library cannot be used.
2. This ODL file uses about 4.5KW of program address space.
3. There are 19 overlays.
4. This ODL file supports sequential files only.

LB:RMS12X

1. An RMS resident library cannot be used.
2. This ODL file uses about 6KW of program space.
3. There are 17 overlays.
4. This ODL supports all file organizations.

NONE

1. No RMS code will be used.
2. No RMS resident library will be used.

VALID COMBINATIONS WITH NO RMS I/O

1. HISEG — LB:BASIC2 DSKLIB — LB:BASIC2 BRLRES — NONE RMSRES — NONE ODLRMS — NONE	a. program size max=16KW. b. fast task building. c. maximum use of shared code. d. identical to COMPILE/TSK.
2. HISEG — LB:BP2COM DSKLIB — LB:BP2COM BRLRES — NONE RMSRES — NONE ODLRMS — NONE	a. program size max=28KW. b. fairly slow task building. c. allows about 5KW more space. d. minimal use of shared code.
3. HISEG — NONE DSKLIB — LB:BP2COM BRLRES — NONE RMSRES — NONE ODLRMS — NONE	a. program size max=31KW. b. slow task building. c. allows the most application code. d. no code is shared.

4. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — LB:BASICS
RMSRES — NONE
ODLRMS — NONE

a. program size max=24KW.
b. fairly fast TKB.
c. good use of shared code.
d. allows less application code than #3.

SEQUENTIAL FILE RMS I/O SUPPORTED ONLY

1. HISEG — LB:BASIC2
DSKLIB — LB:BASIC2
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS11S

a. program size max=16KW.
b. TKB speed is good.
c. RMS has 19 overlays.
d. program area includes RMS code.

2. HISEG — LB:BP2COM
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS11S

a. program size max=28KW.
b. TKB speed is poor.
c. RMS has 19 overlays.
d. program area includes RMS code.

3. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS11S

a. program size max=31KW.
b. TKB speed is poor.
c. RMS has 19 overlays.
d. program area includes RMS code.

4. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — LB:RMSSEQ
ODLRMS — LB:RMSRLS

a. program size max=28KW.
b. TKB speed is fair.
c. RMS is not overlaid.
d. program area doesn't include RMS.

5. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — LB:BASICS
RMSRES — NONE
ODLRMS — LB:RMS11S

a. program size max=24KW.
b. TKB speed is good.
c. RMS has 19 overlays.
d. program area includes RMS code.

6. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — LB:BASICS
RMSRES — LB:RMSSEQ
ODLRMS — LB:RMSRLS

a. program size max=20KW.
b. TKB speed is fast.
c. RMS is not overlaid.
d. program area doesn't include RMS.

ALL TYPES OF RMS I/O SUPPORTED

1. HISEG — LB:BASIC2
DSKLIB — LB:BASIC2
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS11X

a. program size max=16KW.
b. TKB speed is slow.
c. RMS has 57 overlays.
d. program area includes RMS code.

2. HISEG — LB:BASIC2
DSKLIB — LB:BASIC2
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS12X

a. program size max=16KW.
b. TKB speed is slow.
c. RMS has 17 overlays.
d. program area includes RMS code.

3. HISEG — LB:BP2COM
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS11X

a. program size max=28KW.
b. TKB speed is very slow.
c. RMS has 57 overlays.
d. program area includes RMS code.

4. HISEG — LB:BP2COM
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS12X

a. program size max=28KW.
b. TKB speed is very poor.
c. RMS has 17 overlays.
d. program area includes RMS code.

5. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS11X

a. program size max=31KW.
b. TKB is the slowest.
c. RMS has 57 overlays.
d. program area includes RMS code.
e. optimized for space.
f. runs very slowly

6. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — NONE
ODLRMS — LB:RMS12X

a. program size max=31KW.
b. TKB is very slow.
c. RMS has 17 overlays.
d. program area includes RMS code.

7. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — LB:BASICS
RMSRES — NONE
ODLRMS — LB:RMS11X

a. program size max=24KW.
b. TKB speed is fair.
c. RMS has 57 overlays.
d. program area includes RMS code.

8. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — LB:BASICS
RMSRES — NONE
ODLRMS — LB:RMS12X

a. program size max=24KW.
b. TKB speed is fair.
c. RMS has 17 overlays.
d. program area includes RMS code.

9. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — NONE
RMSRES — LB:RMSRES
ODLRMS — LB:RMSRLX

a. program size max=24KW.
b. TKB speed is good.
c. RMS is not overlaid.
d. program area doesn't include RMS.

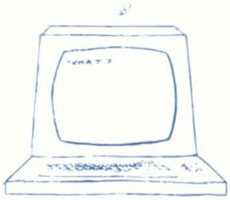
10. HISEG — NONE
DSKLIB — LB:BP2COM
BRLRES — LB:BASICS
RMSRES — LB:RMSRES
ODLRMS — LB:RMSRLX

a. program size max=16KW.
b. TKB speed is fast.
c. RMS is not overlaid.
d. program area doesn't include RMS.
e. optimized for speed.

LETTERS to the RSTS Pro . . .

. . . is YOUR column! Send us your comments, suggestions, or notes of interest to the RSTS community. We'd enjoy hearing from you.

DEAR RSTS MAN:



DEAR RSTS MAN: Regarding your answer to Round and Happy, *RSTS Professional, Vol. 2, #2, May/June 1980,* R&H was seeking a solution to the problem of rounding dollars and cents. The algorithm you provided him with will work but is itself inefficient.

The algorithm read:

```
DEF FNR(X)+INT(X*10.**2%+.5)/10.**2%
```

This function is required to multiply 10.*10. twice for each execution. We tested the efficiency of this algorithm verses one using the constant 100. with the following program, called TIMER:

```
2       EXTEND
999     X1+TIME(0%) / X2+TIME(1%)
32767 PRINT "ELAPSED TIME+";TIME(0%)-X1; &
        "CPU TIME +";(TIME(1%)-X2) / 10. &
```

We ran the program three times with the following three lines:

```
1000 X+INT(I*10.**2%+.5) / 10.**2%FORI+1 TO 10000
1000 X+INT(I*100.+.5) / 100. FOR I+ 1 TO 10000
1000 X+1. FOR I+ 1 TO 10000
```

(The last was to ascertain the amount of time needed for a LET statement)

The result:

```
ELAPSED TIME + 10 CPU TIME + 8.4
ELAPSED TIME + 4 CPU TIME + 3.7
ELAPSED TIME + 2 CPU TIME + 2.1
```

Thus the function using 10.**2% was slower by about 4.7 seconds over 10000 executions. This means that the function using the constant 100. is 4 times faster than the function using 10.**2%. This difference is related to the intricacies of floating point math on the PDP 11. (Our machine, by the way, is a PDP 11/70 with floating point hardware. The difference would have been even greater on a machine without floating point hardware.)

In general, however, the solution you propose is superior to the one proposed by DEC. I refer, of course, to the SCALE command and String Arithmetic.

The SCALE command is used to set the number of decimal places to be kept in floating point numbers. In practice this can be very confusing to the user as the SCALE command is part of RSTS/E and not BASIC PLUS. Thus the SCALE FACTOR cannot be modified by a program, the user must specify the SCALE. Beyond this, however, the SCALE factor can produce erroneous results for those of us who prefer rounding to truncating as the SCALE of 2 would truncate $99.999 to $99.99.

DEC's alternate (and, according to the BP Language manual, 'more flexible and generally easier to use') method is String Arithmetic. This involves storing all dollar amounts as strings and using the SUM$, DIF$, PROD$, and QOU$ functions to operate on them. This is fine except for two things:

1. Strings generally take up more program space than numeric data.

2. String processing is much slower than numeric processing.

We again took the TIMER program above and substituted the following two lines:

```
1000 X+10.+10. FOR I+ 1 TO 10000
1000 X$+SUM$('10.', '10.') FOR I+ 1 TO 10000
```

The results were:

```
ELAPSED TIME + 2 CPU TIME + 2.4
ELAPSED TIME + 11 CPU TIME + 9.8
```

Again subtracting the 2.1 second baseline we get a vast performance difference. In this case numeric handling is 25 times faster than string functions!!!

As a general bit of advice concerning monetary amounts I would offer the following:

1. Whenever possible keep money in cents. This will reduce the likelihood of error due to the fact that most fractions cannot be represented in binary notation. (See Section 6.8 of the BASIC-PLUS Language Manual.)

2. Avoid the use of String Arithmetic. This method takes up more program space and is much slower. CVT$F and CVTF$ run much faster than NUM1$ and VAL.

3. Whenever dollar amounts are used in arithmetic expressions use the function given above. This will reduce the amount of round off error.

Sincerely,
Richard Carlson, Casher Associates

DEAR RSTS MAN: In response to the rounding question posed, if you need to round negative numbers as well as positive, you could use the following function:

```
DEF FNR(X) + FIX( X * 10**2% + SGN(X) * .5 ) / 10**2%
```

However, both this function and the one mentioned in Vol. 2, #2, will fail for certain values of X unless an appropriate SCALE factor is in effect. The following function, although slower will work for all values of X and without having to use the SCALE factor:

```
DEF FNR(X) + VAL( PLACE$( NUM1$( X ), 2% ) )
```

Sincerely yours,
Mark J. Diaz

Dear Gentlemen:

The RSTS MAN needs all the help he can get. Thanks.

DEAR RSTS MAN: Why doesn't my crash dump work in 7.0?

Clera.Sil

Dear Mr. Sil: *Very large systems in 7.0 (like one with 6 DH's) exceed the capacity of some arrays in ANALYS. An SPR went in long ago. The rest is silence.*

—

DEAR RSTS MAN: Every once in a while a couple of DIBOL programs try and read the same record resulting in a record locked condition. That's not so bad, but occasionally the whole system freezes up solid; I mean I can't even run a system status on the console! The only way to thaw the system out is to Control C on all of the terminals until the offending program has been killed. I am running under DIBOL V4C. This problem has occurred under RSTS Version 6 and 7.

What's wrong?

Frozen Solid

Dear Frozen: *The RSTS Man has seen this exact case several times not only in DIBOL but also in BASIC PLUS and BASIC PLUS 2. The key to the problem lies in how you have "solved" it; i.e., by control C'ing the offending program. There is therefore, one program that is causing the problem. For some reason it has encountered a locked block and is looping on the error waiting for it to become unlocked:*

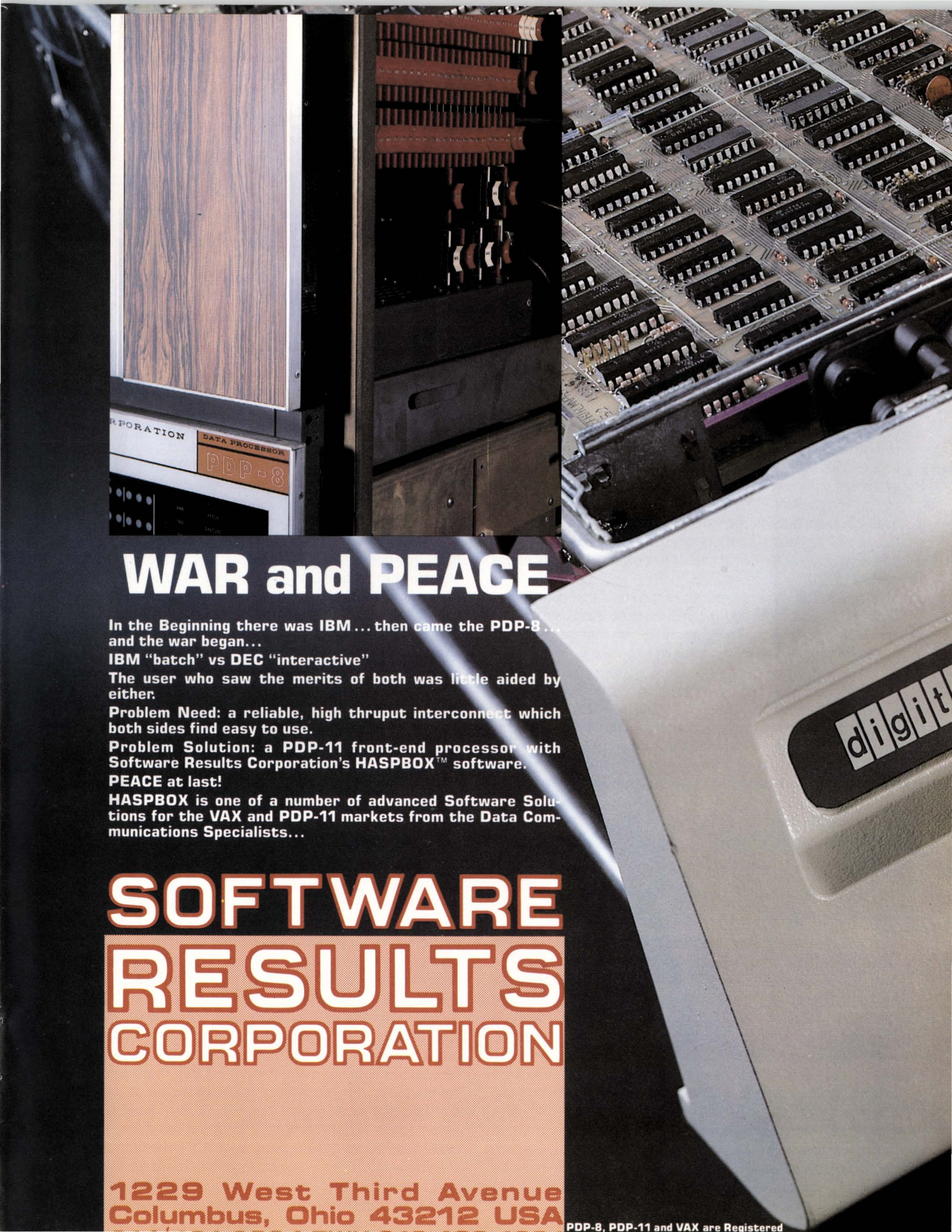
```
15000 GET # CHANNEL% BLOCK RECORD.NUMBER   &
      / IF ERR=19% THEN RESUME !LOOP ON LOCK
```

This can cause a VERY tight loop. If this job has a higher priority than the job that is holding the block locked, the locking job will never unlock the block because it will never be scheduled. Further, it will continue in this CPU bound loop forever. Note, that a control C on this job will deschedule it and allow the locked block to be unlocked. It should not be necessary to KILL the job. Better code for this condition is:

```
15000 GET # CHANNEL% BLOCK RECORD.NUMBER   &
      / IF ERR=19% THEN SLEEP (5) /RESUME
```

The SLEEP(5) will insure that the job will be descheduled and allow others to run, hopefully unlocking the block. DYNPRI or other priority changers will usually handle this situation as they tend to lower the priority of CPU bound jobs. It is possible, however, to start a job at such a high priority that DYNPRI may never lower it enough so that other jobs may run. Some of my System programmers, sometimes raise a job's priority manually (PRIOR now UTILTY) and cause this problem. The Rack or Water Torture will solve these problems.

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



FILE STRUCTURE AND ACCESSING TECHNIQUES

By Gregert Johnson, IIRI International

I still remember the mixture of elation and dismay which attended my first encounter with BASIC-plus and RSTS, on the eve of a System/3 to PDP-11 conversion six years ago. In Record I/O, with its powerful GET/FIELD/PUT triumvirate, one had been granted the gift of infinite flexibility — with it, you could do anything. But there lay the rub — anything, but **you** do it. "Infinite flexibility" is a double-edged gift indeed, for although it implies the freedom to do a great deal, this unfortunately includes the freedom to do nothing, if in fact one does not know **how** to do something. System/3 ISAM may have been limited, but at least it was there and it worked. DEC, on the other hand, did not offer in those days (and, some might argue, does not offer today) an application file management system of sufficient generality and simplicity to be really useful to the commercial RSTS and BASIC-plus user; and what was available from outside software vendors was not always reliable, to say the least. Consequently, there was no other option but to dig in and learn what could be done with the bare bones of Record I/O.

This article shares some of the results of that educational process. The concepts and techniques to be discussed are not specifically limited to BASIC or RSTS, of course; they are, however, ideas eminently well suited to on-line RSTS systems, and can be quite easily and efficiently implemented in BASIC-plus. No coding examples are presented — the intent has been to present something of the conceptual foundations involved, rather than specific implementations. It is hoped that this presentation will prove interesting and useful to those of the RSTS community who wish to know something of what goes on behind the scenes in most data retrieval systems, and who are perhaps unwilling to accept the limitations and devote the resources necessary for using RMS-11 (a group among whom the author numbers himself).

General definitions

To begin with, the basic problem which confronts us is the storage and retrieval of data records from disk files. The **record**, as such, can be thought of as the unit of logical access — it is that minimal part of a data base (file) which we wish to retrieve or store on disk. Each record can be subdivided into sections or **fields** which are the data items **per se** (names, addresses, quantities, etc.). From among the fields of a record, one or more sets of them may be singled out to serve as **keys** which characterize or define the record, either for the purpose of imposing a logical structure among the records of a file, or in order to provide the means of locating individual records (we shall be concerned only with the latter). The keys may or may not be unique: collections of records in a file may or may not be permitted to have keys of equal value, i.e. with identical key fields.

In contrast to the record as the unit of logical access, the **block** is the unit of **physical** access: every disk access (either a read or a write) will involve the transfer of a full block of data (512 bytes in RSTS), whether or not all of that data is used by a program. Several logical records can be included in one block, so that more than one record is read or written during one disk operation. This "multiple blocking" of records can have a profound effect on access mechanisms, as will be seen later.

Since more than one block can be transferred in a single disk operation, the notion of unit of physical access can be generalized to that of the **bucket** (RMS terminology, which we may as well use here), an integral number of blocks which will be transferred during a single operation. The term block and bucket will be used interchangably; the fundamental idea is unit of physical access, the amount of data moved in one disk transfer.

The number of logical records contained in a single disk block (or bucket) is called the **blocking factor** — we shall assume that a bucket always contains an integral number of records, i.e. that records do not span bucket boundaries.

Collectively, all records/blocks/buckets together constitute a **file**. We shall assume that the computer operating system provides random access to each file on the basis of logical block number. This is of course true of RSTS — the data/directory system is in fact a kind of ISAM file structure which allows one to access any block in a file by specifying its position in the logically ordered sequence of blocks imposed on the file (1,2,...,N = total no. of blocks), regardless of the physical location of that block. This mechanism we take as given, upon which all other accessing schemes are superimposed.

The concept of 'Index'

There is one point which should be clarified at once. This is a trivial observation which is so obvious that it often goes unnoticed, and can give rise to confusion. It has to do with the idea of an **indexed file**. Such files are common in data processing, and it is sometimes thought that an index is a necessary part of keyed access, the **sine qua non** without which fast record retrieval is impossible. It is not; an indexed file is simply one in which one data structure (the index) is used as a secondary means to retrieve records from another (the data file). There is no necessary connection between random access and an index — the mechanisms used to access the index records has nothing to do with the fact that those records are being used to retrieve others, usually by means of an address stored in the index record (see Fig. 1). One can in fact have keyed access to data records without the use of intermediate index files; indices are used only when it is either impossible or unfeasible to have the physical location of data records bear a direct relationship to the value of their keys, e.g. when it is desired to access records on different sets of keys (so-called

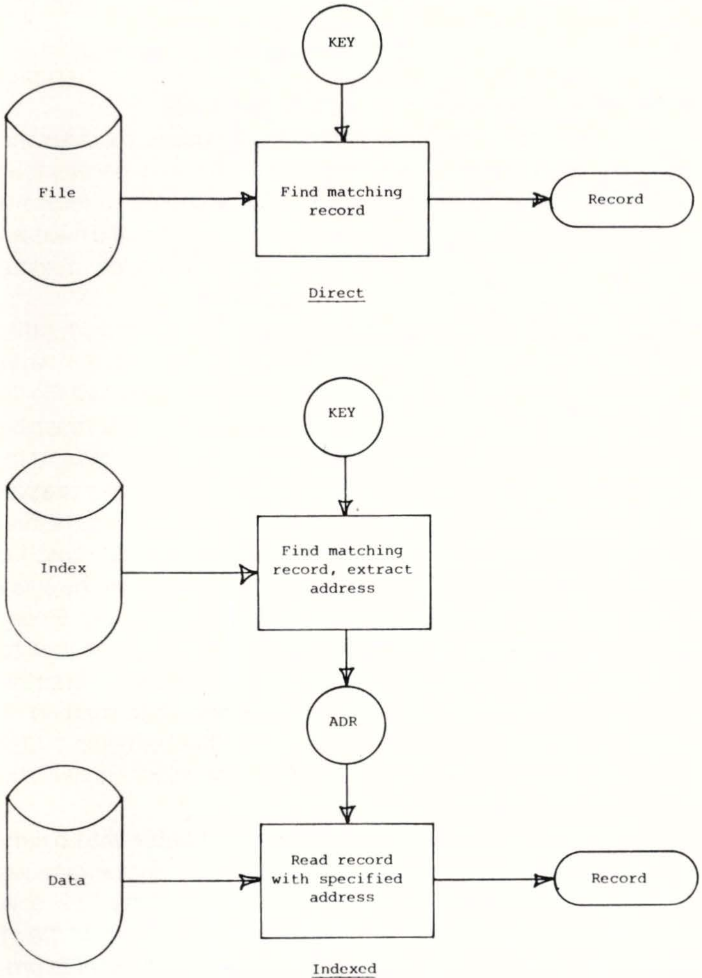


FIGURE 1. Keyed Access: Direct vs. Indexed

"multi-key" retrieval) — but they are not necessary for keyed access **per se**.

In discussing keyed access, we shall assume that it is irrelevant whether the files under consideration are "index" files or "data" files — they could be either, depending on the use to which they are put.

Access mechanisms

In order to achieve efficiencies in the storage and retrieval of information, one should expect that trade-offs of various kinds will be necessary: it seems to be a pervasive "conservation law" that reduction of effort in one area always entails a corresponding increase of effort in another — one always pays a price for value received. Thus, it will come as no surprise that file accessing techniques become more efficient in proportion to the amount of structure that one is prepared to impose upon the files in question.

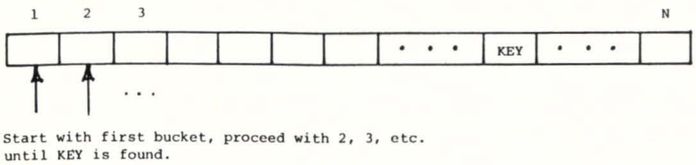
Sequential search

To begin, consider what is perhaps the simplest possible file structure — a **sequential file**, in which records are stored in blocks (or buckets) which are consecutively ordered from 1 to N, where N is the total number of buckets (Fig. 2). We consider the problem of locating a particular set of data in such a file, e.g. that record whose key value is given = KEY. No information is known regarding the location of this record other than that it is located in one of the buckets 1,2,...,N. Under this assumption, we can do no better than to start our search for the desired record by reading the first bucket, and then continuing consecutively with buckets 2, 3, etc. until that containing the record corresponding to KEY is reached. The efficiency of this procedure (or of any retrieval procedure) can be measured by the number of disk accesses which are necessary to find a given record. In this case, we have no way of knowing exactly how many accesses will be necessary to retrieve a particular record, since the location of the records is (presumably) random, meaning that we have no information regarding it. All we can say is that when this sequential searching procedure is used on each of the records in the file, the number of bucket accesses will vary uniformly from 1 to N. On the average, therefore, it will be necessary to read half the file to locate a random KEY value:

Average accesses = ½ N.

The number of accesses involved in an **unsuccessful** search (for a key not present in the file) is always N, since one can never be sure that a key is not in the file until all have been examined.

It may be worthwhile to observe that the term **sequential** in this context refers not so much to the organization of the file as to the searching technique used. This is the only technique possible with truly sequential files (e.g. card or magnetic tape files), where one has no choice but to read the buckets consecutively. Direct access disk files (such as those supported by RSTS) can be accessed randomly by relative block number, but this capability is of little use where we have no information regarding the location of records on the basis of their keys. We could, of course, randomly cast about a file, looking for the desired KEY in a hit or miss fashion, although it would make no



Average accesses = ½N

FIGURE 2. Sequential Search

sense to do so (the average number of accesses to retrieve records in this case would be N, twice that for a sequential search). The only use made of the direct accessing capability in a sequential search is simply to read blocks in such a way that once a bucket has been examined (for record containing KEY) and discarded, it will not be read again; the order of processing need not in fact be 1,2,... — that's simply the easiest way to proceed.

The effort necessary to maintain a file of this sort is minimal. The most troublesome maintenance activity, viz. the adding of new records to the file, is trivially easy in this case — one need only insert a new record into the last bucket (number N), or append a new bucket if the last is full. This will require two disk accesses to accomplish — one to read, another to write. Of course, we pay for this low maintenance in terms of inefficient accessing characteristics — finding a record in a file of 1K (1024) buckets will require 512 disk accesses, on the average.

Binary Search

Once we decide that we are willing to expend some effort in maintaining a degree of additional structure in a file, dramatic improvements in accessing characteristics become possible. Thus, suppose that we can define a collating relation among the key values (e.g. alphabetical order); suppose also, for simplicity, that all keys are unique, so that no two records have the same key. Then, if steps are taken to ensure that records appear in the file in collated sequence, such that all keys in a bucket are greater than all keys in preceeding buckets, we can make use of the structure thus provided to reduce greatly the accesses necessary to find a given record. The technique, called a binary search, proceeds as follows: instead of starting the search with the first bucket of the file, read the middle one, half way between the first and the last (Fig. 3). By examining the keys found in that bucket, we obtain information which immediately cuts the amount of work to be done in half: namely, we know in which half of the file KEY must reside (if in fact it is not in the bucket just read) — the other half, containing keys greater than KEY, is thus eliminated from further consideration and need not be touched. This procedure is then repeated with the remaining half of the file: read **its** middle bucket, and use the keys found there to eliminate half of the remainder. Continue in like manner until the remainder to be considered is simply a single-bucket; then, either KEY is found there or it isn't in the file. The number of disk accesses necessary to reach this point is just equal to the number of times N, the number of buckets, must be divided by 2 in order to get a result of 1, plus one (for the initial access):

$$\text{Accesses} = \log_2 N + 1$$

(Recall the definition of the "logarithm to the base 2 of N": it is simply the number of times N must be divided by 2 to get 1, or, equivalently, the number of times 2 must be multiplied by itself to get N. For example — $\log_2 8 = 3$; $\log_2 1024 = 10$).

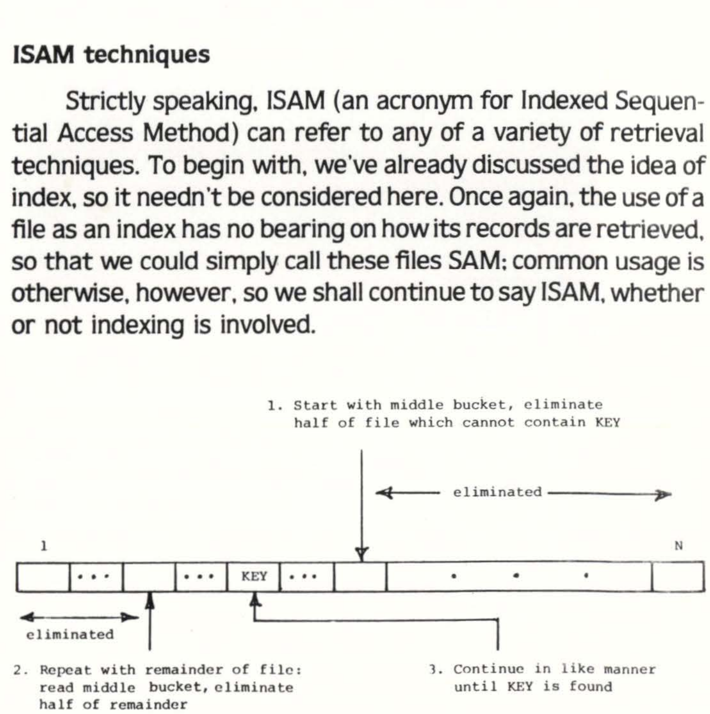
To see what a dramatic improvement this is, consider the example used earlier for a sequential search — whereas that technique required an average of 512 accesses to locate a record in a file of 1024 buckets, a binary search would necessitate a maximum of 10+1 = 11 accesses, a reduction of some 98%!

Such improvement is not obtained without considerable cost, however. This becomes evident when we consider what must be done when new records are loaded to a sorted file. In order to retain the sorted sequence, a new record must be inserted into that position in the file which corresponds to its position in the collated key sequence — this requires that all higher keyed records be shifted one position higher in the file; in other words, the entire higher portion of the file must be re-written. For random additions, this will require on the average N disk accesses (½N to read, the same to write). Binary search is therefore **not** a technique which lends itself well to on-line applications with highly dynamic files. However, in situations involving relatively stable files it is not a bad method. It also has the virtue (along with sequential files) of being 100% efficient with regard to utilization of disk space, unlike the schemes to be discussed below.

Significant as is the improvement obtainable with binary search, it often is unable to provide the instantaneous response usually desired in on-line applications: 11 disk accesses hardly represents perfection, which would be 1 access to retrieve a record. Progress toward this goal is provided by ISAM, which is in a sense a generalization of binary search.

ISAM techniques

Strictly speaking, ISAM (an acronym for Indexed Sequential Access Method) can refer to any of a variety of retrieval techniques. To begin with, we've already discussed the idea of index, so it needn't be considered here. Once again, the use of a file as an index has no bearing on how its records are retrieved, so that we could simply call these files SAM; common usage is otherwise, however, so we shall continue to say ISAM, whether or not indexing is involved.



$\text{Accesses} = \log_2 N + 1$

FIGURE 3. Binary Search (Sorted File)

In its simplest form, ISAM is simply the sequential search described above, with the added restriction that the records be stored in sorted key order. Indeed, this is the essence of IBM System/3 ISAM, mentioned earlier. Keeping the records sorted has two purposes — first, it has the effect of shortening unsuccessful searches: as soon as a key greater than KEY is reached, we know that the rest of the file need not be considered; second, and much more important, the sorted order enables a search shortening technique to be used which is illustrated by what is called a "core index" in System/3 terminology. It works in this fashion: when a file is opened at the start of processing, certain of the keys and their record addresses, namely those at the end of each disk track, are read into a table which is then kept in memory for the duration of the run. Then, whenever a random key retrieval is required, a look-up in this "core index" table is first performed, the result of which is that it is known on which track the desired key must reside, before any disk access has been attempted. Then, a single arm movement is all that is necessary to position the read/write head over the track containing the desired key. Thus, one effectively realizes perfection: one disk access for a random retrieval.

This trick is simple and effective, but a moment's reflection will reveal its inherent limitations. First of all, if the file is large enough, the table of end-of-track keys and addresses will be too large to fit into available memory. Although it would be possible to construct the table using fewer keys (e.g. the keys found at the end of every other track), with correspondingly greater accesses necessary, this is an added complication which is in fact not supported by System/3 (at least it wasn't six years ago). Another problem is that the "core index" so constructed cannot take account of changes which may have taken place in the file since it was opened — in other words, the method fails in a dynamic, on-line environment. Finally, the maintenance problems associated with all sorted files must be dealt with: the addition of new records necessitates the restoration of sorted key order in one way or another. This problem is worse in an on-line environment, where several users may be adding new records simultaneously.

Nevertheless, the "core index" suggests a generalization of binary search which leads to full-blown ISAM. Recall the essence of that technique: by reading a bucket and examining its contents we were provided with information which narrowed the search, specifically divided it in half. That is in fact what the core index does — by performing a series of accesses when the file is opened, we are put in possession of information which allows us to know in what portion of the file (on which track) any given key will be found; in fact, if the number of entries in the table (= number of tracks in the file) is B, then the amount of disk work required to find a key is divided by B (why B has been chosen to represent this number will become clear in a moment). Let us now enquire whether we can devise a searching scheme which is such that whenever a disk access is performed we are able to use information so obtained to restrict the domain of the search; specifically, if at some point in the process we have narrowed the search down to some portion of the file, we should like another disk access to narrow it down still further, say by a factor of B. Implementation of this recursive procedure leads to what is usually referred to as ISAM (although it is sometimes called "B-tree structure"), illustrated in Fig. 4. In that diagram, the lower row of boxes represents the buckets of the file which contain the records **per se**; the trick has been to include additional buckets, collectively referred to as a **directory**, which provide the search-narrowing information. Each directory bucket contains B records, each of which consists of the highest keys and addresses of B other buckets. The directory is arranged in levels: the buckets in one level refer to buckets in the next lower level, with the lowest level being the data record buckets themselves. B is the **record blocking factor** defined earlier, the number of records which can fit in one bucket, and which can thus be read in a single disk access. Whenever such an access is performed, the information obtained is such that we are able to choose which one of the buckets in the next lower level should be accessed next.

Fig. 4 illustrates the technique for an example in which the blocking factor = 3 records per bucket. The search begins by reading the single highest level directory bucket, called the **root**, whose location is presumed to be known (perhaps fixed). By examining the keys contained in that bucket, we are able to establish that the desired record (with key = KEY) must be located in the middle third of the file (if present at all), presumably because KEY is less than the second key in the bucket, but greater than the first. We have thus eliminated 2/3 of the file from further consideration. Using the address of the bucket corresponding to the second key (represented by the darker arrow) we perform another disk access, reading the middle bucket from the next lower level. The keys stored in that bucket in turn allow us to narrow the search to the right third of the already narrowed file, so that now only 1/9 of the file

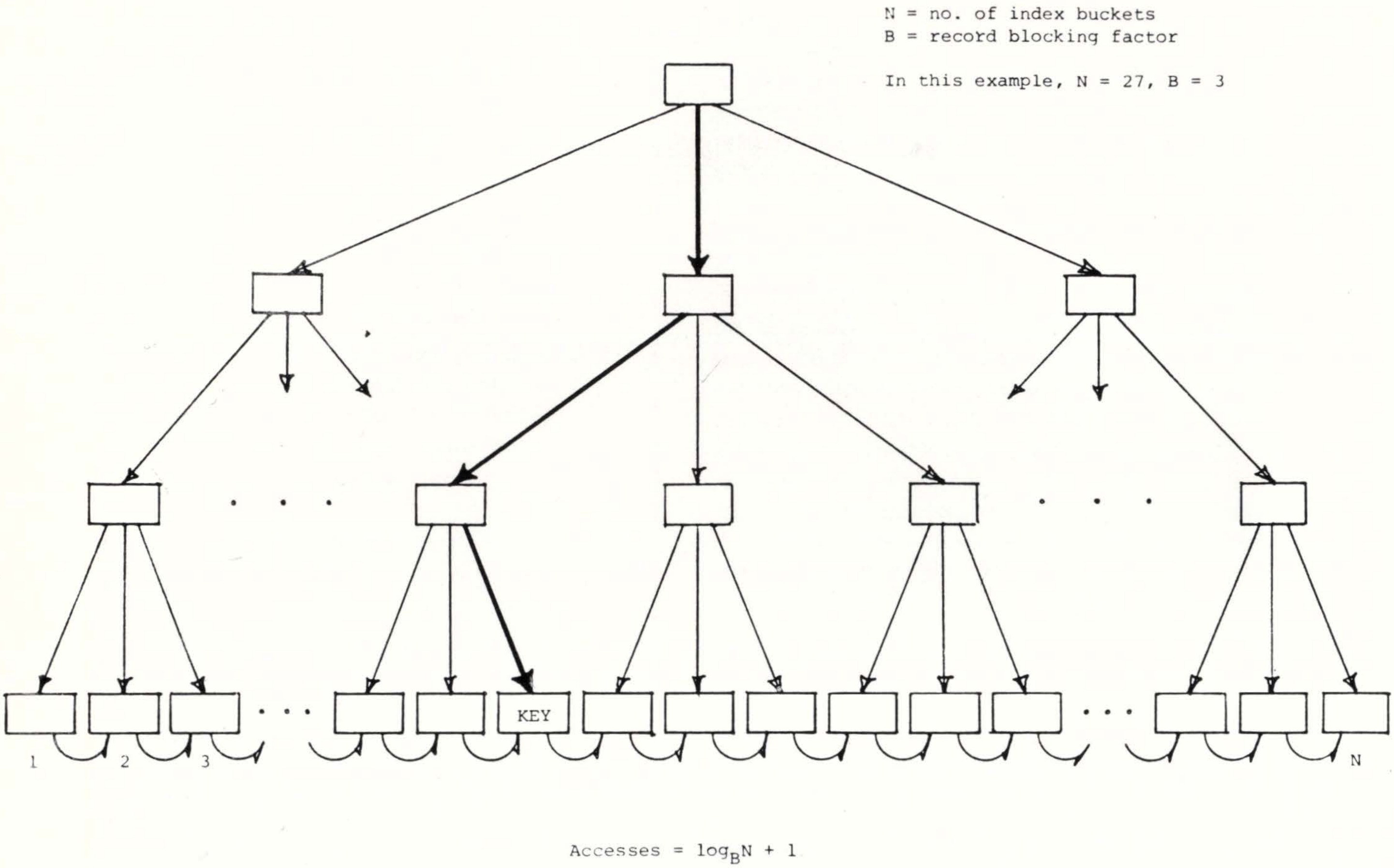


FIGURE 4. ISAM File Structure

remains to be considered. Reading the indicated directory bucket from the next level results in another 1/3 reduction, so that now only 1/27 of the file can possibly contain the desired record; since in this example the file contains only 27 buckets, just one more access is necessary to obtain the bucket which contains KEY — a total of four disk accesses has been required.

In general, with N buckets and a blocking factor of B, the disk accesses required to locate any key in the file is

$$\text{Accesses} = \log_B N + 1.$$

We can see that fewer accesses will be necessary for a given file size N the greater the blocking factor; and multiplying the size of the entire file by B will result in the addition of only one access for retrieval. Fig. 5 illustrates ISAM retrieval characteristics for various file sizes and blocking factors. The horizontal dashed line represents perfection: one disk access for a random retrieval; this is a goal which can be approached by ISAM, but never reached.

In Fig. 4, the record buckets at the bottom are pictured as being linked horizontally by pointers (from 1 to 2 to 3 etc.). These pointers represent addresses which connect the buckets in sequential key order, and which thus provide the one really important advantage of ISAM file processing: namely, that at whatever point one happens to be in the file, it is possible to continue processing sequentially in collated key order. This provides so-called "approximate key search" and "generic key search" capabilities. Thus, it is possible to specify only the first part of KEY (the first few characters), and then use the searching mechanism to retrieve the first key which matches as much of the key as given; following the chain of sequential pointers then enables one to obtain **all** records which match as much of the key as has been given. Note that the buckets have not been drawn contiguously, as in Fig. 2 and 3; this is indicative of the fact that the file and directory buckets need not be located in contiguous logical positions, as was necessary for binary search — the pointers (both directory and sequential) provide the necessary concatenations.

We're almost afraid to ask about the overhead costs associated with these marvelous retrieval efficiencies—as one might suppose, they can be considerable. To begin with, we have an overhead in disk storage because of the directory tree structure which has been built on top of the original file—this amounted to a 48% increase in the example of Fig. 4 (13 directory buckets added to 27 file buckets). Part of this disk overhead must be devoted to the pointer addresses which are used to implement the tree structure and sequential ordering. The most serious overhead, however, is the processing overhead associated with file maintenance, particularly the loading of new records. Complex software is necessary to maintain the tree structure and bucket pointers (there has to be a reason why RMS-11 requires 23K words of memory!) when the file & directory buckets are filled. Usually an attempt is made to keep these buckets partially empty, especially at the lowest (record) level; then, the addition of a new record is a relatively trivial matter of inserting it into the free space, with perhaps some minor rearrangement within the bucket. When the free space is exhausted, however, the phenomenon of "bucket splitting" takes place, where additional buckets must be added to the

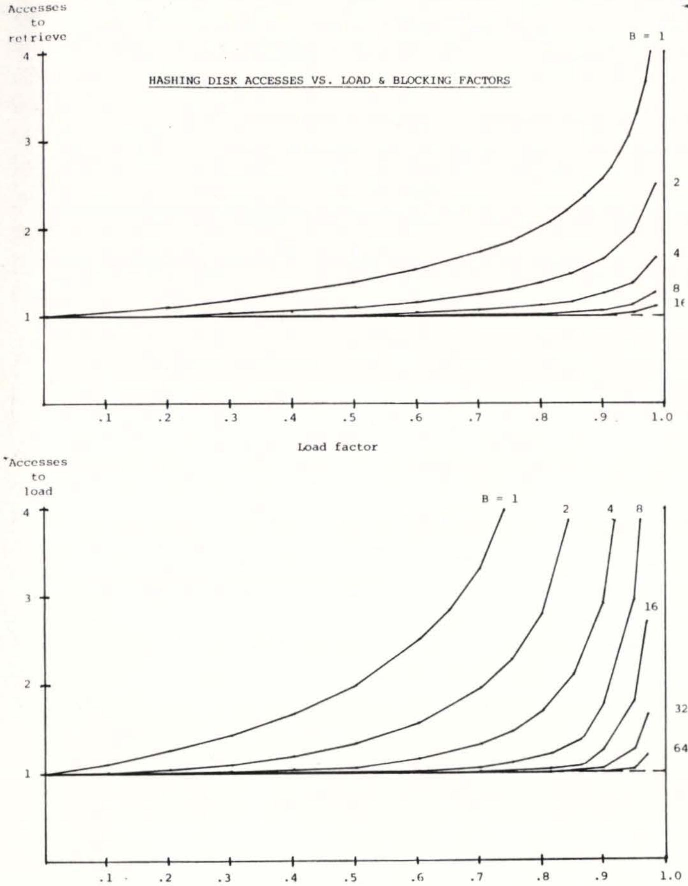


FIGURE 5. ISAM Disk Accesses vs. File Size & Blocking Factor

directory at one or more levels. These new buckets must be linked with the old via address pointers, and all must be done in such a way that simultaneous processing by several on-line users does not corrupt the file structure. In general, ISAM is very expensive: one pays a high price for the benefits of sequential processing in an on-line environment.

In those applications where one is able to dispense with sequential processing capability, however, there is a retrieval technique which comes as close to magic as anything this writer has seen in data processing: this is the method of key hashing, to which the remainder of our discussion will be devoted.

Key hashing

The virtues of simplicity can hardly be over-estimated. This is especially true in data processing, which is a complex endeavor to begin with, where ill-considered complications can result in systems which are confusing and unmanageable, as well as expensive and inefficient. This is not to say that complexity as such should be avoided (even if it could be); but complex systems should be composed of modular units which are as simple in concept and implementation as possible. It is only thus that our creations remain comprehensible to us, and retain the flexibility necessary for rational growth.

One occasionally comes upon an idea which, although conceptually very simple, is charged with an astonishing potential for applications of great power and flexibility, the sort of concept which a mathematician would call "elegant." Record access via key hashing is such a concept.

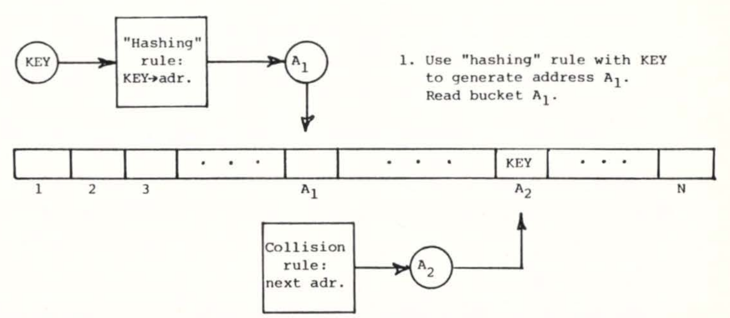
The fundamental problem facing us is that of finding stored data. Now, one of the best ways of finding something is to know where you put it in the first place — and this is the essential idea of hashing. If, in fact, we knew the bucket address of a record beforehand, then only a single disk access would be necessary to retrieve it. This is the case automatically in "direct access" files, where each record is identified by its relative record number, which is to say its position in the file. Our problem, however, is to be able to know something of a record's position on the basis of a more general alphanumeric key — we need some way of converting the key to an address.

Suppose that we start with an empty file, and a record has been submitted for loading (Fig. 6). This record has a key value = KEY, which is to be used to locate the record in the file. Assume further that we have some kind of rule which is such that given any alphanumeric key, we can produce a number from it. It doesn't matter what this rule is, it's a kind of "black box" into which we can put a key on one side and get a number out the other. We put only one restriction on this rule, namely that the numbers it produce lie in the range 1 to N, where N is the size of the file (in buckets). It's obvious what comes next: we just use the number given by this "hashing" rule as the address for loading the record.

So far, all this is trivial and somewhat arbitrary; it seems that we have simply thrown the record into a more or less random location with little rhyme or reason. The significance of what we've done becomes evident, however, when we consider the converse problem and ask how we can retrieve the record we've just loaded. The answer is simple: we just use exactly the same procedure we used for loading — put the key through the same hashing rule, and the result will be the address of the record.

We can continue in this manner: a second record is loaded by putting its key into the hashing rule, resulting in another address which is used to load the new record. That record, too, is immediately retrievable by re-applying the rule to its key.

We seem to have achieved perfection, and with very little effort, a single disk access to retrieve a record from its key.



2. For record load: if empty space or a deleted record exists in bucket, load new record into it. If bucket is already full, use collision rule to generate another address (A_2); read bucket A_2 and repeat at 2.

3. For record retrieval: if KEY is found in bucket, success. If not, then if empty space exists in bucket, KEY is not in file. Otherwise, use collision rule to generate next address A_2, read bucket A_2, repeat at 3.

Expected accesses for record load & retrieval are independent of the absolue file size (N): they depend only on the load factor (ratio of used to total record space) and the blocking factor (number of records per bucket). See Fig. 7.

FIGURE 6. Access via Key Hashing

This seems too good to be true, and, alas, it is — for the fact is that we have no guarantee that as we continue loading records to the file, we won't encounter a key which results in an address which has already been used by an earlier loaded record. This will become more and more likely as the file is gradually filled. We have to face the possibility (or rather the virtual certainty) that sooner or later our hashing rule is going to provide an address which is unusable, because it's already occupied.

There are several ways to handle such "collisions", as they are called. One of the simplest is suggested by the observation that when a collision occurs, we are essentially in the same situation as when we started, viz. trying to find a storage location for a record, except that now part of the file is unavailable — we have, in effect, a smaller file to deal with. This immediately suggests that we provide another rule, like the hashing rule, which generates another address which would then be used to store the record. This "collision rule" could then be repeated until we finally found an unoccupied bucket. The only additional restriction to be placed on the collision rule is that it should not generate addresses previously generated for the same key. In this way we are able eventually to find free storage space for all records to be loaded (see Fig. 6).

Note the effect of record blocking on this procedure: if the blocking factor B is equal to 1, then only one record can be stored in a bucket. Any attempt to load another record to a bucket which already holds a single record would immediately cause a collision. If B is greater than 1, however, then several records could be loaded into the same bucket before a collision occurred which forced an application of the collision rule. A large blocking factor therefore reduces the number of accesses required to load or retrieve.

Thus far we've said almost nothing concerning the actual form of the hashing and collision rules. There are in fact many simple ways to implement them. The basic idea in designing such rules is to minimize the chance of collision. It is easy to define the **worst** possible hashing rule: given any key, let the address always be 1 (or 2, or any other constant value). Then, all records would be hashed to the same address, guaranteeing a collision on every load. The ideal, or course, would be a hashing rule which produced no collisions at all. For a given set of keys it is conceivable that a particular rule could be devised which in fact produced unique addresses for all of the keys in the set. This would imply, however, that the keys themselves were somehow incorporated in the rule; such a rule would therefore have to contain at least as much information as a list of the keys themselves, and this translates into large space requirements and execution overhead. Furthermore, such a rule would be useless with keys not expressly built into it. Perfection is therefore out of the question.

Since perfection is denied us, what is the next best? For an arbitrary collection of unique keys, the best hashing rule is one which generates addresses which are spread as uniformly throughout the file as possible, so that clustering in certain buckets does not cause more collisions than necessary. A good hashing rule is therefore a kind of "pseudo-random number generator" which produces all addresses from 1 to N with equal probability.

Lest this sound too esoteric, let me quickly demonstrate a simple example of such a hashing rule. It is not the best rule, but it does work quite well enough to be useful.

First, we must somehow convert the alphanumeric key into numerical data. One very simple way of doing this is to use numerical equivalents for each character in the key string, e.g., their binary ASCII assignments (all ASCII characters can be represented by eight bit integers in the range 0 to 255). The hashing rule we propose is simply to take the ASCII values of the characters forming the key and multiply them together. Thus, for example, suppose KEY is the string ABC. These three characters have ASCII values of 65, 66 and 67; multiplying them together gives 287430. We have thus converted an alphanumeric key into a single numeric value.

This is not enough, however — recall that the rule must produce addresses which lie between 1 and the size of the file; we have no reason to suppose that this number will be in this range. Suppose, however, that we divide the product by the file size N, and then look at just the remainder; this remainder will lie between 0 and N-1, so that if we add 1 to it, we wind up with a number which necessarily falls somewhere between 1 and N: this we take as the address.

If, for example, the file in question contains 101 buckets, then dividing the "hash value" by the file size gives 287430/101 = 2845 with a remainder of 85; the address therefore is 85+1 or 86. We can apply the same rule to other sample keys, with these results:

KEY	Address
ABC	86
ABD	33
AAA	7
ZZZ	84

This rule behaves exactly as we wish it to: for an arbitrary set of keys it produces bucket addresses which are widely scattered through the file. Furthermore, it is exceedingly simple and efficient.

Unfortunately, this rule has a property which makes it less desirable than others which could be devised: the key BAC, or any other permutation of the original key ABC, will be assigned the same address (86) — all keys which are permutations of the same characters will in general be hashed to the same address, resulting in collisions and hence in additional disk accesses. Rules which avoid this to some extent are easily concocted, however. One such is simply to add the position in the key to a character's value before multiplying; thus, for the key ABC we would compute the product of 65+1, 66+2 and 67+3. This rule produces somewhat better results than the first, as shown by an experiment below.

So, assuming we have a hashing rule, what to do about collisions? Perhaps the simplest collision rule is just to advance sequentially to the next block following that found to be full, and continue until one with free space is found. This is indeed simple, and it works; unfortunately it doesn't work very well, because it has a tendency to produce tumor-like clusters of buckets which grow larger as the file is loaded — landing in one of these clusters then obliges one to continue all the way to the end of it before free space is found. Far better is the following, which can be used when steps are taken to ensure that N is a prime number, i.e., not divisible by any number greater than 1 and less than itself (it is not at all difficult to arrange this): having computed the "hash value" H (say) and divided by N to get a remainder which in turn is increased by one and then

used as the bucket address, repeat the division of H but use a divisor smaller than N, say N1. If we take the remainder of **this** division, and add 1 to it, we obtain a number between 1 and N1 (call it J). This number is then used to jump to another bucket, simply by adding J to the hashed address. If the bucket arrived at is also full, we add J to the address and jump again, until we finally come to a block with free space. (If trying to jump would overshoot the end of the file, simply continue at the beginning, just as if the file were connected end to start in a circular fashion.) Requiring that N be prime ensures that any series of jumps of this sort will land on every bucket in the file before returning to the original bucket given by the hashing rule (although we of course hope that free space would be found long before that, say after only one or two collisions). So, for example, using N1 = 99, the key ABC would jump through the file in the progression 86 (the first try), 19, 53, 87, . . . ; the key ABD in the progression 33, 100, 66, 32, ... We thus obtain a very fair approximation to a uniformly "random" generation of bucket addresses.

Hashing as described thus far has been portrayed as an efficient, straightforward procedure. But just how good is it? Can we say anything quantitative about what sort of behavior one might expect from it under real circumstances? It turns out that we can, in basically the same way we treated sequential access earlier. For, although we can't know beforehand how many disk accesses the retrieval of an arbitrary record will take, because of the random distribution of records, we can make use of that very randomness to estimate the **average** number of accesses which could be expected for a large number of loads or retrievals. This is a harder problem than the

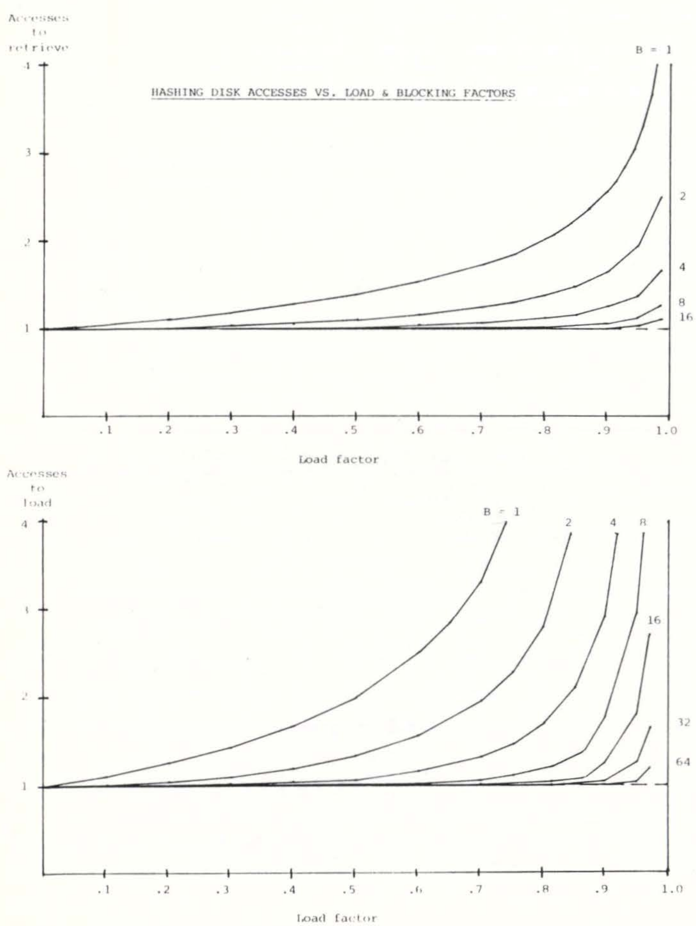


FIGURE 7. Hashing Disk Accesses vs. Load & Blocking Factors

one dealt with earlier, but if we make some reasonable assumptions the theory of probability can be brought to bear to obtain useful estimates. Thus, if we assume 1) that the records are indeed uniformly distributed throughout the file; and 2) that there are a relatively large number of buckets in the file (more than 50, say), then the expected accessing statistics can be computed without too much trouble, with results depicted in Fig. 7.*

The first thing that strikes our attention in these results is that the accessing behavior does not depend at all upon the absolute size of the file, but only on the proportion of file space which is occupied by records, a number which we'll call the **load factor**. Thus, for a file which is 75% full, we would obtain the same number of expected acceses per retrieval with files of all sizes, whether they contained 1,000 blocks or 10,000. This is in sharp contrast to the methods examined earlier, each of which depended in some way upon N, the number of buckets in the file.

Next, we notice that our "line of perfection" (the dashed line at 1) is almost invisible because it is closely mimicked by hashing behavior, particularly with larger blocking factors — with files that are not too heavily loaded and in which the records are multiply blocked, hashing can retrieve keyed records with only a single disk access almost every time.

To come down out of the realm of the theoretical, here is the result of an experiment with a real file, using several different hashing rules:

Average retrieval accesses

Loading:	34%	70%	90%
Method 1	1.1566	1.3999	1.7662
Method 2	1.1231	1.2972	1.5892
Better Method	1.0036	1.0969	1.3511
Ideal	1.0027	1.0557	1.2250

The file used in this test contained 1373 records, with a blocking factor of 4 128-byte records per block, each with a 5-byte key (the reason for the odd record count is that the test file was not really a test file at all, but a live inventory master which was in use at the time). Methods 1 & 2 are the simple examples described above; the "better method" is another simple one which takes more pains than the others to avoid collisions, and to produce a more uniform distribution of addresses. Such rules are not hard to devise; one is encouraged to experiment. The values in the bottom row are those which would be expected with an ideal hashing rule, producing a perfectly random distribution of addresses. The three methods tested represent progressively closer approximations of this ideal.

*For the mathematically inclined, here are the formulae:

Expected accesses to load $= 1/(1-f^B)$

Expected accesses to retrieve $= \sum_{k=0}^{\infty} f^{Bk}/(Bk+1)$

where f is the load factor.

The advantages of hashing are obvious — but what of the disadvantages? We've already alluded to one, namely the impossibility of sequential processing. In some situations this may render the technique unusable, e.g. those where generic or approximate key searches are absolutely required. We can discern another potential drawback from Fig. 7. Although files with large blocking factors can be fairly heavily loaded before disk accesses increase appreciably, there always comes a point beyond which further loading results in a very rapid deterioration in efficiency, because of an increase in the number of collisions entailed in loading records to a file with very little free space. This is reflected in the explosive rise in all the curves of Fig. 7 as the load factor approaches unity. This means that in hashed files some space must be kept empty to ensure retrieval, but especially loading, efficiency; there is thus an overhead in disk space which must be allocated when using hashed files.

Another disadvantage is the obverse of the very property of hashed files which makes them so efficient, namely the fact that there is a very close relationship between key values and record locations. This relationship depends, as we have seen, upon the size of the file. Hence, should it become necessary to change the size of a file, perhaps to enlarge it in order to provide more space, one is obliged to reorganize it completely — all records must be reloaded into new addresses. It is not possible simply to append more blocks at the end of a hashed file, as can be done in principle with ISAM structures. This is usually not too much of a problem, however, if proper attention is given to allocating sufficient file space during the planning stage.

Finally, the accesses involved in loading and accessing individual records can never be predicted with absolute certainty, since hashing depends on probabilities. Most records will probably be retrievable with one or two accesses, especially if the file is not full; one has no absolute guarantee of this, however.

Nevertheless, the efficiencies of hashing usually far outweigh its drawbacks, and when it can be used it is unsurpassed in performance, especially in on-line environments.

Hashing vs. ISAM: a comparison

We've already remarked on some of the differences between ISAM and hashing performance. It is difficult to make more specific quantitative comparisons which take account of the various trade-offs involved because of the fundamentally different structure of the two methods (the "apples vs. oranges" dilemma). In an attempt to devise a meaningful comparison, we shall define the "equivalent hashed file" associated with any given ISAM file: namely, that hashed file which has the same blocking and load factors as does the ISAM file. We have not mentioned the concept of load factor in reference to ISAM files before; we define it now as the proportion of space occupied by the data records themselves to that occupied by the entire structure (data + directory). The ISAM load factor is thus a measure of the disk storage overhead incurred by the directory tree structure, in the same way that the hashed file load factor indicates the overhead (empty file space) which must be present to ensure efficient performance. It can be shown that the load factor for ISAM files is approximately equal to 1-1/B, where B is the blocking factor (the greater the number of buckets in the file, the better the approximation). The equivalent hashed file for an ISAM file which has a blocking factor of B is therefore a hashed file with blocking factor B and load factor 1-1/B. Fig. 8 compares the accesses necessary for retrieval and loading of records with the two methods for various file sizes. A word of explanation is in order for the entries given for ISAM loads, which are perhaps not quite fair. These numbers, computed from a formula given in the RMS-11 User's Guide, represent the disk accesses which are necessary when a record is loaded to an ISAM structure which is completely full and which therefore involves bucket splitting at all levels. By keeping free space in the file buckets, with a corresponding increase in storage overhead, the occurrence of bucket splitting is greatly reduced. In general, the figures given are upper limits on the loading accesses, the retrieval accesses providing a lower limit. The hashing values are statistical expectations associated with an ideal hashing rule; note that although increasing the load factor increases disk accesses in a file with a given blocking factor (cf. Fig. 7), the increasing blocking factor conspires to keep the acccess characteristics within bounds (the retrieval and loading averages in the last columns respectively approach limits of 1 and 1.582 as the blocking factor becomes large).

File size: 100

Blocking	Load Factor	ISAM (Retrieve/Load)	Hashing (Retrieve/Load)
2	.500	8 / 73	1.099 / 1.333
4	.750	5 / 40	1.078 / 1.463
8	.875	4 / 31	1.047 / 1.523
16	.938	3 / 23	1.026 / 1.553
32	.969	3 / 23	1.014 / 1.568
64	.984	3 / 23	1.007 / 1.575

File size: 1000

Blocking	Load Factor	ISAM (Retrieve/Load)	Hashing (Retrieve/Load)
2	.500	11 / 115	1.099 / 1.333
4	.750	6 / 50	1.078 / 1.463
8	.875	5 / 40	1.047 / 1.523
16	.938	4 / 31	1.026 / 1.553
32	.969	3 / 23	1.014 / 1.568
64	.984	3 / 23	1.007 / 1.575

File size: 10000

Blocking	Load Factor	ISAM (Retrieve/Load)	Hashing (Retrieve/Load)
2	.500	15 / 185	1.099 / 1.333
4	.750	8 / 73	1.078 / 1.463
8	.875	6 / 50	1.047 / 1.523
16	.938	5 / 40	1.026 / 1.553
32	.969	4 / 31	1.014 / 1.568
64	.984	4 / 31	1.007 / 1.575

FIGURE 8. Comparative Access Statistics: ISAM vs. Equivalent Hash File

This table shows that in terms of pure performance in random processing, hashing quite definitely carries the day. There will be times, of course, when sequential processing (and hence ISAM) is simply indispensible; but perhaps not as many as one might think. With hashing, sequential processing is "just a sort away". ISAM may seem to obviate the necessity for sorting, but this can be illusory. In some processing situations which are necessarily sequential, such as transaction activity reports, the information which is to govern the sequence of processing may be unknown right up to the time of report generation, e.g., transaction summary quantities. One then has to sort regardless of the mechanism used for random processing. As always, efficient system design requires careful thinking and planning. Fig. 9 summarizes the differences between ISAM and hashing which are relevant to the decision as to which should be used in specific applications.

Conclusion

This discussion has been presented to RSTS users by another RSTS user. The techniques I've described are not examples of textbook exotica; on the contrary, they are quite implementable under RSTS, using BASIC-plus, with very gratifying results. It is possible to construct efficient and powerful file handling systems using the tools which are at hand, and which have been at hand for some years. One may in fact elect to make use of RMS-11. This is a viable alternative, but, as we've seen, there are others. The key, as always, is simplicity. It is only necessary to cultivate economy of thought and simplicity of gesture; then, with a little knowledge, all things **are** possible. Go forth, experiment, explore!

Type	Advantage	Disadvantage
ISAM	Supports approximate & generic key searches, as well as sequential processing. Uniform accesses for all records in file. Can be expanded without file reorganization.	Complex file structure. Accesses increase with increasing file size; relatively large number of accesses. Decreases relatively slowly with increasing blocking factor. Storage overhead: pointers and directory tree. Program overhead: complicated software necessary to maintain pointers & tree structure. Processing overhead: difficult file maintenance can necessitate large number of accesses for record loads.
Hashed	Simple file structure. Very fast (average) access. Accesses independent of file size, decreases rapidly with larger blocking factor. Low program overhead: simple software for all operations. Low processing overhead: few disk accesses necessary for all operations.	No generic or approximate key search (key must be completely specified); no sequential processing. Access characteristics can vary for different records in file. File expansion requires reorganization. Storage overhead: part of file must remain empty for efficient processing.

FIGURE 9. Relative Advantages & Disadvantages of ISAM vs. Hashed Files

FROM THE EDITORS . . .

continued from page 4

Carl B. Marbach

example that should be studied and learned from. So . . .

We started one day by a user showing me:

ODD ADDRESS TRAP
XXXX XXXXX 000000 XXXXXX

This occurred in our cash receipts program which has been running in BASIC PLUS for over two years. It happened on about the 50th receipt. We started over and finished with no more trouble but I was concerned. More and more of these began to occur, one or two a day. Strange things were happening to some disk files; corruption, i.e., strange data in the middle of a file, a program that became data, and finally complete loss of data in a critical data file. Some of the ODD ADDRESS TRAPS were happening in PIP! I decided that because it was so intermittant (once or twice a day) that it was perfect for the RDC to run all night, surely they would pounce on the error and it could get fixed. The report in the morning was a clean bill of health — and for a sick patient that is bad news. Well, the weekend was coming. I let them have it all of Saturday and Sunday. Monday's report was phoned to the branch, something about cache, they were very secretive about what test failed or how they knew it was cache. Field service replaced all of Cache except for one board for which they had no operating replacement. No help. Back to RDC to a RSTS specialist. Yep, we were getting odd addresses and traps to 4 he said after looking at the ERRDIS printout. End of specialists help! One more night that week on the RDC produced no diagnosis. I was still generating only one to two errors a day, how could my local branch help if the mighty RDC couldn't find it in a 60 hour run? Let's try one more weekend. No dice.

Monday morning the Branch came in to replace Memory Management on a guess, all but one board for which they didn't have a working replacement. My own personal theory on the problem after discussions with others, was that the program that was in memory was getting changed; either in swapping or when it was read in, this could explain the CK checksum errors ERRDIS was also reporting. "Do you have a diagnostic that runs itself, copies itself to the disk, reads itself back in and runs again?", I asked. "Yes." "Did the RDC run it?" "Can't tell. Don't know." "Let's try it." It failed gloriously.

Now that there was a diagnostic that failed repeatedly, I declared the machine down and gave it to field service. Twelve hours later a bad diode on a massbus card and a bad foreign massbus interface were found. Who did what to whom is not known but the diode was clearly blown and shorted. Replacing this card, the diagnostic ran, and so has RSTS without anymore errors. The foreign massbus card was also replaced by us.

Throughout the two weeks of trouble, the following became obvious:

1. The branch and the RDC never effectively communicated or discussed the problem.

2. Every call I made to the RDC was to a new person who I had never talked to before. If there was any continuity, I didn't know about it.

3. Subsequent investigation shows that, incredibly, the RDC never ran this diagnostic because they thought it wasn't necessary.

4. When I thought that they were running extended times to find the intermittant problem, they weren't. You have to request that specially, I was told.

5. No one at the RDC cared that this problem had been going on for two weeks. The branch only becomes concerned when the machine is "down". The fact that I was having intermittant problems was my concern, not theirs.

6. I don't know how to deal effectively with the RDC. I do know how to deal with my branch; they are real live people who I know and have a good working relationship with.

7. The RSTS community needs to know more about the RDC and how to use it. It is MY machine and there will be NO secrets about it.

8. If you are having an intermittant problem there is no substitute for declaring it "down" and making it the branch's responsibility.

The RDC is a good idea. They are staffed by some of the most pleasant people I have ever dealt with; "Good morning Carl, let me get some information and we'll try to help you as soon as possible." I like the idea of 15 minute response time and having the branch bring the right part when the system won't boot. I get a questionnaire each year about the field service branch asking how they have performed in the last year. It even has a place for remarks. Each year I give them high marks and consider myself lucky to have the branch I do. I have never been given a similar opportunity to score the RDC or even comment on them. The pleasant voices and quick service aren't enough.

Lastly: If you have comments please let me know; we'll publish as many as we can in the spirit of improving the RDC concept and its use.

MESSAGE: RDC, you're good but you need us (the RSTS community) to help you be as good as you can be. We want to help.

[Next Issue: How the RDC found and solved a problem in record time.]

continued from page 4 **R. D. Mallery**

c) elimination of overhead
  within monitor
  within FIP
  eliminate swapping totally.

PERIPHERAL SPEED
a) full speed disc transfers (RM instead of RP)
b) multiple disc controllers
c) swapping disc emulators for critical files.

MONITOR IMPROVEMENTS
a) multi thread FIP
b) tree-structured directories.

A big problem is that there are some limits in the monitor structure that will make it difficult to use any improvements.

The major limit is the ceiling on space available for small buffers. Four mapping registers (PAR's) in memory management are used to map what is referred to as 'permanently mapped memory'. A number of monitor tables must reside here, and any remaining space is left for small bufers.

One solution is to change some application of small buffers to use XBUF (a la the line printer driver in V7.0). Certainly, CCL's are a waste of small buffers. The real payoff would be a re-work of the terminal driver to use XBUF.

My machine is stalled at 50 jobs and 96 ports due to lack of small buffers. A solution to the small buffers and another MB would clear the way to 112 ports and 63 jobs. I am forced to acquire a second CPU to get 10 more jobs. That means spending nearly $130,000 instead of $25,000 to get those same jobs on my first system. Those are mighty expensive job slots.

Given the abundance and variety of possible improvements and the money to be made from them, all it will take is some pressure exerted by some large users (preferably national accounts) exerted through their product lines to get them implemented.

Dear Dave:

Sorry to be so late getting this letter to you.

I would like to confirm, in writing, what I told you on the phone about the reason why Murray Stinson did not publish your announcement in the Marketplace section of the RSTS SIG newsletter.

The marketplace was established to make information available as to software which might be supplied free or at some unspecified cost. The only publications which might have been listed in the Marketplace were documentation, such as a RSTS user's guide and RUNOFF documentation. Another point is that these were in the form of books or manuals, which were not periodical in publishing frequency.

I checked with Stuart Katz and Maryann Oskirko to confirm the intention of the Executive Board in authorizing the current format of the Marketplace.

If you wish to pursue the issue, Stuart is the next person in the chain between myself and the Board.

Sincerely,

Bruce L. Gaarder, RSTS SIG Chairman

We published this letter intact because we think that all RSTS SIG members should know what goes on behind the scenes, and what they may be missing.

LETTERS to the RSTS Pro ...

... continued from page 6

SENERUG's next meeting was announced for August 28 at Thames Valley State Technical College in Norwich, Conn. with guest speaker Tony Unger of Memorex Corp. on the topic of system networking. Details may be gotten by writing to Box 3043, Pawtucket, R.I. 02861.

—

Dear Sir:

I found Vol. 1, No. 1 while browsing at Crisis Computer Corp. in Santa Clara. I am an ex-PDP-10 hacker from the A-I community, and have been heavily involved with RSTS for the last 3 years, in the transaction processing area. I would like to know if you are still publishing 'R-P', and if back numbers are available. I am willing to write short notes on smoothing the rough edges of RSTS. This magazine is a bold and useful step, and I hope you make a go of it.

Sincerely,
Thomas A. Gafford
G. Systems

We're still here, Thomas. Welcome! We look forward to hearing from you again.

—

RSTS Professional:

Having received my copy of Vol. 2, No. 2 recently may I say again what a refreshing magazine you produce, highly technical but fixed firmly in the real world.

Thank you,
Mark Thornber,
Buckingham, England

*Thank **you**, Mark. We firmly intend to stay here!*

—

Dear Sirs,

We return herewith the additional invoice submitted by your goodselves. Although your current issue indicates that there is an additional charge for overseas, the February/March issue from which we completed the registration card, clearly states that the subscription is 20 dollars. It therefore seems that, at least for the first year, you will have to bear any additional costs yourselves as the registration card was completed as per your invitation.

We look forward to receiving the balance of our three issues with interest, as it is certainly a most 'professional' product.

Yours faithfully,
Capt. J. S. McKenzie
Chairman
Maine & General Group Limited
Douglas, Isle of Man

*Well Captain, you forgot to say it, so we'll say it for you — **Gotcha!***

Our BINARY CLOCK is still gaining fame. Here are the latest late responses from the photo contest of Vol. 2, No. 1, p. 64. The correct time is 3:51:14. We're going to send a prize to those who gave the correct answers, however, because we published this answer in the last issue of the RSTS PROFESSIONAL, we cannot accept entries received after July.

This is a Binary Clock and the time is 3:51:14. Judy Hess, Information Systems, Suncor, Toronto, Ontario.

The time is 3:51 and 14 seconds. Please send prize if still available. Alan C. Sibert, American Used Computer Co., Boston, Massachusetts.

Alan, only half the prize is available for you. We got only half the answer. You told us what the object "said", but not what it "is".

It's clear what the picture is! It's a clock, of course. It reads, 3:51:14. Right? Harry Plantinga, MCS, H.H. Cutler Co., Grand Rapids, Michigan.

Almost, Harry. Your answer is a little cloudy. It's a BINARY clock, not just any clock, of course.

Correct me if I'm wrong, but it looks like a BCD (binary coded decimal) clock with the numbers 3:51:14 illuminated. The reason I say correct me if I'm wrong is that I thought BCD normally read 1 2 4 8 from the bottom up? It doesn't tell a legitimate time that way, however, so I'm guessing it's BCD reading 1 2 4 8 from the top down. If I get my choice of prize, send me the clock! Gordon G. Jones, Director, Academic Computing Services, University of Wisconsin, Menomonie, Wisconsin.

Dear Gordon, BCD, 3:51:14 - correct; bottom up - wrong; top down - correct; choice of prize - wrong. Hope this helps.

The mystery picture is a binary clock which reads 12:54:42. If I had had the money to buy the one I saw in a store I would have, but the things are incredibly expensive! I rather imagine that they would take time to learn to read also, unless one was already very well versed in the reading of binary. Lee S. Van Deest, Nashville, Tennessee.

Sorry, Lee. Check the photo again and take a little more time.

—

DO YOU REMEMBER THIS?

(Photo contest, RSTS Professional, Vol. 2, No. 2, p.34.)

A RSTS T-shirt is on its way to the readers listed below who correctly identified exactly what was going on, which was the building of a Heathkit H19 Video Terminal.

[The RSTS Pro has now bought and built an H19/VT52 compatible terminal. Look for an article on this terminal in an upcoming issue.]

Photo contests will appear in the RSTS PROFESSIONAL occasionally and readers will have until publication of the next issue to submit their answers. We may, from time to time, limit the number of correct answers eligible to receive prizes.

Here are the entires in order of receipt:

Dear Editors:

Regarding "What is going on here?" in "Pro" Vol. 2, # 2, p. 34, it appears a backseat driver is giving assistance to an assembler of a Heathkit H19 or H88 or H89 computer or terminal kit, probably in this case working on the Video Circuit Board, Pack 2. Hard to say, though, since the photo is of such high quality . . .

It's a great magazine. Keep up the quality output.

James A. Meilahn
Waunakee, WI 53597

Dear James, you can have $3 or $4 or $5. (P.S. Thanks for the compliment — we think our readers are great too.)

—

Dear Sirs:

The picture shows the assembly of the Video Circuit Board in the Heath H19 CRT Kit.

Sincerely,
Ollie L. Treadway
Software Resources Inc., Raleigh, N.C.

Fantastic Ollie, you must have used a magnifying glass!

—

Gentlemen:

It appears to be a Health [sic] Kit Microprocessor Trainer being built.

Very truly yours,
Anthony Monteiro, Electrical Engineer
Town of Hudson, Massachusetts
Office of Light & Power

Dear Anthony, we agree. This sort of thing could ruin your health.

—

Dear RSTS Pro People:

I observe on page 34 another contest picture. I must confess that this one is much less obvious than the last . . . I will list my guesses in order.

1. The woman with the smile is assembling a microcomputer.
2. The woman with the smile is disassembling a microcomputer.
3. The woman with the smile is building a single-board VAX, which she intends to use as a video game because she is a devious type.
4. Sour cream.
5. What is really going on is that the other person in the picture is brushing lint off the woman with the smile.
6. The two people in the photograph are building a microcomputer which they intend to use to take over the editing and layout functions of your magazine. You will shortly be outmoded.
7. What do you mean, 'exactly'?
8. Stay with us now, as we learn two new words in Turkish.

Thank you much, anyway. I always like a ridiculous challenge.

Cheers,
Jon Singer
Colorado Video, Inc., Boulder CO

Jon, you were 4th again. And to top it off, we like your 4th answer the best. Sorry we can't cheer you up this time with another prize but keep trying — you may end up with a T-shirt for every day of the week.

—

Dear RSTS Professional,

The lady on page 34 is repairing a printed circuit board.

Howard Fear
Programmer
Computer Resources Corp.

Not quite, Howard.

—

Dear RSTS Professional:

Do I know what it is? Sure I do!!! I've just finished messing up the very board that the pictured lady is working on. It appears to me to be a very cute lady in the process of constructing the video portion of a Heathkit Computer Terminal (either H19 or H89 — it's hard to tell). Our school is in the process of converting (hopefully) to this type of terminal with hopes of making minor repairs locally.

Keep 'em coming,
Mark Muri
Rose-Hulman Institute of Technology,
Terre Haute, IN

No wonder you messed it up, Mark. One has to know exactly what he/she is working on in order to get it right.

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

GOODIES

By Eddie Cadieux

PROPOSED ADDITIONS TO THE PDP-11 INSTRUCTION SET

BBW	Branch Both Ways
BEW	Branch Either Way
BH	Branch and Hang
BMR	Branch Multiple Registers
BOB	Branch on Bug
BOD	Beat on Drum
BOI	Byte Operator Immediately
BPO	Branch on Power Off
BST	Backspace and Stretch Tape
CDS	Condense and Destroy System
CLBR	Clobber Register
CLBRI	Clobber Register Immediately
CM	Circulate Memory
CPPR	Crumple Printer Paper and Rip
CRN	Convert to Roman Numerals
DC	Divide and Conquer
DMNS	Do What I Mean, Not What I Say
DMPK	Destroy Memory Protect Key
DO	Divide and Overflow
EIOC	Execute Invalid Op-Code
EMPC	Emulate Pocket Calculator
EPI	Execute Programmer Immediately
EROS	Erase Read Only Storage
EVGH	Emulate A VG on a Hazeltine
EXCE	Execute Customer Engineer
EXPP	Execute a Political Prisoner
FSRA	Forms Skip and Run Away
HCF	Halt and Catch Fire
IBP	Insert Bug and Proceed
IIP	Ignore Inquiry and Proceed
LCC	Load and Clear Core
MLR	Move and Lose Record
PBC	Print and Break Chain
PDSK	Punch Disk
PI	Punch Invalid
POPI	Punch Operator Immediately
PVLC	Punch Variable Length Card
RASC	Read and Shred Card
RIRG	Read Inter Record Gap
RPM	Read Programmers Mind
RSD	On Read Error Self Destruct
RSTOM	Read From Store Only Memory
RWOC	Read Writing on Card
SPSW	Scramble Program Status Word
SRSD	Seek Record and Scar Disk
SRZ	Subtract and Reset to Zero
SSJ	Select Stacker and Jam
STROM	Store in Read Only Memory
TDB	Transfer and Drop Bit
UER	Update and Erase Record
WBT	Water Binary Tree
WIRG	Write Inter Record Gap

GOODIES

By Eddie Cadieux

FOLDED, SPINDLED & MUTILATED,
SHAFTED, ELECTRONICALLY, INC.

ON BEHALF OF COMPUTER VICTIMS EVERYWHERE,
GO TO 8-5-12-12 AND GET OFF MY 2-1-3-11 EXCLAMATION POINT

Giant Credit Corp.
1929 Depression Street
Conglomerate, WI

Dear Computer:

I am 407-923-3101-D. The "D", as I can tell from your warm and friendly ink-jet letters, means you think of me as a deadbeat. I am not.

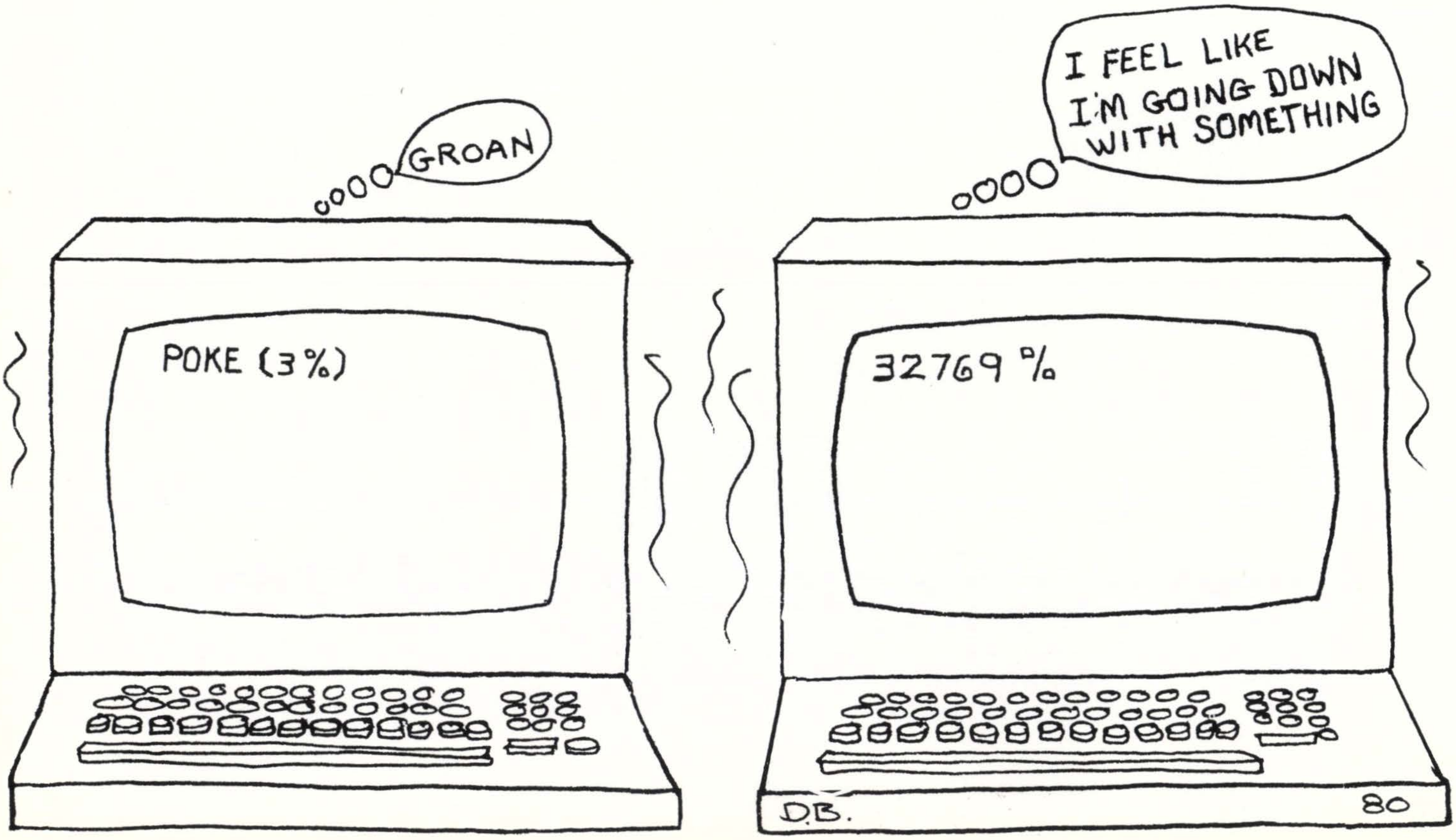
On January 3rd of this year I purchased an electronic TV game from Marty's Mall Outlets of Chicago and I guess they sold you the "paper" for my balance due of $87.13. You wrote me a letter thanking me and sending me a payment book. I made one payment.

Then my electronic game went haywire. Not only did I electrocute a perfectly good cat, the Civil Aeronautics Board claims the interference from my set caused a 727 to become a stretch 727 in mid-air.

To top it all off, the game blew up my $1200 TV set and the ensuing fire destroyed not one room of my apartment but all three apartments above me. No one was hurt in the fire but during the evacuation I saw my wife emerge from Clyde Metcalf's apartment and — since I hate Clyde — I am now divorced.

I feel no obligation to pay another dime for the blasted unit and I suggest you ink-jet Marty's for the balance. You won't get it from —

407-923-3101-D
The D stands for Disgusted



SETUTL.MAC

System Startup Utility

By Steven L. Edwards, Software Techniques, Inc.

Watching a RSTS system bring itself to life can be an emotional (almost religious) experience. The first time. After that it soon becomes quite frustrating to have to sit and wait for 5 minutes while the system crawls back to life.

The most frustrating thing about system startup is that it is precisely the same thing each time. With V7.0 we have the capability to do a 'silent startup.' 'Silent startup' is nice but it just hides the problem. The problem is that the CUSPS have to interpret the commands forced to them, when what we really need is a program that already knows how to start up our system and does it. Fast.

SETUTL is a system startup utility written in MACRO-11. The program will add CCL's, libraries, logicals, RTS's, and set terminal characteristics. Since the macros to set terminals take about 10 pages of code, these macros were edited out for publication. As you can see from the sample run below, SETUTL is a solution.

```
INIT     V7.0-07A          RSTS V7.0-07 Development

Command File Name? ↑C
>
1  KB0  [1,2]   ...MCR+RSX     ↑C(0R)  1(28)K+3K     0.7(+0.7) +0
RUN SETUTL

Setutl  V7.0-03 Software Techniques

** SETUTL ** Adding Logicals
** SETUTL ** Adding CCL's
** SETUTL ** Adding Libraries
** SETUTL ** Adding Run-time Systems
** SETUTL ** Adding Setting terminals
>
1  KB0  [1,2]   ...MCR+RSX     ↑C(0R)  1(28)K+3K     1.5(+0.8) +0
```

To use the program, edit the source file and add the commands your installation needs after the label 'START:'. Sample commands are included. Because of the complexity of the RTS command, commands to add the DEC standard RTS's are also included.

The program generates threaded code to facilitate the generation of memory efficient and highly modular code. The generated code is also highly run-time efficient because all of the real work is done at assembly time. Using this program, and 'silent startup,' we can bring an 11/70 completely up (including operator services) in 50 seconds. SETUTL runs in about 1 second. On an 11/34 SETUTL runs in about 2 seconds.

If you find any bugs in this program or would like to see some other commands added, please contact me.

To receive the full version of this program (including the terminal macros) on a 9-track 800 BPI tape, send $25 to Software Techniques, 10841 Chestnut Street, Los Alamitos, CA 90720.

```
        .ENABL  LC

TITLE   SETUTL,<SET SYSTEM CHARACTERISTICS>,03,13-JUL-80,<SLE>

;       PACKAGE:        FREEBEE

;       WRITTEN BY:     STEVEN L. EDWARDS

;       DATE:           19-MAY-80

;       SOFTWARE TECHNIQUES
;       LOS ALAMITOS, CALIFORNIA 90720

;       THIS  INFORMATION  IN  THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
;       NOTICE  AND  SHOULD  NOT BE CONSTRUED AS A COMMITMENT BY SOFTWARE
;       TECHNIQUES.

;       THIS  SOFTWARE  IS  UN-RELEASED  AND  SOFTWARE  TECHNIQUES HAS NO
;       COMMITMENT  TO  SUPPORT  IT AT THIS TIME, UNLESS STATED ELSEWHERE
;       IN WRITING.

;       THIS  SOFTWARE  IS  BEING  MADE AVAILABLE FREE OF CHARGE TO DECUS
;       MEMBERS  FOR  USE  ON  A  SINGLE  COMPUTER  SYSTEM.  RIGHT  OF
;       DISTRIBUTION TO A THIRD PARTY IS NOT GRANTED.  SO THERE.

;
;       MODIFICATION HISTORY
;
;       VER/EDIT        DATE            REASON
;       --------        ----            ------

;       7.0-01          19-MAY-80       INITIAL CONCEPTION.
;       7.0-02          28-MAY-80       FIX FILL MACROS, ADD HEADER.
;       7.0-03          13-JUL-80       ADD "ENDING IN ERROR" STUFF.
;       7.0-03NT        17-JUL-80       SPECIAL VERSION FOR THE RSTS
;                                         PROFESSIONAL (NO TERMINALS)
```

```
;
;       GENERAL DESCRIPTION
;

                THE  PURPOSE  OF  THIS  PROGRAM  IS  TO  SET  ALL  SYSTEM
;       CHARACTERISTICS IN AS LITTLE TIME AS POSSIBLE.  THIS PROGRAM WILL
;       SET UP CCL'S, LOGICAL'S, LIBRARIES, AND RTS'S.


                THE FORMAT OF THE VARIOUS COMMANDS IS:
;

;       CCL'S

;               ADDCCL  <COM>,<MAND>,<PROGRAM NAME>[,PRIV[,LINE]]

;               WHERE 'COM' IS THE UNIQUE PART OF THE COMMAND, AND 'MAND'
;                 IS NOT.

;       LOGICALS

;               ADDLOG  <PHYSICAL>,<LOGICAL>

;       LIBRARIES

;               ADDLIB  <NAME>,ADDR,POS,STAY,<P.FLAG>

;               WHERE  'POS'  IS THE POSITION IN THE LIBRARY LINKED LIST;
;                 P.FLAG IS THE LIBRARY DESCRIPTOR WORD.

;       RUN-TIME SYSTEMS

;               ADDRTS  <NAME>,ADDR,P.SIZE,P.MSIZ,POS,STAY,<P.FLAG>,<P.DEXT>

;               WHERE  P.SIZE  IS  THE  MAXIMUM  USER SIZE; P.MSIZ IS THE
;                 MINIMUM  USER SIZE; POS IS THE POSITION IN THE RUN-TIME
;                 SYSTEM  LINKED  LIST;  P.FLAG  IS  THE  RUN-TIME SYSTEM
;                 DESCRIPTOR  WORD;  P.DEXT  IS  THE  DEFAULT  EXECUTABLE
;                 EXTENSION

;       SEND

;               SEND    <TEXT>

;               THIS COMMAND ALLOWS SETUTL TO SEND A MESSAGE TO KB0:.

;       ASSEMBLY INSTRUCTIONS:

;               EDIT  THE  SOURCE  FILE TO INCLUDE THE COMMANDS NEEDED BY
;                 YOUR INSTALLATION AFTER THE LABEL 'START:'

;               MAC SETUTL=COMMON,SETUTL        ; INVOKE MAC.TSK
;                                               ; COMMON.MAC IS ON THE SYSGEN
;                                               ;    TAPE

;               TKB SETUTL=SETUTL               ; INVOKE TKB.TSK

;
;       GLOBAL SYMBOLS
;

        .MCALL  EXIT$S

        .SBTTL  VARIABLES

        .SBTTL  ORDER THE PROGRAM SECTIONS.

        ORG     CODE                    ; PROGRAM CODE AREA.

        ORG     TEXT                    ; COMMAND AND PROGRAM NAME TEXT.

        ORG     THREAD                  ; THREAD FOR PROGRAM CONTROL.

;
;       VARIABLE DESCRIPTION AND INITIALIZATION
;

        .PSECT  CODE

ERR:    .WORD   0                       ; ERROR COUNTER.

PRIV    =       100000                  ; DEFINE PRIV CCL BIT VALUE.

STAY    =       128.                    ; DEFINE THE STAY FLAG FOR RTS'S
                                        ;   AND RESLIB'S.

..CTY.  =       0.                      ; CONSOLE TERMINAL FOR SENDS.

        .SBTTL  ADDCCL MACRO

;
;       MACRO TO SETUP FOR CCL.
;

.MACRO  ADDCCL  UNQ, CMD, PRG, PRV=0, LIN=0
        TMPORG  TEXT                    ; ENTER TEXT AREA.
$TEMP0  =       .
        .ASCII  / - ADDCCL /
$TEMP1  =       .
        .ASCII  /UNQ/
$TEMP2  =       .-$TEMP1
        .ASCII  /CMD/
.REPT   11-<.-$TEMP1>
        .BYTE   0
.ENDR
        .ASCII  / = /
$TEMP3  =       .
        .ASCII  /PRG/
$TEMP4  =       .-$TEMP3
        .BYTE   15,12
$TEMP5  =       .-$TEMP0
        UNORG
        TMPORG  THREAD                  ; INTO OUR THREAD.
        .WORD   FSS                     ; DO A FILENAME STRING SCAN.
        .WORD   $TEMP4                  ; LENGTH OF STRING TO SCAN.
        .WORD   $TEMP3                  ; ADDRESS OF STRING TO SCAN.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP5                  ; LENGHT OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        .WORD   CCL                     ; SET THE STRING FOR A CCL.
        .WORD   $TEMP1                  ; ADDRESS OF COMMAND.
        .WORD   $TEMP2                  ; LENGTH OF UNIQUE.
        .WORD   PRV+↑D<LIN>             ; LINE NUMBER.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP5                  ; LENGHT OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        UNORG
.ENDM   ADDCCL

        .SBTTL  ADDLIB MACRO

;
;       MACRO TO SETUP FOR LIB.
;

.MACRO  ADDLIB  NAME, P.LOAD, POS=0, STY=0, P.FLAG=0
        TMPORG  TEXT                    ; INTO THE TEXT AREA.
$TEMP0  =       .
        .ASCII  / - ADDLIB /
$TEMP1  =       .
        .ASCII  /NAME/
$TEMP2  =       .-$TEMP1
        .BYTE   15,12
$TEMP3  =       .-$TEMP0
        UNORG
        TMPORG  THREAD                  ; INTO THE THREAD AREA.
        .WORD   FSS                     ; DO A FILENAME STRING SCAN.
        .WORD   $TEMP2                  ; LENGTH OF STRING TO SCAN.
        .WORD   $TEMP1                  ; ADDRESS OF STRING TO SCAN.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP3                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        .WORD   LIB                     ; DO THE ADD LIB.
        .WORD   ↑D<P.LOAD>              ; LOAD ADDRESS.
        .WORD   0
        .WORD   0
        .BYTE   ↑D<POS>                 ; LINKED-LIST POSITION.
        .BYTE   ↑D<STY>                 ; STAY FLAG.
        .WORD   <P.FLAG>                ; FLAG WORD.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP3                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.

        UNORG
.ENDM   ADDLIB

        .SBTTL  ADDLOG MACRO

;
;       MACRO TO SETUP FOR LOG.
;

.MACRO  ADDLOG  PHY, LOGICAL
        TMPORG  TEXT                    ; INTO THE TEXT AREA.
$TEMP0  =       .
        .ASCII  / - ADDLOG /
$TEMP1  =       .
        .ASCII  /PHY/
$TEMP2  =       .-$TEMP1
$TEMP3  =       .
        .ASCII  /LOGICAL/
$TEMP4  =       .-$TEMP3
        .BYTE   15,12
$TEMP5  =       .-$TEMP0
        UNORG
        TMPORG  THREAD                  ; INTO THE THREAD AREA.
        .WORD   FSS                     ; DO A FILENAME STRING SCAN.
        .WORD   $TEMP2                  ; LENGTH OF STRING TO SCAN.
        .WORD   $TEMP1                  ; ADDRESS OF STRING TO SCAN.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP5                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        .WORD   FSS1                    ; DO A FILENAME STRING SCAN.
        .WORD   $TEMP4                  ; LENGTH OF STRING TO SCAN.
        .WORD   $TEMP3                  ; ADDRESS OF STRING TO SCAN.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP5                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        .WORD   LOG                     ; DO THE ADD LOGICAL.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP5                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        UNORG
.ENDM   ADDLOG

        .SBTTL  ADDRTS MACRO

;
;       MACRO TO SETUP FOR RTS.
;

.MACRO  ADDRTS  NAME, P.LOAD=0, P.SIZE, P.MSIZ, POS=0, STY=0, P.FLAG=0, P.DEXT
        TMPORG  TEXT                    ; INTO THE TEXT AREA.
$TEMP0  =       .
        .ASCII  / - ADDRTS /
$TEMP1  =       .
        .ASCII  /NAME/
$TEMP2  =       .-$TEMP1
        .BYTE   15,12
$TEMP3  =       .-$TEMP0
        UNORG
        TMPORG  THREAD                  ; INTO THE THREAD AREA.
        .WORD   FSS                     ; DO A FILENAME STRING SCAN.
        .WORD   $TEMP2                  ; LENGTH OF STRING TO SCAN.
        .WORD   $TEMP1                  ; ADDRESS OF STRING TO SCAN.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP3                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        .WORD   RTS                     ; DO THE ADD RTS.
        .WORD   ↑D<P.LOAD>              ; LOAD ADDRESS.
        .WORD   ↑D<P.SIZE>              ; MAX USER JOB IMAGE SIZE.
        .WORD   ↑D<P.MSIZ>              ; MIN USER JOB IMAGE SIZE.
        .BYTE   ↑D<POS>                 ; LINKED-LIST POSITION.
        .BYTE   ↑D<STY>                 ; STAY FLAG.
        .WORD   <P.FLAG>                ; FLAG WORD.
        .RAD50  /P.DEXT/                ; DEFAULT EXTENSION.
        .WORD   ERROR                   ; CHECK FOR AN ERROR.
        .WORD   $TEMP3                  ; LENGTH OF TEXT.
        .WORD   $TEMP0                  ; ADDRESS OF TEXT.
        UNORG
.ENDM   ADDRTS

        .SBTTL  SEND MACRO

;
;       THIS MACRO ALLOWS THE USER TO SEND A TEXT STRING TO KB ..CTY.
;

.MACRO  SEND    STR
        TMPORG  TEXT                    ; INTO THE TEXT AREA.
$TEMP0  =       .
        .ASCII  /** SETUTL ** /
        .ASCII  /STR/
        .BYTE   15,12
$TEMP1  =       .-$TEMP0
        UNORG
        TMPORG  THREAD                  ; INTO THE THREAD AREA.
        .WORD   SND                     ; ROUTINE TO DO THE SEND.
        .WORD   $TEMP1                  ; LENGTH OF STRING.
        .WORD   $TEMP0                  ; ADDRESS OF STRING.
        UNORG
.ENDM   SEND
```

```
        .SBTTL  PROGRAM CODE

;
;       MAINLINE PROGRAM
;

        .PSECT  THREAD
THREAD:

        TMPORG  TEXT                    ; INTO THE TEXT AREA.
$TEMP0  =       .

        .BYTE   15,12
        .ASCII  /Setutl V/
        .BYTE   SYSVEL/400&377,SYSVEE&377
        .BYTE   SYSVEE/400&377,'-
        .BYTE   $$$VER&377,$$$VER/400&377
        .ASCII  /NT/
        .ASCII  /       Software Techniques/
        .BYTE   15,12,12
$TEMP1  =       .-$TEMP0
        UNORG
        TMPORG  THREAD                  ; INTO THE THREAD AREA.
        .WORD   SND                     ; ROUTINE TO DO THE SEND.
        .WORD   $TEMP1                  ; LENGTH OF STRING.
        .WORD   $TEMP0                  ; ADDRESS OF STRING.
        UNORG

START:
        SEND    <Adding Logicals>

        ADDLOG  <SY:>,<RPT>
        ADDLOG  <DB1:>,<OV>
        ADDLOG  <DB2:[1,1]>,<LB>
        ADDLOG  <SY:[1,9]>,<STEVE>
        ADDLOG  <SY:[101,101]>,<FUN>

        SEND    <Adding CCL's>

        ADDCCL  <@>,<>,<[1,2]ATPK.*>,PRIV,30000
        ADDCCL  <BAS>,<IC>,<[1,3]SWITCH.TSK>,PRIV,30050
        ADDCCL  <BP2>,<>,<LB:[1,2]BASIC2.TSK>,,30000
        ADDCCL  <IFL>,<>,<[1,2]RMSIFL.TSK>
        ADDCCL  <LI>,<>,<[1,3]DIR.TSK>,PRIV,30000
        ADDCCL  <RT>,<11>,<[1,3]SWITCH.TSK>,PRIV,30050
        ADDCCL  <UT>,<ILTY>,<[1,2]UTILTY.SAV>,,8192
        ADDCCL  <ZAP>,<>,<[1,3]ZAP.TSK>

        SEND    <Adding Libraries>

        ADDLIB  <DB0:[0,1]BASICS>,<39>
        ADDLIB  <DB0:[0,1]RMSRES>,<93>

        SEND    <Adding Run-time Systems>

        ADDRTS  <TECO>,0,24,4,0,0,<PF.KBM>,<TEC>
        ADDRTS  <BP2COM>,0,28,1,0,0,<PF.KBM>,<TSK>
        ADDRTS  <RT11>,0,28,2,0,0,<PF.KBM!PF.CSZ!PF.EMT!377>,<SAV>
        ADDRTS  <BASIC2>,0,16,1,0,0,0,<TSK>
        ADDRTS  <BASIC>,0,16,2,0,0,<PF.KBM!PF.CSZ>,<BAC>
        ADDRTS  <RMS11>,0,28,1,0,0,0,<TSK>
        ADDRTS  <BASICD>,0,12,2,0,0,<PF.KBM!PF.CSZ>,<BAC>

        .SBTTL  SET UP THE THREADED EXIT.

        TMPORG  THREAD                  ; INTO OUR THREAD.
        .WORD   EXIT                    ; EXIT PROGRAM.
        UNORG

        SEND    <%Ending in error.>

        TMPORG  THREAD                  ; INTO OUR THREAD.
        .WORD   EXIT                    ; EXIT PROGRAM. (AGAIN)
        UNORG

        .PSECT  CODE

SETUTL: MOV     #THREAD,R4              ; SET UP THREAD.
        JMP     @(R4)+                  ; JUMP THROUGH THREAD.

EXIT:   TST     ERR                     ; ANY ERRORS?
        BEQ     10$
        CLR     ERR                     ; SO THAT WE EXIT NEXT TIME.
        JMP     @(R4)+                  ; JUMP THROUGH THREAD.

10$:    EXIT$S

        .SBTTL  SUBROUTINES

;
;       DO A UU.CCL.
;

CCL:    CMP     #134745,@#FIRQB+FQEXT   ; DID THE USER SPECIFY "???"
        BNE     2$                      ; IF SO THEN,
        MOV     #177777,@#FIRQB+FQEXT   ;   FORCE A "-1."
2$:     MOV     (R4)+,R0                ; ADDRESS OF COMMAND.
        MOV     #11,R1                  ; LENGTH OF COMMAND.
        MOV     #FIRQB+FQSIZ,R2         ; AREA FOR COMMAND.
3$:     MOVB    (R0)+,(R2)+             ; MOVE A BYTE.
        SOB     R1,3$
        MOV     #FIRQB+FQFUN,R0         ; ADDRESS OF FUNCTION CODE.
        MOVB    #UU.CCL,(R0)+           ; FUNCTION CODE.
        CLRB    (R0)+                   ; OBLIGATORY ZERO.
        MOVB    (R4)+,(R0)+             ; UNIQUE LENGTH
        INC     R4                      ; FUDGE A LITTLE.
        MOV     (R4)+,@#FIRQB+FQCLUS    ; LINE NUMBER.
        .UUO                            ; DO THE CALL.
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       INFORM USER OF ANY ERRORS.
;

ERROR:  TSTB    @#FIRQB                 ; ERROR?
        BNE     10$                     ; TOO BAD.
        CMP     (R4)+,(R4)+             ; BUMP PAST STRING STUFF.
        BR      99$                     ; END OF ROUTINE.

10$:    INC     ERR                     ; INCREMENT OUR ERROR COUNTER.
        MOV     #FIRQB+FQFUN,R1         ; ADDRESS OF FUNCTION CODE.
        MOVB    #ERRFQ,(R1)+            ; FIP FUNCTION CODE.
        MOVB    @#FIRQB,(R1)            ; ERROR NUMBER.
        CALFIP
        MOV     R1,R2                   ; SAVE FIRQB+FQFIL.
        CLR     @#XRB                   ; MAKE SURE WE HAVE A NULL WORD.
20$:    TSTB    (R1)+                   ; CHECK OUT THE SCENE.
        BNE     20$                     ; GO FOR THE NULL!
        SUB     R2,R1                   ; NOT TO GET PERSONAL BUT,
        CALL    SETXRB                  ;   HOW LONG IS IT?
        MOV     R1,(R0)                 ; LENGTH OF MESSAGE.
        MOV     (R0)+,(R0)+             ;   "      "     "
        MOV     R2,(R0)                 ; ADDRESS OF ERROR TEXT.
        .WRITE
        CALL    SETXRB
        MOV     (R4)+,(R0)              ; LENGTH OF MESSAGE.
        MOV     (R0)+,(R0)+             ;   "      "     "
        MOV     (R4)+,(R0)+             ; MESSAGE ADDRESS.
        .WRITE
99$:    JMP     @(R4)+                  ; ON TO NEXT COMMAND.

;
;       DO A .FSS ON A STRING.
;

FSS:    CALL    SETFQB                  ; CLEAR FIRQB.
FSS1:   CALL    SETXRB                  ; CLEAR XRB.
        MOV     (R4)+,(R0)              ; LENGTH OF STRING.
        MOV     (R0)+,(R0)+             ;   "      "     "
        MOV     (R4)+,(R0)              ; ADDRESS OF STRING.
        .FSS
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       ADD THE LIBRARY.
;

LIB:    MOV     #FIRQB+FQFUN,R0         ; OFFSET TO FUNCTION CODE.
        MOVB    #UU.RTS,(R0)+           ; FUNCTION CODE.
        MOV     #16.,(R0)               ; ADD LIBRARY SUBFUNCTION.
        MOV     #FIRQB+FQEXT,R0         ; RESET THE POINTER.
        MOV     #5,R1                   ; NUMBER OF WORDS TO COPY.
10$:    MOV     (R4)+,(R0)+             ; MOVE A WORD...
        SOB     R1,10$                  ; UNTIL WE ARE DONE.
        .UUO                            ; UUO HOOK.
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       DO A UU.SLN.
;

LOG:    MOV     #FIRQB+FQFUN,R0         ; OFFSET TO FUNCTION CODE.
        MOVB    #UU.SLN,(R0)+           ; FUNCTION CODE.
        MOV     #1,(R0)                 ; ADD SLN SUBFUNCTION CODE.
        .UUO                            ; UUO HOOK.
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       DO A UU.RTS
;

RTS:    MOV     #FIRQB+FQFUN,R0         ; POINT TO FQFUN.
        MOVB    #UU.RTS,(R0)            ; .UUO FUNCTION CODE.
        MOV     #FIRQB+FQEXT,R0         ; POINT TO FQEXT.
        MOV     #6,R1                   ; SIX WORDS.
10$:    MOV     (R4)+,(R0)+             ; MOVE A WORD.
        SOB     R1,10$
        .UUO
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       DO THE SEND.
;

SND:    CALL    SETXRB                  ; CLEAR XRB.
        MOV     #6,(R0)+                ; "BROADCAST TO KEYBOARD"
        MOV     (R4)+,(R0)+             ; LENGTH.
        MOV     (R4)+,(R0)+             ; ADDRESS.
        MOV     #TTYHND*400,(R0)+       ; TTY HANDLER INDEX.
.IF     NE      ..CTY.
        MOV     #..CTY.,(R0)            ; KEYBOARD NUMBER.
.ENDC
        .SPEC                           ; SPECIAL FUNCTION CODE.
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       SET THE TERMINAL.
;

TRM:    CALL    SETFQB                  ; CLEAR FIRQB.
        DEC     R0                      ; MOVE BACK A BYTE.
        MOV     #15,R2                  ; NUMBER OF WORDS TO COPY.
1$:     MOV     (R4)+,(R0)+             ; MOVE A WORD...
        SOB     R2,1$                   ; UNTIL WE ARE DONE.
        .UUO                            ; UUO HOOK.
        JMP     @(R4)+                  ; ON TO THE NEXT COMMAND.

;
;       COMMON SUBROUTINES.
;

SETFQB: MOV     #FIRQB,R0               ; ADDRESS OF FIRQB.
        MOV     #FIRQB+FQFUN,-(SP)      ; PUSH FUNCTION ADDRESS.
        MOV     R1,-(SP)                ; PUSH R1.
        MOV     #FQBSIZ/2,R1            ; SIZE IN WORDS.
        BR      CLEAR                   ; JUMP TO COMMON CODE.

SETXRB: MOV     #XRB,R0                 ; ADDRESS OF XRB.
        MOV     R0,-(SP)                ; PUSH ADDRESS OF XRB.
        MOV     R1,-(SP)                ; PUSH R1.
        MOV     #XRBSIZ/2,R1            ; SIZE IN WORDS.

CLEAR:  CLR     (R0)+                   ; ZAP A WORD.
        SOB     R1,CLEAR                ; UNTIL DONE.
        MOV     (SP)+,R1                ; POP R1.
        MOV     (SP)+,R0                ; POP R0.
        RETURN

        .END    SETUTL
```

RSTS DISK DIRECTORIES, Part 3

By Scott Banks, Systems Design

3.1 We're back . . .

I apologize for the delay in getting Part 3 to press. Thanks to all the nice folks who complained about it's absence in the last issue. For newcomers, two prior articles* introduced the basic directory concepts and definitions. In this third article we will explore the RSTS directory structure via practical programming examples. The eight figures from Part 2 are reproduced here for your convenience.

3.2 Open the UFD as a File

As stated earlier, all UFDs are themselves files. RSTS allows us to open any UFD as though it were a file. Internally, FIP deals with UFDs somewhat differently than data files. For the user, however, I/O operations appear to function identically. To open the directory for [1,2] on the system disk, we use this statement:

```
100    OPEN '[1,2]' FOR INPUT AS FILE 1%
```

Because we have supplied a PPN but no filename, FIP knows that we mean the UFD for this account. If an account on a disk other than the system disk were desired, it must be explicitly named (e.g., 'DB1:[2,3]').

To open a UFD requires privilege. When a UFD is opened without a MODE modifier (or MODE 0), the UFD is write-protected. In this way, the system is protected from the hazards of transient programmers and programs. By definition, persons (and their computers) who are involved in casually writing to UFDs are necessarily transient.

It is possible to write to a UFD by using MODE 16384 at open time. This allows grand and sweeping gestures like 'Directory Repair' (post-crash, presumably). Do not ever attempt this until such time as you are approached to write directory structure articles for an international RSTS publication. In a future article I plan to discuss some of the techniques that can be successfully used in crash recovery as well as directory maintenance. For the present, we will concentrate solely on inspection of the UFD.

3.3 Virtual Arrays

As it happens, the virtual array facility of Basic-Plus is ideally suited for our directory investigation. If you are not familiar with this feature, become so at this time. For UFD manipulation, we will use a very specific dimension statement:

```
110    DIM #1, U%(3583,7)
```

I have chosen to use the integer array U% to map the disk directory. The first subscript will choose the blockette number desired. The word within that blockette is described by the second index value. This scheme will automatically worry about which physical disk block of the directory is required. The first disk block contains blockettes 0 through 31, the second contains 32 through 63, and so on. The value 3583 is simply one less than the maximum number of blockettes a UFD could possibly contain. (My faithful readers will quickly calculate: 7 clusters of 16 blocks of 32 blockettes each...).

Figure 2 depicts the UFD Label blockette (the LB is always the first blockette of the first block — therefore blockette 0). To access the link word in the LB, we use U%(0,0). A -1 will be found in U%(0,1), and zero values in U%(0,2) through U%(0,5). Packed in U%(0,6) will be the project and programmer numbers of this UFD. Part of the definition of the Label blockette is that 'UFD' (in RAD50) is stored in word 7. This is easily verified by printing the value of U%(0,7), -31692 if you want to try it.

3.4 Link Words

Once having opened a UFD and DIMensioned a virtual array, any word in any blockette may be accessed. The link word of the LIB is the starting point for a directory 'walk'. This link will point to the name blockette of the first directory entry. Look at Figure 8, the breakdown of a typical link word. Ignoring the special bit flags, there are 3 fields of interest. The link tells us which cluster we want, which block within that cluster, and which blockette within that block. To use the virtual array, we must perform a little math to determine the unique blockette to whom all this attention is directed.

Fortunately, as with normal data files, FIP ties all the clusters together leaving us only to deal with block numbers. The virtual array logic connects all the blocks together sequentially. The two factors needed to convert a cluster/block/blockette link word to a blockette index are the number of blockettes per block and the UFD clustersize. There are always 32 blockettes to a block. The clustersize for a given UFD must be accurately determined, however. Assume we include this line of code:

```
120    CLU% = U%(31%,0%)
```

Figure 3, a view of the File Directory Cluster Map (FDCM) reveals that word 0 holds the clustersize. As you recall,

*RSTS Professional Vol. 1, No. 1, p.30 and Vol. 2, No. 1, p.45.

blockette 31 of each and every block of the UFD contains an identical copy of the FDCM. The most direct approach is to use blockette 31. CLU% is now available for use in our calculations.

Let's implement the link word transformation as an integer function. The input parameter will be the link word itself. The result of the function will be that blockette number for which we yearn. Here's one way to write the function:

```
10000 DEF FNLINK%(L%) =
   ((( L% AND 3584% ) / 512% ) * CLU%
 + ( SWAP%( L% AND -4096% ) / 16% )) * 32%
 + ((  L% AND 496% ) / 16% )
```

A quick synopsis of this function is in order. After the DEF statement, the first line isolates link word bits 9 through 11— the cluster number. This is multiplied by the clustersize CLU% to contribute to the block offset. Bits 12 to 15 are isolated in the second line yielding the block number (again, within the cluster). This block number is added to that derived in the first line. Multiplying this sum by 32 provides a count of just how many blockettes must be 'skipped' to arrive at the proper block offset. The third line isolates bits 4 through 8, the blockette number, which is summed with the offset. This blockette number is returned to the calling expression.

A null link word is one that points 'nowhere'. Except for the special bit flags, a zero word is defined as null. Rather than test the link word for zero, test the result of FNLINK. The function completely disregards link word bits 0 through 3.

3.5 Promenade

The ability to step through a UFD, entry by entry, is the basic ingredient of nearly all directory-intensive programs. We begin by picking up the first link word from the LB. If this pointer is null, there are no entries in the directory. Normally, it will refer us to the Name blockette (NB) of the first entry. Word 0 of this NB will point to the next NB. The process continues until a null link is encountered. This subroutine will walk through a directory:

```
1000    PTR% = FNLINK%(U%(0%,0%))           !SET FIRST LINK          &
                                            !
1020    GOTO 1190 UNLESS PTR%               !NULL    EXIT            &
        \PTR% = U%(PTR%,0%)                 !POINT TO NEXT NB        &
        \GOSUB 2000                         !PROCESS THIS ENTRY      &
        \GOTO 1020                          !LOOP FOR NEXT ENTRY     &
        !
1190    RETURN                                                       &
        !
```

This routine calls a subroutine at line 2000 for each directory entry. If 2000 printed the name of the entry (contained in the NB), a directory listing would result. For example:

```
2000    PRINT RAD$(U%(PTR%,1%));            !PRINT THE FILENAME      &
              RAD$(U%(PTR%,2%)); '.';                                &
              RAD$(U%(PTR%,3%));            !AND EXTENSION           &
        !
2020    PRINT '<';                          !THEN PRINT PROTECTION   &
          SWAP%(U%(PTR%,4%)) AND 255%;      !  CODE ON SAME LINE     &
            '>';                                                     &
        !
2180    PRINT                               !FINISH THIS LINE        &

        !
2190    RETURN                                                       &
        !
```

Line 2000 prints the filename, including a dot between the name and extension parts. I peeked at Figure 4 to get the layout of the Name blockette. The high byte of word 4 was used in line 2020 to print the protection code of this entry.

Of course, if the routine at line 2000 compared the file's name with some known filename, a search would be possible. In this way one could locate any specific file in a directory. Similary, such a technique allows selective listing of directories. Only files matching *.BAS may be desired, for example.

The DIRECT program depends upon just this directory walk to do it's job. The real difference between this utility and our sample program is the attention paid to detail of the options provided.

3.6 Accessing the Accounting Blockette

In our previous example routine, all the information we required appeared in the Name blockette. Remember that the NB contains a pointer to the Accounting blockette. To print the date of last access, found in the AB, we add this line:

```
2040    AB% = FNLINK%(U%(PTR%,6%))          !POINT TO ACC BLKETTE    &
        \PRINT ' '; DATE$(U%(AB%,1%));      !DATE OF LAST ACCESS     &
        !
```

Once again we use the FNLINK% function. The NB's link to the AB is converted to a blockette number. Word 1 of the Accounting blockette, according to Figure 6, is the date of last access for this directory entry. The variable AB% may be further imposed upon to discover other information in the AB.

Version 7.0 of RSTS has a large files option. If this option is selected at system generation time, files greater than 65535 blocks are allowed. For such a file, Figure 6 needs some modification. I have added Figure 9, the format of the Accounting blockette for a large file entry. Note that word 5, which normally would store the first three characters of the run-time system name, is zero. It is this zero word that defines the entry as a large file. No RTS name may ever have a zero first word (blank names are not legal). Also, and very reasonably, no executable file may be larger than 65535 blocks (as they require RTS names).

On a large file system, some special attention must be paid to deriving the proper filesize. If word 5 of the AB is non-zero, word 2 will be this file's true size. Since the filesize is a 16-bit integer, we must convert negative values up to the range of 32768 through 65535. Alternately, word 5 may be zero. The low-order byte of word 6 will then contain the most-significant byte of the filesize. Multiply it by 65536 and add it to word 2. This will print the file size:

```
2060    SIZE = U%(AB%,2%)                   !FIRST TRY AT SIZE       &
        \SIZE = SIZE + 65536.               !CORRECT FOR NEGATIVE    &
              IF SIZE < 0.                  !  SIZE VALUES           &
                                            !CORRECT IF LARGE:       &
        \ SIZE = SIZE + 65536. * U%(AB%,6%)                          &
              UNLESS U%(AB%,5%)                                      &
        \PRINT USING '######', SIZE;                                 &
        !
```

If we wished to report the run-time system name as well, it is important to remember that it should be supressed if word 5 is zero.

While we are on the topic of large files systems, it is worth mentioning that the open file count bytes (back in word 5 of

PDP* 11 & 11/70 Disk Capacity Headaches

For Fast Relief . . . DMS Disk Sub-Systems Are The Best Prescription

COMPARE PRICE - PERFORMANCE

	RWP 06	DM 70 / 300
DRIVE CAPACITY (Mbytes)	174.4	253.7 (45% LARGER)
DATA RATE (K / words / sec)	337.9	491.5 (45% FASTER)
PRICE	$44,000.	$23,900. (45% LOWER)
	RJM 02	**DM 11 / 80**
DRIVE CAPACITY (Mbytes)	67	67 (EQUAL)
DATA RATE (K / words / sec)	337.9	491.5 (45% FASTER)
PRICE	$24,000.	$13,350. (45% LOWER)

COMPARE DELIVERY

DATALEASE MEMORY SYSTEMS can deliver and have your disk sub-system on-line in less time than it takes DEC* to process an order. (Typically within 30 days)

COMPATIBILITY

DATALEASE MEMORY SYSTEMS offers disk sub-systems that emulate the RM02, RM03, RP04, RP05, and RP06. The sub-systems are completely software transparent and best of all will run standard diagnostics.

NATIONWIDE SERVICE & SUPPORT

DATALEASE MEMORY SYSTEMS provides installation and maintenance nationwide through the Engineering Services Division of CDC. Our own highly experienced systems analysts supervise every installation and do not step out of the picture until you are completely satisfied.

Call toll free 800-854-0350

DATALEASE

DATALEASE MEMORY SYSTEMS
2770 East Regal Park Drive
Anaheim, California 92806
In California (714) 632-6986

*PDP and DEC are registered trademarks of Digital Equipment Corporation

the NB) are not maintained in a large file system. This information is kept in memory for this version of FIP and is available via PEEK( ) calls. The DIRECT program handles all these large files tangles and is worth reading.

3.7 Retrieval and Attribute Blockettes

To obtain information in either the Retrieval or Attribute blockettes, we must convert their links to virtual array indices:

```
RET% = FNLINK%(U%(PTR%,7%))        ! INDEX TO RB
ATT% = FNLINK%(U%(AB%,0%))         ! INDEX TO FIRST ATB
```

Note that the Attribute blockette link word is in the AB. Unlike the AB, which must exist for every entry, the RB and (very often) ATB links may be null. Test for zero, after the FNLINK call, before attempting to recover data.

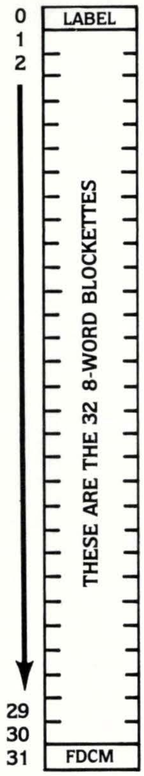


FIGURE 1. The First Block of a UFD

0	LINK TO FIRST NAME BLOCKETTE	
1	—1	
2	0	
3	0	
4	0	
5	0	
6	PROJECT #	PROGRAMMER #
7	—31692. = 'UFD' (RAD50)	

FIGURE 2. UFD Label Blockette

Word 0 of the Retrieval blockette contains a link word to the next RB (see Figure 7). This link is null at the end chain. To obtain list of all the clusters for all the clusters for a given file, step through the RBs in the same fashion as the main NB walk.

3.7 The Complete Example Program

A running version of the simple directory listing program appears in Figure 10. You can add anything you want to it, just to test your understanding. Since the UFD is automatically opened read-only, there is no possibility of harm to your system.

Coming next issue . . .
Inside the MFD

0	UFD CLUSTERSIZE
1	DCN #0
2	DCN #1
3	DCN #2
4	DCN #3
5	DCN #4
6	DCN #5
7	DCN #6

FIGURE 3. FDCM Blockette

0	LINK TO NEXT NAME BLOCKETTE	
1	FILE NAME	
2	(RAD50)	
3	EXTENSION (RAD50)	
4	PROTECTION CODE	STATUS
5	R/O REGARDLESS OPEN COUNT	OPEN FILE COUNT
6	LINK TO ACCOUNTING BLOCKETTE	
7	LINK TO FIRST RETRIEVAL BLOCKETTE	

FIGURE 4. Name Blockette

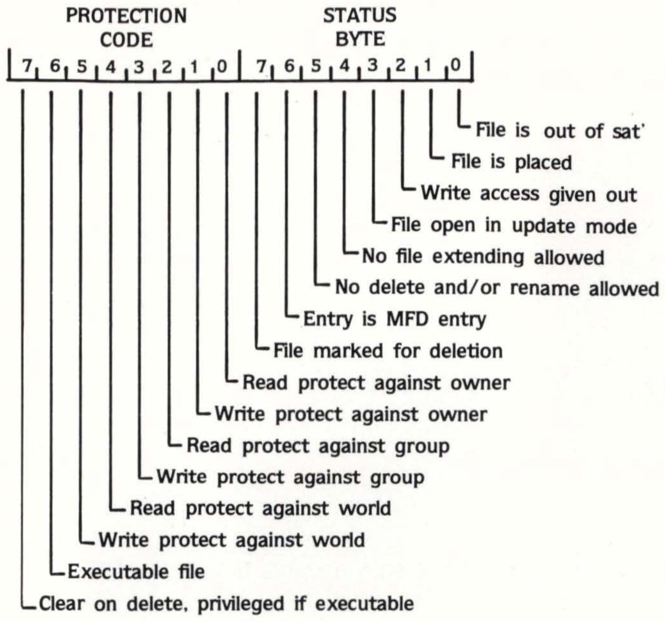


FIGURE 5. NB Status (Word 4)

0	LINK TO ATTRIBUTE BLOCKETTE
1	DATE OF LAST ACCESS
2	FILE SIZE IN BLOCKS
3	DATE OF CREATION
4	TIME OF CREATION
5	RUN-TIME SYSTEM NAME
6	(RAD50)
7	FILE CLUSTERSIZE

FIGURE 6. Accounting Blockette (not for large-files system)

0	LINK TO NEXT RETRIEVAL BLOCKETTE
1	DCN #0
2	DCN #1
3	DCN #2
4	DCN #3
5	DCN #4
6	DCN #5
7	DCN #6

FIGURE 7. Retrieval Blockette

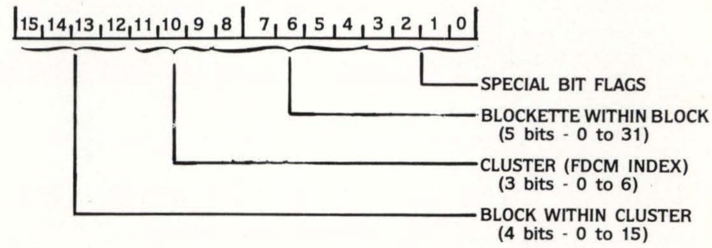


FIGURE 8. Format of Link Word

0	LINK TO ATTRIBUTE BLOCKETTE	
1	DATE OF LAST ACCESS	
2	LSB FILE SIZE IN BLOCKS	
3	DATE OF CREATION	
4	TIME OF CREATION	
5	0	
6	0	MSB FILESIZE
7	FILE CLUSTERSIZE	

FIGURE 9. Accounting Blockette (for large file)

```
1         EXTEND
100       OPEN '[1,2]' FOR INPUT AS FILE 1%
110       DIM #1, U%(3583,7)
120       CLU% = U%(31%,0%)
200       GOSUB 1000                          ! LIST DIRECTORY        &
        !
290       GOTO 32767                                                  &
        !
1000    ! WALK THROUGH DIRECTORY                                      &
        !                                                             &
          PTR% = FNLINK%(U%(0%,0%))           ! SET FIRST LINK        &
        !
1020      GOTO 1190 UNLESS PTR%               ! NULL - >EXIT          &
        \ GOSUB 2000                          ! PROCESS THIS ENTRY    &
        \ PTR% = FNLINK%(U%(PTR%,0%))         ! POINT TO NEXT NB      &
        \ GOTO 1020                           ! LOOP FOR NEXT ENTRY   &
        !
1190      RETURN                                                      &
        !
2000    !                                                             &
        !       PRINT THIS DIRECTORY ENTRY                            &
        !                                                             &
          PRINT RAD$(U%(PTR%,1%));            ! PRINT THE FILENAME    &
                RAD$(U%(PTR%,2%)); '.';                               &
                RAD$(U%(PTR%,3%));            ! AND EXTENSION         &
        !
2020    PRINT USING ' <###>',                 ! THEN PRINT PROTECTION &
                SWAP%(U%(PTR%,4%)) AND 255%;  !     CODE ON SAME LINE &
        !
2040      AB% = FNLINK%(U%(PTR%,6%))          ! POINT TO ACC BLKETTE  &
        \ PRINT ' '; DATE$(U%(AB%,1%)); ! DATE OF LAST ACCESS         &
        !                                                             &

2060      SIZE = U%(AB%,2%)                   ! FIRST TRY AT SIZE     &
        \ SIZE = SIZE + 65536.                ! CORRECT FOR NEGATIVE  &
                        IF SIZE < 0.          !       SIZE VALUES     &
                                              ! CORRECT IF LARGE:     &
        \ SIZE = SIZE + 65536. * U%(AB%,6%)                           &
                        UNLESS U%(AB%,5%)                             &
        \ PRINT USING ' ######', SIZE;                                &
        !                                                             &

2180    PRINT   ! FINISH THIS LINE                                    &
        !
2190    RETURN                                                        &
        !                                                             &

10000   !                                                             &
        !       CONVERT LINK WORD TO VIRTUAL ARRAY INDEX              &
        !                                                             &
          DEF FNLINK%(L%) =                                           &
          ((( L% AND 3584% ) / 512% ) * CLU%                          &
          + ( SWAP%( L% AND -4096% ) / 16% )) * 32%                   &
          + (( L% AND 496% ) / 16% )                                  &
        !
32767     END                                                         &
```

FIGURE 10.

THE ALTERNATIVE TO DBMS

By H. L. Singband, Director, MIS Merchandata

Just the other day I was sitting in my office leaning back in my chair, feet up, and feeling pretty good. For the past three or four weeks all our programs had been working as planned, closings had taken place on time and without error, our new systems development was giving every indiction of going well, and best of all, neither the comptroller nor the marketing manager had asked for any new 'reports'. Yup — everything was finally beginning to shape up and I started thinking about the big raise that I was definitely due to receive. Of course, in retrospect, this was the perfect time for Murphy's law to take effect and naturally it did. The phone rang and it was our marketing manager, Sell-em Stan, and he had a great idea but he needed a report that was 'vital' — "Would I mind very much seeing him right after lunch?"

About one-thirty I walked down to Sell-em's office and immediately put my guard up. I knew that he had something up his sleeve because he offered me one of his rare Cuban cigars and ushered me over to his real leather chair (usually reserved for only the biggest customers). Then he hit me with it — "I need to know how many pairs of knickers we've sold to golfers over 65 in the last six months." Even though he was discreetly waving this years bonus schedule as he asked, I was still able to answer: "Gee — Stan I'd love to help you out on this one but there's a modulator stuck in the Unibus and all of the programmers are busy patching the backup package." Before he could figure out what that meant, I beat it back to my office.

The point of this little bit of fictionalized truth is that it happens to most of us in the data processing community all too often and for the most part we've done very little to ameliorate the problem, which simply stated is not being able to deliver a large part of the data we are trustees of in a form usable by management.

The reasons that we are not generally in a position to do this are many. Perhaps the majority of these reasons are beyond our control but on the other hand very many (the more significant ones?) are well within our supposed area of expertise and amenable to our advice if not under our total control. My aim in this article is not to go over all of the reasons which have created this situation but rather to focus on the one which I feel is most important relative to the objective of producing the information that management needs within the time span during which it still has value. What management needs is accessible data stored in a flexible form.

Well, you say, that's easy enough, buy and install a Data Base Management System. Well, I say, maybe not (see RSTS Professional, Vol. 2, No. 1, p. 5). Don't be shocked, this may sound like apostasy but it really isn't. I won't deny the generally accepted benefits of a DBMS which are well known —

1. reduced data redundancy
2. improved accuracy
3. easier file and program maintenance
4. better security through centralized control
5. data independence
6. greater applications programmer productivity
7. programming standards

In addition to these, many DBMS packages offer other features such as query languages, restart and recovery routines, a data dictionary, and a report generator. And don't forget the real grabber — "Wow! Will this stuff look great on my resume."

It sure would, but what you should remember is that you don't get something for nothing and is what you're planning on getting worth what you (read your company) are going to pay for it both up front and for implementation? The answer is maybe. If you have the management understanding, commitment and support, if you have the systems programming talent available to thoroughly understand and tune the package, if you have the hardware resources to cover the DBMS overhead, if you have someone who can function as an effective data base administrator, if you have an operational users committee, if you have management who can (or even want to) use such a sophisticated tool, and if your company has the money and patience to see the implementation through then you should give some very serious thought to going along with a DBMS package.

These are not specious arguments against the concept of a DBMS, but I have no doubt that these are the minimum requirements for the successful implementation of any DBMS. You should always keep in mind that a DBMS is not merely some software used to access data or a set of formal rules or a data organization scheme or a collection of integrated data. The implication of DBMS goes far beyond the elemental software technology from which it is constructed and into the realm of how it is perceived by management. It is here, in how your boss perceives the DBMS concept and understands it and most importantly, what does he expect from it in terms of enhanced performance and therefore profits that your potential for

success or failure lies. Bear in mind that your performance will be judged, in the ungracious and austere light of results, not in the soft, romantic rays of blue sky promises. Most, if not all managers, have been exposed to data processing for some time and are familiar enough with its jargon so as to have at least a rudimentary feeling for its potential as well as for its limitations. Therefore, if you broach the DBMS concept to them they will likely grasp its implications quite readily and will consequently have their expectations raised to a point beyond what you intuitively know to be attainable. However, even at this early stage you have lost the war unless your organization has available to it all of those indispensable components which were noted above along with many that I, and probably you too, have not even considered.

So, where does this leave you — stuck with one of DEC's outdated offerings (IAM) or maybe with their overblown, overcomplicated, oversold and ungainly RMS-11. This latter offering may have come to you at no charge along with Version 7 of RSTS but I would ask two questions: Why are they passing it along at no extra charge (they sure do charge plenty for the good stuff don't they?), and second, What will it cost to implement both in labor and additional hardware? (I always try to remember that all vendors make their money on hardware and provide the software so that customers will buy their hardware.) No, you're not restricted to these baroque products, there is another generic group of data management software available which I'll call File Management Systems (FMS).

The FMS approach to the solution of the problem we're trying to relieve i.e., delivering the right data to the right person in time for him to get some value from it, may not have had the elaborate press coverage that DBMS has enjoyed but it's a lot more likely to provide some practical and realizable help at a much more reasonable cost. What most FMS's don't have is a DDL (data definition language), a schema, elaborate recovery sub-systems, or sophisticated inquiry capabilities. What the better File Management Systems do have is one or more file organization options (ISAM, sequential, relative, and/or hashed), standard reading and writing procedures, relative economy in terms of hardware and system overhead, and most of all the're far easier to implement than is a DBMS and in many cases may even be integrated into existing systems and programs with minimal changes to current code.

At this point it would be a good idea to note the fact that computer people are not in the same business that we were in as little as five years ago. In the past we were computer people or data processors, but today we are (or should be if we are not yet) in the information management business. The difference between these callings may seem superficial but is in fact significant. The implication of data processing is one of bits and bytes with overtones of high technology while information management suggests a results oriented role where the computer is just one of the instruments necessary to achieving a desired goal.

Now that we've clarified what business we're in, we can continue and unequivocally state that the best way for the average computer department to solve its data delivery problem and to get into the information management buisness is to use the software tools that will allow it to do so. As I've said above, the DBMS option is certainly an alternative but may be impractical because of its cost and complexity whereas the FMS option is more likely to be the appropriate choice for the medium and smaller organization and in many instances for the larger organization as well.

What I'd like to do now is describe a file management system that I have used for several years and I think that this will illustrate the fact that a data management system need not be overcomplicated to be effective.

This particular package consists of a set of subroutines used for data manipulation and a set of utilities which provide the necessary support capabilities such as file creation and sorting. The system is written in Basic-Plus and runs under RSTS/E Version 6B or later and, if all modules are used, will require 4KB. Although the system is simple in design, it can readily be used in applications that require considerable sophistication in file handling.

My operational experience with this package has been positive because it is easy to learn, easy to use, and lends itself to both simple and complex applications. Programmers who have been introduced to this system have been able to make virtually full use of it in about two days of reading and experimenting with it. I have used it in many applications from the relatively straightforward (accounts receivable) to the intricate (inventory modeling).

Files can be designed so that one or more (up to 250) different types of record can reside within the same file. This facility allows you to segment the data so that related data within the generic file can be conveniently grouped. As an example, the data often ascribed a place in a customer master file can be divided into familial bunches with each bunch containing fields more closely intra-related than they are to the fields of the other bunches. viz. name and address, credit history, sales history, and billing data. (See Fig. 1.) By allowing

```
N/A              Cr hist          Sls hist           Billing data

Cust No.         Cust No.         Cust No.           Cust No.
Name             A/R balance      Sls this mth       Ship via
Street           D+B rating       Sls 1 mth ago      Terms
City             Crnt paymts      Sls 2 mth ago      Discount %
State            Ovr 30 ''        Sls YTD            Salesman
Zip              Ovr 60 ''        Sls last yr        Commission %
-----------------------------------------------------------------------
added later
Phone            Credit limit     Returns YTD
```

FIGURE 1.

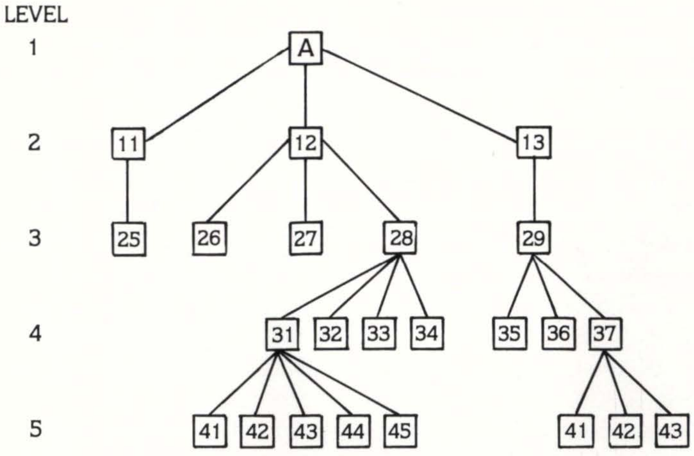


FIGURE 2.

you to split a file into multiple record types your systems and programming effort is given a greater degree of flexibility in terms of file enhancements, system changes, and data accessibility. By way of illustration, in Figure 1, as additional data are required you need only revise the structure of the file and add the necessary fields. If you prefer or if circumstances recommend, you may just as easily define another record type to achieve similar results.

After files have been designed and built, access to them can be accomplished through several methods—hashing, multi-key ISAM, sequential (by key index), and relative record. Any given file can be accessed using one or more of these methods simultaneously. For multi-key ISAM and sequential processing, the supporting indices and key files can be defined and created at any time using the supplied utilities either stand alone or chained to (and back again) from an application program.

Both file organization and file accessibility are based on the concept of hashing which is what makes this package somewhat unique and also what makes it simple to use. (See Scientific American, April 1977.) Records are loaded into the file based on a relative address generated from the key by the hashing algorithm and are retrieved in a like manner. This provides you with a speed of random access exceeded only by relative record processing so that you can expect to retrieve any record in a file (using hashing) in 1.2 or fewer average seeks. The advantage here over ISAM to on-line processing should be obvious.

This system supports a variety of data storage techniques (ASCII, binary, floating point, Radix-50, etc.) allowing the programmer to minimize the amount of storage needed and to mix storage types within the record. In addition, a generalized record sort is provided as part of the set of utilities.

An example of one of the more involved file structures to which this system has been applied is shown in Figure 2. Note the presence of duplicate keys at level five and potentially at any of the other levels (except the highest).

This is a hierarchical structure and you would naturally expect to access the records using links or pointers. However, this FMS allowed us to make use of its hashing and multi-key ISAM capabilities simultaneously thus avoiding the need for embedded links and the problems and overhead in maintaining them and recovering from their corruption. Access to records in a level without duplicate keys is accomplished by hashing directly to that record without having to go through the ISAM index thus noticeably improving response time. In fact, most of the organizations to which this structure is applicable only require duplicate key handling at relatively few levels so that access to the file is very often through the hashing algorithm thereby making for excellent response time. This was a particular design requirement since these files are most often used interactively. (In the interests of clarity, several intervening levels were omitted). The insertion and deletion of levels can be effected rather easily. Deletion is the easier and is accomplished by flagging each record in that level with the appropriate character. Insertion of levels is only slightly more difficult because it calls for a data entry operation.

Travel up, down, and across this structure is along the path of an ISAM index. This is necessary because, as noted above, there are no embedded links or pointers in the data file. Since the file management package permits the user to control the buffer size of the files, the index can be assigned a large buffer size therefore drastically reducing the number of physical disk accesses to the index and keeping the overall number of accesses down, thus maintaining good speed even for sequential passes of the data file.

I hope this all too brief example will at least cause you to give some consideration to the less complex file management systems that are available before committing yourself and your organization to a full DBMS which may in fact not be necessary.

Mr. Singband is Director MIS at Merchadata, Brockton, MA. Merchadata provides inventory modeling and simulation systems to the retail industry.

BIG DISKS

Carl B. Marbach

FLASH! DEC announces the RM05, a 300 MB disk subsystem for unibus and MassBus computers. Price to be about the same as the RP06 Disks.

Before . . . . .

There was no 300 MB disk available. The world of big CDC drives was limited to the "foreign" manufacturers; they were there and still are. Who are they and how do they do it? Or as was once remarked: "Who does what, with which, and to whom?"

Foreign unibus devices are fairly straightforward and have been available for a long time. Foreign printers, tapes, memory, multiplexers and others have been common for some time. Even DEC people seem used to them. Foreign disks are now common also. Many suppliers make unibus interfaces for large disks. Earlier models of some manufacturers were multi-board and/or required an external formatter. Currently, single board unibus controllers are widely available.

Foreign Cachebus controllers entered the field later than their unibus counterparts, but the demand for them outstrips the smaller machine requirements. It stands to reason that an 11/70 or VAX is more likely to invest the bucks in a large disk than an 11/03. Although there may be others the larger suppliers of this type of interface are Emulex, Diva, and Systems Industries. How these are marketed will be discussed later. Some Cachebus controllers require a unibus slot in addition to the 4 RH70 controller slots, some also require additional black boxes, and some are totally contained within the RH70 slots. "Better" is in the eyes of the beholder; an external black box (formatter) may give more flexibility (two computers, one black box, two RH70, giving total access to two data bases?), but certainly takes up more room, power and possibly expense. The trend is towards the 4 board RH70 controller.

The disks themselves are mostly CDC, Ampex, Trident (Xerox), or soon some of the Winchester variants. CDC seems to be the most abundant and DEC has chosen this drive to be the RM05. Look for more and more Winchester disks to be available for the foreign controllers while DEC will be about two years later if History teaches us anything.

O.K. You are still confused. I like to make lists which help me see what is going on, here is one suggested by Kent Winton at Systems Industries:

1. Computer Manufacturer
   a. DEC
2. Disk Drive Manufacturer
   a. CDC
   b. AMPEX
   c. Trident/Century/Xerox
   d. New Winchester variants
   e. Datum
3. Controller Manufacturer
   a. Emulex
   b. AED
   c. Dataram
   d. Xebec
4. System Manufacturers
   a. Systems Industries
   b. DIVA
   c. Plessey
   d. Others
5. Brokers
   a. Data Systems Services
   b. DataLease
   c. Advanced Digital Products
   d. Nordata
   e. Others

I called a sample of these people and asked them questions about their products and how they sell them.

Walt Mitchell of Data Systems Services markets Emulex controllers and mostly CDC drives. He stresses the fact that they will take full responsibility for, and have the expertise to, install the drive and the controller in your system. They are located on the West Coast and fly the installation people to you. They are selling at a rate of about $24,000,000 a year which is a bunch of disks; and they are growing. Maintenance is handled by CDC in the field at most of his sites. The Emulex Controller is a 4 board set that goes into the RH70 slots and uses no unibus slots or external boxes for the 11/70. For Unibus systems, it is a single board controller. Media compatibility, which they provide, is a big plus from DSS; their RP06 subsystems can be read on a DEC RP06 and vice versa. Walt and his company have a big commitment to this market.

Matt Goldbach of DATALEASE sells Emulex Controllers and mostly CDC drives. The SC-70 cachebus controller utilizes the 4 RH70 slots for the 11/70. The SC-11 Unibus controller is a two board unit that utilizes one unibus slot but uses the space for two; the newer SC-21 is a true single board unibus controller. The SC-01 supports the Q-bus on the LSI-11's. So far this year they have sold about 100 drives

Vince Meturo of Advanced Digital Products sells Emulex Controllers with mostly CDC drives. They have their own staff of field engineers in San Diego, Los Angeles and Seattle who install and service their drives and controllers. In other parts of the country they utilize CDC for service and fly the installers to your sites. They also market non disk add on's from ABLE, MOSTEK, CIPHER and others. When I heard about the 15 engineers on their staff I asked what they did. They are designing a Pascal Microengine for the unibus, called the PDQ/3. You'll hear more about this later.

Kent Winton of Systems Industries (SI) was kind enough to give me the outline shown above. The original SI controllers for the 11/70 utilized the 4 board RH70 slots, one Unibus slot, and an external formatter. Currently, the Unibus board is no longer required. They will have a 4 board RH70 only interface available soon. The External Formatter will not go away for those who want it. With it, the user can create a totally redundant two computer, two disk, two formatter system where each computer can read the other's disk, and in the event of a failure the data base can be maintained. This is the flexibility we talked of earlier. The Unibus controller uses one unibus slot and the external formatter. Service and installation are handled by their nationwide service people or by CDC. Others such as GE, Grumman, ICE, Braegen and even DEC are maintaining their systems in selected places. They currently have about 12,000 installations and are spending about $1,700,000 annually on R&D. They will have a Q-bus controller in the future.

All of these companies have successful installations, some may have had unsuccessful ones. The larger your customer base the more likely you are to have at least one who doesn't like you. Your decision must be based on your specific requirements of price, performance, installation/support, and service. There are lots of big disks out there (even 600 MB). One could be yours!

REMOTE ANNUNCIATOR FOR OPERATOR MESSAGES

By Norm Seethoff

Overview

Many DEC computer sites cannot afford the luxury of a full time operator to monitor messages printed on the console keyboard. Hence, important console messages (such as the action requests of the RSTS Operator Services Package) often go unnoticed for extended periods of time. A simple piece of hardware is described whose visible and audible alarms are triggered by an Ascii bell character sent to the console terminal as part of an operator message. The device is low in cost and easy to assemble from readily available components. It is easily attached to an LA36, and can be attached to most other terminals after a quick inspection of the terminals bell circuitry.

Purpose of the Device

The device described was first implemented as a means of reminding an operator of any pending messages or requests which had been printed on the operator's console. These messages were frequently being overlooked if the operator was not near the console when they were printed out. As a result, batch jobs were waiting excessively long for tape mounts, and forms changes requests waited for hours. The circuit described below was assembled to monitor the bell control logic on the operator's console and provide both a visual and audible signal when a bell character was sent to the console. These signals would then serve as a reminder to the operator to check the console for a pending request or message.

Use of the Device

Externally, the device consists of two switches, three leds and an audible alarm. A toggle switch is used to enable or disable the alarm circuitry. A green led is lighted to indicate the circuit is enabled. A yellow led flashes once a second to indicate the circuit is disabled. If the monitor circuit has been triggered by a bell sent to the console terminal, a red led flashes once a second and an audible alarm sounds for one second every minute. The flashing red led and the audible alarm remind the operator to check the console for a message requiring intervention. A pushbutton switch is then used to reset the alarm. The device is connected to the terminal with miniature clip leads and a cable of the desired length. The alarm unit can even reside outside the computer room if the cable length is not excessive.

Circuit Description

The circuit used to implement this remote annunciator consists of three extremely simple sections: a monitor and display section, a timing signal section, and a tone generator section. The total parts count is very low, with only seven integrated circuits, three leds, three switches, a speaker or audio transducers, and some assorted resistors and capacitors required. A schematic drawing of the circuit is shown in Figure 1.

The monitor and display section is controlled by the Enable/Disable switch S1. When S1 is open, the input to the latch is held high and the output of the inverter is low, causing the green led to light indicating the circuit is enabled. Closing S1 extinguishes the green led and gates the 1 second blink signal to the yellow led which flashes indicating the circuit has been disabled. This section receives inverted control signals from the terminal, which are buffered and shaped by the Schmitt trigger inverter U1. This signal is then used to clock the state of the Enable/Disable switch into the latch U3. This latched state Enable/Disable state is then ANDed with the 1 second blink signal to drive the visible alarm. It also can be used to drive the control input of the audible alarm directly. A latched alarm condition is cleared by momentarily depressing one of the reset switches. Two switches are shown on the schematic to allow one to be located next to the console in cases where the annunciator is located in another room.

The timing signal section consists of two 555 universal timers used to generate a free-running 1 hertz signal and a controllable 1/60 hertz signal. The 1 hertz signal is used to flash the disable and alarm leds. This signal is generated by U5 and its associated components. Its frequency may be adjusted by varying the values of the RC components on the left side of U5. The triggerable 1/60 hertz signal is used to control the audible alarm. It is controlled by the latched enable signal. When this signal is high, it will cause a one second pulse to be applied to the tone generator once a minute. The frequency and duty of this signal can also be adjusted by varying the RC components shown on the left side of U6.

The tone generator section consists of another 555 used as an oscillator under control of the 1/60 hertz signal. The output of this 555 is used to drive a Panasonic miniature audio transducer (a small speaker could be substituted if desired). The output of the transducer is approximately a 2500 hertz tone, whose frequency may be adjusted by varying the RC components shown on the left side of U7.

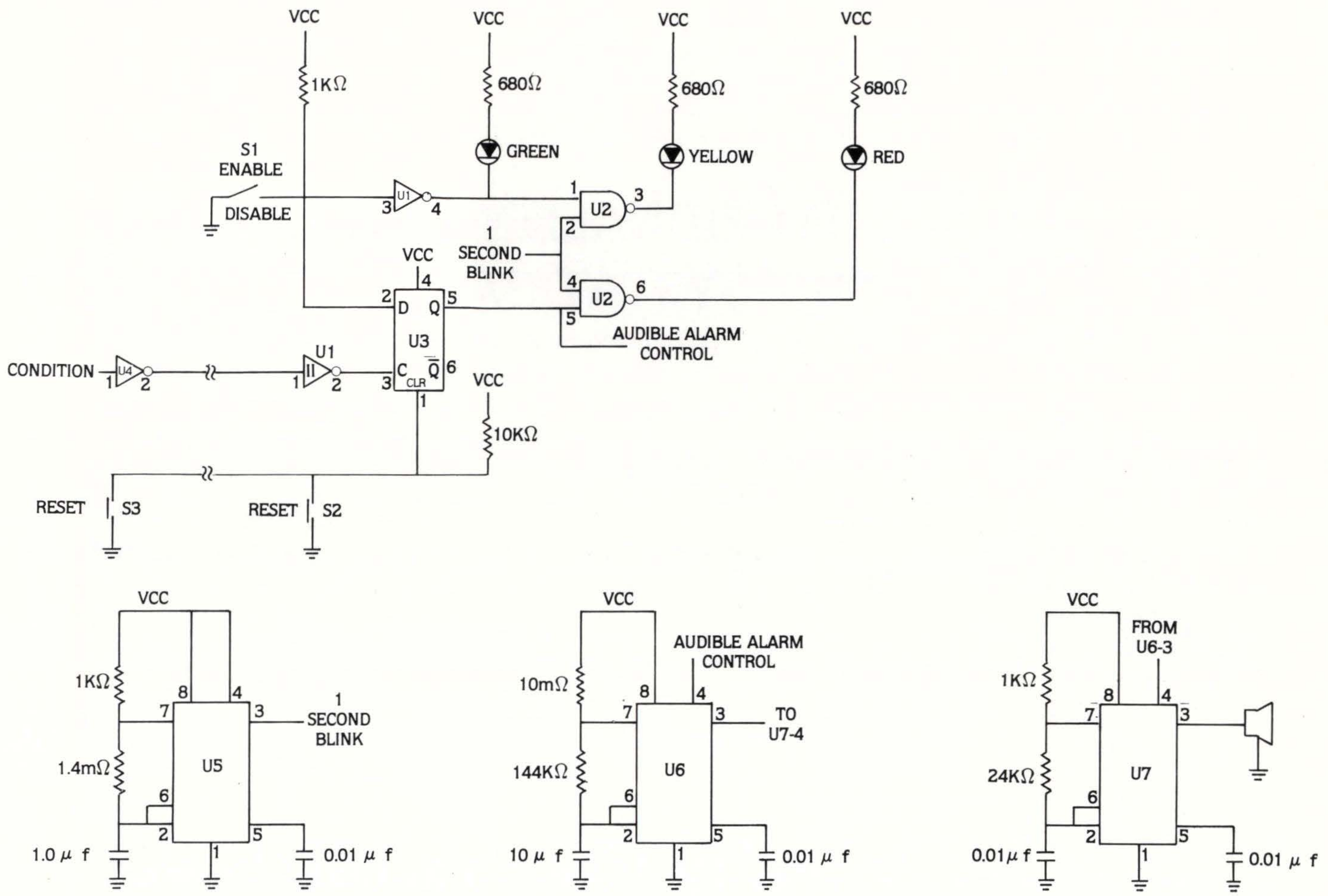


FIGURE 1.

Connection to an LA36

Connection to an LA36 is facilitated by the use of a miniature clip lead and snap-on connectors for the VCC and ground connections. The connection point for the control input may be found on page A-12 of the LA36 Maintenance Manual. The latched bell signal is available on E36 pin 5 of the main logic PCB attached to the printer's rear cover. VCC and ground connections may be made at the connector lugs provided near the lower edge of the board. The inverter to be used as a driver may be mounted on insulative material and placed within the LA36. Alternately, there is an unused inverter available on the LA36. This inverter is formed by pins 11 and 10 of a 7404, generally found at location E8. Pin 11 of E8 can be connected directly to pin 5 of E36, and pin 10 of E8 used to drive the output cable directly.

Extension to a Multi-Line Circuit

The circuit is easily expanded to accommodate multiple inputs while using the existing timing circuitry and tone generator. Rather than driving the 1/60 hertz control for the tone generator directly, the outputs of multiple latches may be combined prior to driving the 1/60 hertz control input. This is readily accomplished by breaking the connection from U3 pin 5 to U6 pin 4,, and connecting the complemented output of each latch to the input of a NAND gate, and then using the output of this gate to drive the 1/60 hertz control input. Using commonly available ICs, it is possible to assemble a six channel unit using only 12 active components (not including those required for the signal buffers in the terminal itself).

Components Required

Ref.	Qty.	Item Description
U1	1	74LS14 hex schmitt trigger inverter
U2	1	74LS00 quad 2 input NAND gate
U3	1	74LS74 dual D latch
U4	1	7404 hex inverter
U5	1	555 universal timer
U6	1	555 universal timer
U7	1	555 universal timer
	3	680 ohm resistor
	3	1 K ohm resistor
	1	10 K ohm resistor
	1	24 K ohm resistor
	1	144 K ohm resistor
	1	1.4 M ohm resistor
	1	10 M ohm resistor
	4	0.01 uf capacitor
	1	1.0 uf capacitor
	1	10.0 uf capacitor
S1		SPST toggle switch
S2		Pushbutton switch
S3		Pushbutton switch
	1	Green LED
	1	Yellow LED
	1	Red LED
	1	Panasonic 16R02C audio transducer

THREE FUNCTIONALITY PATCHES FOR SYSTAT.BAS

By G. Johnson, IIRI International

Each of these patches applies to Version 7.0 of SYSTAT.BAS.

Patch No. 1 — Display open channel information for each job (privileged users). Using the switch /G results in a line printed for each channel currently open for the job. If a disk file is opened on the channel, then the file specification is displayed, along with the currently accessed block number and the total file size. If the device is not disk, then only the device identification is shown. This patch is applicable only to large file systems.

```
Example:

User types 'SY/17/G', system prints:

17 +    4,8    KB32    ARINQO   10/28K   KB A12    34:37.8   -8/6   BASIC
        Chan 0   KB:
        Chan 1   DB1:[  4,107]CUSMAS.    <58>   8094 of 25920   Rd,RR
        Chan 2   DB1:[  4,107]INVIDX.    <48>    488 of 878     Rd,RR
        Chan 3   KB:
        Chan 12  NL:
```

Patch No. 2 — Specific file names for /O and /W switches. This patch extends the usefulness of the /O and /W switches (open files, open files and jobs accessing them) by allowing filenames to be specified, as well as devices (e.g., /W:DB1:[1,2]TEST.FIL causes data to be displayed only for the file DB1:[1,2]TEST.FIL). In addition, the /W switch will also display the current block being accessed by each job which has a file open, as well as the total file size.

```
Example:

User types 'SY/W:DB1:[4,107]CUSMAS', system responds with

Open Files (DB1:(4,107)CUSMAS only) and Jobs accessing them:

DB1: -- File                    Op/RR    Size     Clu   Status
DB1:[  4,107]CUSMAS.    <58>    3/7     25920     64    Upd
                  26    Rd, Wr, Up          1
                  39    Rd, RR          25839
                  37    Rd, RR          18375
                  36    Rd, RR          22370
                  29    Rd, RR          14078
                  17    Rd, RR           8094
                  12    Rd, Wr, Up       8754
                  10    Rd, Wr, Up       9504
                  9     Rd, RR           4289
                  24    Rd, RR          25055
```

Patch No. 3 — Repetitive operation. Appending the switch /I:s to the output specification results in regeneration of the output every s seconds.

Patch Code — These listings are designed to be used with $CPATCH. In each case, the checksum is given at the end.

Patch no. 1 -

```
*H/1080<tab>/V<cr>
1080<tab>J%=INSTR(1%,S$+"/","/") &<cr>
*0AI<cr>
1070<tab>I%=FNT1% &<cr>
<tab>\ I%=INSTR(1%,S$,"/G") &<cr>
<tab>\ IF I%=0% THEN CHAN.INFO%=0% ELSE CHAN.INFO%=-1% &<cr>
<tab>\ GOTO 1160 UNLESS P% AND M2%(29%) &<cr>
<tab>\ S$=LEFT(S$,I%-1%)+RIGHT(S$,I%+2%) &<cr>
<tab>\ DISK.NAMES$="" &<cr>
<tab>\ DISK.NAMES$=DISK.NAMES$+CVT%$(SWAP%(PEEK(I%+M2%(5%))))+CHR$(48%+J%) &<cr>
<tab><tab>FOR J%=0% TO PEEK(I%+M%(5%)) &<cr>
<tab><tab><tab>FOR I%=0% TO M2%(9%)-2% STEP 2%<cr>
<esc>*H/10130<tab>/V<cr>
10130<tab>PRINT #0%,S$; &<cr>
*14AV<cr>
<tab>\ GOTO 10020 &<cr>
*I<cr>
<tab>\ GOSUB 14100 IF CHAN.INFO% &<cr>
<esc>*H/10635<tab>/6AV<cr>
<tab>\<tab><tab><tab><tab>S$="" &<cr>
*0AI<cr>
<tab>\<tab><tab><tab><tab>GOSUB 10637 \ GOTO 10640<cr>
<esc>*I/10637/JDV<cr>
10637<tab><tab><tab><tab><tab>S$="" &<cr>
*H/10640<tab>/0AI<cr>
10639<tab><tab><tab><tab><tab>RETURN<cr>
<esc>*H/15000<tab>/V<cr>
15000<tab>! &<cr>
*0AI<cr>
14100<tab>! &<cr>
<tab>&<cr>
<tab>&<cr>
<tab>!<tab>O P E N   C H A N N E L   I N F O R M A T I O N &<cr>
<tab>&<cr>
<tab>&<cr>
<cr>
14110<tab>IOB%=PEEK(JDB%) &<cr>
<tab>\ FOR CHN%=0% TO 30% STEP 2% &<cr>
<tab>\<tab>GOTO 14140 IF J2%<0% UNLESS CHN% &<cr>
<tab>\<tab>WCB%=PEEK(IOB%+CHN%) &<cr>
<tab>\<tab>GOTO 14140 UNLESS WCB%>0% &<cr>
<tab>\<tab>ST%=PEEK(WCB%) &<cr>
<tab>\<tab>DRV.IDX%=ST% AND 255% &<cr>
<tab>\<tab>IF DRV.IDX% THEN &<cr>
<tab><tab><tab>F$=CVT%$(SWAP%(PEEK(M2%(5%)+M2%(9%)+DRV.IDX%-2%))) &<cr>
<tab>\<tab><tab>U%=SWAP%(PEEK(WCB%+2%)) AND 255% &<cr>
<tab>\<tab><tab>F$=F$+NUM1$(U%) UNLESS F$="NL" OR (F$="KB" AND U%=J2%) &<cr>
<tab>\<tab><tab>F$=F$+":" &<cr>
<tab>\<tab><tab>GOTO 14130 &<cr>
<tab><tab>! NON-DISK DEVICE<cr>
14120<tab><tab>CUR.BLOCK=65536.*(SWAP%(PEEK(WCB%+4%)) AND 255%) &<cr>
<tab><tab><tab>+32768.+(PEEK(WCB%+6%) EQV 32767%) &<cr>
<tab>\<tab>FCB%=PEEK(WCB%+8%)-28% &<cr>
<tab>\<tab>F$=MID(DISK.NAMES$,3%*(PEEK(FCB%+24%) AND 255%)+1%,3%)+":" &<cr>
<tab><tab><tab>+"["+FNN$(3%,SWAP%(PEEK(FCB%+4%)) AND 255%)+"," &<cr>
<tab>\<tab>S$=SPACE$(3%) \ LSET S$=NUM1$(PEEK(FCB%+4%) AND 255%) &<cr>
<tab>\<tab>F$=F$+S$+"]" &<cr>
<tab>\<tab>F$=F$+RAD$(PEEK(FCB%+6%))+RAD$(PEEK(FCB%+8%)) &<cr>
<tab><tab><tab>+"."+RAD$(PEEK(FCB%+10%)) &<cr>
<tab><tab><tab>+"<"+NUM1$(SWAP%(PEEK(FCB%+12%)) AND 255%)+">" &<cr>
<tab><tab><tab><tab>UNLESS (ST% AND 16384%) OR (ST% AND 256%) &<cr>
<tab>\<tab>FILE.SIZE=65536.*(SWAP%(PEEK(FCB%+24%)) AND 255%) &<cr>
<tab><tab><tab>+32768.+(PEEK(FCB%+26%) EQV 32767%) &<cr>
<tab>\<tab>W.WCB%=PEEK(WCB%+12%) &<cr>
```

```
<tab>\<tab>GOSUB 10637 \ W$=CVT$$(LEFT(S$,LEN(S$)-2%),2%) &<cr>
<tab><tab>! DISK DEVICE<cr>
14130<tab><tab>PRINT #0%,TAB(7%);"Chan";CHN%/2%;TAB(16%);F$; &<cr>
<tab>\<tab>PRINT #0%,TAB(44%);FNN$(6%,CUR.BLOCK);" of "; &<cr>
<tab><tab> NUM1$(FILE.SIZE);TAB(61%);W$; &<cr>
<tab><tab><tab>UNLESS DRV.IDX% &<cr>
<tab>\<tab>PRINT #0%<cr>
14140<tab>NEXT CHN% &<cr>
<tab>\ RETURN &<cr>
<ff>
<esc>*EX<cr>
```

Checksum: 28778.

Patch no. 2 -

```
*H/1060<tab>/8AV<cr>
<tab>\<tab>O.DEV$=LEFT(O.DEV$,I%-1%) IF I% &<cr>
*KI<cr>
<tab>\<tab>O.DEV$=O.DEV$+":" UNLESS I% &<cr>
<tab>\<tab>FILE.MATCH$=RIGHT(O.DEV$,INSTR(1%,O.DEV$,":")+1%) &<cr>
<esc>*V<cr>
<tab>\<tab>CHANGE SYS(CHR$(6%)+CHR$(-10%)+O.DEV$+":") TO M% &<cr>
*G/.DEV$/4DV<cr>
<tab>\<tab>CHANGE SYS(CHR$(6%)+CHR$(-10%)+O.DEV$) TO M% &<cr>
*3AI<cr>
<tab>\<tab>GOTO 1160 UNLESS M%(23%) &<cr>
<esc>*AI<cr>
<tab>\<tab>GOTO 1160 IF O.DEV$="SY" &<cr>
<esc>*AI<cr>
<tab>\<tab>PPN.MATCH%=M%(5%) +SWAP%(M%(6%))  &<cr>
<tab>\<tab>NM1.MATCH%=M%(7%) +SWAP%(M%(8%))  &<cr>
<tab>\<tab>NM2.MATCH%=M%(9%) +SWAP%(M%(10%)) &<cr>
<tab>\<tab>EXT.MATCH%=M%(11%)+SWAP%(M%(12%)) &<cr>
<esc>*V<cr>
<tab>! CLEAR THE 'MULTIPLE ERRORS' FLAG. &<cr>
*H/10610<tab>/AV<cr>
<tab>\ PRINT #0%," (";O.DEV$;" only)"; IF LEN(O.DEV$) &<cr>
*G/.DEV$;/I/":";FILE.MATCH$;/V<cr>
<tab>\ PRINT #0%," (";O.DEV$;":";FILE.MATCH$;" only)"; IF LEN(O.DEV$) &<cr>
*H/10630<tab>/AV<cr>
<tab>\<tab><tab><tab>PRINT #0%,SO$;NUM1$(U%);":"; &<cr>
*I<cr>
<tab>\<tab><tab><tab>GOTO 10650 IF PEEK(FCB%+4%) -PPN.MATCH% &<cr>
<tab><tab><tab><tab><tab>   OR PEEK(FCB%+6%) -NM1.MATCH% &<cr>
<tab><tab><tab><tab><tab>   OR PEEK(FCB%+8%) -NM2.MATCH% &<cr>
<tab><tab><tab><tab><tab>   OR PEEK(FCB%+10%)-EXT.MATCH% &<cr>
<tab><tab><tab><tab><tab>UNLESS FILE.MATCH$="" &<cr>
<esc>*V<cr>
<tab>\<tab><tab><tab>PRINT #0%,SO$;NUM1$(U%);":"; &<cr>
*H/10640<tab>/AV<cr>
<tab><tab><tab><tab><tab> TAB(19%);LEFT(S$,LEN(S$)-2%) &<cr>
*G/-2%)/I/;/V<cr>
<tab><tab><tab><tab><tab> TAB(19%);LEFT(S$,LEN(S$)-2%); &<cr>
*AI<cr>
<tab>\<tab><tab><tab><tab>S$=NUM1$((SWAP%(PEEK(WCB%+4%)) &<cr>
<tab><tab><tab><tab><tab><tab>AND 255%)*65536. &<cr>
<tab><tab><tab><tab><tab> +32768.+(PEEK(WCB%+6%) EQV 32767%)) &<cr>
<tab>\<tab><tab><tab><tab>PRINT #0% TAB(36%);FNS$(5%) &<cr>
<esc>*V<cr>
<tab>\<tab><tab><tab><tab>WCB%=W.WCB% AND (NOT 31%) &<cr>
*EX<cr>
```

Checksum: 43723.

Patch no. 3 -

```
*H/1060<tab>/V<cr>
1060<tab>E9%=0% &<cr>
*AI<cr>
<tab>\ S.SAVE$=S$ &<cr>
<esc>*H/1080<tab>/V<cr>
1080<tab>J%=INSTR(1%,S$+"/","/") &<cr>
*0AI<cr>
1072<tab>I%=INSTR(1%,S$,"/I:") &<cr>
<tab>\ IF I%=0% THEN AWHILE%=0% ELSE &<cr>
<tab><tab>J%=INSTR(I%+3%,S$+"/","/") &<cr>
<tab>\<tab>AWHILE%=VAL(MID(S$,I%+3%,J%-I%-3%)) &<cr>
<tab>\<tab>S$=LEFT(S$,I%-1%)+RIGHT(S$,J%) &<cr>
<tab>! SLEEP DELAY FOR REPETITION<cr>
<esc>*H/1300<tab>/V<cr>
1300<tab>IF 0%<>0% THEN &<cr>
*0AI<cr>
1300<tab>IF AWHILE% THEN SLEEP AWHILE% &<cr>
<tab>\ S$=S.SAVE$ &<cr>
<tab>\ GOTO 1060 &<cr>
<tab>! WAIT, AND THEN REPEAT WHAT'S JUST BEEN DONE<cr>
<esc>*L<cr>
1300<tab>IF 0%<>0% THEN &<cr>
*3JC/2/V<cr>
1302<tab>IF 0%<>0% THEN &<cr>
*EX<cr>
```

Checksum: 57108.

FUNCTIONS PUT THE **FUN** BACK INTO PROGRAMMING

By Steve Holden, M-O-S Software Ltd.

IN the course of writing many thousands of lines of Basic Plus, and reviewing at least as many more written by other programmers, I have come to the conclusion that it is much easier to write good Basic Plus than bad.

Since this is rather a bold statement, perhaps I should expand it a little before continuing. By easier I mean that a given aim can be achieved with a total expenditure of less time. The given aim is usually the production of a set of programs forming a system, which I can turn over to clients knowing that if any thing goes wrong then the job of fixing the programs will be straightforward, not a Sherlock Holmes exercise.

Good code is Basic Plus that I can let any of my programmers loose on, confident that he will be able to learn any unfamiliar techniques from the listing rather than having to apply to the original author for an explanation. Documentation is expensive to produce and maintain, so we feel it helps to minimize the need for it.

Bad code is the stuff that sends you into a cold sweat in the middle of the night when you realize that you've acutally let a client loose on it. Poorly structured, with no standards and no documentation of the methods which the programmer has had to invent for himself, this is code you can't have confidence in.

If you don't agree with these informal definitions (and you don't have to), perhaps I should explain that in my business (small PDP-11 software house) the most important attributes of our programs are maintainability, usability and generality, in that order. We do not feel that using clever tricks to make a RSTS/E program run in four minutes rather than five is a sensible way to capitalize on the enormous investment we are making in software production.

Our programs must be maintainable because I don't know if I'm going to be in the office when a client complains about one of my programs, and client satisfaction gets a very high priority on our list. They must be usable because we can sell them faster if potential clients can see how our systems hang together, and they must be general because re-inventing the wheel is a treasonable activity. Programmer time is our major asset, and we can't afford to waste it.

To help us achieve these goals and yet still maintain throughput in the programming department we have built several libraries of standard routines. Most of these routines are functions, since that usually gives a cleaner software interface. People have complained to me that this is inefficient, but my reply is that computers can be inefficient far quicker than I can.

We also have a wonderful pre-processor which glues everything together, puts in line numbers and does a bundle of other things to increase our productivity—but I'll save the sales pitch for another article. Maybe after you read this one it will be easier to understand why we decided to go that way.

CURSOR ADDRESSING

The first concession we decided to make to maintainability was a ban on writing code which depended on any particular hardware. Since this is not always practical, the general rule is to write so as to minimize the impact of hardware changes.

We have standard string variables to perform functions like clear the screen, home the cursor and so on. These are initialized in our standard program skeleton.

We also hide the cursor addressing mechanism in a function. Rather than pass an X and a Y parameter we use a string. In early experiments I found tht I easily forgot which way round the X and Y should go. We also felt that two arguments for a cursor address was a bit excessive, since there is a maximum of five arguments allowed to any function.

So we use an addressing scheme which letters the rows and numbers the columns: "C20" is our standard account number entry position, on the twentieth print position of the third row. It seems to work quite well, and is a positive aid to program readability. The actual code of the function we started using with our CDC terminals is shown in Figure 1.

You can see that we always write in extend mode. This aids readability so much that I can't imagine any excuse for not using it, although I understand there are some strange individuals who prefer to remain in the stone age. So, for example, to print a message on the bottom line of our 24 x 80 screens, we write:

```
PRINT #KB% FNA$("X1"); "THIS IS THE MESSAGE";
```

which works fine. We can, of course, store pre-computed screen addresses in string variables if we want to save on CPU time, but in practice we rarely do this.

```
DEF FNA$(POSN$)                        ! ADDRESS THE CURSOR     &
        =CHR$(155%)+"1"                ! ESCAPE SEQUENCE        &
        +CHR$(VAL(RIGHT(POSN$,2%))+31%) ! ROW ADDRESS           &
        +CHR$(ASCII(POSN$)-33%)        ! COLUMN ADDRESS         &
```

FIGURE 1. A Standard Cursor Addressing Function

ERROR HANDLING

One of the commonest ways to indicate an error in a subroutine or function is to set a flag. We use the variable E% as reserved for this purpose, with the convention that a non-zero value indicates an error. If such an indication is set, our standards say that the variable ERM$ should be set to contain an explanatory text. Many of our functions indicate errors in this way. Obviously, we could handle this by writing:

```
IF E%                                                   ! if error flagged        &
THEN        PRINT FNA$("X1");CHR$(7%);ERM$;             ! then print message      &
            / GOTO    <RETRY>                           ! and loop back           &
```

But this has some disadvantages. Firstly, it is verbose, and I don't like keying in more than I have to. Secondly, it means that any following code must be on a new line (i.e., it needs a new line number), and this breaks the flow of the logic when you're reading it. If it isn't readable then it isn't maintainable, so our answer was FNERROR%, shown in Figure 2.

```
DEF FNERROR%                                    ! CONDITIONAL ERROR       &
\ FNERROR%=E%                                   ! RETURN FLAG VALUE       &
\ IF E%                                         ! IF WE'RE SAYING YES     &
  THEN   E%=0%                                  ! CLEAR ERROR FLAG        &
         \ PRINT #KB% FNA$("X1");               ! MOVE TO BOTTOM LINE     &
           SPACE$((80%-LEN(ERM$))/2%);          ! CENTRE THE MESSAGE      &
           ERM$;CHR$(7%);                       ! PRINT IT AND BEEP       &

FNEND                                           ! ......................  &
```

FIGURE 2. A Conditional Error Printing Function

This function can be used in conditional statement modifiers, which simplifies the error handling code quite a lot, as we can now write

```
GOTO   <RETRY>   IF FNERROR%                    ! loop if any problems    &
```

which has the merit of readability and will fit anywhere in a multi-statement line quite comfortably. If you're beginning to sense the direction we're taking then the next function will be no surprise.

Some functions can only return one error; this happens a lot when we're trying to do things like find a record. Our record reading functions often have one string argument (the key we're looking for), and return either logical record number of a zero to indicate no record was found. If the record wasn't found, then we always want to print an error message, but the message may vary according to circumstances. Hence FNEP% (Figure 3).

```
DEF FNEP%(E$)                                   ! UNCONDITIONAL ERROR     &
\ E%=-1%                                        ! FORCE ERROR CONDITION   &
\ ERM$=E$                                       ! SET MESSAGE UP          &
\ FNEP%=FNERROR%                                ! PRINT, RETURN TRUE      &
\ FNEND                                         ! ......................  &
```

FIGURE 3. A Function To Force Error Message Printing

This allows us to indicate error conditions to the user with statements like this:

```
GOTO 2000 IF FNEP%("NO SUCH RECORD")            ! barf and loop back      &
      UNLESS FNAC.READ%(KEY.$)                  ! if record not found     &
```

which will also slot into a multi-statement line quite nicely, and is quite readable once you get the idea that FNEP% always returns true and always prints a message.

We have, by this means, bought for ourselves peace of mind. We know that the error flag is always cleared once the error condition has been handled, and that the user will get his messages neatly centered on the bottom line of the screen.

Logically, of course, there is no need for the FNEP% function to be called with literal arguments. One development we have considered is to have either a virtual array or a detached error-lookup routine to which any program can apply with an integer to receive a system-wide error message. This would then allow us to convert a single message file into French, German or whatever, and increase our flexibility at small cost.

Just in case anyone is wondering, I should point out that FNERROR% uses SPACE$ ( ) instead of TAB ( ) because we never know what RSTS thinks our current position is. This is a condition local to our site. We always run in echo-control modes, and never print unnecessary carriage returns due to a high fill factor forced upon us by the use of brand X printers hung on the auxiliary outputs of our brand X displays.

THE NITTY GRITTY : USER INPUTS

So, we are now able to position the cursor and print error messages until the cows come home. The next frequent requirement of programs is to get data from the users — they don't all do it, but your users are likely to be unhappy if they can't tell their software what's been happening lately. For this purpose we habitually use functions. The primary input routine gets a string, and all other input functions make use of this one to read a response from a defined cursor position.

The call is FNI$ (POSN$,MIN%,MAX%), where POSN$ is a standard cursor address string, MIN% is the length of the shortest string you're prepared to accept, and MAX% the length of the longest. The code for the function we use in production has been cut down a bit to give us Figure 4, but the guts are there. The omitted code deals with things like supplying default values, detecting special system-wide responses and so on, which wouldn't be interesting or useful. Besides, I have this sneaking feeling that I've already said too much!

```
         DEF FNI$(POSN$, MINLEN%, MAXLEN%)          ! GET STRING FROM USER   &

XXX10    W9$=FNA$(POSN$)                            ! BUILD ADDRESS STRING   &
          \ W9$=W9$+SPACE$(MAXLEN%)+W9$             !    TO CLEAR FIELD      &
          \ PRINT #KB% W9$;                         ! CLEAR AND MOVE BACK    &
          \ PRINT #KB%, RECORD 256%, CHR$(MAXLEN%)  ! DEFINE THE FIELD       &
          \ GET #KB%                                ! GET A RESPONSE         &
          \ FIELD #KB%, RECOUNT AS W1$              ! ALLOW ACCESS TO IT     &
          \ W1$=CVT$$(W1$,157%)                     ! REMOVE FROM BUFFER     &
          \ PRINT #KB% FNA$("X1"); VDU.CLL$;        ! CLEAR ANY ERROR MSG    &
          \ W1%=LEN(W1$)                            ! GET RESPONSE LENGTH    &
          \ FNI$=W1$                                ! AND THE RESULT IS...   &
          \ GOTO XXX50 UNLESS W1%<MINLEN%           ! STRAIGHT OUT IF THE    &
                          OR W1%>MAXLEN%            ! LENGTHS CHECK OUT      &

XXX20    E$=" CHARACTER"+FNP$(MAXLEN%)              ! START TO BUILD ERROR   &
          \ IF MINLEN%=MAXLEN%                      ! IF EXACT NUMBER        &
            THEN  E$=NUM1$(MINLEN%)+E$              ! NEEDED THEN SAY SO     &
                  \ GOTO XXX40                      ! AND GO PRINT ERROR     &

XXX30    IF MINLEN%                                 ! IF RANGE REQUIRED      &
         THEN     E$=NUM1$(MINLEN%)+" TO "          ! THEN SAY X TO Y        &
                  +NUM1$(MAXLEN%)+E$                ! CHARACTERS REQUIRED    &
         ELSE     E$=NUM1$(MAXLEN%)+E$+" MAXIMUM"   ! ELSE X CHARACTERS MAX  &

XXX40    GOTO XXX20 IF FNEP%(E$+", PLEASE")         ! LOOP TO TRY AGAIN      &

XXX50    FNEND                                      ! ---------------------- &

19100    IF ERL=XXX10%                              ! ERROR IN FNI$          &
         THEN     RESUME XXX10 IF FNEP%("ILLEGAL INPUT - PLEASE RETRY")      &
```

FIGURE 4. How To Get Inputs Without Worrying

FNP$(X%) is a simple little function which returns a small letter s unless its argument is one, which helps to keep things grammatical. I won't trouble you with the code for it.

The other input routines are largely a matter of making sensible use of FNI$, and so I'll give you their interface descriptions without the code. If you think this scheme is good enough to use you'll no doubt have your own firm ideas about how such things ought to be organized.

FNI%(POSN$, MIN%, MAX%) gets an integer at the cursor position represented by POSN$, not less than MIN% and not greater than MAX%.

FNI(POSN$, MIN, MAX, PLACES%) gets a real number at cursor position POSN$, in the range MIN <= X <= MAX, and with no more than PLACES% digits after the decimal point.

FNYN%(POSN$) simply gets a Y or an N from the user at the designated cursor position, returning TRUE (—1%) or FALSE (0%) accordingly.

Other input routines with tighter validation requirements always use FNI$, so there is only ever one read from the keyboard. This makes control-Z trapping the easiest thing in the world, and allows you to unclutter your error handlers of things like tests for all the various places at which the user might have hit the control-z combinations.

DISK HANDLING

So now we can all be experts in formatted screen handling. The next requirement is some means of handling disk traffic so that we only read a block in when such a transfer is actually required. We thought that, for small systems at least, it might be a good idea to work exclusively in terms of logical records. We put into our standard program skeleton a dimension statement for the following arrays:

RECBLK%( ) the I'th element holds the number of logical records per block on the file open on Channel I.

CORBLK%( ) holds the block number currently in the buffer as a result of the last get statement executed.

ALTER%( ) if ALTER%(I%) is non-zero then some logical record in the buffer has been amended since being read, and therefore the block must be written back before it is overwritten by a further get.

OFS%( ) holds the offset within the buffer of the logical record currently of interest on this channel.

These are principally used by FBGETB%(CH%,REC%) which we use to get logical record number REC% in for the file open on channel CH%. The code is shown in Figure 5.

```
        DEF FNGETB%(CH%,REC%)                   ! GET A LOGICAL RECORD     &
        \ BLK%=(REC%-1%)/RECBLK%(CH%)+1%        ! COMPUTE BLOCK NUMBER     &
        \ RECSIZ%=512%/RECBLK%(CH%)             ! COMPUTE RECORD SIZE      &
        \ RN%=REC%-(BLK%-1%)*RECBLK%(CH%)       ! AND SUB-RECORD NUMBER    &
        \ OFS%(CH%)=(RN%-1%)*RECSIZ%            ! AND RECORD OFFSET        &
        \ GOTO XXX40 IF BLK%=CORBLK%(CH%)       ! WE ALREADY HAVE IT       &

        PUT #CH%, RECORD CORBLK%(CH%)           ! WRITE CURRENT BLOCK      &
                IF ALTER%(CH%)                  ! IF ANY AMENDMENT         &

        GET #CH%, RECORD BLK%                   ! READ THE RIGHT BLOCK     &
        \ ALTER%(CH%)=0%                        ! SHOW IT'S CLEAN          &
        \ CORBLK%(CH%)=BLK%                     ! AND SHOW WHICH BLOCK     &

XXX40   FNGETB%=-1%                             ! ALWAYS RETURN TRUE       &
        \ FNEND                                 ! ------------------------ &
```

FIGURE 5. Get A Block — But Only If You Must

This allows us to be reasonably profligate with our read requests, and yet be sure that we aren't bunging up the disks with useless I/O. If that isn't worth the CPU time then surely nothing is, when you consider the difference in elapsed time between accessing an in-core record and going to the disk to read a block.

CONCLUSIONS

I don't claim that this is the best way to write Basic Plus — the requirements at different installations are so diverse that there is no single best way. These functions, and many others like them, represent a lot of time spent working out a way that would allow us to throw software together reasonably fast.

The idea behind this article was not primarily to blow our trumpet (although sadly nobody else does this for us), but to offer ideas about improving programmer productivity in the Basic Plus environment. I hope other users will be giving us the benefit of their experience in future editions of the RSTS PRO.

Our efforts are mostly geared towards minimizing the impact of changes, both in hardware systems and user requirements, on our software development effort. One indication that this works came when we were able to change our complete sales and purchase ledger packages to handling two types of VDU (CDC and Hazeltine, with very different command structures) with only two man-days effort.

We do have other functions, some of which we regard as proprietary. We hve also developed our software in a way which will allow onward migration to Basic-Plus 2. This has mostly involved hiding the "naughty bits" in functions in the standard libraries.

The Version 7 BASIC-Plus-2 "Build" that won't . . .

By Jeffrey R. Harrow, Lockheed-Georga Co.

Among the newly supported capabilities and features which DEC has included in RSTS/E Version 7, the old familiar BASIC-Plus System Default Run-Time System can now give way to other System Default Run-Time Systems, notably BASIC-Plus-2. As usual with such a radical change, some problems and confusion have developed.

For our scenerio, let's assume that we're building a new RSTS/E system, offline, from scratch:

1. During the first part of the SYSGEN dialog, we come to the question:

RSX as default RTS?

Although you really don't want RSX as your final default RTS, you must say "YES" here, so that you will get the RSX utilities brought in and patched PRIOR to building BASIC-Plus-2.

2. In the BASBLD dialog, you will come to the following questions concerning the BASIC-Plus-2 HELP function:

Generate HELP files <NO>
HELP Files Device <SY:>
HELP Files Account <[1,2]>

You should choose a dedicated account for the HELP text files, as there are several hundred standard help files created, and this is a bit much to add to the already overcrowded [1,2]. Let's say we choose [1,12]. The dialog will finish, various files will be PIPed from the distribution medium, and you will finally see:

>RUN BP2HLP.TSK

This program creates the various standard HELP text files used by BP2, rather than PIPing a large number of files from the distribution medium. However, under the scenerio of a new system build which we are using, regardless of how long you wait for BP2HLP.TSK to finish, nothing will happen!

A CTRL/T will only show that ATPK is Sleeping and using no CPU time (which is normal during a BUILD). If, however, you have an 11/70 with those "antiquated" front panel lights, you'll notice that the system is quite busy (no idle time).

The problem is that the account which you've specified to contain the HELP files ([1,12]) doesn't yet exist, and the BP2HLP.TSK program is (apparently) looping without reporting any error message to the terminal!

If you CTRL/C out at this point, you'll find that none of the HELP files have been created and indeed, the [1,12] account doesn't exist.

The solution is evident, of course, and that is to create the HELP file account prior to beginning the BASBLD procedure, just prior to step 6.5.3 in the SYSGEN manual.

Let's try:

>RUN $REACT <CR>
?Can't find file or account
>

Well, of course . . . $REACT, a CUSP, has not been built yet! If you've built a BASIC Plus Run-Time System, you could make it your current System Default Run-Time System by going back into the INIT code and making BASIC the Default Run-Time System using DEFAULT, then re-starting RSTS/E and entering the appropriate SYSCALLs to setup the password in RAD50 and create the account. If you did NOT build BASIC Plus, then of course you could write a short MACRO program to do the same thing . . .

I wouldn't be painting this dark picture if there wasn't an easy alternative: Fool $PATCPY, which is already there, into creating the account for you, as follows:

>RUN $PATCPY <CR>
<PATCPY's Banner Line >

Enter distribution drive/PPN <SY:[1,2] > : <CR>
Enter output drive/PPN <SY:[200,200]> : SY:[1,12] <CR >
%Can't find file or account — SY:[1,12]
Attempt to create SY:[1,12] <NO> ? YES <CR >
Account SY:[1,12] created with your password.

Package to patch? CTRL/C
>

Now, continue with step 6.5.3, and when BASBLD finally runs BP2HLP.TSK, it should finish in less than 5 minutes on an 11/70.

3. After you've successfully built and autopatched BP2, you should be CERTAIN to add any patches to the BP2 components which were published in the Software Dispatch subsequent to the production of your Autopatch tape!! This should be done PRIOR to building the CUSPs! If you don't, you might have to re-build all of your CUSPs again, a long process, to include modified modules or fix compiler generated errors.

With these additions to the SYSGEN procedures, building a new system with BASIC-Plus-2 as the System Default Run-Time System should procede without any problems.

Maryann Oskirko, Executive Director of DECUS, accepts her RSTS Professional T-shirt after attending a N.Y. Metro Lug Meeting.

John Runyon, President of the N.Y. Metro Lug, accepts his RSTS Professional T-shirt following a monthly meeting in Manhattan.

List of Advertisers

TUNE7.BAS

Release Version 1.0

By Chuck Zappala, Teltone Corporation

This article addresses the problem that Mr. Wendell Peterson, United Industry, Inc., described in the **RSTS Professional**, (Vol. 2, No. 2, p.6.)* The program displays system performance in the graphics mode on a VT-52 or VT100 terminal. This histogram-like display provides information of directory and data read and write caching hits as well as the more traditional statistical data obtained from STATUS in RSTS/E version 6C. I kept the program as simple as possible to make it easier for the end user to customize. There are enough comments in the program to achieve this. In addition, there is embedded code with comments that allow the user several options in the presentation of the data on the screen.

The program is used primarily to observe the effects of the cache statistics by varying the XBUF size, cluster size and data residency time. These values are shown as percent of total hits for both data read and data write. The program is run while the system is on line and used much the same way the old STATUS program was used.

Since the program is relatively short, it should be painless to enter as source code. The program is written for a privileged account and it could be modified to run in a non-privileged account very easily. I would be very happy to hear from others regarding the usefulness of this program.

* **Mr. Peterson's letter:** . . . How about an article on measuring system performance. We're running version 7.0 of RSTS and have tried using the programs STATUS and QSTATS. The output from these two are not very compatible and they seem somewhat at odds. I have tried DEC, DECUS and other users for information on QSTATS, but to no avail. . . .

PROGRAM DESCRIPTION

This program is based on a program called HISTOG.V52 by Brian D. Lockrey, ITT North Technology Center, 4565 Columbus Pike, Deleware, Ohio 43015, 614—548-4301 ext. 278. The original program performed a few graphic functions on the performance of a RSTS/E ver 6 system. The following modifies the original to include more information, especially the cache performance on a RSTS/E ver 7 system.

NOTICE

TELTONE assumes no responsibility for the accuracy of data obtained by this program, nor for the interpretation of that data. It is strongly recommended that you consult a system software specialist before undertaking any system monitoring and tuning operations.

The release of this program does not imply a commitment by TELTONE to provide any maintainability, future releases or revisions of this program. However, comments and improvements are welcomed.

This program may be used freely among the DIGITAL/DECUS community. However, the program is prohibited from being used in any publication without a credit line.

Chuck Zappala, Teltone Corp., 10801 120th Avenue, N.E., Kirkland, WA 98033, 1-206-827-9626.

```
!         MODIFICATION HISTORY:
!
!         DATE                VER                   DESCRIPTION
!
!
100       EXTEND
          \ DIM EXEC(70)
          \ DIM CACH(70)
          \ DIM SYS.12%(30)
          \ DO% = -1%
          \ HOME.CLEAR$ = CHR$(155%)+CHR$(72%)+CHR$(155%)+CHR$(74%)
          \ GRAPHICS.ON$ = CHR$(155%) + "F"
          \ GRAPHICS.OFF$ = CHR$(155%) + "G"
```

```
        \ BLOCK.BAR% = 97%
                !
                ! DIM EXEC ARRAY FOR DATA STORAGE FOR REAL TIME
                ! DIM CACH ARRAY FOR DATA STORAGE FOR REAL TIME
                ! DIM SYS  ARRAY FOR MONITOR TABLES, PART 2
                ! DEFINE COMMON VARIABLES FOR SCREEN
                !
125     BASE.STATS% = PEEK(156%) &
        \ DISK.BASE% = PEEK(BASE.STATS%) &
        \ JSTATS.BASE% = PEEK(BASE.STATS%+2%) &
        \ CACHE.BASE% = PEEK(BASE.STATS%+6%) &
                ! GET THE MONITOR STATISTICS TABLES BASE. &
                ! GET THE DISK STATS BASE, JOB/CPU BASE, AND CACHE BASE. &
        \ IF (DISK.BASE% AND JSTATS.BASE% AND CACHE.BASE%)=0% THEN &
                PRINT "STATISTICS FEATURE NOT AVAILABLE" &
        \       GOTO 32767 &
                ! DISK AND JSTATS ARE 0 IF MONITOR STATS NOT CONFIGURED. &
                ! ALL MUST BE CONFIGURED FOR TUNE7.BAS TO RUN. &
!
128     CHANGE SYS(CHR$(6%) + CHR$(-12%)) TO SYS.12%
                \ SYS.12%(I%) = SYS.12%(I%) + SWAP%(SYS.12%(I%+1%))
                  FOR I% = 3% TO 29 % STEP 2%
                \ JOBCNT% = SYS.12%(13%)
                        !
                        ! GET MONITOR TABLES, PART 2.
                        ! STORE CURRENT NUMBER OF JOBS.
                        !
130     IDENT.STG$="V7.0"
132     PRINT HOME.CLEAR$;
135     PRINT IF CCPOS(0%)
        \ PRINT:PRINT "TUNE7.BAS";CHR$(9%);IDENT.STG$;CHR$(9%);
          CVT$$(RIGHT(SYS(CHR$(6%)+CHR$(9%)+CHR$(0%)),3%),4%)
                !
                ! PRINT THE PROGRAM AND SYSTEM ID
                !
140     PRINT
150     INPUT 'ENTER SLEEP TIME IN SECONDS';SLEEP.TIME
160     PRINT
175     PRINT 'THE FIRST DISPLAY SHOWS AVERAGES SINCE LAST START UP.'
                \ PRINT 'JOBS ON SYSTEM IS ALWAYS CURRENT.'
                \ PRINT 'USE <RETURN> OR SLEEP TIME TO UPDATE THE DISPLAY.'
                \ PRINT 'USE <CTRL-C> TO TERMINATE PROGRAM.'
                \ PRINT 'THIS PROGRAM HAS DATA DISPLAY OPTIONS.'
                \ PRINT 'REFER TO DOCUMENTATION IN THE PROGRAM.'
                \ SLEEP 10%
                        !
                        ! SLEEP.TIME IS THE NUMBER OF SECONDS BETWEEN
                        ! STATISTIC DATA GATHERING.
                        ! THE FIRST PAGE OF THE DISPLAY STATISTICS
                        ! ARE AVERAGES DISPLAYED MOSTLY IN
                        ! PERCENT.  KB.CHAR. IN AND OUT ARE DISPLAYED
                        ! IN CHAR. PER SECOND.  NOTE THAT CHAR. OUT
                        ! ARE DIVIDED BY TEN, SO THAT THE BAR GRAPH
                        ! IS PREVENTED FROM OVER FLOWING
```

```
200     WHILE DO%
        \       PRINT HOME.CLEAR$;
        \       JOB.COUNT = PEEK(JOBCNT%) AND 255%
        \       GOSUB 10010
        \       GOSUB 10030
        \       PRINT '                   SYSTEM STATISTICS AND TUNING AID FOR RSTS/
E VER. 7 ';"<";SLEEP.TIME;">"
        \       FAKE.IT$ = FNBAR$('JOBS ON SYSTEM  ',JOB.COUNT)
        \       X = FNPERCENT(SIN.TOT.HITS,SIN.TOT.READS)
        \       FAKE.IT$ = FNBAR$('TOTAL CACHE HITS',X)
        \       X = FNPERCENT(SIN.DIR.HITS,SIN.DIR.READS)
        \       FAKE.IT$ = FNBAR$('TOTAL DIR  HITS ',X)
        \       X = FNPERCENT(SIN.DATA.HITS,SIN.DATA.READS)
        \       FAKE.IT$ = FNBAR$('TOTAL READ HITS ',X)
        \       X = FNPERCENT(WRITE.HITS,WRITE.CHECKED)
        \       FAKE.IT$ = FNBAR$('TOTAL WRITE HITS',X)
        \       X = FNPERCENT(CACHE.TIME,UPTIME)
        \       FAKE.IT$ = FNBAR$('TOTAL CACHE TIME',X)
        \       X = FNPERCENT(SYS.CHARGED,UPTIME)
        \       FAKE.IT$ = FNBAR$('SYSTEM CHAR TIME',X)
        \       X = FNPERCENT(SYS.UNCHARGED,UPTIME)
        \       FAKE.IT$ = FNBAR$('SYS UNCHARG TIME',X)
        \       X = FNPERCENT(CHARGED,UPTIME)
        \       FAKE.IT$ = FNBAR$('TOTAL CHARG TIME',X)
        \       X = FNPERCENT(UNCHARGED,UPTIME)
        \       FAKE.IT$ = FNBAR$('TOTAL UNCHG TIME',X)
        \       X = FNPERCENT(LOST.TIME,UPTIME)
        \       FAKE.IT$ = FNBAR$('LOST TIME       ',X)
        \       X = FNPERCENT(NULL.TIME,UPTIME)
        \       FAKE.IT$ = FNBAR$('NULL TIME       ',X)
        \       X = FNPERCENT(USER,UPTIME)
        \       FAKE.IT$ = FNBAR$('USER TIME       ',X)
        \       X = FNPERCENT(FIP.NEEDED,UPTIME)
        \       FAKE.IT$ = FNBAR$('FIP NEEDED      ',X)
        \       X = FNPERCENT(FIP.DISK.WAIT,UPTIME)
        \       FAKE.IT$ = FNBAR$('FIP DISK WAIT   ',X)
        \       X = FNPERCENT(FIP.SAT.WAIT,UPTIME)
        \       FAKE.IT$ = FNBAR$('FIP SAT WAIT    ',X)
        \       X = FNPERCENT(FIP.CPU,UPTIME)
        \       FAKE.IT$ = FNBAR$('FIP CPU TIME    ',X)
        \       X = FNPERCENT(FIP.IDLE,UPTIME)
        \       FAKE.IT$ = FNBAR$('FIP IDLE TIME   ',X)
        \       X = FNPERCENT(FIP.WAITING,UPTIME)
        \       FAKE.IT$ = FNBAR$('FIP WAIT TIME   ',X)
        \       X = FNPERCENT(IO.TIME,UPTIME)
        \       FAKE.IT$ = FNBAR$('TOTAL I/O TIME  ',X)
        \       X = FNPERCENT(KB.CHAR.IN,UPTIME)
        \       FAKE.IT$ = FNBAR$('CHAR/SEC IN     ',X)
        \       X = FNPERCENT(KB.CHAR.OUT,UPTIME)
        \       FAKE.IT$ = FNBAR$('CHAR/SEC OUT/10 ',X/10)
        \       SLEEP SLEEP.TIME
        \ NEXT
        \ STOP
```

```
        !
        ! THE IS THE MAIN LOOP OF THIS PROGRAM.
        ! GET THE CURRENT JOB COUNT FROM THE MONITOR TABLE, PART 2.
        ! THE DATA IS COLLECTED BY THE GOSUB'S AND TWO MAIN
        ! FUNCTIONS DISPLAY THE STATISTICS.  FNPERCENT COMPUTES
        ! THE PERCENTAGES OF TWO ITEMS RETURNED FROM THE
        ! STATS TABLES.  FNBAR PRINTS THE GRAPHIC BARS.
        ! NOTE THAT THE FUNCTION FNX OBTAINS REAL TIME STATS
        ! DATA, WHICH IS HARD CODED IN, FOR THE CACHE TABLE.
        ! THE SAME IS TRUE FOR FUNCTION FNZ BUT FOR THE JOB TABLE.
        ! TO OBTAIN 'RUNNING STATISTICS' USE THE FUNCTION FNC
        ! IN PLACE OF FNX AND USE THE FUNCTION FNJ IN PLACE OF
        ! FNZ.  SEE LINES 10010 THRU 17000.
        !
10010           SIN.TOT.READS       =     FNX(0%) &
        \       SIN.DIR.READS       =     FNX(4%) &
        \       SIN.TOT.HITS        =     FNX(8%) &
        \       SIN.DIR.HITS        =     FNX(12%) &
        \       CHERST              =     PEEK(CACHE.BASE%+16%) &
        \       CHENUM              =     PEEK(CACHE.BASE%+18%) &
        \       CHENUE              =     PEEK(CACHE.BASE%+20%) &
        \       CHFCNT              =     PEEK(CACHE.BASE%+22%) &
        \       CHDCNT              =     PEEK(CACHE.BASE%+24%) &
        \       SIN.DATA.READS      =     FNX(26%) &
        \       SIN.DATA.HITS       =     FNX(30%) &
        \       NOLOOK.READS        =     FNX(34%) &
        \       INSTALL.READS       =     FNX(38%) &
        \       MCLU.READS.CAS      =     FNX(42%) &
        \       MCLU.READS.NOTC     =     FNX(46%) &
        \       SEG.READS           =     FNX(50%) &
```

CHEERS & JEERS

Carl B. Marbach

Cheers to: The Aladdin/Stanley Thermos Company. They are the ones who make stainless steel thermos bottles that won't break. They come in many sizes. If you have ever dropped a glass vacuum bottle and heard the crash as all the glass breaks, try a stainless steel unbreakable one; they won't break even when dropped. I had two (one for hot, one for cold) for about 12 hard years. Camping in Newfoundland to California; dropped from high places; dented all over; well traveled!

Well, both stoppers had broken and the cups wouldn't screw on properly so I finally packaged them both up and sent them to Aladdin with a request that they repair and return. They returned ... Two Brand New Bottles! I guess they really mean these to be lifetime investments. Thanks and CHEERS!

• □ •

Jeers to: Intertec Company, makers of the Superterm printing terminal (and now CRT's), I bought one of these about 6 months before the 180 CPS DECwriters came out. First, I had trouble getting ribbons for it, calls to the factory were ignored! Then, promises were not kept. Bad Scene. The printhead began having trouble and my Distributor couldn't get parts. More calls to the factory; my order for a new printhead was returned — not available. A product languishes in the field because the factory won't support it! JEERS to Intertec.

```
      \        WRITE.CHECKED    =        FNX(54%) &
      \        WRITE.HITS       =        FNX(58%) &
      \ RETURN
               ! GET CACHE INFO: PEEKS ARE 16-BIT, FNX OR FNC ARE 32-BIT &

10030          TICKS            =        PEEK(JSTATS.BASE%) &
      \        UPTIME           =        FNZ(2%) &
      \        UPSECS           =        UPTIME/TICKS &
      \        FREE.SMALL.BUFF  =        PEEK(FREES%+2%) &
      \        SYS.CHARGED      =        FNZ(8%) &
      \        LOST.TIME        =        FNZ(12%) &
      \        SYS.UNCHARGED    =        FNZ(16%) &
      \        NULL.TIME        =        FNZ(20%) &
      \        USER             =        UPTIME - NULL.TIME - LOST.TIME - &
                                         SYS.CHARGED - SYS.UNCHARGED &
      \        CHARGED          =        USER + SYS.CHARGED &
      \        UNCHARGED        =        NULL.TIME + LOST.TIME + SYS.UNCHARGED
      \        FIP.NEEDED       =        FNZ(24%) &
      \        FIP.IDLE         =        FNZ(28%) &
      \        FIP.WAITING      =        FNZ(32%) &
      \        FIP.CPU          =        FIP.NEEDED - FIP.WAITING &
      \        FIP.CODE.WAIT    =        FNZ(36%) &
      \        FIP.DISK.WAIT    =        FNZ(40%) &
      \        FIP.SAT.WAIT     =        FNZ(44%) &
      \        FIP.OTHER.WAIT   =        FNZ(48%) &
      !        CPU.PRI.0        =        FNZ(52%) &
      !        CPU.PRI.1        =        FNZ(56%) &
      !        CPU.PRI.2        =        FNZ(60%) &
      !        CPU.PRI.3        =        FNZ(64%) &
      \        CPU.PRI.4        =        FNZ(68%) &
      \        CPU.PRI.5        =        FNZ(72%) &
      \        IO.TIME          =        CPU.PRI.4 + CPU.PRI.5 &
      \        CACHE.PRI.0      =        FNZ(78%) &
      \        CACHE.PRI.1      =        FNZ(82%) &
      \        CACHE.PRI.2      =        FNZ(86%) &
      \        CACHE.PRI.3      =        FNZ(90%) &
      \        CACHE.PRI.4      =        FNZ(94%) &
      \        CACHE.PRI.5      =        FNZ(98%) &
      \        CACHE.TIME       =        CACHE.PRI.0 + CACHE.PRI.1 + &
                                         CACHE.PRI.2 + CACHE.PRI.3 + &
                                         CACHE.PRI.4 + CACHE.PRI.5 &
      \        KB.CHAR.IN       =        FNZ(110%) &
      \        KB.CHAR.OUT      =        FNZ(114%) &
               ! GET JOBS INFO: PEEKS ARE 16-BIT, FNZ OR FNJ ARE 32-BIT &
               !        CPU.PRI.0        CPU executing user program &
               !        CPU.PRI.1        CPU executing the null job &
               !        CPU.PRI.2        not used by RSTS &
               !        CPU.PRI.3        CPU executing monitor code &

10090 RETURN
               ! END OF THE DEFININTION OF CACHE AND JOB STATS
15100 DEF* FNC(P%) =   (32768. + (PEEK(CACHE.BASE% + P%) EQV 32767%))+ &
                        (PEEK(CACHE.BASE%+P%+2%)*65536.) &
      !
               ! FUNCTION FNC EXTRACT A 32-BIT CACHE STAT INTEGER
               ! FROM THE MONITOR AREA AND CONVERT IT TO FLOATING POINT.
               ! P% = ADDRESS FOR THE PEEK
               !
```

```
15200   DEF* FNJ(P%) = (32768. + (PEEK(JSTATS.BASE%+P%) EQV 32767%)) +
                (PEEK(JSTATS.BASE%+P%+2%)*65536.)
                !
                ! FUNCTION FNJ EXTRACT A 32 BIT JOB STATS INTEGER
                ! FROM THE MONITOR AREA AND CONVERT IT TO FLOATING POINT.
                ! P% = ADDRESS FOR THE PEEK.
                !
16000   DEF* FNZ(Q%)
                \ Q = FNY(JSTATS.BASE% + Q%)
                \ FNZ = Q - EXEC(Q%/2%)
                \ EXEC(Q%/2%) = Q
                \ FNEND
                        !
                        ! COMPUTE DATA AS REAL TIME SINCE LAST
                        ! SLEEP TIME INTERVAL FROM JOB STATS
                        ! TABLE
                        !
16005   DEF* FNX(G%)
                \ G = FNY(CACHE.BASE% + G%)
                \ FNX = G - CACH(G%/2%)
                \ CACH(G%/2%) = G
                \ FNEND
                        !
                        ! COMPUTE DATA AS REAL TIME SINCE LAST
                        ! SLEEP TIME INTERVAL FROM CACHE STATS
                        ! TABLE
                        !
16010   DEF* FNY(Q%)
                \ Q = PEEK(Q%)
                \ Q = Q + 65536. IF Q < 0.
                \ FNY = PEEK(Q% + 2%) * 65536. + Q
                \ FNEND
                        !
                        ! RETURN A PEEKED VALUE FROM
                        ! THE JOB OR CACHE TABLE
                        !
17000   DEF FNBAR$(T$,X)
        \ X1 = X / 2
        \ X% = X
        \ PRINT T$; ":"; TAB(10%); X%; TAB(22%);
          GRAPHICS.ON$; STRING$(X1,BLOCK.BAR%); GRAPHICS.OFF$
        \ FNEND
                !
                ! GET THE NUMBER OF BARS AND DIVIDE BY 2.
                ! CONVERT IT TO AN INTEGER.
                ! T$ = STRING NAME OF VARIABLE.
                ! PRINT VARIABLE  NAME THEN TAB AND PRINT ITS VALUE.
                ! TAB AGAIN TO START THE BAR.
                ! TURN ON GRAPHICS, AND USE STRING FUNCTION TO PRINT
                ! THE NUMBER OF BARS, THEN TURN OFF GRAPHICS.
                !
18000   DEF FNPERCENT(Q1,Q2)
        \ Q2 = .01 IF Q2 = 0.0
        \ FNPERCENT = Q1 / Q2 * 100.0
        \ FNEND
        !
        !       CALCULATE PERCENTAGES
        !
32767   END
```

GXZY XMQRZPD RST L*

By Joel Schwartz, M.D.

*The answer to this cryptogram is on page 79.

I RECENTLY came across several new games on the computer. The first was a terrific new computer simulation of the popular game **Gromitz.** I played other computer versions of this game but this program is by far the best. The game is played with exactly the same rules everyone is familiar with so there is no need for me to go into great detail. The only bad part is that this version is limited to 27 players. But if you can be patient for your turn this really is not a problem.

The second new game is called **Rotten Peanuts.** You start out in Libya, with enough food for 262 days and 250,000 dollars. The object is to negotiate with a foreign government without getting caught by the secret police or your brother. The outcome of the game is too predictable; but you can play it again by promising to reform.

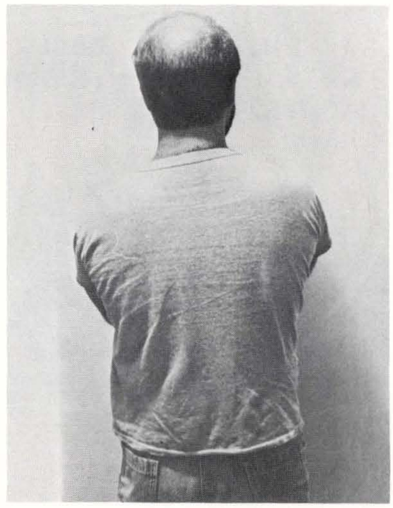
The third game is called "**Find the President.**" By asking different candidates selected questions you have to find out who the president is, what he said that was true, and what time he contradicted himself. The game can be tricky since sometimes more than one candidate can tell the truth!

The fourth game is called **Shalom**! The computer randomly gives you seven letters and you have to find as many Hebrew words as possible. Point values are given to each word and there are extra points for pornbrew (pornographic Hebrew). This is a great teaching game for the whole family. Remember, that Japanese goes top down, while Hebrew goes with reverse line feeds and carriage returns.

Another game I found very interesting is "**Why TECO.**" Everyone starts out at DEC and as they advance through DECUS they answer the life long question, "Why TECO?."

The game I liked best of all was called **Doubletalk.** This game involved uniquely at least four or five principles, who at any one time constantly procrastinated or possibly vociferously attacked their objective. The best part was near the inception of the middle when all five players were approaching the end.

And finally a letter from a lady in Terre Haute, Indiana —



Dear Dr. Schwartz:
Enough of this fooling around. I want to see a picture of you from the waist up — without a mask!
Sincerely,
Elinor Schwartz

GOODIES

By Eddie Cadieux

'ODE TO A PROGRAMMER'

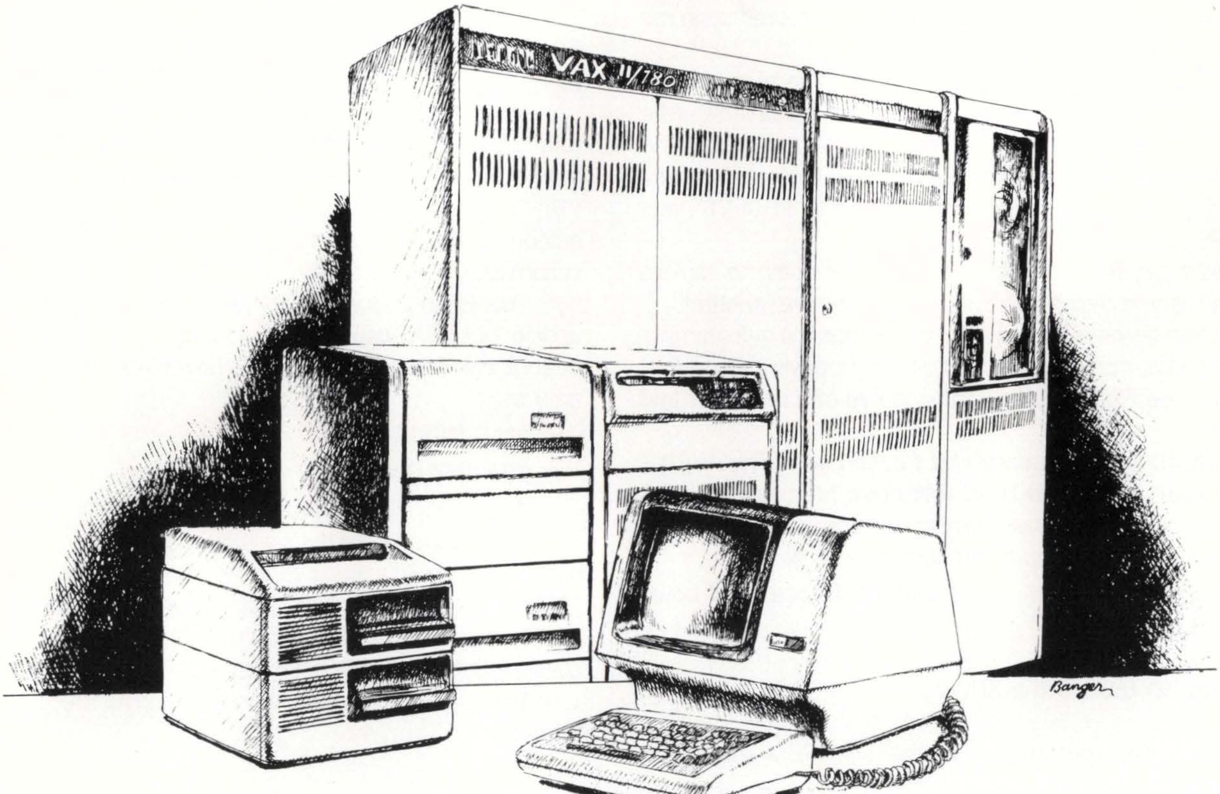
"No program is perfect," they say with a shrug.
"The client is happy — what's one little bug?"
But he was determined, the others went home,
He dug out the listings, deserted, alone.

Night passed into morning, the room was cluttered
with core dumps and punch cards, "I'm close," he
muttered.
Chain-smoking, cold coffee, logic, deduction,
"I've got it!" he cried, "Just change one instruction!"

Then change two, then three, as year followed year,
And strangers would comment, "Is that guy still here?"
He died at the console of hunger and thirst.
Next day he was buried face down, nine edge first.

And his wife, through her tears, accepting her fate,
Said, "He's not really gone — — —,
he's just working late!!"

WORD PROCESSING



PDP-11 AND VAX

MULTI-USER

Written entirely in MACRO-11 machine language, you can have up to 90 people using the full power of WS-11 word processing simultaneously without impairing the performance of data processing.

EASY TO USE

Color coded commands are clearly labeled on the keyboard. Menus display all choices on the screen. No technical language or cryptic commands to remember. The screen immediately shows all changes you make to your document. No surprises.

FROM PDT TO VAX

WS-11 can be used on the full PDP-11 family using its own operating system, RSTS/E, CTS-500, RSX-11M or VAX/VMS. Transition from one system to another can be made without any change to your documents.

POWERFUL FEATURES

Just a sample: Right margin justification, Centering, Search and replace, Libraries of stored text, Simultaneous editing and printing, Support for a wide range of printers including Xerox, NEC, Qume, Sanders and DEC.

$2000-$3500

WORD SYSTEM 11

NDS

NORTHWEST DIGITAL SALES
668 S. Sunset Avenue
West Covina, CA 91790
(213) 960-2895

More I/O FROM MACRO — Quickly and Easily!

By Bob "MACRO MAN" Meyer

Hi again! Macro Man here . . . (an alias bestowed upon me by his excellence, Richard D. Mallery, III).

As promised in my last article, we're going to move into some more MACRO this month. [Oh, boy!] For those of you who MISSED my last article . . . [the nerve...] go to your secret hiding place, [I find that most of my readers keep their copy in the John . . .], dig out your last issue of the "RSTS PROFET", and CATCH UP!

Ok, let's get serious. Today, class, we're going to explore the mysteries and depths of the .READ directive [ahhhhh . . .]. My reason for choosing the .READ call is that it's quite similar to the .WRITE directive you saw [if you were paying attention] in the last issue. [Besides, I've got one hell of a hangover and it's easy.]

The .READ Call is as easy and fun to use as the .WRITE that you keyed in (?) and tried last time; both use the XRB [Remember him?] and pass similar parameters. Let's step through the sample program for some more details.

Starting from 'MOV #XRB,R0' [three lines down from the label 'INPUT:']:

1) Point R0 toward the XRB.

2) Specify an input buffer length. [This is a maximum; anything more will be truncated.]

3) As DEC would say 'Must be ZERO !'. [This becomes good ol' RECOUNT.]

4) Tell RSTS where to put the data received.

5) Specify MSB [Most significant bits.] of the block number to be read, and tell Him to use channel zero.

6) Specify LSB [Guess.] of the block number to be read. [Random access devices only.]

7) Next is the 'Wait Time'. [Like 'WAIT 5%' in Basic.] [Or -1% for Run-Time System read [beyond the scope of this hangover . . .]]

8) Finally, some device-dependant modifiers can be inserted. A few familiar modifiers for the KB are:

print #1%, record 1%	!output in Binary mode
print #1%, record 256%	!declare Field [for Echo Control]
get #1%, record 8192%	! read without wait

Next, the .READ is executed [EMT 002], and faithful RSTS does the rest. [What'll they think of NEXT!]

The demo program on the following page is setup to show the use of both the .READ and .WRITE calls. Those of you who did your homework last month and keyed in the PRINT routine can use that chunk of code [with some help from Teco [?Why TECO? — ?How TECO?], and use that module with this "new improved" version. The sequence of events [starting from the label 'INPUT:'] as follows:

First, we put the address of the prompt to be printed in R0, followed by the length of the prompt in R1. [We'll see an easier way to do this later.]

Next, the input code described above is executed [hopefully . . .] and some rhetoric from the user's terminal should be in the buffer area. Now we setup some parameters for the PRINT routine [note that the length of the keyed in data anxiously awaits you in the second word of the XRB], and return the test data back to the user. [GIGO]

The Build procedure for these modules was in the last article, but for those of us who's pages are a little sticky, [with peanut butter, of course] here's how to assemble and link your new toy:

MAC INPUT=INPUT

TKB INPUT=INPUT

As before, no prefix files are needed for this guy.

Next issue, [weather permitting . . .] we'll get into ways of making these I/O routines easier to implement by using the MACRO capabilities of the assembler, and creating an object library from them. And if you're REAL good, maybe we'll concoct a RESLIB out of these critters!

Well, last call was 10 minutes ago . . . me go home now. Enjoy!

```
.TITLE  INPUT   INPUT DEMO
.IDENT  /1.0/

;
;INPUT.MAC -- SIMPLE KB INPUT DEMO IN MACRO
;
;4/25/80 BOB MEYER
;

;
;DEFINE SOME CONSTANTS
;
.READ   =       104002                  ;RSTS 'READ' DIRECTIVE
.WRITE  =       104004                  ;RSTS 'WRITE' DIRECTIVE
XRB     =       000442                  ;START OF TRANSFER REQUEST BLOCK

;
;DEFINE TEXT/BUFFER AREA
;
BUFFER: .BLKB   80.                     ;80. BYTE KB BUFFER
.ENABL  LC
PROMPT: .ASCII  /Type a line:/          ;PROMPT MESSAGE
PRMLEN  =       .-PROMPT                ;LENGTH OF PROMPT MESSAGE

;
;START OF MAIN CODE
;
INPUT:  MOV     #PROMPT,R0              ;POINT TO PROMPT
        MOV     #PRMLEN,R1              ;AND TO LENGTH OF PROMPT
        CALL    @#PRINT                 ;PRINT IT

        MOV     #XRB,R0                 ;SETUP XRB FOR INPUT
        MOV     #80.,(R0)+              ;SETUP BUFFER LENGTH
        CLR     (R0)+
        MOV     #BUFFER,(R0)+           ;WHERE TO PUT THE INPUT
        CLR     (R0)+                   ;CHANNEL NO. FOR INPUT
        CLR     (R0)+                   ;BLOCK
        CLR     (R0)+                   ;WAIT TIME
        CLR     (R0)+                   ;MODIFIERS

        .READ                           ;LET RSTS DO THE INPUT

        MOV     #BUFFER,R0              ;POINT TO USER'S INPUT FOR PRINTING
        MOV     @#XRB+2,R1              ;GET LENGTH OF USER'S INPUT
        CALL    @#PRINT                 ;PRINT IT BACK ON THE KB:
        BR      INPUT                   ;LOOP

;
;SUBROUTINE TO PRINT ON THE TERMINAL
;
PRINT:  MOV     #XRB,R2                 ;POINT TO XRB
        MOV     R1,(R2)+                ;PUT LENGTH IN XRB
        MOV     R1,(R2)+                ;TWICE
        MOV     R0,(R2)+                ;POINT TO STRING TO BE PRINTED
        CLR     (R2)+                   ;CHANNEL ZERO
        CLR     (R2)+                   ;BLOCK ZERO
        CLR     (R2)+                   ;NO WAIT TIME
        CLR     (R2)+                   ;NO MODIFERS

        .WRITE                          ;DO THE PRINT

        RTS     PC                      ;RETURN TO CALLER

        .END    INPUT
```

SOFTWARE MANAGEMENT

By Richard A. Marino, Data Processing Design, Inc., Placentia, California

This article was transcribed, condensed and edited from a seminar given by the author at the Toronto 1980 DECUS conference.

Why do we want to talk about software management? If you and your programmers write code that works correctly the first time and runs year after year with no major problems; if you are able to meet deadlines and keep schedules with unbelievable consistency, then you may not need software management. But if your systems are late, have numerous bugs, then welcome to the group. While this article will probably not solve any of these problems, it may help you identify some problems and provide some ideas for possible solutions.

What I will discuss is managing software products and managing software people. First of all, if any of the ideas I present here seem useful to you, be careful. What often begins as a good common sense idea in the mind of a manager sometimes gets translated into something quite different when implemented by the staff. It may not be easy to convince managers, not to mention programmers, of the virtues of a new idea, but one needs to try to introduce new ideas and to convince people to try new methods and new solutions.

One of our first observations is that programming is a unique activity. Not quite engineering, not quite art, it has its own special rewards. First, there is the joy of making something of one's own design. That's the creative satisfaction. Second, there is the pleasure of making something others will find useful or helpful. Third, there is a fascination with the sheer complexity of software. Fourth, there is the joy of learning and doing new and different things. And finally, there is a definite delight in building things from imagination, much like building castles in the air. There is often little substance other than pure thought and pure creativity and part of the fun of programming is the joy of creativity.

Along with the rewards of programming there are also some disadvantages. First, there is the demand for perfection. Human beings are not used to being perfect, very few human activities demand it, but computers demand it. There is also the problem of having outside forces set your objectives, your goals, and your schedule. There is also the unfortunate dependence on others. Unlike the artist, it is difficult for a programmer to create in isolation. And software lacks the permanence of a painting or a building, and may in fact be obsolete soon after completion.

What is perhaps the largest problem of this or any human activity? When anything fails, the cause is often communication. As recounted in the book, "The Mythical Man Month", the story of the Tower of Babel is a prime example of communication problems. In reviewing that project there was a clear mission, sufficient manpower and materials, and enough time, but poor communication and organization destined the project to fail.

Product Planning

How many projects are completed within the initial estimates? Almost none. Why does this happen time after time? There are many reasons, four of which are the most common. First, poor estimating techniques. Second, we often confuse effort with progress. Third, we have no way to adequately measure progress and fourth, we are unable to deal well with schedule slippages.

As the title of the book "The Mythical Man Month" indicates, the "man month" as a unit of measure is a dangerous myth. A man month is a suitable measure for a perfectly partitionable task, such as picking cotton. Ten people can do the work 10 times faster than one person. An unpartitionable task would, for example, be bearing a child. It takes nine months, no matter how many women are involved. Software tasks have very complex interrelationships and the problems that occur often relate to communication. In most cases software tasks are not unpartitionable; adding people does not always get the task done sooner. On large projects especially the time spent communicating with the many people involved increases the time needed to complete the task, and at some point, adding people delays the task further, rather that completing it sooner.

Estimating

If I had to define a typical division of effort on a large project, I would estimate 1/3 planning, 1/3 coding (probably an overestimate), and 1/3 debugging and testing. The planning part is a part we normally ignore, the coding part is the easiest to estimate, and testing is often forgotten, ignored, or completely underestimated. And we have not even considered documentation, another necessity that often negatively impacts true completion of a project. We are wrong in our estimates time after time, partly because we are optimists, as most humans are, and partly because we overestimate the time a person spends on programming (as opposed to other tasks).

Work Environment

Part of the reason our estimates are wrong, especially in the amount of time actually spent doing active, productive work, is that programmers often spend their time, rather than working on the new project, in responding to questions about previous projects, in doing clerical duties, and in other less productive activities. Certainly, part of this problem is management, for in more cases than not, the tools we give a programmer are not the best. At many installations from terminals to computer time, we shortchange the programmer. As with any workmen, better tools do produce better results.

Another important factor in software management is documentation. This topic in itself could occupy an entire article. Let's at least say it requires early planning, careful implementation, and maintenance. It is vital both to the product development, product delivery, and ongoing product support. In my own case we attempt to produce a draft of documentation before coding even begins. An additional benefit of this method is that it defines the goals relatively precisely, at least from the user's viewpoint, of the project.

Structured Techniques

Structured techniques are not only for the large computer shops with giant projects and hundreds of programmers. You may have the attitude that if you ignore them they will go away. In fact, structured techniques mean much more than just GOTO-less programming and if understood can offer benefits in projects of any size. Let's briefly review some of the more popular structured techniques.

Structured Design. This technique demands careful analysis of the problem. Some of its goals include production of a project abstract, a list of goals, a list of objectives, and the constraints that may affect a project (both positively and negatively). Structured design may include functional specifications or prose descriptions of the project at many levels of detail.

Structured Programming. Probably the most widely discussed of the structured techniques, it attempts to apply standards to the use of computer languages. In particular, Structured Programming emphasizes reducing or eliminating the use of constructs such as GOTO's and promoting the structures DO WHILE and IF THEN ELSE. Structured Programming also stresses modular programming and other techniques that improve the understandability and maintainability of software.

Top Down Design. Top down design implies a technique of design that breaks projects into major functions and then minor functions, and then smaller functions, designing a structure with the largest and highest level areas first. This means, for example, designing the user interaction before defining the file access routines.

Top Down Testing. This is one of the most useful techniques and can be applied to even small projects. It implies the development of software so that major interfaces are tested early. Parts of the system requiring low level routines are tested using "stubs" which might simply exit or provide constant or random data in order to provide testing of higher level structures.

Some of the benefits of top down testing and top down design include the ability to see limited working versions early in the development process. Debugging is easier and programmer and user moral are often improved. Parts of the project can be seen working early in the development cycle. Problems with top down testing and design include the increased likelihood of communication problems in larger projects and the requirement for more detailed designs since with projects requiring more than one person interactions between routines must be carefully and relatively completely designed very early in the development process.

Chief Programmer Teams. As originally suggested by Mills, a programming team should be comparable to a surgical team. It has been shown that programmers differ greatly in productivity. However, there are some super-programmers. These people can design and code faster and more efficiently than most programmers. A super-programmer, much like a surgeon, is supported by a team with such members as an administrator, an editor, a language lawyer, a librarian, a tool smith, and a tester. Such teams reduce the communciation requirements and provide high productivity through specialization. The super-programmer can devote time to the overall design and to the coding of critical routines and support people can handle the less critical tasks.

Structured Walk Throughs. Though not as applicable for very small projects or in very small environments this technique is one of the most useful structured techniques. The goal is egoless programming with peer group reviews of design and code for mutual benefits. An atmosphere is created where discussion and critique takes place and where looking for flaws, weaknesses, and ambiguities, becomes a common goal. Walk throughs include the review of specifications, designs, code, and testing techniques. Its aim is building a true team, not in the sense of the super programmer and chief programmer team but a team of relative equals. In conducting walk throughs a presentation is made and members of the group walk through the detail (code, design, etc.) with the author. Managers should be kept out of walk throughs as they are often an inhibiting factor and give the impression that walk throughs and the errors detected contribute to the evaluation of employees. Improper attitude and lack of organization are also typical problems. Walk throughs should be kept short and should strive to detect, not correct, errors.

Other structured techniques include design reviews, the use of psuedo-code in design, and team programming. There are also many others described in the references at the end of this article.

Software Staffing

In software management we also encounter problems with staffing. Staff selection is certainly one of them. The demand for software people is great while the supply is short. Schools often do not provide the right education for practical employment. Head hunters constantly raid one company to supply another. Tests are meaningless and we must often hire based on little more than gut level evaluations.

Another major problem is training. Unfortunately there are few options available. Internal training (if you have inhouse expertise and are willing to dedicate their time to training) is probably best. Training from DEC is a second and somewhat less desirable option largely because of the substantial differences in the quality of the training between Digital offices and different sessions.

Another major problem of managing a software staff is measuring productivity and performance. Here there are no simple answers, but regular reviews that include both evaluation and the setting of future goals are critical to staff management and moral.

There are areas where management can make some overall contributions. First, provide good hardware configurations aimed at development as well as production. Encourage detailed specifications and written communication during design and development. Schedule with kid gloves and provide as much support as possible to your staff.

Software Management

Software management is a very, very difficult task. With all the pressures and all the problems, many of which I have mentioned, the manager himself is often overburdened. Part of this problem results from a fact of life called fire fighting. A manager spends so much time trying to put out fires, trying to correct problems, trying to keep his head above water, that he doesn't have enough time to review, to plan, to evaluate policy, and to develop a strategy. There just isn't enough time. And that is one of the biggest problems of all.

REFERENCES

There are many books about Software Management and Software Engineering. A visit to your local university bookstore might help you find these titles or others that would be valuable reading.

Jenson and Tonies, Software Engineering, Prentice Hall

Preston, Communication for Managers, Prentice Hall

Yourdon, Techniques of Program Structure and Design, Prentice Hall

Brooks, The Mythical Man Month, Addison Wesley

Van Tassel, Program Style, Design, Efficiency, Debugging, and Testing, Prentice Hall

Aron, The Program Development Process, Addision Wesley

Kernighan, Plauger, The Elements of Programming Style, McGraw Hill

Yourdon, Managing the Structured Techniques, Prentice Hall

Metzger, Managing a Programming Project, Prentice Hall

Kerzner, Project Management, Van Nostrand and Reinhold

Weinberg, The Psychology of Computer Programming, Van Nostrand and Reinhold

Cohen, Principles of Technical Management, AMACON

CLASSIFIED

Send Classified Ads to: RSTS Classified, P.O. Box 361, Fort Washington, PA 19034. ($1.00 per word, first 12 words free with one year's subscription.)

XEROX 1750 PRINTER w/Metal Print Wheel, (219) 232-3992. Ask for Jim Welborne.

COMPREHENSIVE COMPUTER SERVICES in St. Louis wants Part-time Programmer. Rapid Growth to Full Timer. Steven Birenbaum, 7305 Manchester, Suite 300, St. Louis, MO 63143.

M&G IN THE ISLE OF MAN for Real Time Commercial Software! Capt. J.S. McKenzie

---

PROPERTY/CASUALTY INSURANCE Systems Consulting, Development, Implementation, Trouble-Shooting. CTS-500 BASIC+2 RPGII COBOL. Ivan Cofresi, P.O. Box 40627, Minillas, Santurce, P.R. 00940.

MOST® COMPUTER SYSTEMS — Superior Software for Industrial Engineers — RSTS/E, PDP-11, IBM 370. Robert Dietrich (412) 351-4100.

RSTS/E CONSULTING & TRAINING, BASIC-Plus, BASIC-2, MACRO-11, DDG, Werner L. Stunkel, 347 Cherokee, Lake Forest, IL 60045. (312) 234-5172.

HINDITRON — A Leading DEC Software House in India. H.S. Mehta.

CONSUMER ELECTRONICS WHOLESALE! Luvor Int'l, 39 W. 29th, NYC 10001, Steve Ostry.

PDP S/WARE. MAILING LISTS, Utilities for RSTS/E, Tenpin Bowling Package. Realtime Software. TWX 36751, REALT HX. J.V. Gabriel.

QUARK CONSULTING GROUP (St. Louis, Boston): Consulting, Contract Programming. 4+ years RSTS experience: BASIC+, BASIC+2, FORTRAN, MACRO11, VAX-MACRO, Graphics, CAD/CAM, System Utilities. (314) 721-1818.

COMPEX DATA SYSTEMS: Digital's new commercial o.e.m. in Holland. P.O. Box 253. 5340-AG OSS.

LOOKING FOR RSTS sites using SI 9400 disk controller. Call 417—836-1130, Don Johnson.

DREAMS, An Electronic Mail System for RSTS. Get free report. Tom Burtnett, Dickinson College, Carlisle, PA 17013. 717—245-1513.

TECHNICAL ECONOMICS Project Control Software with Graphic Capability. 415—525-7774, Vera A. Kahn.

BARRY JONES CONSULTING/Programming, S.F. Bay area, Financial Modeling, Business Systems. 415—566-0153.

SKYPEK SOFTWARE SERVICES: Macro-11, DECnet, System Programming, Application Packages, Consulting, 404—325-7433.

OPPORTUNITY FOR DATA PROCESSING MINICOMPUTER PROFESSIONALS
If you are a CAREER oriented, Data Processing Professional who would like to join one of the most exciting, leading software companies in the world, where your professional growth is limited only by your own ability, we may have an opportunity for you. We are seeking someone with 1 to 2 years experience, with exposure to commercial systems application, and RSTS/E a plus. If a successful future is your goal, please submit resume, references and salary history to:
V.P. Technical Operations
AMCOR Computer Corporation
1900 Plantside Drive
Louisville, KY 40299
502/491-9820
No Agencies Please!

CUPID — The ultimate in application software development aids for DEC datasystems. T.U.B.S. Ltd., 30 New Walk, Leicester, LEI 6TW, England.

WILL TRADE anything for OMSI Pascal, call 901—278-7730 after 5 pm, Rob McAdams.

8KW DEC PDP-12 with RK01, LA30-P $4000. James N. DeVries, 401—246-1200.

JOB COST SOFTWARE for CTS 300-500. Zebec Data Systems, 713—782-3471.

BUY SELL SWAP DEC Equipment Computel Systems LTD. 212—351-5395.

HUMANS, KLINGONS, ROMULANS, Announcing SPB! The ultimate, Frank Farmer. 313—286-8800.

LOWRY AND ASSOCIATES — Engineering Software — Time and Frequency Domain analysis, Mathematical modeling, Analog systems and circuits. Irvine - 714—751-3820.

MACRO SYSTAT 7.0 LARGE FIP
Super Fast — almost no CPU, all standard features + core & open file by job. Works with RSX or BP2 RTS's. Nine track 800 or 1600 only. Send $40.00 cash, check or m/o. Bob Meyer, 9 Lockwood Ave., Fieldsboro, NJ 08505.

HARDWARE & SOFTWARE DESIGN
Mini — Micro
RSTS, A Specialty
Scott Banks, Systems Design, 1108 Morefield Rd., Phila., PA 19115. 215—677-4836.

NEW USED DEC Hardware for immediate delivery. Rent, Lease, Buy, MLPI The Rental People, 604—682-7541.

WANTED FOR CASH: 11/70 w or w/o FPP minimum DEC memory, no peripherals, high-boy cabinets. DEC Service Letter, Dave Mallery 215—364-2800.

RSTS INTERNALS, TUNING, RT11/RSX MACRO under RSTS consultation & programming — call MACRO MAN, Bob Meyer, 609—298-9127.

RSTS/E to RSTS/E direct file transfer over DH/DZ keyboard line. For info write: RANDY PARK, Box 202, Northgate Station, Seattle, WA 98125.

WE DO CUSTOM software work — experienced in RSTS/E and BP2 Accradata Services. Call (317) 447-0541.

DELAWARE VALLEY

TIMESHARING

Eastern Pennsylvania
Philadelphia
South Jersey

DEC 11/70 RSTS/E

24 hours - 7 days
Most commerical packages
Custom work, Raw time

NATIONWIDE DATA DIALOG
70 James Way
Southampton, Pa. 18966
215-364-2800

FOR RSTS/E SOFTWARE consulting in New England, Call (617) 895-5090.

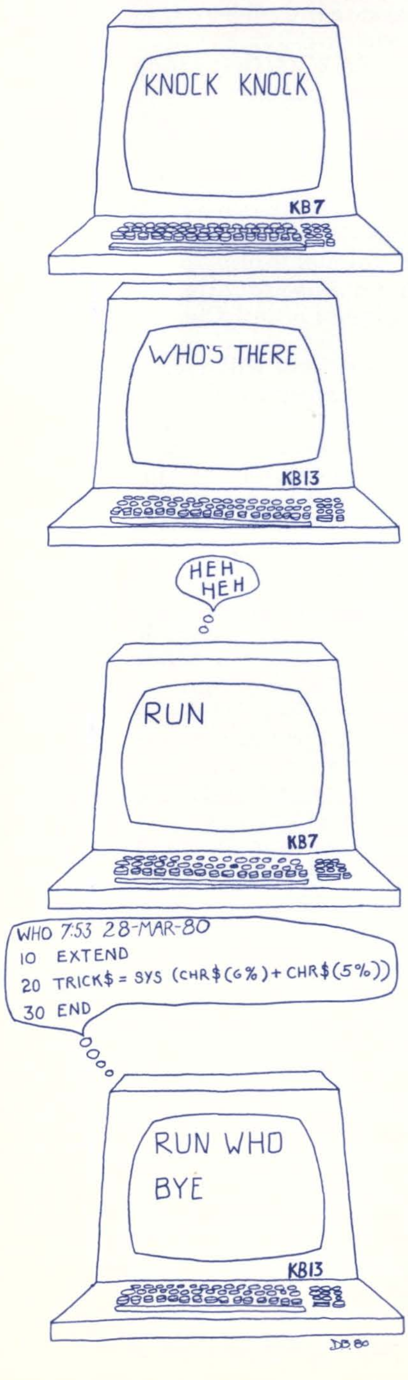
HEAVILY USED DBMS for RSTS/E. Excellent Product. Reasonable Cost. Many Uses. Sauer Computer Systems, Inc. (314) 962-0382.

TIME SHARING AVAILABLE: very reasonable. CTS 500, Basic+2, shared business packages available. Call (219) 753-0411, ask for Dave.

DEC PDP 11/20 FOR SALE, 4K like new $2500, 615/244-6094.

RSTS TIMESHARING available. Metro Detroit Area. Excellent Rates. Call (313) 924-1020.

RSTS BP2, Commercial applications and performance consulting. Gerald Tolman, Concord, MA, 617—369-0635.

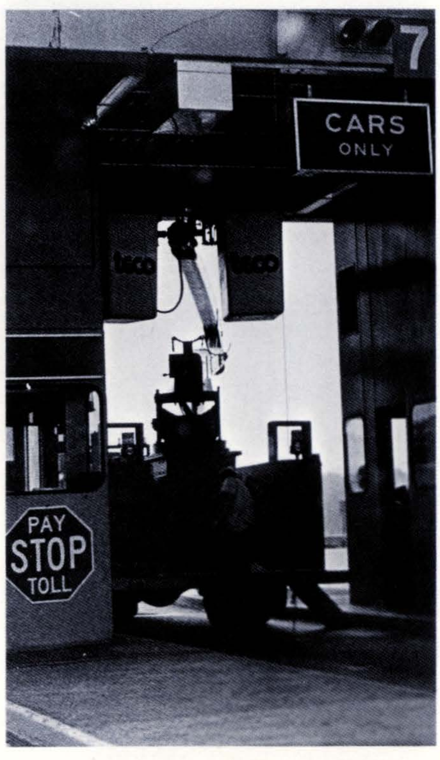


GROSSMAN GRAPHICS

The producers of the *RSTS PROFESSIONAL* are preparing for telecomphotoset - telephone communications phototypesetting.

Our Intelligent Communications Interface is the latest in typesetting technology. We are now able to receive data directly from your computer or word processor to our phototypesetter. Once these keystrokes have been "captured", we will mold your copy to your specifications, or to our design if you wish, and send to you camera ready mechanicals or final printed material.

YOUR NEXT
ANNUAL REPORT, BOOK, NEWSLETTER, . . .
IS JUST A PHONE CALL AWAY!
SAVE TIME AND MONEY call or write for further details:
5041 frankford avenue, 2nd floor, philadelphia, pa 19124 • (215) 357-0782



How TECO? Why TECO?
A prize if you tell us?

RSTS RESCUE SQUAD

We salvage all kinds of disasters:

- unreadable disks
- ruined UFDs and MFDs repaired
- immediate response
- telephone DIAL-UP
- on-site
- software tools
- custom recovery

Brought to you by **M Systems** *and a well known (and read) Disk Directory expert.*

CALL 24 HOURS

215—542-7008

NEWS RELEASES

Occasionally we are requested to print news that may be of interest to the RSTS community. We are happy to offer this feature to our readers. We reserve the right to print only as time and space permit. We cannot return any photos or manuscripts sent to this feature. Send news releases to: *RSTS NEWS RELEASE, P. O. Box 361, Ft. Washington, PA 19034.*

April 28, 1980

COMDATA ANNOUNCES INTEGRAL MODEM FOR TELETYPE 43 TERMINALS . . .

Skokie, Illinois — ComData has added another modem to its extensive line of low speed, low cost modems with the introduction of its model 154E2-12. The new modem, easily installed without modification in Teletype Corporation's series 43 terminals, provided Bell 103/113 compatible 300 baud full-duplex data communication with direct connection to private lines or the dial network via FCC approved standard jacks. Standard features include system diagnostics via keyboard control of analog and digital loopback.

These new modems are in production and available from stock at a single quantity price of $175.00 with the widely known ComData warranty. ComData is located at 8115 N. Monticello, Skokie, Illinois 60076. For further information call (312) 677-3900.

— — — — — — —

April 29, 1980

RABBIT-3 RSTS/E JOB ACCOUNTING AND PERFORMANCE MONITORING SYSTEM . . .

RAXCO Inc., West Palm Beach, Florida announces RABBIT-3, a job accounting and performance monitoring system available on DEC PDP-11's running RSTS/E Version 7.

RABBIT-3 captures user detail accounting information for each job on a session by session or per program basis. CPU time, connect time, device time, DCT's, disk usage, PPN are logged. Additionally job state conditions such as I/O wait, keyboard wait, run state and other RSTS/E monitor states are available.

RABBIT-3 is a self contained stand alone system. Written totally in PDP macro assembler for maximum efficiency, RABBIT-3 provides accounting and system monitoring information in ASCII stream data files for program processing. Optionally produced are RMS indexed files which can be accessed by on-line data base retrieval programs, such as DATATRIEVE. Fast and small, RABBIT-3 runs in 5K core with only 1% (appx.) system degredation.

System crashes pose no problems. using a crash recovery subsystem, RABBIT-3 will automatically recover and restart from a "system down". No accounting or monitoring information is lost.

RABBIT-3 provides the system manager with complete system monitoring information via a user log file. Who used what resources, where, when and how much as it actually occurred is available for analysis. This audit trail may be used to detect computer security breaches, fraud or abuses.

The accounting department can utilize RABBIT-3 output for billing purposes by processing the user accounting information through a billing program or report writer. RABBIT-3 output may be utilized as input to RABBIT-1, a comprehensive billing, invoicing and crosscharging system. RABBIT-2, a system for graphic analysis of computer performance will also accept RABBIT-3 output by the end of the 2nd quarter, 1980.

The RABBIT-3 system is provided via a minitape or diskette complete with installation instructions and system manager options. Actual installation is complete with a running system in less than an hour.

RABBIT-3 is available on a rental basis for $99 per month. It may also be permanently licensed for $2500, which includes macro code.

RAXCO, Inc., (305)842-2115

— — — — — — —

May 6 1980

SSI MOVES, ADDS MARKETING, PRODUCTION SPACE . . .

Fort Lauderdale, Fla. — Southern Systems, Inc. (SSI), printer system manufacturer, added some 15,000 square feet of office and manufacturing space, with a June move into new corporate headquarters in Fort Lauderdale and construction of a new Clearwater, FL plant.

SSI's marketing and sales staff will occupy the new corporate offices, at 2841 Cypress Creek Road, Fort Lauderdale, FL 33309.

Construction on the Clearwater manufacturing facility began in May, with an October completion anticipated. The new SSI plant, located at the St. Petersburg-Clearwater Airport, will house both manufacturing and research/development staffs.

"With our expanded sales in the U.S. and new sales in Europe, our needs in all facets of the company are growing rapidly. The moves were essential to accommodate larger staffs and larger production facilities," said Joseph Horn, SSI president.

SSI has doubled sales each year for the past four years and is projecting continued rapid growth due to expanded marketplaces and distribution channels.

The firm supplies end-users and OEMs with impact printer systems ranging from 200 lpm to 1500 lpm.

Specialists in printer system technology, SSI provides printer systems to be used on computers from DEC, Hewlett-Packard, Data General, Perkin-Elmer, Burroughs, TI, Honeywell, SEL and many others.

At the new Fort Lauderdale office, Telex is 522135, telephones, (305) 979-1000 and (800) 327-5602.

— — — — — — —

May 1980

PDP-11/70 USERS CAN NOW INSTALL THE MAXIRAM MEMORY SYSTEM DIRECTLY ON THE CACHE BUS. IT REDUCES COSTS: GAINS DRAMATIC PERFORMANCE IMPROVEMENTS WITH UP TO 50% INCREASED THROUGHPUT...

El Segundo, California — PDP-11/70 users can now take advantage of a unique system to gain dramatic performance improvements to achieve up to a 50% increase in throughput. The Maxiram Memory System, a solid-state non-rotating equivalent of RS04/RS03 fixed-head disc drives, can be attached directly to the

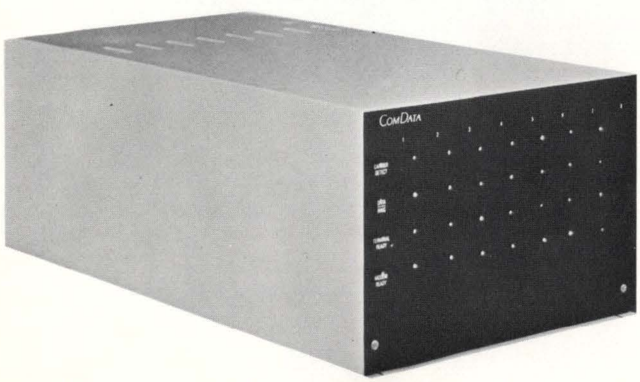
cache bus of PDP-11/70 computers. The system, called the Maxiram-11/70HS, was previously available only on the Unibus, and in that configuration can be attached to any PDP-11 processor. By using the Maxiram-11/70HS directly on the cache bus, the Unibus traffic can be reduced, the data flow improved, and the throughput increased.

The Maxiram System consists of a high speed controller located in one of the RH70 locations in the processor plus a 19-inch rack-mounted chassis containing the solid-state random access memory. Capacities from 512K bytes to 8 Megabytes can be attached to the high speed controller.

The cost savings and performance benefits are dramatic. The greatly increased throughput is due to the 36-bit wide word transfer directly into main memory without disturbing the cache memory, and the zero latency time. Reliability is also greatly improved and maintenance costs are enormously reduced compared to the equivalent disc units. The system combines the convenience of a peripheral with the reliability of solid-state memory.

Whether attached to the Unibus of any PDP-11 processor, or to the cache bus of the PDP-11/70, typical applications for the Maxiram are: a swapping device, a program overlay storage, scratch files, remote user storage, and any situation requiring frequent access to large blocks of data.
For more information, contact Imperial Technology, Inc., 831 S. Douglas Street, El Segundo, California 90245, or call (213) 679-9501.

— — — — — — — —



June 2, 1980
COMDATA RELEASES 8 CHANNEL RACK MOUNTED MODEM ASSEMBLY. . .

Chicago, Illinois — ComData recently began shipment of it's new model 330-8R Modem Assembly which satisfies the needs of small time-shared computer and multiplexer sites. Up to 8 rack-mounted printed circuit card modems operating at 300, 1200 and 2400 baud, including frequency division multiplex channels, may be included. LED indicators are displayed through a black plexiglass front panel. Two units may be mounted side-by-side in a standard 19 inch cabinet for later expansion.

The new unit measures 9 inches wide, 6 inches high and 17 inches deep. List price is $295.00 and delivery is from stock.

ComData is located at 8115 N. Monticello, Skokie, Illinois 60076. 312—677-3900.

— — — — — — — —

June 19, 1980
NEW DISTRIBUTOR FOR OMSI PASCAL-1 ON DEC MACHINES . . .
RTP, the City based Systems, Software and Bureau house, have signed a UK distributorhsip for OMSI Pascal-1.

In use since 1975, over 600 DEC sites in 22 countries are claimed.

Steve Collins, RTP's Technical Director, said that OMSI Pascal had first been considered last year when planning the implementation of a large Database system.

> "Being RSTS/E based, we needed to make the move to a compile code language, and one that could mirror complex data structures.
>
> The very fast OMSI Pascal execution time and the availability of SYS call equivalents made it an obvious choice. Add to this OMSI Pascal's portability through RSTS/E, RT 11 and RSX operating systems, with target machines LSI to VAX and you have an extremely powerful product.
>
> We always considered Pascal was more than just a teaching language, and a year later, our expectations have been more than realized.
>
> Neat OMSI features like the in-built Debugger and Performance Profiler have enabled our staff to quickly obtain optimum results and implement a large commercial system within months."

RTP also offer the facility on their own bureau for customers to develop and run Pascal programs. In addition, RTP will shortly be providing a "Commercial Computing Pascal" course, and intend to promote other OMSI products — for example a fast SORT for PDP 11 users.

For further information please contact: Steve Collins or Ed Patient at 588-0667.

— — — — — — — —

June 25, 1980
SSI NAMES SOUTHERN CALIFORNIA REP . . .
Fort Lauderdale, Fla. — Southern Systems, Inc. (SSI), computer printer system manufacturer, has named Western Scientific Marketing Inc. as sales representatives for the Southern California area.

Western Scientific is located at 5402 Ruffin Road, San Diego, Calif. 92111, and also has offices at Carson, Calif. According to James W. Rule, SSI vice president/marketing, Western Scientific will handle computer printer sales in the area south of Santa Barbara.

SSI's printer lines encompass medium and high speed impact systems, including the latest in matrix and band technology. The systems are compatible with all processors of DEC, DG, Hewlett Packard and Interdata as well as most processors of Burroughs, TI, Honeywell and SEL.

Southern Systems recently relocated its corporate offices to 2841 Cypress Creek Road, Fort Lauderdale, FL 33309. Telephone is (305) 979-1000.

— — — — — — — —

July 15, 1980
ENHANCEMENTS TO RABBIT-1 JOB REPORTING AND BILLING SYSTEM FOR VAX AND RSTS/E...

West Palm Beach, Florida — RAXCO announces three major enhancements for RABBIT-1, a job accounting session billing and cross-charging system available for VAX/VMS and PDP11/RSTS/E Version 7 users.

All enhancements are upward compatible with previous releases and represents RAXCO's rapid response to requests from users. They are particularly useful to the new users of RABBIT-1.

Feature 1, a new RABBIT-1 System setup procedure prompts the user and assists in the creation of the account file, resource rate table, discount information and report specifications. Once established, these tables and files may be dynamically modified to produce various user reports without the need for reprogramming.

Feature 2 of RABBIT-1 automatically submits a daily job to determine disk file utilization for each user. This information, while available for immediate processing, is "rolled forward" each day for month end processing.

Feature 3 is a single command for complete RABBIT-1 System execution. After completing the initial system setup, all that's required to run report writer is one command. The RABBIT-1 System will then generate all reports specified in the setup phase each time the complete system execution command is invoked.

The RABBIT-1 System is available on a rental or purchase program for as little as $99 per month. A PDP 11/RSX11/m version of RABBIT-1 is scheduled for release the fourth quarter of 1980.

Other RABBIT Systems available include RABBIT-2 and RABBIT-3.

RABBIT-2, Performance Analysis accepts RABBIT-1 data as input producing complete system appraisal information.

RABBIT-3 is a Job Accounting and Performance Monitoring System which creates user data on a session by session basis. The output of RABBIT-3 may be utilized by RABBIT-1 and 2, and by DATATRIEVE.

RAXCO markets a complete line of operational support, financial planning and data management systems for DEC computing equipment.

For more information, contact: Raxco, Inc., 3336 N. Flagler Drive, West Palm Beach, FL 33407, telephone (305) 842-2115.

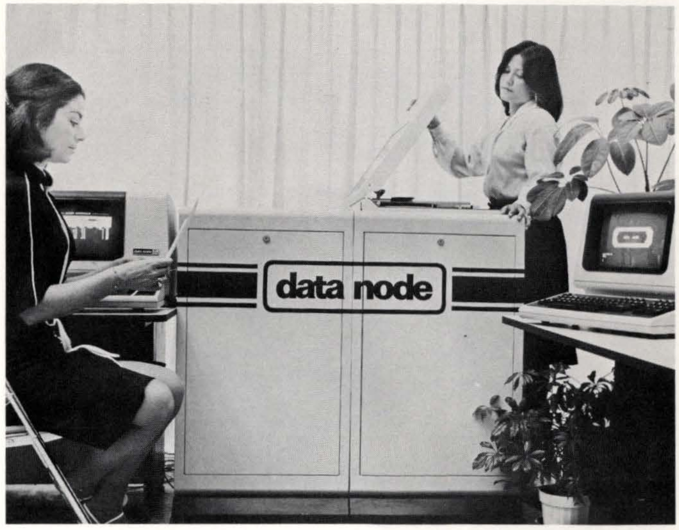
— — — — — — — —

June 15, 1980
DATA NODE, INC.

Sunnyvale, California — Data Node, Inc. is pleased to announce the availability this September, of the Data Node I many-terminal micro computer/mini computer network system.

Designed for the commercial marketplace requiring many-terminal access to a common data base, the Data Node I features very fast response times and competes, in certain vertical markets, with maxi-minis such as DEC 11/70, HP 3000 and the DG Eclipse. Offering significant performance increases at substantially reduced costs, the Data Node I supports up to 60 terminals at three to five times the data based performance range of these traditional maxi-minis.

Traditional computer systems supporting multiple terminals or jobs exhibit a common characteristic of



general purpose machines — they reach a saturation point where the system's management of these jobs, and their attendant paging or swapping, begins to consume more resources than the jobs themselves. It is at this point that response time falls off drastically.

A typical traditional system would have an average job size of 16K words (32K bytes) paging or swapping in and out of the main processor. The processor and system software itself, has traditionally been designed as a jack-of-all trades manager, data handler and computation machine.

Data Node has overcome this traditional bottlenecking by designing a special purpose central data base machine (the node — or Data Node) and combines this central data master, via an ASCII protocol, with microcomputers residing in the terminals, themselves.

Data Node I features a PDP 11/34 with high speed cache memory as its node engine, and 64K byte Z80A microcomputers in otherwise standard DEC VT100 terminals as its Intelligent Node Terminals (INT/200).

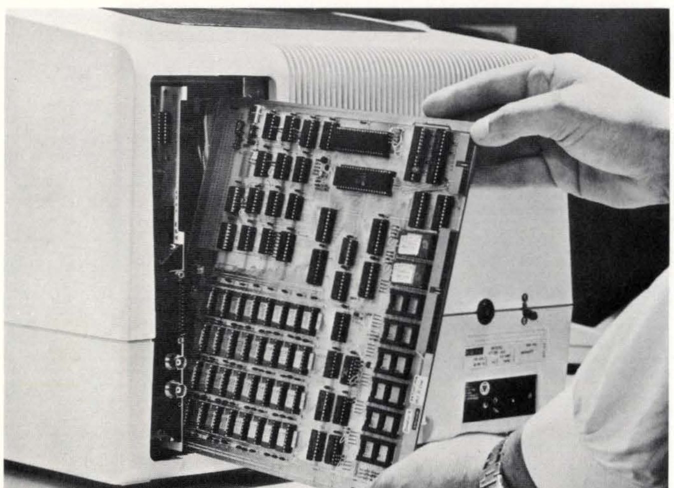
The central node software on the PDP 11 is named Node Central and runs in conjunction with DEC's very popular RSTS/E or Data Systems 500 operating system. Node Central allows the down-line loading of the INT/200 terminals and performs all the data management functions requested by the programs resident in the terminals. This special purpose design of the Data Node system with its offloading of data

entry and inquiry computation onto the INT/200 terminals allows the average job's data I/O buffering in the PDP-11 to remain at 2K words giving an 8-to-1 advantage over traditional systems.

Proprietary designs of data storage and management by Node Central enchance that advantage by such features as file scans at speeds approaching the hardware limits of the disks themselves, and two revolution disk key updates.

The PDP 11 will be retained in future Data Node systems as an Attached Processor as there will always be those non-interactive tasks better suited to running in the more muscular, directly ported main attached processor.

While the INT/200 microcomputer terminal features an extended BASIC as its first language, with built in Node Protocol and screen handling verbs, any microcomputer allowing assembly language subroutines could be readily modified, in software, to act as networked terminals. Options for the INT/200 terminals include local diskette and printer support, and all connections to the network are industry standard RS232 with speeds up to 9600 baud, and higher speeds to larger processors can be accomodated on special order.



Configured with the printer and diskette options, the INT/200 terminal supports the CPM microcomputer operating system for those users wishing to develop stand-alone applications with full time or occasional linkage to the Data Node itself.

The normal office environment was used as a particular design goal in Data Node I development. Over six months went into cabinet design alone with major emphasis placed on attractiveness, quietness, maintainability and compactness. The standard system shown in the accompanying photograph houses the full blown PDP 11/34, two high performance 80 mega byte Winchester disk drives, and a full size industry standard nine track 800/1600 bpi magnetic tape drive. The system is remarkably quiet and very attractive, yet allows complete maintenance access to all components. No special sub flooring or temperature maintenance is necessary, other than normal office requirements, and the use of Winchester disk drives significantly enhances reliability and reduces maintenance costs.

The unique horizontal mounting of the tape drive unit at waist height has received unanimous approval from office personnel in beta test sites for ease of use.

The standard Data Node I system consists of the central system with PDP 11/34, cache memory, 256K bytes of main memory, 16 line microprocessor controlled multiplexer, two 80 mega byte Winchester disk drives and controller, industry standard 45 ips 800/1600 bpi magnetic tape drive and controller, four INT/200 microcomputer terminals, and a 300 line-per-minute printer. The standard Data Node I system as described lists for $98,850 and includes perpetual single use licenses for Node Central, INC/200 BASIC and RSTS/E. Additional INT/200 terminals are available for $3,850 each.

Data Node, Inc. also expects a large initial market in add-on kits to existing DEC RSTS/E installations. These kits consist of the Node Central software which runs concurrent with the standard RSTS/E operating system, and plug-in-and-go microcomputer board to a standard VT100 requires no modification to the terminal and features a keyboard selectable "transparent mode" so the user may switch between use as a standard terminal and the INT/200 mode. Use of this Data Node/RSTS add on kit can allow existing RSTS/E users to significantly offload their systems, increase throughput, and expand their responsive terminal load. PDP 11/34, 11/44, and 11/70 systems can all utilize this add-on kit. The add-on kit is priced at $325 per month for the Node Central software and support, and the INT/200 conversion microcomputer boards and BASIC license at $1,850 per terminal.

Throughout the development of Data Node I and the current development of the several-hundred terminal Data Node II, there has and will be particular attention paid to the sophisticated user/system house customer. All Data Node systems will feature:

1. Medium sized PDP 11 attached processors for non-interactive batch jobs.

2. The highest possible performance within system limits, with the widest possible range of fully expandable systems — from four to eventually, four hundred terminals, and linked Data Node networks.

3. Availability, to the program-in-the terminal, of Node Central's data dictionaries so the sophisticated programmer can develop general purpose table-driven applications.

4. Strict protocol compatible development of central Data Node systems making transparent to the terminal whether its connected to a 4, 40 or 400 terminal system.

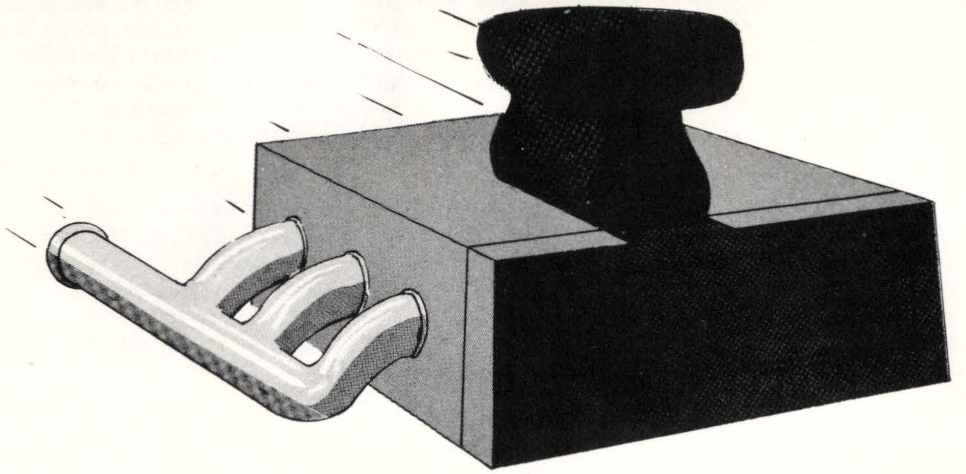
The introduction of Data Node I marks a culmination of development begun in 1974 and initiates a new philosophy of many terminals sharing a common data base, and is the first system allowing microcomputer application developers a many-terminal high performance capability without the attendent high cost, high risk operating system development cycle.

Direct end-user sales by Data Node, Inc. during the first year will be concentrated on the West Coast, but inquires from larger, established systems houses and OEMs are invited.

All inquires should be directed to the Marketing Manager, Data Node, Inc., 432 Toyama Drive, Sunnyvale, Ca. 94086. Inquires should be made on company letterhead and allow two to four weeks for response.

The answer is: RZ KBLQMTSLP MNOZ

RSTS/E SUPER CHARGER



- **ZERO TO 60** data node's add-on kit allows any RSTS/E system to support up to sixty terminals with excellent response times.
- **DROP IN KIT** completely transparent to existing applications, the data node add-on kit includes Node Central software providing data base and network interfacing for RSTS/E, and plug-in-and-go microcomputer boards for your VT100s (or ours).
- **HIGHER PERFORMANCE** you can offload your programs onto the microcomputer in the terminal and experience up to an 8-to-1 reduction in memory requirements per job on the PDP-11.
- **BETTER MILEAGE** the superior data management of Node Central, the super quick data retrieval and sorting,and the offloading of computation onto the microcomputer terminals allows the data node boosted system to achieve transaction rates of as much as 10,000 to 20,000 transactions per hour.
- **LOW COST** the small monthly license fee of Node Central, and the low cost the VT100 microcomputer boards, combined with the much higher transaction rates of the data node boosted system are much more cost effective than the high cost and turmoil of going to a larger system (if you can).
- **TEST DRIVE** our special introductory offer gives you the opportunity to try the data node add-on kit with a 30 day, money back guarantee.

data node

FOR DETAILS on the data node add-on kit, our special introductory offer, and the full blown data node I system, fill out this coupon, clip it to your letterhead, and mail to:

Marketing Manager, Service Products Group
Data Node, inc.
432 Toyama Drive Sunnyvale, Ca. 94086

NAME ____________________ TITLE ____________________

FIRM ____________________ ADDRESS ____________________

CITY ____________________ STATE __________ ZIP __________

Paydirt for VAX and PDP-11 user
One blue-chip card delivers 16 DZ lines per slot.

Key Features

1 SELF TEST
(ON-BOARD LED DISPLAY)

Self test feature is unique. No other DZ has it.

Executes – at every power on.

Identifies – any malfunction and directs attention to the specific area of the DZ board affected. No lights, no problem.

Controls – fault isolation/repair by means of related options.

2 CONFIGURATION CONTROL
(ON-BOARD ADDRESS/VECTOR PENCIL SWITCHES)

Complete configuration control – not matched by other DZ's.

Compatibility – assured with all DEC address/vector/interrupt level disciplines.

Easy Integration – no etch-cuts, no jumpers.

Priority Selection – plug provided just like with DEC.

Automatic Assignment – one setting establishes base address & vector for both logical controllers.

3 STAGGERED LOOP-AROUND
(BUILT-IN MAINTENANCE AID)

Another Able exclusive found only on the DZ/16

Complete Checking – provides the only way to effect total parity/framing error check. Uses one UART to drive another for fault isolation. Alternative internal loop-around gives partial check only.

Diagnostics – support loop-around capability which is built-in to DZ/16 panel. Connectors are built-in. Guess where the other kind are any-time you need them.

Now you can add twice as many DZ lines to your PDP-11 in half the space and at a lower cost than ever before. Our new DZ/16 is a microprocessor-based controller which fits 16 asynchronous communications channels into a single board but sells for much less than the two-board DZ11-E it replaces. There's no waiting either. You'll probably have your card plugged in and running less than 30 days after we get your order.

The unique multiplexer installs in any standard hex-width slot and presents only one load to the Unibus. It supports all DZ11 baud rates, provides modem control on all lines and is compatible with DEC diagnostic and operating system software. The data format is program-selectable for each channel.

This isn't the first time we've been first. It won't be the last. The advantages we've sent your way again and again will keep coming. Get the most out of your VAX or PDP-11. Write today for details on our remarkable line of memory, communications and general-purpose cards for use in the PDP-11 family.

Able, the computer experts

ABLE COMPUTER, 1751 Langley Avenue, Irvine, California 92714. (714) 979-7030. TWX 910-595-1729.

ABLE COMPUTER-EUROPE, 74/76 Northbrook Street, Newbury, Berkshire, England RG13 1AE. (0635) 32125. TELEX 848507 HJULPHG.