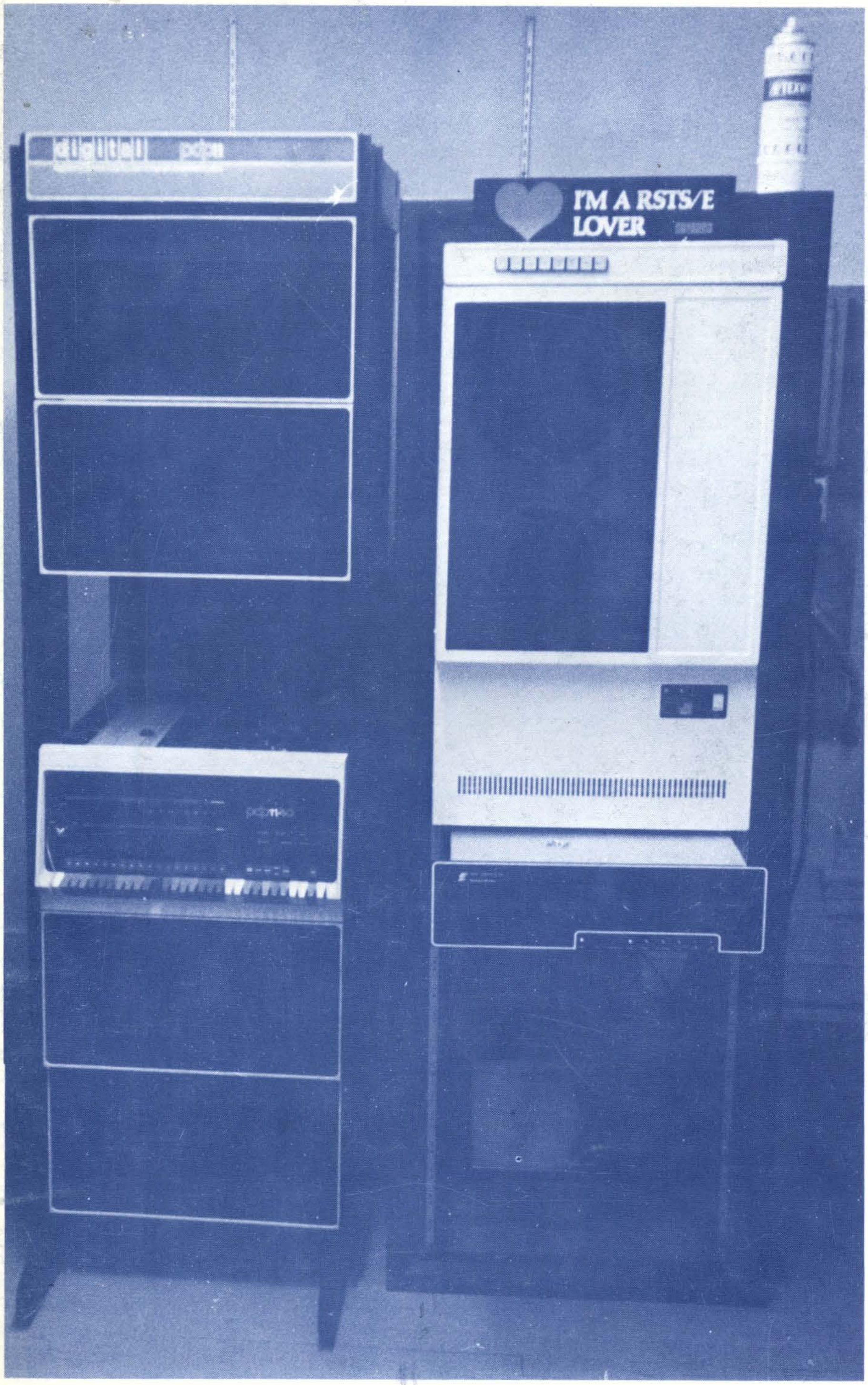
RSTS PROFESSIONAL

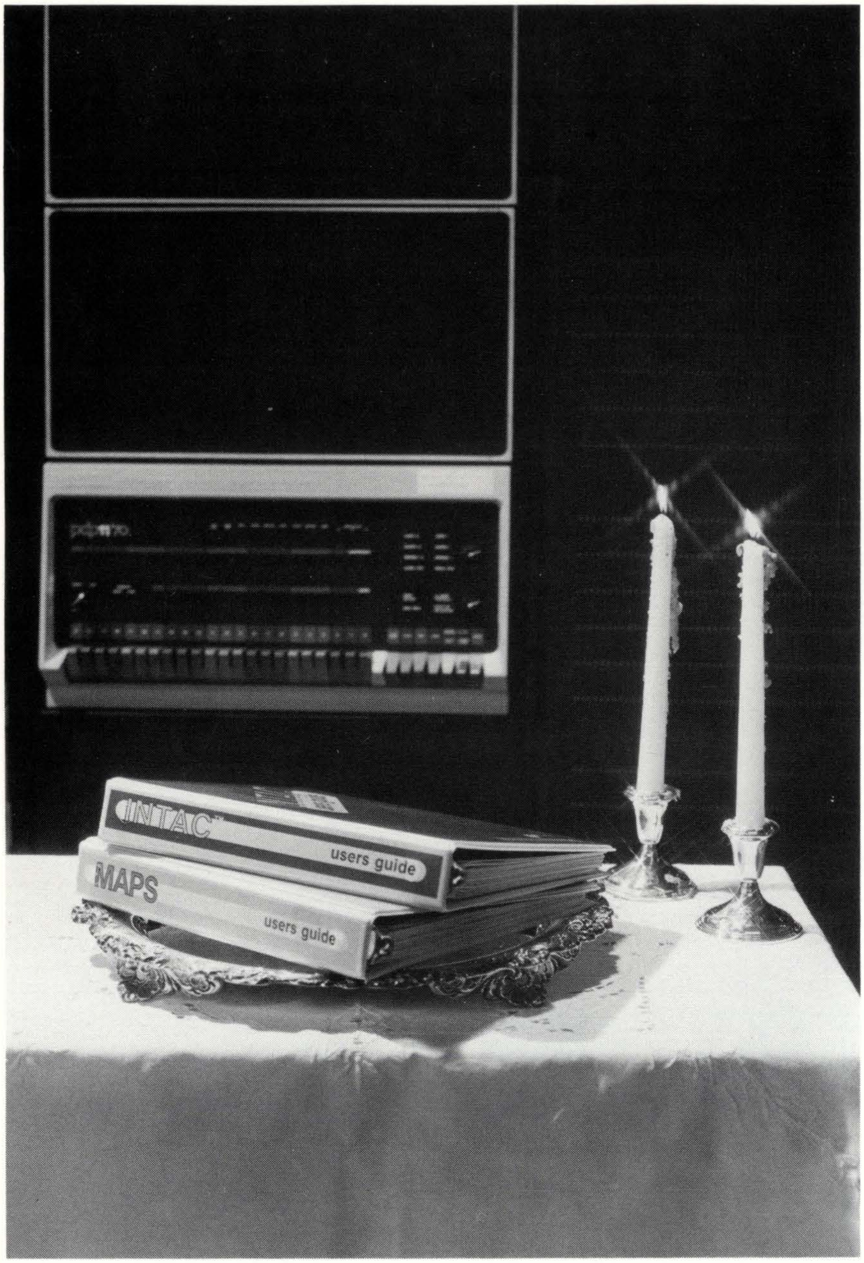
Volume 3, Number 1

March 1981

$7.50/issue, $20.00/year



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 . . .

- Scott Banks on Disk Directories
- Bob "Macro Man" Meyer on How to Macro
- Technical PDP-11 Internals
- Continued RSTS Internals
- The RSTS Pro Installs Their Computer
- Spring Miami Decus Coverage
- RSTS Software Hall of Fame
- VAX X.25 Communications
- Modeling the VAX
- The Doctor Returns
- More . . .

The RSTS Professional Magazine, March 1, 1981, Vol.3, No. 1. 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. . .

In case you haven't noticed, the RSTS PRO has created a thriving new marketplace for software. This was one of our original goals, six issues ago. I would like to single out two of the many products that I have personally bought and paid for. The first is OMSI Sort Version 1.6. I have used OMSI sort since its original release as a macro replacement for SQWIK and MQWIK, the old DEC type-1 header sort. It was the best software buy I ever made, yielding staggering reductions in run time. Version 1.6 contains the final additions to a fine product — the macro rendition of XQWIK and OQWIK, the extract and reordering portions of the sort. Amazingly fast and truly flexible. Bravo!

The second entrant to my 'Hall of Fame' is a package from Software Techniques called DSKIT. I ordered it as soon as I saw the proofs of their ad in the last issue. To make a long story short: last week, using DSKIT, I created 130 accounts and fully extended their centered UFD's in three minutes and forty seconds (a job that used to take four to eight hours). I then copied the full contents of a 300 MB RM05 equivalent to this new 'well structured' disk in 45 minutes, optimizing clustersize and contiguity on the process. I (for once) was speechless. Software Techniques delivered excellent response to the few minor problems I unearthed in the very short learning process. The documentation is excellent. In short, this package is the 'final solution' to structured disks, eliminating all of the time and complexity and reducing the job to one of the simplicity of a SAVRES.

Dave Mallery

Love at First Sight

I'm in LOVE. Head-over-heels in love. I am overcome when I see the twinkle in the darkness, but even in the light I am overjoyed when I come into the room. Our relationship has been short, but I am looking forward to many years of happiness. I look forward to that time of day when I can be alone, just the two of us talking to each other.

Now, before I really get into trouble with Helen, my spouse of some years, let me explain that my New Love is a rather inanimate (? sometimes I wonder ?) Lovely, old, well cared for, rugged, PDP 11/40 that we plan to make over into a New, Well Cared for, New peripheraled, Speeded up, more functional, Great RSTS Machine. That PDP 11/40 was built in 1973 and even has the old style BA-11 Box and real (!) lights and switches — looks like an 11/70 if you don't look too closely. No light show is evident. . . no null job? . . . does any one know why? We don't.

This 11/40 is a real story; how it was bought (and others rejected), how it was delivered, the new memory (256KB on a single board from Data Systems Services), the new disk (160MB Winchester that looks like two RM03's from System Industries), the new Tape (800/1600 of course) also from DSS, the new Multiplexer (A Dmax from Able), the new line printer (Southern Systems), the new cabinet (Surplus RCA), and others good and bad things. We'll tell the story in the the next few issues, don't miss it.

The RDC has invited us to visit and see how they do their remote diagnosis. That story should be ready for the next issue.

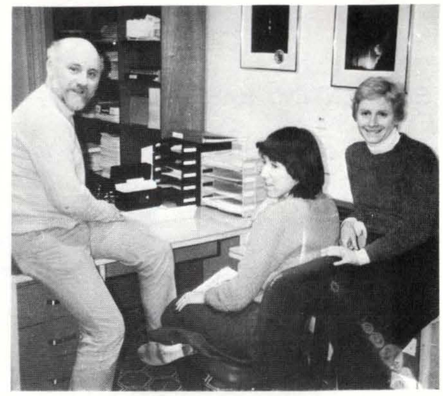
Our VAX-SCENE has been well received and it is significantly expanded in this issue and will continue to get the attention it deserves. Lets hear it from all of the VAX people out there — your doing some neat things, tell us about them.

If you haven't seen the NEW RSTS brochure, ask your salesperson for one. This make RSTS seem like the up and coming product it is! Be sure to look closely at the SYSTAT...What are all those foreign(!) Run-Time systems? I had to go to Canada (see the Canadian DECUS article) to get one, I haven't seen a salesperson in over a year. While you have the DEC rep on the phone ask to see the RSTS, VAX and RSTS/VAX performance data that is available from the performance group in Merrimack; let us know if they can't or won't get it for you — it's got some surprises.

We're well on our way into our second year and it is more: more work, more fun, more people, more magazine. RSTS is a growing product and were going to grow along with it — keep giving us the help we need: more articles, more letters, more of you—tell a friend about RSTS and about the RSTS PROFESSIONAL.

Carl Marbach

RSTS PROFESSIONAL*

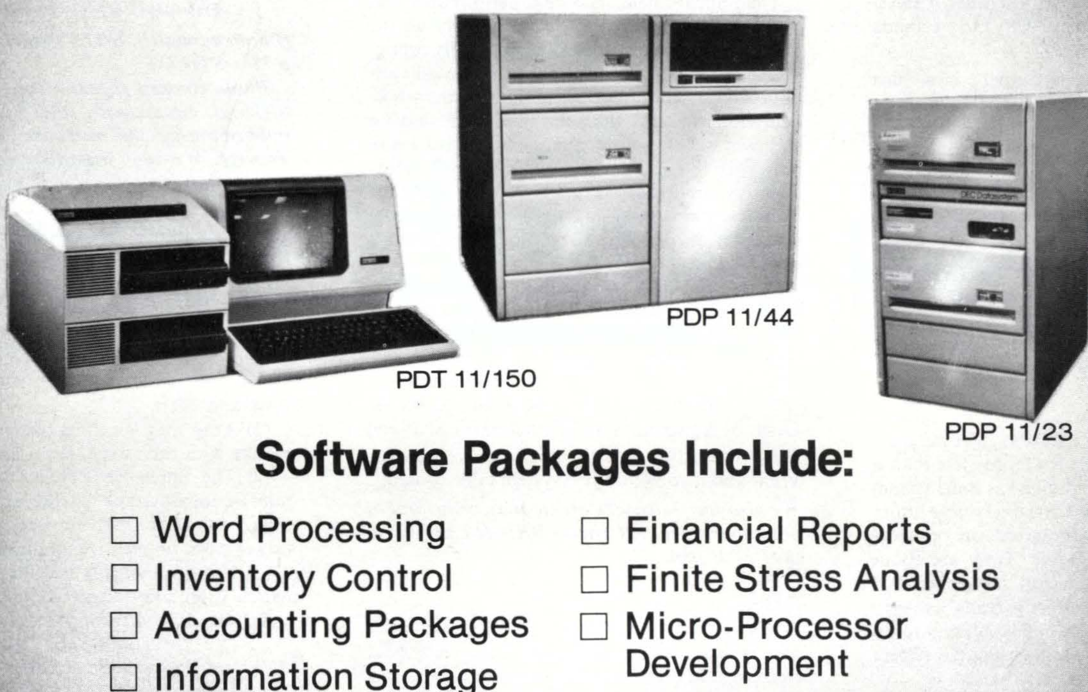


Editorial Information: We will consider for publication all submitted manuscripts and photographs, and welcome your articles, photographs and suggestions. All material will be treated with care, although we cannot be responsible for loss or damage. (Any payment for use of material will be made only upon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

NOW AVAILABLE FOR PROMPT DELIVERY FROM

HAMILTON

DEC PDP 11
FAMILY OF MINI-SYSTEMS



Software Packages Include:

- ☐ Word Processing
- ☐ Inventory Control
- ☐ Accounting Packages
- ☐ Information Storage and Retrieval
- ☐ Financial Reports
- ☐ Finite Stress Analysis
- ☐ Micro-Processor Development
- ☐ Commercial Graphics

RENTAL • SALE • SERVICE • SUPPORT

IMMEDIATE DELIVERY FROM STOCK
ALL RENTALS INCLUDE MAINTENANCE.
PURCHASE OPTION
LEASE PLAN AVAILABLE

HAMILTON
Hamilton Rentals
One Penn Plaza, New York 10119

212-695-1936
TOLL FREE 800-223-2430

LONDON PARIS DÜSSELDORF TORONTO/CALGARY

LETTERS to the RSTS Pro . . .

Dear Dave and Carl,

In your recent report on the U.K. RSTS SIG Meeting you said that it was attended by people "from all over the U.K.".

There were also several participants from Ireland — RSTS is alive and well over here too.

Keep up the good work. Kind regards.

Yours sincerely,
David A. Reynolds
Peat Marwick Mitchell - Dublin

—

Dear Sir:

In your editorial "Go VAX Young Man!" [*RSTS Professional,* Vol. 2, #3], you mention that a product which allows the PDP 11/70 clock rate to be raised has appeared. I am interested in such a product, but have been unable to find any information on it.

Would you please send me the names and addresses of those who are marketing this product.

Yours very truly,
N. McRae
Computer Facilities Analyst
Hardy Associates (1978) Ltd., Canada

Dear Mr. Mallery:

Your article indicated that there is a "fast clock" available for the DEC PDP 11/70.

I would appreciate it if you could tell me the manufacturer so that we can evaluate it as a means to increase our capacity.

H.J. Mainwaring
Staff Engineer, Computer Systems
Cadillac Motor Car Division

You can obtain a modified TIG board for the 70 from: TIG Board, Nassau Systems, P.O. Box 19329, Cincinnati, OH 45219.

This is currently being tested by a major 70 user and the results should be in soon.

—

Dear Sirs,

I have been managing a PDP11/55 time sharing system running under RSTS for less than a year, and when I started it looked as if the system would need to be enhanced in the very near future as the response times to the users was on occasions completely unacceptable. There are about 120 accounts on our system containing over 9000 files and although some effort is being made to reduce these numbers, it has been difficult to do this since the number of people using the system is large, and the job mix is very varied.

It was with great interest, therefore, that I attended the seminar at the Festival Hall, and I returned full of ideas and enthusiasm. Right, I would rebuild the system disk with nice contiguous directories. Then the trouble started. Did I have to run REACT to generate 124 accounts. No need. System functions are available to write a simple program to establish these the same as the existing disk. Now it is a bit of a fiddle to get the right size of file to fill the lower disk. This is not too difficult, however, so it is done and I start opening null files to fill up all the directories. About three in the morining I give up. It's only half done and it will take all night. I realize that since our system requires new accounts to be installed fairly frequently that it is not a once only job.

Thinking again I realize that a UFD fully extended with one null file is a fairly simple structure so why not build all the UFDs into a large contiguous file and patch the DCN of each into the MFD. The file into which the directories are built is in [1,0] and it may be removed after the build by changing the link. This does not clear the SATT so that the directories are protected. Surprisingly, apart from a minor bug in the alignment of clustersize 8 directories (the DCNs must be odd numbers) this worked first time and a disk contianing all the accounts with seven cluster directories was built from an initialized disk in about 12 minutes.

Placed files were then transferred using PIP followed by a wildcard PIP of everything else. Unfortunately, since our system uses new files first all the directories were reversed, and a time consuming REORDR was required.

The next time the disk was copied the new disk was initialized old files first, and the program INVERT was run to change it after the copy was complete. This saves the REORDR time and builds a better directory. Since the control file started the system after the copy to allow the copy to run unattended at night, the program INVERT must tell the monitor that the change has been made because the status bit on the disk is only looked at during the mount operation.

The improvement in system performance was enormous. Not only the directories but all the other files on the new disk were physically contiguous (or nearly so) and the disk access was improved by about 50%. The time for the whole operation was less than the BACKUP that we had used weekly before, and the backup disk was fully runnable in the event of a disaster. I subsequently found that the retrieval of a single file from the old disk (using PIP) became a trivial operation as opposed to getting a file from a BACKUP set.

I am enclosing a description of the process and listings of the programs involved for your information and possible publication as I think it may be useful to other users with similar problems.

Yours faithfully,
Michael J.D. Mowat, B.Sc., M.B.C.S.
Dept. of Agriculture and Fisheries for Scotland

P.S. Your articles on directories are very useful. What about something on system table contents.

We also feel Michael's article may be helpful to other users. See "All Things BRIGHT and Beautiful", this issue.

—

Editors,

I have been a subscriber to your magazine for over a year and have found much enjoyment from reading the *RSTS Professional.* It has proven to be an invaluable source of information on the RSTS Operating System and TECO.

Sincerely,
John J. Walczy, Royal Oak, MI

—

Dear Mrs. Noakes,

Following your request for contributions to the *RSTS Professional,* please find enclosed listings and documentation for half a dozen user subroutines callable from Basic-Plus-2.

I hope someone may be able to make use of them — either they provide functions not available or are very much faster than those provided. The execution time given is for an 11/34A processor.

May I add my thanks and praise for last month's DECUS Commercial SIG meeting which I found interesting and useful [that was the "Dave & Carl Show" at Festival Hall].

Yours sincerely,
M. A. Jackson, Nielsen Business Services

Readers will find Mr. Jackson's subroutines in this issue, "PDP/11 — UTILITIES."

Dear RSTS Professional:

I would like to take a moment to thank the people who have written the excellent articles for the "RSTS Pro". I'm sure their efforts have been of great use to many people. I would like to especially thank Scott Banks, Steve Davis, Steve Edwards and your own Dave Mallery and Carl Marbach.

I feel we in the RSTS community have a responsibility to provide our peers with this type of information and I applaude those who have already done so. For my own part I am starting a year long series of articles on monitor internals. I haven't seen anybody do anything like this yet so I hope it will be helpful. A copy of the first article is enclosed.

Sincerely,
Mike Mayfield
Northwest Digital Software

We all thank you, Mike, for your appreciation. We accept your offer for a series of articles on monitor internals, the first of which appears in this issue.

Readers: Mr. Mayfield's article is titled, "RSTS/E Monitor Internals, Part 1."

—

DO YOU REMEMBER THIS?

(Photo contest(?), RSTS Professional Vol. 2, #3, p.75. - STILL!)

Photo contests appear in the RSTS PROFESSIONAL occasionally and readers 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.

Because no one has gotten this right yet, we'll save the answer 'til the next issue. Following are the latest silly attempts.

"Tampa Elec. Co. truck."

Jeffrey Neu, New York, NY

Wrong!

—

Boys and Girls,

(1) Your mag is getting better and more informative with each issue. Keep up the goodies.

(2) The unresolved TECO "what is it" is a utility-company-type "Cherry Picker", commonly found somewhere near the top of a pole (lower case, no creative ethnic slur intended).

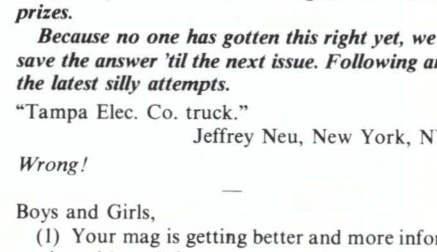
(3) I've enclosed my mailing label. Counting issues, I think my subscription must be about to expire but I need further clues.

Bye/F, Douglas P. Herman
Herman Management Company, Inc.
El Cajon, CA 92021

(1) Thank you. (However, flattery won't get you a T-shirt - usually!).
(2) How TECO, Why TECO remains unresolved.
(3) A clue follows.
(4)

—

Dave, Two of them, even (whatever they are . . .)



—Bill, Merrimack, N.H.

Good Grief! They're Even Contemptibly Omnipresent.

. . . continued on page 95

Segmenting BASIC-PLUS-2 Applications

By Al Cini, Computer Methods Corporation

This is first in a series of articles on analysis and design of BASIC-PLUS-2 Applications.

INTRODUCTION

The advantages of a "modular" (many parts) over a "monolithic" (all-one-piece) software system are, these days, a matter of conventional wisdom. Most of us somehow know we write modular programs, but few of us stop to reflect on how or why we do.

Modular software system concepts are adapted from mechanical and electronic engineering notions, whose practitioners learned a long time ago how expensive "one big thing" can be to build, repair, and modify. In fact, while large-scale integration gets the credit for the third generation of computers, its the shrewd packaging of these chips into **functionally independent** components, and the creative assembly of these components into ever-expanding variety of processors and peripherals, that have made computing the boom industry it is.

It's largely the same in software. Functionally independent modules allow us to work separately on different elements of a project, with confidence that these conceptual "parts" will work properly together when ultimately assembled. Throughout all of engineering, a complete and cohesive design and faithful adherence to its rules during implementation go a long way toward taking the worry out of laying that last brick, soldering that last chip, and taskbuilding that last program. Having to tear up a routine because it doesn't "fit" into a system isn't any less exasperating than smoking a printed circuit board in a poorly designed backplane, or cracking the block of an expensive test engine—anticipating the connections, strains, and pressures among the parts assures the correct integration of the whole, and avoids costly trips back to the drawing board.

UNDERSTANDING MODULES AND MODULARITY

In programming, a **module** is a group of program statements with the following characteristics (Myers, p. 11):

1. The statements are contiguous (i.e., appear together in the program listing).
2. The statements are bounded by identifiable delimiters (such as SUB/SUBEND, DEF/FNEND statements; note that in the case of GOSUB, these boundaries can be very vague).
3. The statements are collectively referenced by a name (or statement number).
4. The statements can be referenced, by the module name, from any other part of the program (a recursive module can even call itself).

The term "module" usually evokes thoughts of external subprograms (PL/1 procedures, FORTRAN subroutines), but its broad definition includes internally contained code sections (COBOL paragraphs, BASIC-PLUS functions and GOSUB-type subroutines) as well as complete concurrently or serially executed programs. Logically, software modules are the realization of our ability to analyze a big problem into manageable parts. In practical terms, software modules determine the feasibility of our work: we arrange them to promote system reliability, simplify maintenance delegate coding responsibility, and optimize memory utilization.

DETERMINANTS OF "GOOD" SOFTWARE MODULES

Before we go any further, I'd like to apologize for the inevitable use of the word "structure" in the text which follows. To say that "structure" in our business is an overworked word is to say that France is a place where a lot of French people live. Introduced formally during the late sixties, "structure" as it modified "programming" identified a coding discipline in which procedural programming languages (RPG isn't one of these; COBOL, BASIC, and FORTRAN are) could be organized into programs using combinations of three fundamental control mechanisms: **sequence** (one statement after another), **selection** (IF-THEN-ELSE), and **iteration** (DO-UNTIL, WHILE-NEXT). Structured programming stirred up some pseudo-controversy (GO TO statements, as it happens, are not included in the basic control devices), got lots of trade press attention, and spawned a host of "structured" disciplines as well as whole libraries of pamphlets and books on each. (A discussion of structured BASIC-PLUS and BASIC-PLUS-2 programs can be found in RP vol. 1 no. 1). "Structure" has become an industry buzzword which raises hackles, provokes debate, and unfortunately has come to have little useful meaning.

For our purposes, we can define "structure" along the lines of its non-DP usage: a structure is a complex entity consisting of discrete but interrelated parts. The parts can be isolated and examined independently and, frequently, are themselves structures of smaller parts. The smallest possible parts in programming are sequences, which may appear within selection or iteration control mechanisms. Modules are composed of such elements, and combine with other modules to make programs and systems.

The "goodness" of a module can be evaluated along two dimensions (after Stevens et. al.): **cohesion** and **coupling** (figure 1).

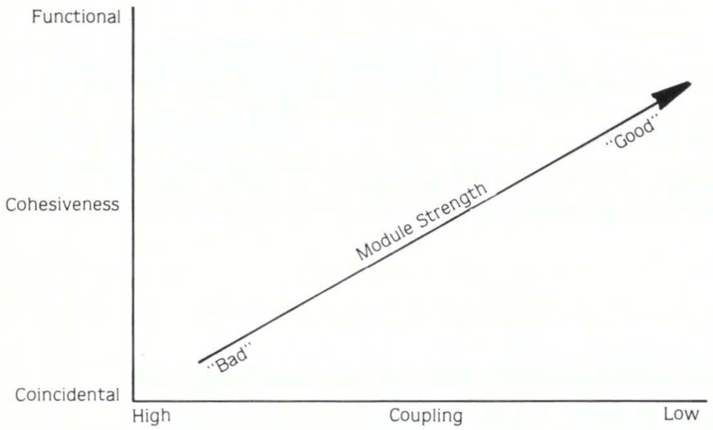


FIGURE 1. Dimensions of Module Strength

Cohesion relates to the strength of the relationship among the elements which comprise a module with respect to the function a module performs. A highly cohesive module is one in which the elements work together to contribute to the execution of a specific function; an incohesive module is one in which the elements are assembled at random. As you might imagine, there are strata of modular cohesiveness along the continuum:

1. Coincidental. When there is no functional relationship among the elements in a module, we can say that the module is coincidentally cohesive. FORTRAN or BASIC-PLUS-2 programs which are too large to fit in memory can sometimes be "hacked" into overlays by randomly assigning program parts to subroutines; such subroutines are coincidentally cohesive modules.
2. Logical. Modules whose components are similar with respect to the work they do, but which do not contribute to the execution of a specific function, are logically cohesive. For example, including all READ statements in a READ subroutine, or building a subroutine which edits all social security, phone, and zip code numbers, are instances of logical modularity. "ON ERROR GO TO 19000" essentially invokes a logical module whose job is to "handle all errors."
3. Temporal. Like logically bound modules, temporally bound modules consist of elements which perform similar services. In addition, however, such elements in temporally cohesive modules are related by the time when their services are called for within a program. Housekeeping procedures like "initialization", "termination", and "clean-up" routines are examples of temporally cohesive modules.
4. Functional. A functionally cohesive module is one in which all constituent elements contribute to the execution of a single function. The function, further, is keyed to an operation within the overall application; it is not a computer function (read, or write), but an application function (accept telephone number, compute overtime pay). Functional cohesion is the strongest form of intramodular relationship.

In discussing functional modules, analogies with mathematical functions are frequently drawn. In mathematics, "Y = F(X)" represents a mapping of a domain of parametric inputs (X) onto a range of outputs (Y) according to a set of transformation rules (F). Likewise, a computer system transforms a set of input data into a required set of output data via a precisely defined series of procedures. A **functional module** is one which performs one specific function against a set of input data, producing a precisely determined set of output data.

Coupling describes the devices by which a calling program communicates with a called module. In general, **low** coupling—in which the connection is minimal and isolated—is better than **high** coupling—in which the connection is complicated and diffuse.

The discrete argument list (such as that between a calling program and a separately compiled FORTRAN or BASIC-PLUS-2 subprogram) represents the lowest form of coupling mechanism, because it provides a simple interface between program components which is clearly identifiable in the listings. A functional module which communicates via an argument list can be removed from a system and tested independently without any special consideration.

Named COMMON areas (available in FORTRAN and BASIC-PLUS-2 environments) offer a more complicated interface (the relationship between a COM or MAP and its host program/called subprograms is not clear in the listings), but implementation problems like limits on the size of argument lists make them necessary.

Blank COMMON areas are treacherously easy ways of hooking two programs together in a hurry, but extremely complicated to sort out during maintenance. They should be avoided.

Data reference coupling—when a module directly references a data element within its calling program without going through the argument list—is the only way to connect a calling BP2 program with an internal function or GOSUB subroutine (in DEF/FNEND, the argument list, which is read-only within the function, is of limited value).

In this case, a "conceptual" list of data items passed between caller and called must be kept in the programmer's head and documented in comments, since it doesn't appear in the listing.

Code coupling—when a module executes statements in its caller or vice-versa except through the normal call/return devices—is the highest form of coupling and the source of the worst kinds of maintenance headaches. GOSUB subroutines and COBOL paragraphs chould be clearly delimited within the program, and should NEVER branch outside their boundaries.

STRUCTURED DESIGN.

There are several approaches to "structured design" in the literature these days, all of which offer a different way of representing the relationship among calling and called modules in a system. We can represent a "hierarchy" of functions and sub-functions pictorially using a hierarchy chart (figure 2). In such a chart, calling programs are arranged in super-ordinate positions with respect to called sub-programs, and the functional "parts explosion" of an application becomes evident.

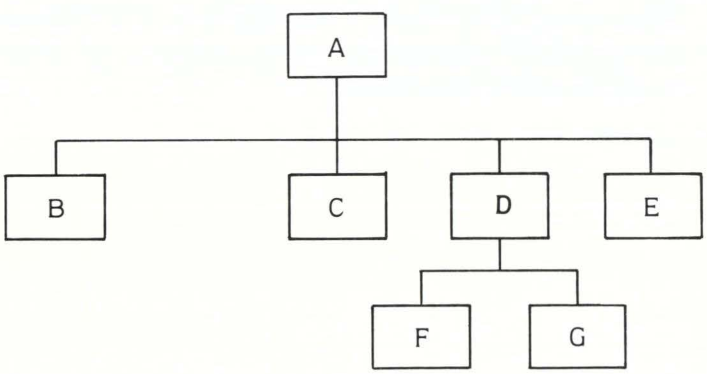
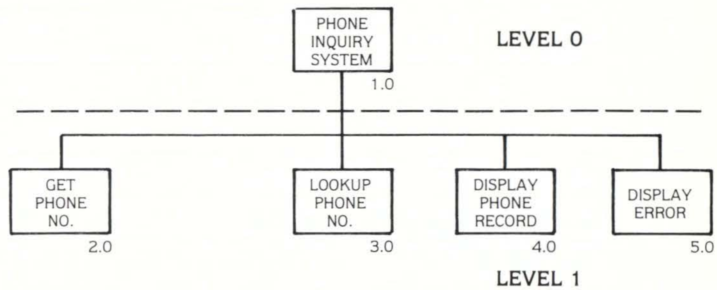


FIGURE 2. Hierarchy Chart

For example, let's build an interactive display program which accepts a telephone number from a CRT, then displays either the name/address of the telephone owner or a "no such phone number" message (figure 3).

The hierarchy chart for our hypothetical phone inquiry system shows that we've refined the overall phone inquiry job into four sub-functions: accept a phone number from the terminal; look for the phone number in a file; display phone information; and display error message. Each of these "modules" represents an independently programmable/testable function. The numbering scheme in the hierarchy chart is patterned after IBM "HIPO" documentation standards, and may be replaced by program names or any other unique identifier.

The Get Phone Number module (2.0) accepts no arguments from its caller (in fact, certain environment-specific arguments, such as logical units numbers, might need to be passed—we've omitted them from this functional diagram) and returns either a collected phone number or a flag which signals that the operator is finished. The "finish" flag can be set by typing escape or CTRL/Z or a null phone number; in any case, it's important to note that the decision about what to do when the flag is raised is the responsibility of the caller (1.0), which will execute the functions on level 1 iteratively until then (see the program listings in figure 4).



ARGUMENTS:

IN	OUT
2.0 —	PHONE NUMBER, ALL FINISHED FLAG
3.0 PHONE NUMBER	PHONE RECORD*, ERROR FLAG
4.0 PHONE RECORD*	—
5.0 ERROR MESSAGE	—

***NOTE: This argument is passed via a named common (i.e. MAP) area.**

FIGURE 3. Phone Inquiry System Hierarchy Chart

The Lookup Phone Number module (3.0) accepts a phone number from the caller, and looks for it in a phone number file (this is just a GET statement with some error handling using BP2 and RMS). If it finds it, it returns a record containing information about the phone number account. If not, it raises an error flag. Again, it is the responsibility of module 1.0 to interpret and deal with the error flag. In this case, the caller decides whether to call module 4.0 (Display the Record) or 5.0 (Display Error) based on information returned by 3.0.

Inspecting the listings, you'll find that 2.0, 4.0, and 5.0 call screen formatting functions which do not appear on the hierarchy chart. Screen formatting routines, and other utility service routines which are frequently called at the lower hierarchy levels are really separate "support hierarchies" which can be documented separately (RMS routines are a good example of a support hierarchy; such routines are excellent candidates for overlay co-trees,

```
10      ! MODULE 1.0 - PHONE INQUIRY SYSTEM
1000    <OPEN FILES>
1500    MAP (PHONE) PH.PHONE.NO$ = 10%, PH.NAME$ = 40%
2000    CALL GETPHN (PHONE.NO$, ALL.FINISHED.FLAG%)             &
\       UNTIL ALL.FINISHED.FLAG%                                &
\               CALL LOOKUP (PH.PHONE.NO$. ERROR.FLAG%)         &
\               IF ERROR.FLAG% THEN
                        CALL ERRDIS (ERT$(ERROR.FLAG%))         &
                ELSE                                            &
                        CALL PHNDIS                             &
!               ENDIF.                                          &
!
2090            CALL GETPHN (PHONE.NO$, ALL.FINISHED.FLAG%)     &
\       NEXT                                                    &
!
32767   END                                                     &
!


10      SUB GETPHN (PHONE.NO$, ALL.DONE.FLAG%)                  &
!
1000    CALL CRTPUT (1%, 10%, "PHONE#:")                        &
\       CALL CRTGET (PHONE.NO$, 10%)                            &
\       ALL.DONE.FLAG% = PHONE.NO$ = CHR$(27%)                  &
\       PHONE.NO$ = CVT$$(PHONE.NO$, 4%)                        &
\       SUBEXIT                                                 &
!
32767   SUBEND                                                  &
!


10      SUB LOOKUP (PHONE.NO$, ERR.FLAG%)                       &
!
1000    ON ERROR GO TO 1090                                     &
\       GET #1%, KEY #0% EQ PHONE.NO$                           &
\       ERR.FLAG% = 0%                                          &
\       GO TO 1099                                              &
!
1090    ERR.FLAG% = ERR                                         &
\       RESUME 1099                                             &
!
1099    ON ERROR GO TO 0                                        &
\       SUBEXIT                                                 &
!
32767   SUBEND                                                  &
!


10      SUB PHNDIS                                              &
!
1500    MAP (PHONE) PH.PHONE.NO$ = 10%, PH.NAME$ = 40%          &
!
2000    CALL CRTPUT (5%, 10%, "PHONE#" + PH.PHONE.NO$)          &
\       CALL CRTPUT (6%, 10%, "OWNER:" + PH.NAME$)              &
\       SUBEXIT                                                 &
!
32767   SUBEND                                                  &
!


10      SUB ERRDIS (ERR.MSG$)                                   &
!
1000    CALL CRTPUT (5%, 10%, "ERROR:" + ERR.MSG$)              &
\       SLEEP 2%                                                &
\       SUBEXIT                                                 &
!
32767   SUBEND                                                  &
!
```

(Note the CRTPUT subroutine accepts a line and column CRT cursor position, and a text string to display. The CRTGET subroutine accepts into a character string variable a field of specified length from the CRT).

FIGURE 4. Program Listings

which is a topic for a future article). The listings also show that the contents of any flag variables are never assumed; that is, all flags are explicitly set or cleared within the module whose function determines the value of the flag. A flag variable should never be assumed to be initially zero/non-zero, and it should never be necessary to initialize a flag you expect back from a subprogram before calling the subprogram.

THE EFFECTS OF FUNCTIONAL MODULARITY ON SYSTEM CHARACTERISTICS.

A system which is refined into functional modules is less likely to fail during integration testing (since the integration testing is built into the design), more readily adapted as user needs change (the functions of each module don't chnage, but new ones may be added and called from superordinate modules to handle new requests), and more readily repaired after a failure (fault isolation is keyed directly to the function which is performed incorrectly, and the failing software module can be removed and tested/debugged independently of the rest of the system). In addition, functionally modular systems tend to be more available (unrelated parts of the system can be kept in service while other parts are repaired) and efficient (the hierarchy chart is a different way of expressing the ODL file; overlays are a design product rather than an after-the-fact hatchet job).

REFERENCES

Glenford J. Meyers, **Reliable Software through Composite Design** (New York: Petrocelli/Charter, 1975).

W.P. Stevens, G.J. Myers, and L.L. Constantine, "Structured Design," IBM Systems Journal, Vol. 13, No. 2 (May 1974), pp. 115-139.

HIPO — A Design and Documentation Technique, IBM, GC20-1851-1, 1975.

Don't connect remote terminals to your Unibus*-based computer

UNTIL YOU'VE READ ABOUT DCA'S SYSTEM 205 UNIBUS-INTERFACE STATISTICAL MULTIPLEXOR.

A DZ11 emulator and stat mux in one.
That's what DCA's System 205 gives you. In fact, a single DCA 205 acts as a host-end multiplexor for PDP-11, VAX or 2020 computers while eliminating up to 16 DZ11 modules!

Supports up to 128 remote terminals.
With a 205 and another DCA statistical multiplexor at the remote end, you can support up to 128 remote asynchronous terminals. Any DZ11's already in use for remote-site service can be transferred to local service.

An easily-affordable point-to-point datacomm network.

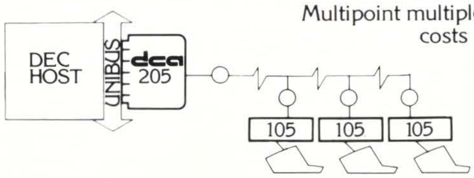
Requires just a single Unibus slot.
That's right, the 205 only uses one Unibus hex slot, freeing up from three to fifteen additional slots. Costly and unsightly cables are eliminated, too.

Superb response time.
The 205 is, in effect, a DZ11 and statistical multiplexor in one integrated unit. So the character delay that normally occurs between a DZ11 and a multiplexor is eliminated. Result: Your terminal users will enjoy a crisper echo.

Statistical multiplexing.
Our 205 gives you all the benefits of DCA's statistical multiplexing, including full ARQ error control that maintains data integrity, network transparency, reduced CPU overhead — and more.

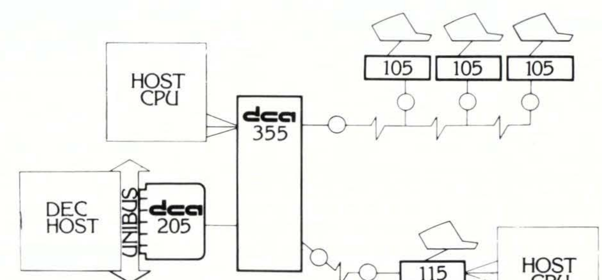
Multipoint multiplexing.
This isn't a promise, it's a fact: DCA's customers have been multidropping several remote terminal locations to a single telephone line since 1979. And multipoint multiplexing with the 205 lets you enjoy the lowest cost communications possible from Unibus-based DEC computers.

Multipoint multiplexing lowers costs dramatically.



Reliable, proven performance.
Like our other statistical multiplexors, the 205 is a field-proven, reliable system. In 1974, DCA introduced statistical multiplexing — we've been the technological leader ever since!

Unparalleled growth potential.
For networks that are bound to grow, DCA is the clear-cut choice. Because complete system compatibility lets you start with a 205 today, and grow into a larger, more powerful network tomorrow. And DCA users enjoy the lowest-cost network growth in the industry!



Any terminal user can access any host in the network.

Excellent cost efficiency.
There is no better value for DEC users — the 205 will dramatically reduce your datacomm costs while providing high throughput and reliable operation. Interested? Send in the coupon or call DCA at 404/448-1400 today.

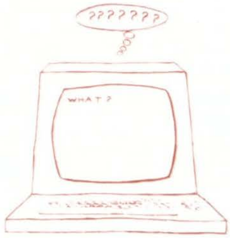
☐ Send me more information on DCA's System 205.
☐ Please have a sales representative call.

Name ____________
Title ____________
Company ____________
Address ____________
City ________ State ________ Zip ________
Telephone ____________
(Area) (Number) (Ext.)

dca
ENGINEERED TO EXPAND

Digital Communications Associates, Inc.
Dept. S-2 135 Technology Park/Atlanta
Norcross, GA 30092

DEAR RSTS MAN:



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

DEAR RSTS MAN: Perhaps you can explain the whys and wherefores of the drop/regain temporary privileges SYS call particularly as they are used by $QUE and $ATPK when both are protected <232>.

I have had 2 problems — one with $QUE, the other with $ATPK.

$QUE — Trying to queue a job from a program using the method given in the Systems User Guide [QUE.11 User's Guide, *RSTS Professional*, Vol. 2, # 4], QUE "bombed" with a "Protection Violation at line 1040" error message. The particular line was the one opening $QUEUE.SYS for read access. That statement was preceeded by the SYS call trying to "regain" its temporarily dropped privileges. $QUEUE.SYS had a protection code of <60> at the time but why does that make a difference to a program with privileges. — The error did ***not** happen* if my calling program was itself privileged.

$ATPK — Trying to perform the contents of an indirect command file using $ATPK. I also chained to it from a user program. Its code causes it to drop its privileges temporarily — read Monitor tables I and II — regain its privileges and perform several PEEK functions. Of course, it too died when executing the first PEEK even though it had allegedly regained its privileges. However, it works if called as a CCL and also works if the calling program is privileged.

I have performed some of these tests on two other machines — one running Release 7.0 and one running Release 6C — results were the *same* on all machines.

What's going on — HELP!

Mike Farrell

Dear Mike: *Old RSTS adage: "When you chain from non-privilege to a privileged program at a line number, the privileged program permanently drops privilege." Be glad that it does! That's why CCLs have ;PRIV in their definitions.*

DEAR RSTS MAN: Regarding your answer to 'Frozen Solid,' I've had the same problem of two programs in DIBOL UPDATE mode locking each other—and themselves—out of a record. I work on a PDP-11/70 using 6C; the friend who introduced me to *RSTS PROFESSIONAL* has also experienced the problem under 7.0.

The solution you gave may work in BASIC-PLUS, but it fails in DIBOL. In DIBOL, one may access records that cross block or cluster boundaries, which means that a lock on one record may in fact lock two blocks. The second program may access the first block properly but find the second locked; when it goes back to get the record again, it finds the first block locked—by its read the first time through.

The addition of one line of code, however, frees things up. The following is our standard solution to the problem (minus our own external subroutines, which add nothing to the understanding of the situation):

```
          GOTO GTREC1                 ;JUMP OVER ERROR ROUTINE AND GET RECORD
GTREC0,   OFFERROR                    ;TURN OFF THE ERROR TRAP
          XCALL ERROR (ERN,ERL)       ;WHAT ERROR OCCURRED?
          IF (ERN.NE.40) GOTO BOMB    ;PERMIT ONLY 'RECORD LOCKED'
          UNLOCK CHAN                 ;UNLOCK THE RECORD AFTER ATTEMPTED READ,
                                      ;   THUS PERMITTING ACCESS ON NEXT TRY
          SLEEP TIME                  ;ADJUST TIME DEPENDING ON THE FILE AND
                                      ;   THE FUNCTION — WE USUALLY USE 1 SEC
                                      ;   — AND LOOP BACK TO THE READ
GTREC1,   ONERROR GTREC0              ;SET THE ERROR TRAP
          READ (CHAN,RCRD,NDX)        ;RANDOM-ACCESS READ FOR RECORD NO. NDX
          OFFERROR                    ;TURN OFF THE ERROR TRAP AND PROCEED
```

Obviously, the 'GTREC0' error handling routine can be modified to deal with any other errors the user may want to permit. 'BOMB', of course, is a bomb-out routine that would close any open files, report the error, and get out, leaving some poor programmer to try to figure out what went wrong.

We've found that adding the UNLOCK CHAN instruction handles the problem adequately in a number of DIBOL shops, even in the presence of the phantom priority booster.

What it doesn't handle is another DIBOL problem involving the UPDATE mode. In several of our relative files, we've been able to READ the final record successfully but have been unable to WRITE it back. We suspect the problem occurs when the final record coincides with a cluster boundary, but haven't investigated it further; as standard practice, we simply create an extra record in every relative file and never use it. Do you know why this happens, and is there any more elegant solution than our very practical one?

Very truly yours, Phil Anthony, Programmer, Joy Shops Inc.

Readers, DEC??

Points of Interest

By Wilf Forrow, Schroder Life Group, Portsmouth PO1 2AW

TERMINAL PRINTERS

RSTS normally outputs data quite happily to a powered down terminal. This is a problem for a printer such as an LA120, especially if the spooler deletes the file thinking it was printed O.K. Does anyone know how to prevent this other than by remembering to type control/S before powering the printer down?

MAGNETIC TAPE DENSITIES

Who thinks 1600 bit per inch is risky for secure tapes? I used to, but I have changed my mind. The U.S. National Bureau of Standards recommend that 800 bit per inch be avoided as this is the least reliable of all current densities. (Presumably, 1600 bpi is safer because of the different recording method). I still get some mag tape errors in my error log, but at least a full RM03 back up fits on two tapes instead of three and runs considerably faster.

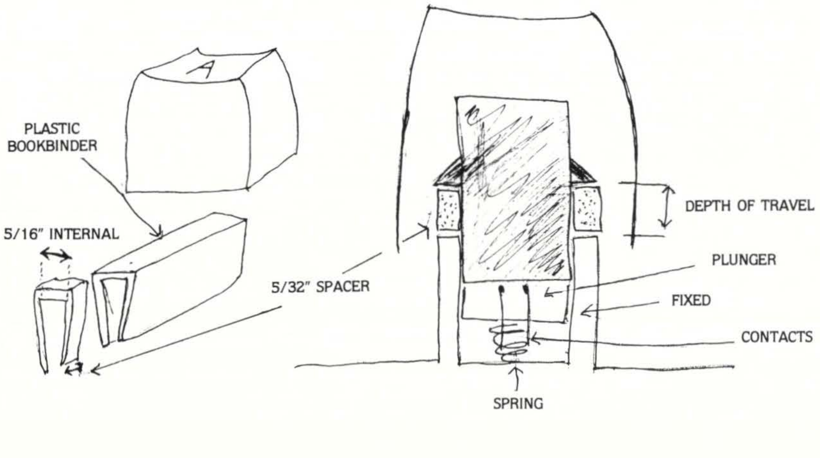
The bureau also recommends for long term storage without loss of data:—

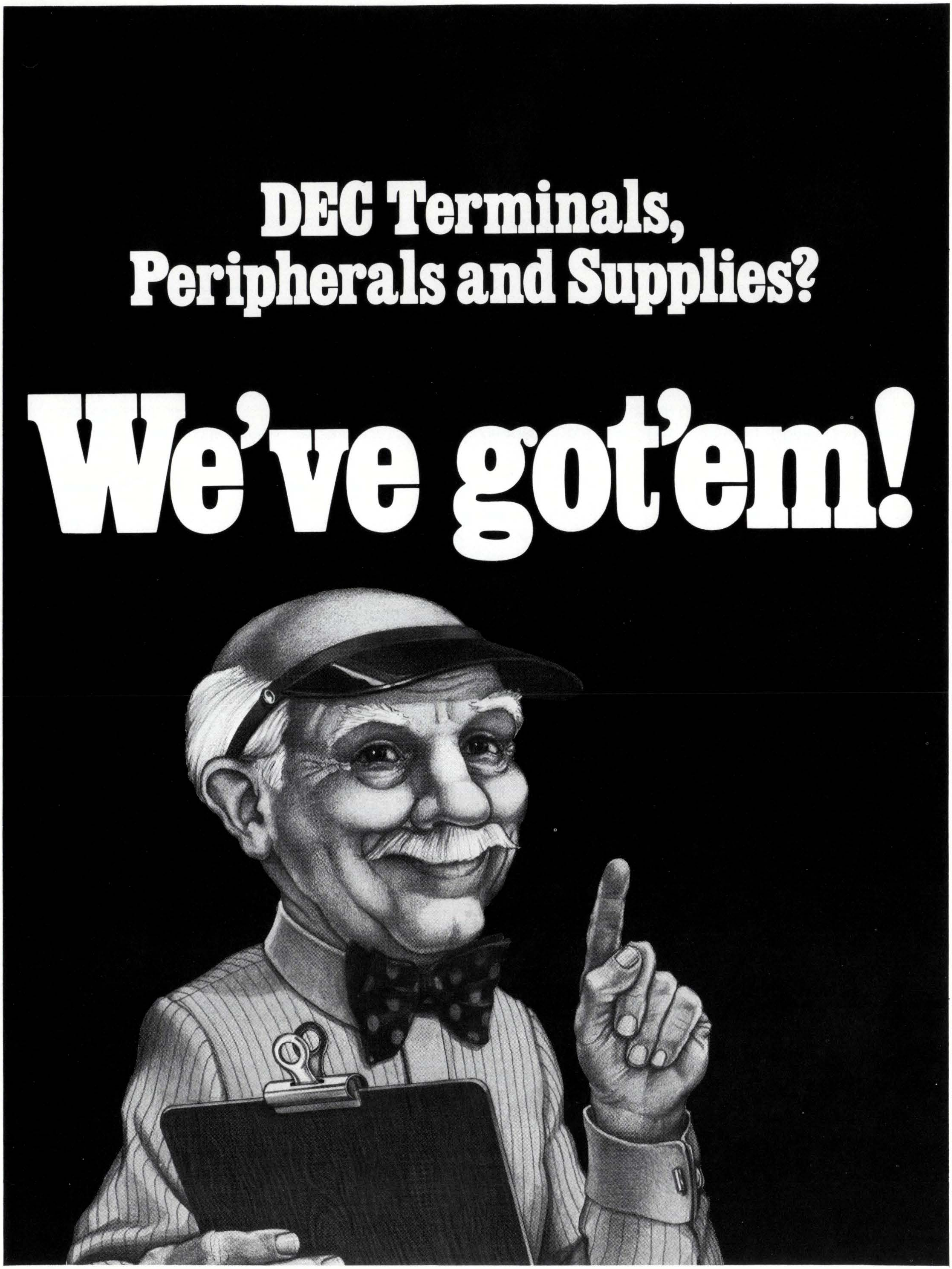
1. Purchase good quality tapes.
2. Avoid tapes with more than 4 or 5 write skips.
3. Avoid tapes over 5 years old.
4. Transport and store tapes vertically in cannisters.
5. Keep tapes clean, at constant humidity and temperature.

DISABLING TERMINAL KEYS ON VT100's, LA120's ETC.

Some installations may want to disable selected keys, for example CONTROL, NO SCROLL, BACK SPACE, BREAK. Here is a simple method which is reliable, reversible, cheap and will not invalidate your warranty. It works by placing a spacer below the keytop so that the key is locked in the 'up' position (so you can immediately tell that the key is disabled). A section from a plastic book-binder is ideal — do not use metal in case it falls into the electronics.

1. Obtain a length of plastic bookbinder (5/16 of an inch internal measurement, approximately).
2. Cut and file smooth a section approximately 5/32 of an inch long, (cut several and keep the best).
3. Gently lever off the keytop.
4. If the plunger comes out with the keytop, remove it and put it back in the keyboard.
5. Insert the spacer below the shoulders. It should be a tight fit.
6. Replace keytop and check that the key is disabled.




DEC Terminals,
Peripherals and Supplies?
We've got'em!

CTRL/F Monitor Support

Version: V7.0-07

Nothing is more inevitable than an idea whose time has come.
- Emerson

1.0 Introduction

When RSTS/E V7.0 was (finally) released in early 1980 and the Large File Processor was unveiled it became clear that information about open files was now contained in memory (as opposed to being pointed to on disk.) I set about at that point to develop a routine that would allow users of the system to strike a control key to display their open files, much the same way CTRL/T displays a "mini-SYSTAT" for the user.

With all the information readily available in memory it merely became a matter of finding out where the information was and how it was stored. In addition, the Terminal Driver would have to be modified to support this added functionality.

It turned out to be fairly easy to solve the above problems. However, writing the CTRL/F code itself turned out to be quite an effort. I had a large amount of information, and several different display formats. Not to mention the trouble of debugging. Many lonely hours of using MONODT and a listing of the TTOPNF code were spent before the final version was complete.

What follows is the installation procedure for CTRL/F Open Files support for your RSTS/E V7.0-07 Large File Monitor. The installation of this option allows the user to strike the <CTRL> and <F> keys simultaneously to get a listing of the files currently open by the job.

TTOPNF WILL ONLY WORK ON LARGE FILE SYSTEMS.

2.0 Installation

Rather than get into a very tedious technical discussion of the exact mechanics of TTOPNF, I will merely present how it may be included in your RSTS/E monitor. To install TTOPNF requires a SYSGEN. Because modifications to the standard files is required, the SYSGEN should be run off disk (as opposed to tape). To do this copy all the files off your SYSGEN tape into the same account as the TTOPNF files. Several things have to be done before you can run the standard SYSGEN procedure to include TTOPNF support. These are:

1. Assemble TTOPNF.MAC
2. Modify the terminal driver (TTDVR.MAC)
3. Insert TTOPNF.OBJ in the RSTS.OBJ library
4. After SYSGEN dialog, modify SYSGEN.CTL to include TTOPNF.

2.1 TTOPNF.MAC

It should be noted that the source as it stands now has been thoroughly tested at several sites for well over a year. Beware, modifications can lead you into trouble.

The assembly procedure for TTOPNF.MAC is as follows:

```
RUN $MACRO
*TTSYSF,TTSYSF/C=COMMON,KERNEL,TTSYSF
ERRORS DETECTED: 0
*^Z
```

NOTE

The files KERNEL.MAC and COMMON.MAC can be located on your RSTS/E Sysgen Medium.

2.2 TTDVR.TEC

To make as simple as possible to modify the terminal driver (TTDVR.MAC on your Sysgen Medium), I developed a TECO macro that will modify TTDVR.MAC to include support for TTOPNF. To modify the standard TTDVR.MAC file found on your RSTS/E distribution, place it in the same account as TTDVR.TEC. Then run TTDVR.TEC. It should produce output similar to the following.

```
RUN TTDVR.TEC
Found "TTDVR.MAC" working
```

When TTDVR.TEC is through the copy of TTDVR.MAC in that account will have been patched to support TTOPNF. If any errors occur they will be TECO errors. Analyze the error and take appropriate action to correct the problem

2.3 Inserting TTOPNF.OBJ in RSTS.OBJ

To insert the TTOPNF module in RSTS.OBJ requires the use of the program LIBR.SAV. This program should currently be in your system library ([1,2]). If not there it can be found on your RSTS/E Sysgen Distribution. Follow this procedure to accomplish the insertion.

```
RUN $LIBR.SAV
*RSTS/X/N=RSTS,TTOPNF
*^Z
```

TTOPNF has been sucessfully inserted if no errors occurred.

2.4 Modifying SYSGEN.CTL

At this point you should run the SYSGEN.SAV program and configure your system. After this is completed, a small modification must be done to SYSGEN.CTL. During the LINKing of the TER phase of the monitor an additional library search must be performed for TTOPNF. If you have included CTRL/T mini-systat support then insert the text "TTOPNF" directly below that in the SYSGEN.CTL file as shown.

```
$R LINK.SAV
TER/Z,TER/A/W,TER=IN:TER,DK:RSTS.STB/X/B: 117000/U: 1000/I/C
TTDVR/C
IN:RSTS
TERPAT
TTSYST
TTOPNF

$R LINK.SAV
```

If TTSYST has not been included insert the text following "TERPAT".

3.0 Appearance

After the new monitor SIL has been installed striking a CTRL/F sequence will produce a listing of your currently open files. It will look similar to this:

```
>TKB
TKB>                          (CTRL/F struck at this point)
1         KB4
8         DB0 [1,71]          TEMP18.TMP*     <60>    0       0       16
15        DB0 [1,2]           TKB   .TSK      <104>   142     169     32      CR
```

Most of the information is self explanatory. From left to right: The channel number, the device open on that channel, the project-programmer number, the filename (the "*" indicates the file has been marked for deletion or is tenative), the protection code, the current virtual block number of the file being accessed, the current virtual size of the file, the clustersize, and then a set of one-character flags. The flags:

C The file is contiguous.

R The file is open in read-only mode.

U The file is open in special file update mode.

L The file has been "placed" on the disk at a specific device cluster.

The file has a block or range of blocks locked.

P The file is permanent and may not be killed or renamed.

There are two special cases of file opens that are seen by TTOPNF. These are UFD opens and "non-file structured" disk opens. They produce output similar to that below.

```
1         DB0                 Non-file structured    0       495513  8
2         DB0 [1,2]           User-file directory    0       112     16      PR
```

On the disk open the "size" becomes the number of virtual blocks the disk contains, and the clustersize is the device cluster size. UFD information is the same as file information. Note the "PR" flags on the UFD open.

4.0 Optional Patch

It may be desirable to allow non-privileged users the ability to use CTRL/F. A patch can be made to the built monitor to enable this. The patch procedure is shown below.

```
RUN $ONLPAT
File to patch? (Monitor SIL with TTOPNF support)
Module name? TER
Base address? ..PRVF
Offset address? 0
 Base    Offset   Old      New?
137554  000000   001404   ? NOP
137554  000002   032761   ? ^C (UP-ARROW C TO EXIT)
```

5.0 Caution

Many letter quality printers use a CTRL/F sequence for acknowledgment. To prevent the monitor from intercepting this sequence set NO CTRL/R on these terminals.

6.0 Information

For any further help or comments contact:

Steven P. Davis
Director Software Engineering
Software Techniques, Inc.
5242 Katella Ave. Suite #101
Los Alamitos, CA 90720

7.0 Acknowledgements

The routines OUTPNT, OUTSIZ, OUTCHR, and DODIVS are slightly modified versions of the same routines found in the TTSYST support code of the RSTS/E monitor.

8.0 TTOPNF.MAC

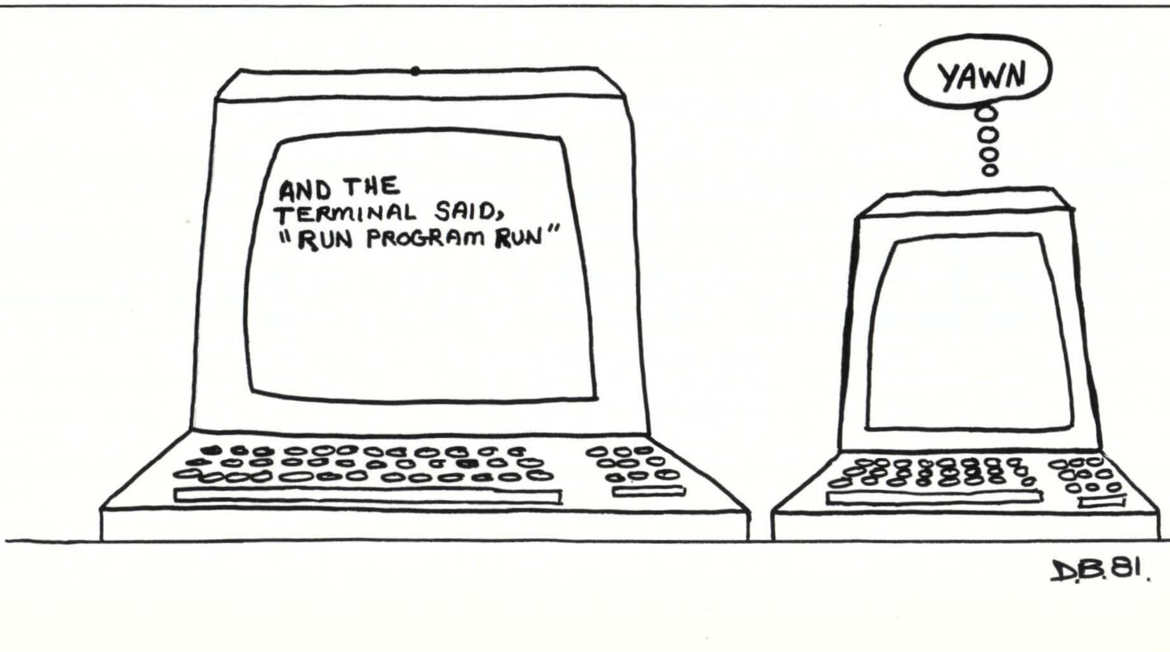
```
TITLE    TTSYSF,<CTRL/F OPEN FILES>,07,28-AUG-80,SPD

; WRITTEN BY: STEVEN P. DAVIS
;
; COPYRIGHT (C) 1980, 1981
; SOFTWARE TECHNIQUES, INC., LOS ALAMITOS, CA 90720
;
; THIS  SOFTWARE  IS PROVIDED FREE OF CHARGE TO MEMBERS OF THE NORTH
; AMERICAN  DECUS  ORGANIZATION  AND  MAY  BE  COPIED  ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE.  THIS  SOFTWARE,  OR  ANY
; OTHER  COPIES  THEREOF,  MAY  NOT  BE  PROVIDED  OR OTHERWISE MADE
; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR  NON-COMMERCIAL  USE  AND
; TO ONE WHO AGREES TO THESE LICENSE TERMS.  TITLE  TO AND OWNERSHIP
; OF THE SOFTWARE SHALL AT ALL TIMES REMAIN IN SOFTWARE TECHNIQUES.
;
; 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.

.SBTTL  SET UP WINDOW POINTERS
.ENABL  LC

.DSECT  ,NOCREF

F$LINK: .BLKW           ;POINTER TO NEXT FCB THIS UNIT
F$FID:  .BLKW           ;FILE ID (LINK TO NAME ENTRY)
F$PPN:  .BLKW           ;PPN OF FILE
F$NAM:  .BLKW           ;FILE NAME
        .BLKW           ;SECOND WORD FILE NAME
        .BLKW           ;EXTENSION
F$STAT: .BLKB           ;FILE STATUS BYTE
F$PROT: .BLKB           ;PROTECTION CODE
F$ACNT: .BLKB           ;OPEN COUNT
F$RCNT: .BLKB           ;READ REGARDLESS COUNT
F$WFND: .BLKW           ;FBB OF FIRST WINDOW
        .BLKW           ;RESERVED, RETRIEVAL ENTRY
F$UFND: .BLKW           ;CURRENT FBB
        .BLKW           ;OF NAME ENTRY
```

```
F$UNT:  .BLKB           ;FIP UNIT NUMBER
F$SIZM: .BLKB           ;FILE SIZE MSB
F$SIZL: .BLKW           ;FILE SIZE LSB
F$CLUS: .BLKW           ;FILE CLUSTER SIZE
F$WCB:  .BLKW           ;FIRST WCB FOR THIS FILE

.DSECT  ,NOCREF

W$IDX:  .BLKB           ;HANDLER INDEX (0=DISK)
W$STS:  .BLKB           ;STATUS FLAGS
W$JBNO: .BLKB           ;JOB   * 2
W$FLAG: .BLKB           ;FLAG BITS
W$PT:   .BLKB           ;PENDING TRANSFERS
W$NVBM: .BLKB           ;NEXT VIRTUAL BLOCK MSB
W$NVBL: .BLKW           ;NEXT VIRTUAL BLOCK LSB
W$FCB:  .BLKW           ;FCB @ F$CLUS
W$REN:  .BLKW           ;RETREIVAL ENTRY NUMBER
W$WCB:  .BLKW           ;NEXT WCB THIS FCB + FLAGS
W$NXT:  .BLKW           ;FBB OF NEXT RETREIVAL WINDOW
        .BLKW           ; (FROM ABOVE)
W$WND:  .BLKW           ;CURRENT WINDOW
        .BLKW           ;WORD TWO OF WINDOW
        .BLKW           ;WORD THREE OF WINDOW
        .BLKW           ;WORD FOUR OF WINDOW
        .BLKW           ;WORD FIVE OF WINDOW
        .BLKW           ;WORD SIX OF WINDOW
        .BLKW           ;WORD SEVEN OF WINDOW

.BSECT                  ;BITS FOR FILE STATUS

US.OUT: .BLKB   .       ;FILE IS PHYSICALLY ON ANOTHER DISK
US.PLC: .BLKB   .       ;FILE HAS BEEN PLACED
US.WRT: .BLKB   .       ;FILE IS OPEN FOR WRITE ACCESS
US.UPD: .BLKB   .       ;FILE IS OPEN FOR UPDATE
US.NOX: .BLKB   .       ;FILE IS CONTIGUOUS
US.NOK: .BLKB   .       ;FILE CANNOT BE KILLED DURING TIMESHARING
US.UFD: .BLKB   .       ;NAME ENTRY IS A UFD
US.DEL: .BLKB   .       ;FILE IS MARKED FOR DELETION

.BSECT                  ;BITS FOR W$STS OR DDSTS

        .BLKB   .       ;OPEN NON-FILE STRUCTURED
        .BLKB   .       ;READ-LOCK ON FILE
        .BLKB   .       ;WRITE-LOCK ON FILE
WC$UPD: .BLKB   .       ;UPDATE MODE ON FILE
WC$CTG: .BLKB   .       ;FILE IS NO EXTEND (CONTIGOUS)
WC$LCK: .BLKB   .       ;LOCK IS ON IN FILE
WC$UFD: .BLKB   .       ;FILE IS A UFD
WC$USE: .BLKB   .       ;WRITE PRIVS GRANTED

.SBTTL  TTSYSF - PRINT OPEN FILES FOR USER
;+
; TTSYSF - PRINT OPEN FILES FOR USER
;
; CALL: R1 -> TO CONSOLE DDB
;
; BACK: C-BIT SET IF CAN'T DO
;-

        DEFORG  TTSYSF

TTSYSF::MOVB    DDJBNO(R1),R3   ;;GET OWNING JOB   *2
        BIC      C<63.*2>,R3    ;;AND ENSURE A VALID JOB   *2
        BEQ     10$             ;;NO JOB, NO OPEN FILES
        MOV     JOBTBL(R3),R3   ;;GET JOB DATA BLOCK POINTER
        BEQ     10$             ;;NO JDB, NO OPEN FILES
        BIT      JFPRIV,JDFLG(R3) ;;ARE WE PRIVILIGED?
..PRVF  ==      .               ;;** PATCH ** TO 'NOP' SO NON-PRIV CAN DO
        BEQ     10$             ;;NO, DON'T SHOW OPEN FILES
        BIT      DDCONS,DDCNT(R1) ;;REALLY CONSOLE DEVICE FOR JOB?
        BNE     DOSYSF          ;;YES, GO SHOW OPEN FILES
10$:    SEC                     ;;MARK AS NO GOOD
        RETURN                  ;; AND RETURN

GLOBAL  <JOBTBL>

.ENABL  LSB

UFDMSG::.ASCIZ  %User file directory%<11>
NFSMSG::.ASCIZ  <11><11>%Non-file structured%<11>
.EVEN   ;THIS COULD BE ANNOYING

DOSYSF: MOV     R5,-(SP)        ;;SAVE AN IMPORTANT REGISTER
        CMPB    DDHORZ(R1),DDHORC(R1) ;;IS CARRIAGE CURRENTLY RESTORED?
        BEQ     10$             ;;YES
        MOV     R3,-(SP)        ;;SAVE THIS POINTER
        CALLX   ASCOUT,R5,CRLF.0 ;;NO, SO RESTORE THE CARRIAGE
        MOV     (SP)+,R3        ;;GET IT BACK
10$:    MOV     JDIOB(R3),R3    ;;POINT TO THEIR I/O BLOCK
        TST     (R3)+           ;;SKIP THEIR CONSOLE TERMINAL
        CLR     R5              ;;LET R5 BE THE CHANNEL COUNTER
20$:    CALL    30$             ;;OUTPUT NEXT CHANNEL
        CMP      15.,R5         ;;PAST LAST CHANNEL
        BGT     20$             ;;NOT YET
        CALLX   ASCOUT,R5,CRLF.0 ;;TO MAKE IT LOOK GOOD
        MOV     (SP)+,R5        ;;RESTORE THE IMPORTANT REGISTER
        CLC                     ;;SAY DONE
        RETURN                  ;; AND EXIT

30$:    INC     R5              ;;SET NEXT CHANNEL NUMBER
        MOV     (R3)+,R4        ;;GET POINTER TO WCB
        BEQ     40$             ;;NOTHING THERE, GO FOR NEXT
        BIT      1,R4           ;;ODD ADDRESS?
        BNE     40$             ;;THAT'S ODD, QUIT
        BR      50$             ;;CONTINUE
40$:    RETURN                  ;;PRE-MATURE RETURN
50$:    MOV     R3,-(SP)        ;;SAVE THIS POINTER
        MOV     R5,-(SP)        ;;SAVE THIS COUNTER
        MOV     R4,-(SP)        ;;SAVE POINTER TO WCB
        MOV     R5,-(SP)        ;;PUT CHANNEL NUMBER ON STACK
        CALL    OUTSIZ          ;;OUTPUT THE CHANNEL NUMBER
        CALL    OUTCHR,R5,<11>  ;;NOW OUTPUT A <TAB>
        MOV     (SP),R4         ;;GET WCB POINTER BACK
        MOVB    W$IDX(R4),R2    ;;IS THIS A DISK?
        BEQ     60$             ;;YEP
        BIT     R2, 1           ;;BOGUS INDEX?
        BNE     120$            ;;QUIT
        MOV     R2,R5           ;;COPY DEVICE INDEX *2
        ADD      DEVNKB-2,R5    ;;POINT TO NAME IN "DEVNAM"
        MOVB    DDUNT(R4),-(SP) ;;SAVE THE UNIT NUMBER
        CALL    OUTPNT          ;;OUTPUT DEVICE NAME
        CALL    OUTSIZ          ;;AND UNIT NUMBER
        BR      110$            ;;AND CONTINUE
60$:    MOV     W$FCB(R4),R4    ;;POINT TO FCB FOR THIS FILE
        MOVB    F$UNT-F$CLUS(R4),R4 ;;GET THE FIP UNIT FOR THIS FILE
        MUL      10.,R4         ;;COMPUTE OFFSET INTO LOGICAL NAME TABLE
        ADD      LOGNAM+6,R5    ;;POINT TO UNIT NUMBER
        MOVB    (R5),-(SP)      ;;SAVE THE UNIT NUMBER
        TST     -(R5)           ;;POINT BACK ONE WORD
        CALL    OUTPNT          ;;PRINT OUT DISK NAME
        CALL    OUTSIZ          ;;PRINT OUT THE UNIT NUMBER
        MOV     (SP),R4         ;;GET BACK WCB POINTER
        MOV     W$FCB(R4),R4    ;;POINT TO FCB AGAIN
        SUB      F$CLUS,R4      ;;POINT TO TOP OF WINDOW
        MOV     F$STAT(R4),-(SP) ;;SAVE FILE STATUS
        MOV     F$CLUS(R4),-(SP) ;;SAVE CLUSTERSIZE
        CLR     -(SP)           ;;FOR MSB AS CLUSTERSIZE
        MOV     F$SIZL(R4),-(SP) ;;SAVE FILE SIZE LSB
        MOVB    F$SIZM(R4),-(SP) ;;SAVE SIZE MSB
        CMP     -(SP),-(SP)     ;;SAVE TWO SPOTS FOR VITUAL BLOCK NUMBER
        BIT      DDNFS,@16(SP)  ;;OPEN NON-FILE STRUCTURED?
        BEQ     70$             ;;GO ABOUT OUR BUSINESS IF NOT
        CALLX   ASCOUT,R5,NFSMSG ;;PRINT OUT THE MESSAGE
        BR      100$            ;;AND GO ON OUR MERRY WAY
70$:    MOVB    F$PROT(R4),-(SP) ;;GET PROTECTION CODE
        ADD      F$NAM,R4       ;;POINT TO FILENAME
        MOV     R4,-(SP)        ;;SAVE THE POINTER
        TST     -(R4)           ;;SKIP THE WORD
        MOV     (R4),-(SP)      ;;SAVE THE PPN
        MOV     (SP),-(SP)      ;;COPY IT TWICE
        SWAB    (SP)            ;;SWAP FOR PROJECT NUMBER
        CALL    OUTCHR,R5,<'[*400+' > ;;NOW OUTPUT A <SP>"[
        CALL    OUTSIZ          ;;OUTPUT THE PROJECT NUMBER
        CALL    OUTCHR,R5,<',>  ;;AND THEN A COMMA
        CALL    OUTSIZ          ;;THEN PROGRAMMER NUMBER
        CALL    OUTCHR,R5,<11*400+']> ;;THEN OUTPUT ]<TAB>
        MOV     (SP)+,R5        ;;POINT TO FILE NAME
        BIT      <WC$UFD*400>,@20(SP) ;;WAS IT JUST A UFD?
        BEQ     80$             ;;NO, TAKE NO SPECIAL ACTION
        TST     (SP)+           ;;SKIP THE PROTECTION CODE
        CALLX   ASCOUT,R5,UFDMSG ;;AND SAY IT'S A UFD
        BR      100$            ;;AND CONTINUE
80$:    CALL    FILNAM          ;;GO PRINT OUT FILENAME
        BIT      US.DEL,16(SP)  ;;ARE WE DELETED?
        BEQ     90$             ;;NOPE
        CALL    OUTCHR,R5,<'*>  ;;OUTPUT A STAR
90$:    CALL    OUTCHR,R5,<60.*400+11> ;;NOW OUTPUT <TAB><
        CALL    OUTSIZ          ;;PRINT PROTECTION CODE
        CALL    OUTCHR,R5,<11*400+62.> ;;NOW OUTPUT ><TAB>
100$:   MOV     16(SP),R4       ;;RESTORE WCB POINTER
        MOV     W$NVBL(R4),2(SP) ;;SAVE VIRTUAL BLOCK LSB IN STACK
        MOVB    W$NVBM(R4),(SP) ;;SAVE VIRTUAL BLOCK MSB IN STACK
        CALL    DBLNUM          ;;OUTPUT NEXT BLOCK
        CALL    OUTCHR,R5,<11>  ;;NOW OUTPUT A <TAB>
        CALL    DBLNUM          ;;OUTPUT FILESIZE
        CALL    OUTCHR,R5,<11>  ;;NOW ANOTHER <TAB>
        CALL    DBLNUM          ;;OUTPUT THE CLUSTER SIZE
        CALL    FILSTA          ;;DO FILE STATUS
110$:   CALLX   ASCOUT,R5,CRLF.0 ;;FINALLY RESTORE CARRIAGE FOR THIS TIME
120$:   TST     (SP)+           ;;SKIP WCB POINTER
        MOV     (SP)+,R5        ;;RESTORE CHANNEL POINTER
        MOV     (SP)+,R3        ;;RESTORE IOB POINTER
        RETURN                  ;;EXIT SUBROUTINE

GLOBAL  <CRLF.0,LOGNAM,DEVNKB>

        .DSABL  LSB

 SBTTL  PRINT OUT FILE STATUS FLAGS

;+
; CALL FILSTA
;
; SP -> FILE STATUS BITS
;
; R4 GETS CLOBBERED
;-

        .ENABL  LSB

FILSTA: MOV     (SP),R4         ;;SAVE RETURN ADDRESS
        MOV     2(SP),(SP)      ;;RESET AS FILE STATUS
        MOV     R4,2(SP)        ;;AND RESET RETURN ADDRESS
        CMP      US.WRT,(SP)    ;;IS THERE STATUS?
        BEQ     50$             ;;NOPE, SKIP IT
        CALL    OUTCHR,R5,<11>  ;;AND YET ANOTHER <TAB>
        BIT      US.NOK,(SP)    ;;PERMANENT FILE?
        BEQ     10$             ;;NOPE
        CALLX   OUTCHR,R5,<'P>  ;;SAY SO
10$:    BIT      US.NOX,(SP)    ;;CONTIGOUS?
        BEQ     20$             ;;NOPE
        CALLX   OUTCHR,R5,<'C>  ;;SAY SO
20$:    BIT      US.UPD,(SP)    ;;OPEN FOR UPDATE?
        BEQ     30$             ;;NOPE
        CALLX   OUTCHR,R5,<'U>  ;;SAY SO
30$:    BIT      US.UPD!US.WRT,(SP) ;;WE HAVE WRITE PRIVELEGES?
        BNE     40$             ;;YES, SO IGNORE
        CALLX   OUTCHR,R5,<'R>  ;;IF NOT, SAY READ ONLY
40$:    BIT      US.PLC,(SP)    ;;ARE WE A PLACED FILE
        BEQ     50$             ;;NOPE
        CALLX   OUTCHR,R5,<'L>  ;;SAY SO
50$:    TST     (SP)+           ;;GET RID OF STATUS
        RETURN                  ;;AND GO BACK

        .DSABL  LSB

.SBTTL  PRINT OUT SINGLE OR DOUBLE INTEGER

.ENABL  LSB

;+
;DBLNUM - PRINT OUT DOUBLE PRECISION INTEGER
;       WITH OR WITHOUT LEFT JUSTIFICATION
;
;       SP -> LSB OF INTEGER, MSB OF INTEGER
;
;       CALL    DBLNUM
;
;       R2 - R5 RANDOM
;-

DBLNUM: MOV     (SP)+,R5        ;;SNAG RETURN ADDRESS
        CLR     R2              ;;CLEAR A HIGH ORDER
        BISB    (SP)+,R2        ;;GET THE HIGH ORDER OF THE INTEGER
        MOV     (SP)+,R3        ;;GET THE LOW ORDER
        MOV     R5,-(SP)        ;;REPLACE THE RETURN ADDRESS
        DIV      10000.,R2      ;;SPLIT THE NUMBER
        MOV     R3,-(SP)        ;;SAVE LOW ORDER
        TST     R2              ;;IS THERE A HIGH ORDER?
        BEQ     10$             ;;NO HIGH ORDER
```

```
        MOV     R2,R3           ;;SET TO OUTPUT IT
        CALL    20$             ;;DO IT
        SEC                     ;;LOW ORDER SHOULDN'T SUPRRESS LEADING ZEROES
10$:    MOV     (SP)+,R3        ;;DO LOW ORDER NOW
        CALL    20$             ;;OUTPUT THE LOW PART
        RETURN                  ;;AND GO BACK

20$:    CALL    DODIVS,R5,30$   ;;GO DO THE DIVISIONS, PRINTING ASCII DIGITS
         .WORD  1000.,100.,10.
        RETURN                  ;;EXIT

30$:    MOVB    50$(R2),R2      ;;GET THE NUMERIC
40$:    CALLRX  CHOUTE          ;;OUTPUT THE CHARACTER

        .ASCII  "$.?"
50$:    .ASCII  "0123456789"    ;;NUMBERS

        .EVEN

;+
; FILNAM - PRINT OUT A FILENAME
;
;       R5 -> TO THREE FILENAME WORDS
;
;       CALL FILNAM,R5
;
;-
FILNAM: MOV      70$,-(SP)      ;;PUSH FOR EXTENSION
        MOV     PC,-(SP)        ;;BACK HERE ONCE
60$:    MOV     (R5)+,R3        ;;GET THE RAD50 WORD
        SEC                     ;;DON'T SUPRESS LEADING ZEROES
        CALL    DODIVS,R5,80$   ;;GO DO THE DIVIDES
         .WORD 50*50,50
        RETURN                  ;;EXIT

70$:    MOV      '.,R2          ;;SET TO OUTPUT A "."
        CALLX   CHOUTE          ;;AND CALL ROUTINE TO DO IT
        BR      60$             ;;AND CONTINUE

80$:    TST     R2              ;;CHECK FOR A SPACE
        BEQ     90$             ;;OUTPUT A SPACE
        ADD      'A-1,R2        ;;ADJUST FOR ALPHABETICS
        CMP     R2, 33+<'A-1>   ;;IS IT IN FACT ALPHABETIC?
        BLO     40$             ;;YES, GO USE IT
        SUB      36+<'A-1>,R2   ;;ADJUST FOR NUMBERIC (ZERO = 0)
        BR      30$             ;;GO DO IT

90$:    MOV      ' ,R2          ;;SET SPACE AS CHARACTER
        BR      40$             ;;AND OUTPUT IT
;+
; OUTSIZ - PRINT A NUMBER 0-255 WITH LEADING ZERO SUPPRESSION.
;
;       SP -> NUMBER (HIGH ORDER BYTE IGNORED), ...
;
;       CALL    OUTSIZ
;
;       R2-R4 = RANDOM
;       SP -> ...
;-

OUTSIZ: CLR     R2              ;;CLEAR THE BUCKET
        BISB    2(SP),R2        ;; AND GET THE K SIZE
        MOV     (SP)+,(SP)      ;;MOVE THE RETURN ADDRESS DOWN
        ;CLC                    ;;C-BIT=0 FROM 'CLR' ABOVE
100$:   MOV     R2,R3           ;;COPY THE NUMBER
        CALL    DODIVS,R5,30$   ;;GO DO THE DIVISIONS, PRINTING ASCII DIGITS
         .WORD 100.,10.
110$:   RETURN                  ;;EXIT
;+
; OUTCHR - PRINT ONE OR TWO CHARACTERS.
;
;       CALL    OUTCHR,R5
;        .BYTE  BYTE1,BYTE2      (BYTE2 NOT PRINTED IF 0)
;
;       R2-R5 = RANDOM
;-

OUTCHR: MOV     R5,(SP)         ;;CLOBBER SAVED R5 WITH RETURN ADDRESS -2
        ADD      2,(SP)         ;; THEN UPDATE TO REAL RETURN ADDRESS

;+
; OUTPNT - PRINT ONE OR TWO CHARACTERS (VARIABLE).
;
;       R5 -> 2 BYTES TO PRINT
;
;       CALL    OUTPNT
;
;       R2-R4 = RANDOM
;       R5 -> PREVIOUS + 2
;-

OUTPNT: MOVB    (R5)+,R2        ;;GET FIRST CHARACTER
        CALLX   CHOUTE          ;; AND OUTPUT IT
        MOVB    (R5)+,R2        ;;GET SECOND CHARACTER (IF ANY)
        BIC      C<177>,R2      ;; AND TRIM THE SIGN BIT
        BNE     40$             ;;OUTPUT SECOND CHARACTER IF ONE EXISTS
        RETURN                  ;;ELSE JUST EXIT

.DSABL  LSB
;+
; DODIVS - DO A SERIES OF DIVISIONS AND EMIT THE QUOTIENTS.
;
;       R3 = NUMBER TO DIVIDE
;       C=1 IF LEADING ZEROS ARE TO BE PRINTED
;
;       CALL    DODIVS,R5
;        .WORD  ROUTINE TO CALL TO EMIT A QUOTIENT
;               (QUOTIENT IS IN R2, MAY NOT ALTER R0, R1, OR R5)
;        .WORD  DIVISOR 1, DIVISOR 2, ...
;       RETURN
;
;       R2-R4 = RANDOM
;
;       NOTE: (DIVISOR N  200) MUST BE 0 FOR N >= 2
;-

DODIVS: MOV     (R5)+,-(SP)     ;;SAVE THE OUTPUT ROUTINE ADDRESS
        BIC     -(SP),(SP)      ;;SAY NOT INTO SIGNIFICANCE YET
        ADC     (SP)            ;;FORCE SIGNIFICANCE IF NEEDED
10$:    CLR     R2              ;;CLEAR A HIGH ORDER
        DIV     (R5)+,R2        ;; AND DIVIDE
        BNE     20$             ;;RESULT <> 0, SAY SIGNIFICANCE STARTS
        TST     (SP)            ;;RESULT = 0, IS IT SIGNIFICANT?
        BEQ     30$             ;;NOT YET...
20$:    INC     (SP)            ;;INDICATE RESULTS ALL NOW SIGNIFICANT
        MOV     R3,-(SP)        ;;SAVE REMAINDER
        CALL    @4(SP)          ;;CALL THE OUTPUT ROUTINE
        MOV     (SP)+,R3        ;;RESTORE REMAINDER
30$:    TSTB    (R5)            ;;MORE?
        BPL     10$             ;;YES, LOOP
        MOV     R3,R2           ;;NO MORE DIVISIONS, SET FINAL RESULT
        TST     (SP)+           ;;DUMP THE SIGNIFICANCE INDICATOR
        CALL    @(SP)+          ;;GO OUTPUT FINAL RESULT
        RETURN  R5              ;; AND EXIT

.END
```

9.0 TTDVR.TEC

```
        EBTTDVR.MAC$$@A/Found "TTDVR.MAC" - working/Y$S/AWR$FR/AWR/SPD$$
        NORG    TTSYST$LI
                ORG     TTOPNF          ;OPEN FILES CODE

        $$
        NTI$CRR::$$-STI$CTT$
        LI; TI$CFF - F HANDLER.
        $$STI$CTT::$$
        LITI$CFF::                      ;;HANDLE CONTROL/F (OPEN FILES)
        $$
        SBNE    40$$S;;$K
        INOPE, IT IS CONTROL/T OR CONTROL/F
        $$
        S40$:$I CMP     R2, 'F-100      ;;CONTROL/F MAYBE?
                BEQ     60$             ;;YEP, HANDLE IT
        $$
        LI50$:$$
        3LI
        60$:    CALL    MAPPED,R5,TTOPNF ;;GO TRY FOR OPEN FILES
                BR      50$             ;;DO SAME AS CONTROL/T
        $$
        STTSYST:$$
        3LI     TMPORG  TTOPNF
        TTOPNF: SEC                     ;;SAY ILLEGAL UNLESS OVERLAID
                RETURN                  ;; AND EXIT
                UNORG

        $$
        N.WORD  'G-100$
        0LI     .WORD   'F-100          ;CONTROL/F (OPEN FILES)
        $$
        N.WORD  TI$CCG$$
        0LI     .WORD   TI$CFF          ;CONTROL/F (OPEN FILES)
        $$
        EX$$
        $$
```



Disc Structure Notes

By Dave Mallery

GOALS

SURVIVAL
PERFORMANCE
SURVIVAL—MFD'S all equal
DCN listings for contiguous files
PERFORMANCE—Seek/Latency/Transfer
Therefore, primary goal is
REDUCE HEAD MOVEMENT
Sources of Head Movement:
Directories—if left unchecked, destroy performance
a) by randomizing their location
b) by randomizing their links.
Non-contiguous files-window turns.

A PROPER DISC

Fully extended, non-deletable, centered, contiguous UFD's
Centered swapfiles (if applicable)
Major data files; centered, contiguous, non-deletable
Recoverable due to
1) copies of MFD contiguous
2) DCN listings of major contiguous files for NFS access.

HOW TO DO IT

1) During DSKINT
a) Pack Clustersize
b) SATT.SYS base
c) MFD Clustersize 8/16
d) do not pre-extend [0,1] & [1,2]
e) do not locate [1,2]

2) At this point, you have a disc with one cluster MFD at beginning and nothing else but [0,1] first cluster. SATT.SYS is centered.

3) Pre-extend the MFD [1,1] —based on expected total accounts. Plan your accounts — give yourself enough extra.

4) Pre-extend [0,1]-you don't need much. Dump the SATT.

5) In [0,1], create enough contiguous files to fill up the SATT to the center.

6) Run $REACT, creating your accounts.

NOTE: You are making MFD entries only. Use UFD clustersize large enough. Create accounts in order of importance.

7) Pre-extend each UFD.

8) Make an image copy(s).

9) Create and Protect your swap files.

10) You will want to:
a) make sys-call to zero an acct. priv or
b) Place a null length, non-deletable file in each directory. The full UFD will remain contiguous.

11) Copy in your main data files. Hopefully, they can be contiguous and protected.

NOTE: they are at the top of the directory! As you need it, balance their locations around the center by deleting some of the bulk files in [0,1].

12) Copy in your BAC and BAS, etc. files.
Make as many contiguous as possible.
Finally all the rest.
As you copy, do the BAC,TSK files first, so they are higher up in the directory.
In [1,2] place login at the top.
Use separate libraries for groups of cusps.
NOTE: B+2 spoolers really pay off.

CARE AND FEEDING

1) Run $REORDER (in B+2) frequently.
2) Don't create accounts unless you must—have extras created and extended.
3) Keep a stash of contiguous space in [0,1].
4) Use contiguous files as much as possible.
5) Print out and keep the DCN's of all contiguous major files you worry about.
6) Make all your discs from the image copy, so your MFD's are identical and interchangable! ♥

BEGINNER'S GUIDE TO MACRO 11 PROGRAMMING IN RSTS/E

by Thomas Courtney, Mark/Ops, Inc., for the SENERUG Reference Library

Chapter 1

Introduction

MACRO-11 is the assembly language for the PDP family of computers. Recently, programming in MACRO-11 has become respectable in the RSTS/E community, and this guide is intended to help you get started. You are going to need the following publications:

1. MACRO-11 Language Reference Manual.

2. PDP11 Processor Handbook for your machine.

3. RSTS/E System Directives Guide.

Actually, everything other than program listings can be found in these manuals somewhere, so the adventurous may wish to proceed to them directly. This guide is not intended to make you a wizard at programming in MACRO-11; it is intended to give you a notion of the structure of the machine you work on, and a firm understanding of how things (like I/O) get done.

Chapter 2

The General RSTS/E Environment

To program in MACRO successfully requires a different "world view" from higher level language programming. Instead of thinking of data and structure and programs as different entities, all three merge into what is hopefully a well ordered heap. After all, how can you expect to do a GET, or a PRINT or a FIELD in assembly language? For if assembly language programming is meant to put the programmer in direct contact with the machine, we would need a pretty sophisticated machine indeed to have such statements as these.

The answer, of course, is that the PDP11 doesn't do all this. [See: "Technical Notes", p. 66] However, it does some things pretty closely, and only waits for the program to put data in the proper locations, and make a call to the Monitor. These are called General Monitor Directives, and are things like .READ, .WRITE, .CCL, etc. Their locations are defined in the file COMMON.MAC, which comes in all RSTS/E kits, and an abbreviated form is in Appendix A.

To RSTS/E, a MACRO job can occupy up to 17777 bytes of memory (all numbers are octal). The monitor directives get their data, and return some too, from the first thousand bytes of this memory. A mapping of this region looks like this:

BYTE RANGE	NAME	DESCRIPTION
0000-0057		User job image or runtime system.
0060-0107		used by Monitor to make the job swappable.
0110-0167		used by Monitor to make the job swappable.
0170-0377		default stack
0400-0401	KEY, USRSP	defines job status
0402-0441	FIRQB	File Request Queue Block
0442-0457	XRB	Transfer Control Block
0460-0657	CORCMN	core common area
0660-0733		controlled solely by the job.
0734-0735	USRPPN	user project, programmer number.
0736-0737	USRPRT	user default protection code.
0740-0777		user logical device name table.

The sections of this area we are most interested in are the FIRQB and the XRB, since these areas are used in most Monitor-job communications.

2.1 The FIRQB.

The FIRQB (File Request Queue Block) contains the information used by the Monitor for I/O requests. Before you do most I/O requests, you have to load the FIRQB area with a variety of values. The general form of the FIRQB is:

OFFSET	NAME	HIGH BYTE	LOW BYTE	OFFSET	NAME
1			returned status	0	FIRQB
3	FQFUN	calfip/.uuo	job number * 2	2	FQJOB
5	FQSIZM	MSB file size	channel number * 2	4	FQFIL,FQERNO
7		project	programmer	6	FQPPN
11		file name (2 words in RADIX-50 form)		10	FQNAM1
13				12	
15		file ext. (1 word in RADIX-50 form)		14	FQEXT
17		least significant bits of file size		16	FQSIZ
21		buffer length		20	FQBUFL,FQNAM2
23		mode		22	FQMODE
25		status flags		24	FQFLAG
27	FQPROT	prot. code	0, high byte real	26	
31		device name (a ASCII characters)		30	FQDEV
33		0, unit real	device unit number	32	FQDEVN
35		cluster size		34	FQCLUS
37		number of entries in directory lookup		36	FQNENT

Most of these entries should be self explanitory. The major idea is that the programmer puts in information needed by whatever Monitor directive, makes the call, and retrieves whatever information the Monitor gave back.

2.2 The XRB.

The XRB is the other major area used for Monitor-job communications. Information specified here are things like the location and size of strings to be processed. The general format of the XRB is:

OFFSET	NAME	HIGH BYTE	LOW BYTE	OFFSET	NAME
1		buffer size in bytes		0	XRLEN
3		bytes transferred		2	XRBC
5		buffer address		4	XRLOC
7	XRBLKM	MSB of block #	channel # * 2	6	XRCI
11		LSB of block #		10	XRBLK
13		wait time for terminals		12	XRTIME
15		device modifier		14	XRMOD,XRBSIZ

Again, almost everything is straightforward. XRTIME is set for terminal input only. If it is 0, the system will wait forever. If it is positive, that is the time it will wait for a response. If negative, it goes into "keyboard Monitor wait" state, which I have never used, but the Directives manual says is used for indirect command files by the Monitor.

Chapter 3

MACRO-11 Programming Basics

MACRO-11 is a fairly complex assembly language, with over 400_{10} instructions in the instruction set, eight register modes, and a variety of assembly time instructions. I am not about to explain all of it, quite simply because I do not know all of it. Instead, I will attempt to explain the small subset of instructions I am going to use.

A statement in MACRO-11 has the form

```
label: operator        operand, operand        ; comment
```

the label and comment are not neccesary to the legality of the statement. Some operators require one or no operands. The types of assembly statements fall into the following general categories:

1. Instructions: these are from the PDP11 instruction set.

2. Directives: these are calls to the RSTS/E Monitor

3. Data formatting: these reserve space for data.

4. Output formatting: these control listing results.

5. Logical control: these instructions allow pieces of a program to be assembled independently.

I will admit there are statements that do not fall into any of these categories. However, I will try not to use any of them here, and firmly believe that for the first shot, we will have enough to chew on.

3.1 The Instruction Set.

These are the nitty-gritty machine instructions. As they say in the processor handbook of you choice, instructions come in three flavors:

1. single operand — these are the operator operand instructions. The ones we will be using are:

MNEMONIC		ACTION
CLR	dst	clears dst (destination) word
CLRB	dst	clears dst byte.
TSTB	tests dst.	

Basically, CLR and CLRB put a 0 in a location specified by the operand. TSTB (and oddly enough, TST) test a location and set the Processor Status word. The Processor Status word contains information on the current status of the machine. Instructions operate on bits 0-3 of this word, called C, V, Z and N respectively. The C bit is set whenever an instruction caused a carry out of the most significant bit of the result. The V bit indicates an operation overflow. The Z bit is set whenever the result of an instruction is cleared. The N bit is set whenever the result of an operation is negative. In general, if the condition does not occur, then the appropriate bits are cleared.

2. double operand — these are the operator operand, operand instructions. The ones we will use are:

MNEMONIC		ACTION
MOV	src, dst	move source to dst.
SUB	src, dst	subtract src from dst.

3. program control — specifies specific actions to be taken by the processor. The ones we will use are:

MNEMONIC		ACTION
BNE	dst	branch to dst if Z bit clear.
JSR	R, nam	jump to subroutine nam
RTS	R	return from subroutine.

JSR and RTS also use a register, called the LINKAGE REGISTER. Upon executing the subroutine call, the current contents of the register are pushed onto the stack, and the return address is put into the register. Upon returning from the subroutine, the contents of the register are used for the return address, and its original contents are restored and popped off the stack.

3.2 Directives.

As you now already know, the Monitor Directives interface between the Monitor and your job. We are only going to use five in the following examples: .READ, .WRITE, .FSS, CALFIP and .EXIT.

1. .READ — this directive reads data from a file or a device previously opened on a channel. Since we're going to use the keyboard, there will be no need to open it, since it defaults to being channel 0. We will have to fill in the information on the XRLEN, XRBC, XRLOC, XRCI and XRBLK. We will get information back on .XRBLK.
2. .WRITE — This directive writes to a file from a user buffer specified in the XRB. The data we must pass it are XRLEN, XRBC, XRLOC, XRCI and XRBLK. I'm not going to talk about what can be returned, since they are errors, and by the time you are ready to deal with them, you can look them up without much hassle.
3. .FSS — the File String Scan directive. This allows you to set up the FIRQB automatically when opening a file. You can also use it to convert strings to RAD50 format.
4. CALFIP — this command does all sorts of I/O related functions, like opening or closing files, assigning/deassigning devices, directory lookup and quite a bit more. We will use it to create a file, as a demonstration of how to set up the FIRQB using the .FSS directive.
5. .EXIT — This returns control to the default runtime system. You need pass nothing to the call, and nothing is returned.

3.3 Data Formatting.

These statements reserve space for data, and sometimes put data into specific locations. The ones we will use are:

MNEMONIC				ACTION
.BLKW	1000			reserves 1000 (octal) words of storage.
.ASCII	/Test/	15	12	puts the ASCII characters T, e, s, t and <cr>, lf into storage.
.ASCIZ	/Test/	15	12	same as .ASCII but appends a null character to the end.
.EVEN				puts in an extra byte if program counter is odd.

3.4 Output Formatting.

These statements take care of how the assembly listing looks.

MNEMONIC	ACTION
.TITLE Test	causes the title "Test" to appear on every output page of the assembly listing.
.IDENT /v07.1/	causes a version number to be added to the title.
.PAGE	causes a page eject in the assembly listing.
.END nam	end of source file input. If "nam" is not null, then this is the entry point procedure. There can be only one entry point procedure built at task building time.

3.5 Logical Control.

These instructions allow the deferring of several actions until link time. We will only use one such statement:

.GLOBL NAM1, NAM2 — causes NAM1 and NAM2 to be external global references. A label is declared GLOBL by using a double semicolon, e.g. label::. A mnemonic variable is made GLOBL by using a double equal sign, e.g. XRLEN = = 0.

Chapter 4

Registers and Addressing Modes.

The PDP11 has eight general purpose registers for use in memory reference addressing. These registers can serve a variety of functions, from holding data to be played with, to containing pointers to pointers to addresses. Two of the general registers, R6 and R7 (the others are R0—R5) have special functions. Register 6, called SP, is the hardware stack pointer. Register 7 contains the location of the next address to be executed, and is called PC or ".". The secret of how all the registers are used is in their addressing modes.

There are eight addressing modes, and these are all explained in the Processor Handbook. Even though they do it better than I, here is a synopsis of what the story is:

1. Mode 0, Register Mode. This uses the register as a simple accumulator. It is the fastest way to do things, whenever possible.

```
INC     R3          ; adds 1 to the contents of R3.
```

2. Mode 1, Register Deferred Mode. The address of the operand is stored in the register.

```
INC     (R3)        ; adds 1 to the contents of the address
                    ; specified by R3.
```

3. Mode 2, Autoincrement Mode. Like Mode 1, but the register is incremented afterwards. It is incremented by 1 if you are using a byte instruction, and by 2 if you are using a word instruction.

```
INC     (R3)+       ; adds 1 to the contents of the address
                    ; specified by R3, then increments R3
                    ; by 2.
```

4. Mode 3, Autoincrement Deferred Mode. The contents of the address specified by the contents of the register are used as the address of the operand. The register is then incremented 1 or 2, for byte or word instructions respectively. The only way I can remember all of this one is to call it "pointer to a pointer" mode.

```
INC     @(R3)+      ; adds 1 to the contents of the address
                    ; specified by the contents of the
                    ; address specified by R3, then
                    ; increments R3 by 2.
```

5. Mode 4, Autodecrement Mode. This is the same as Autoincrement Mode, except it decrements the register before using its contents.

```
INC     -(R3)       ; contents of R3 are decremented by 2.
                    ; the address specified by this number
                    ; are then incremented by 1.
```

6. Mode 5, Autodecrement Deferred Mode. Similar to Autoincrement Deferred, but the register is decremented before anything else happens.

```
INC     @-(R3)        ; R3 is decremented by 2. Adds 1 to the
                      ; contents of the address specified by
                      ; the contents of the address specified
                      ; by R3.
```

7. Mode 6, Index Mode. The base address is added to the index word to get the address of the operand.

```
INC     1000(R3)      ; 1000 is added to R3, and the contents
                      ; of that address are incremented by
                      ; 1
```

8. Mode 7, Index Deferred Mode. The base address is added to the index. The contents of the index is then used as a pointer to the address of the operand.

```
INC     @1000(R3)     ; 1000 is added to R3; and the address
                      ; pointed to by the contents of the
                      ; address in R3 are incremented by 1.
```

There are also 4 modes associated with the use of the PC register.

9. Mode 2, PC Immediate Mode. This is equivalent to autoincrement with the PC.

```
ADD     #10, R0       ; the value 10 is added to R0.
```

10. Mode 3, PC Absolute Mode. This is equivalent to autoincrement deferred mode with the PC.

```
ADD     @#10, R0      ; adds the contents of location 10 to
                      ; R0.
```

11. Mode 6, PC Relative Mode. This is the Index mode with the PC.

```
ADD     MSG, R0       ; adds the contents of MSG to R0.
```

12. Mode 7, PC Relative Mode Deferred. This the Index deferred mode with the PC.

```
ADD     @MSG, R0      ; the contents of the address specified
                      ; by the contents of MSG are added
                      ; to R0.
```

. . . continued on page 88

NEW PRODUCT. See our ad on page 35.

QUE.11 — System Manager's Guide

SYSTEM FILES

If the installation of QUE.11 has been sucessful you will have these files in your system ([1,x] is the QUE.11 account):

Queue control package: (The protection codes shown must be used)

[1,x]QUE11.TSK <124>
[1,x]OP.TSK <124>
[1,x]TEST.TSK <104>
[1,x]LOAD.TSK <124>
[1,x]SHOW.TSK <104>
[1,x]SUBMIT.TSK <104>
[1,x]DO.TSK <104>
[1,X]CANCEL.TSK <104>

Spooling package:
$PRINT.TSK <104>
$SPOOL.TSK <232>
$QUE.TSK <232>

Print control files: (required for each print device)
$NORMAL.LP <40>
$NORMAL.LP0 <40>

In some systems the PRINT, SPOOL and QUE programs are held in the [1,x] library instead of the $ account. This requires a minor alteration to the control files NORMAL.LP etc. Note that these control files must be kept in the $ library.

SYSTEM STARTUP

The only job which must be started is QUE11.TSK. This may be run in any privileged account but the old (if any) QUE11.DAT file should be in the same account.

When QUE.11 starts it prints a ?-mark. This can be answered in two ways:

1. device specification
2. detach statement (always last).

Device Specification

QUE.11 has an internal table with a list of all the physical devices configured in your system. This table is used to schedule queued jobs which reserve devices and associated volumes. You may disallow the use of a class of devices by a command of this type:

? MT:/bar

You may add to the list of devices by giving the mnemonic of a new 'dummy' device, like this:

? CLASS:

Dummy devices are a very powerful aid in job scheduling; they are described in detail later.

Switches: /single /bar
/clock

Detach Statement

This statement ends the initialization. After it is given QUE.11 searchs for a file called QUE11.DAT which holds a list of any previously queued jobs. If the file does not exist in the account under which QUE.11 is running then a new one is set up automatically.

Two switches may be used on the DETACH statement these are /LIST which prints the internal device table and /PRI:n which sets the QUE.11 priority.

These two switches must be given in the order shown here:

?DETACH/LIST/PRI:0

Note that if you defined a dummy device at a previous set-up which is now omitted, an error may occur when the QUE11.DAT file is scanned. Any job which reserved the omitted device will be omitted; a message will be printed to warn you.

CONCISE COMMAND LANGUAGE (CCLS)

You should add this set of commands to the Concise Command Language (remember to alter the START.CTL files for INIT):

CCL SHO-W = [1,x]SHOW.TSK
CCL SUB-MIT = [1,x]SUBMIT.TSK

```
CCL CAN-CEL = [1,x]CANCEL.TSK
CCL DO- = [1,x]DO.TSK
CCL LOAD- = [1,x]LOAD.TSK
CCL OP- = [1,x]OP.TSK

CCL PRI-NT = $PRINT.TSK
CCL QUE-UE = $QUE.TSK;30000
```

If PRINT.TSK and QUE.TSK are held in the [1,x] library then the $ in the commands above must be replaced by [1,x].

The commands do not have to be as shown; any command verb will be accepted but the words shown here correspond with the User's Guide and the HELP options in the commands. [The PRINT command will override the Basic-Plus immediate mode PRINT statement (although '&' will still work). SPOOL is recommended instead of PRINT in this case.]

CLOSE DOWN

QUE.11 is ended by giving the OP CLOSE command. This serves two purposes it shuts down the queue only when all the current jobs have finished and it saves outstanding job requests in the QUE11.DAT file. If QUE11.DAT is stopped in any other way (by a KILL command, for instance) it is likely that some items in the queue will be lost.

The SHUTUP program will not end QUE.11 properly; the OP CLOSE command should be given just before SHUTUP is started. The SYSTAT job table should be checked to make sure that QUE.11 has stopped. A typical closedown might look like this:

```
SYS/2
2 [1,5]  Det QUE.11   19/28K   SL D02...
Ready
OP CLOSE
Ready
SYS/2
Ready
Run $SHUTUP
. . . . . . .
```


OUTLINE OF OPERATION

QUE.11 processes requests by using pseudo-keyboards as the job consoles. It can process up to four batch streams at once if there are four PK:'s free in the system.

Job scheduling is done by testing each job against the following list of requirements which must be satisfied before the job will start:

1. has the system room for another job?
2. has QUE.11 a job slot free (10 available)?
3. is there a pseudo-keyboard free?
4. has QUE.11 enough free I/O channels (1 for immediate mode, 2 for batch or 3 for batch with a log file)?
5. are the reserved devices free (not assigned and not opened)?
6. are the required volumes (paper) loaded?

Note that there is a maximum of 11 channels at step 4. In immediate mode (DO command) only step 4 needs to be satisfied.

STATE CODES

The state column in the QUE.11 status report (SHOW command) uses the codes described here to indicate the reason why a job is waiting to run.
The normal codes that may appear are:

Q Wait
request not yet scanned because some essential resource is unavailable.

V Wait
job requires one or more volumes to be mounted

D Wait
job requires a device that is in use; it will run when the device becomes free.

Abnormal codes are:

Exxx
an unrecoverable error occured in starting the job (xxx is the RSTS/e error code).

The operator should cancel requests which appear with the abnormal codes because they prevent later requests from running.

DEVICE TABLE

As explained already each job request may reserve up to five devices and associated volumes. These may be any of the physical or dummy devices . The job will not be started until all the devices are free and until they each have the required volume loaded on them.

If a job reserves a device without a specific unit number (e.g MT:) the job will run as soon as any device of the type required becomes available.

The list of currently loaded volumes is altered by the load command which has this form:

```
        LOAD dev:volume

e.g     LOAD LP0:WIDE
        LOAD MT2:TAPE79
```

No checking (except for discs) is done to see if the volume is actually the one loaded and it is the responsibility of the operator to ensure that the QUE.11 device list accurately reflects the state of the peripheral devices on the system.

The current status of the device list may be found by typing

```
      SHOW D[evices]
or    SHOW V[olumes]
```

Only the devices which are currently loaded will appear.

DUMMY DEVICES

Dummy devices are treated by QUE.11 in almost exactly the same way as the real devices on the system. For example, suppose that CLASS: was set up as a dummy device. It could be used to schedule large, slow jobs to run at night or at some other convenient time. The request to QUE.11 would be:

```
SUBMIT test for CLASS:NIGHT
```

which might show in the status report as:

Queue index	Owner	State	Reserved Devices
12.2	[11,9]	V Wait	CLASS:NIGHT

When the operator was leaving in the evening the device CLASS: would be loaded with the 'volume' night:

```
LOAD CLASS:NIGHT
```

Remote job entry is an application where a real device may be loaded with a dummy volume. The distant receiver might be given a mnemonic name which would be used when submitting jobs for transmission. These jobs would be released by loading the device RJ: with the mnemonic name.

The main distintion between dummy devices and the real ones is that the dummy devices may be used by more than one job at the same time. Where this is undesirable it can be prevented by appending the /SINGLE switch to the device when QUE.11 is being started.

SPOOLING OF PRINT FILES

Print files are queued to the printer(s) by a package of programs (stored in the '$' account) which make use of the QUE.11 batch control facilities. The files are transfered to the printer by a program called SPOOL.TSK which is run under the control of a command file (also stored in the '$' account). The name of this file is of special significance; it represents the type of paper required while the file name extension indicates the printer device.

This filename convention is important; it the method by which the System Manager controls which devices may be used as queuable printers and the forms may be used on them.

For example, if your system has a printer LP0: and a terminal KB3: which is also to be used as a printer, you might provide these files:

$NORMAL.LP	$NORMAL.LP0
$NARROW.LP	$NARROW.LP0

$NARROW.KB3
$FORMX .KB3

These files would allow normal and narrow paper to be used in the printer LP0: and narrow and FormX in KB3:. Notice that the file name extensions were given in two forms for LP: — this allows the user to specify the printer exactly (LP0:) or by the generic name (LP:) if the unit is unimportant. This does not matter if there is only one printer on your system. On a two printer system the .LP extension would allow files to be queued for either LP0: or LP1: (determined by QUE.11 when the job is scheduled).

When a file is queued for printing with the PRINT or QUE programs a request is submitted to QUE.11 in the same way as a normal batch job request. However the control file is one of the special files described above. The printer and paper are reserved automatically and appear in the QUE.11 status report as for a batch job.

The name of the file(s) to be printed are given as a list of 'parameters' which cause some of the words in the command file to be replaced. The three words affected are:

1. #FS — the filename string (which may contain switches)

2. #SK — replaced by Y if page skip is required;

3. #RQ — used for REQUED information if necessary.

SHOW n may be used to see the parameters for job 'n'.

PRINT COMMAND FILES

The $NORMAL.LP file which is supplied should be used as a basis for new control files. The listing below gives the purpose of the lines in the control file.

1	RUN $SPOOL.TSK	
2	#FS	— the file string
3	&VOL	— the paper name
4	133	— the paper width
5	66	— the paper depth
6	8192	— the open mode for the printer
7		— name of a header file
8	#SK	— the SKIP option
9	&PPN	— the requester's PPN
10	#RQ	— REQUE information.

The following items are all subtituted when the file is read by QUE.11 at run time:

#FS, &VOL, #SK, &PPN, #RQ

At line 7 a file name may be given; the contents of this file will be printed as part of a separator page at the begining of the printout. If this line is left blank no seperator page appears.

At line 6 the value 8192 stops a form-feed when the printer is closed at the end of the job. See the RSTS/E programmers guide for information on other values.

If the operator cancels a job that has already started the job is stopped and the paper is realigned. SPOOL assumes that the printer can handle form-feeds correctly.

SPOOL COMMANDS

The operator can control printing by means of these commands:

1. CANCEL n
   CANCEL dev: (e.g CANCEL LP:)
   Cancel the job before or during printing. The second form may only be used when the job has started to print.

2. LOAD dev:paper
   used to tell QUE.11 that the paper has been changed. If the paper is changed while a job is

actually running there is no effect until it finishes and the next job is selected.

3. OP STATUS n
   OP STATUS LP:
   used to discover the state of the current job. This command and the following ones will only work while the specified job is actually using the printer, otherwise one of the errors
   ?Job not in receive state
   ?Device not controlled by QUE.11
   will occur.

4. OP RESTART n B
   OP RESTART dev: B
   May be used to restart a printout at the beginning (omit B) or at a specified block. This may upset the page numbering if the /SKIP option was chosen.

The SPOOL.TSK program waits when the print buffers are full. This can cause a delay when some of the commands above are given before a response is returned to the operator and action is taken by the program. The most affected commands are the OP commands.

ON2OFF.B2S

By Dave Schott

```
2       !               Program         : ON2OFF.B2S
3       !               Version         : 7.0
4       !               Edit            : 07
5       !               Edit Date       : 10-Nov-80
6       !               Author          : Dave Schott
11      !                                                               &
        !                                                               &
        !       Able Communications & Systems, Inc.                     &
        !       1197 North Tustin Avenue                                &
        !       Anaheim, CA  92807                                      &
        !       (714) 632-0220                                          &
        !                                                               &
        !                                                               &
        !       This software is furnished free of charge to RSTS/E sites.  It &
        !       may be distributed and copied to any site so long as there is no &
        !       direct commercial profit derived from such distribution. &
        !                                                               &
        !       Although this software has been tested and is believed to be &
        !       error-free, neither ACSI nor the author assume responsibility &
        !       for the use or reliability of the software.             &
        !                                                               &
        !       Any problems with the software should be reported to the author &
        !       at the address above, although support of the software is not &
        !       guaranteed.                                             &
        !                                                               &
        !********************************************************************* &

20      !                                                               &
        !                                                               &
        !       M O D I F I C A T I O N   H I S T O R Y                 &
        !                                                               &

30      !  VER/ED       EDIT DATE       REASON                          &
        !                                                               &
                                                                        &

100     !                                                               &
        !                                                               &
        !       G E N E R A L   D E S C R I P T I O N                   &
        !                                                               &
        !       ON2OFF will turn ON/OFF the modem bit on the requested  &
        !       keyboard line.  This is accomplished by re-writing the  &
        !       TTINTF (terminal interface flag) of the requested       &
        !       keyboard line.  Since this program does poke memory it  &
        !       can only be ran sucessfully from [1,1]
999     !                                                               &
        !       M A I N   C O D I N G                                   &
        !                                                               &

1000    ON ERROR GOTO 19000                                             &
                ! establish the standard error handling                 &
        \ PRINT IF CCPOS(0%) <> 0%                                      &
                ! return KB to left margin                              &
        \ ID$ = "V7.0-07"                                               &
                ! set up the version ID tag                             &

1010    PRINT "ON2OFF";CHR$(9%);ID$;CHR$(9%);                           &
        CVT$$(RIGHT(SYS(CHR$(6%)+CHR$(9%)+CHR$(0%)),3%),4%)             &
                ! indentify ourself w/version ID tag                    &

1020    DIM MONITOR1%(30%),MONITOR2%(30%)                               &
        \ CHANGE SYS(CHR$(6%)+CHR$(-3%)) TO MONITOR1%                   &
                ! first part of the monitor tables                      &
        \ MONITOR1%(I%) = MONITOR1%(I%) + SWAP%(MONITOR1%(I%+1%))       &
                        FOR I% = 5% TO 29% STEP 2%                      &
                ! swap those bytes that need to be                      &
        \ CHANGE SYS(CHR$(6%)+CHR$(-12%)) TO MONITOR2%                  &
                ! second part of the monitor tables                     &
        \ MONITOR2%(I%) = MONITOR2%(I%) + SWAP%(MONITOR2%(I% + 1%))     &
                        FOR I% = 3% TO 29% STEP 2%                      &
                ! swap here also                                        &
                                                                        &

1030    MAX.KB% = MONITOR1%(3%)                                         &
                ! maximum # of keyboard on this system                  &
        \ DEVOKB% = MONITOR2%(9%)                                       &
        \ DEVPTR% = MONITOR1%(7%)                                       &
        \ KB0.DDB% = PEEK(DEVPTR% + DEVOKB%)                            &
                ! KB0: DDB                                              &
                                                                        &

1040    PRINT "Keyboard #: ";                                           &
                ! request a keyboard line                               &
        \ KB$ = FNI$ UNLESS ERRFLAG%                                    &
        \ GOTO 1050 UNLESS ERRFLAG%                                     &
        \ GOTO 32767                                                    &

1050    KB% = VAL(KB$) UNLESS ERRFLAG%                                  &
                ! get the keyboard into correct form                    &
        \ GOTO 1055 UNLESS ERRFLAG%                                     &
        \ PRINT ERT$(ERRFLAG%)                                          &
                ! let the user know of an error                         &
        \ ERRFLAG% = 0%                                                 &
        \ GOTO 1040                                                     &
                                                                        &

1055    GOTO 1060 IF KB% > 0% AND KB% <= MAX.KB%                        &
                ! make sure it's a legal keyboard                       &
        \ PRINT "You cannot change the console (KB0:)" IF KB% = 0%      &
        \ PRINT "Keyboard # ";KB%;" is not configured on this system"   &
                                IF KB% <> 0%                            &
        \ GOTO 1040                                                     &
                ! tell user of the goof - and reprompt                  &

1060    KB% = KB% * 2%                                                  &
        \ KB.DDB% = PEEK(KB0.DDB% + KB%)                                &
                ! get the requested keyboard's DDB.....                 &
        \ TTINTF% = PEEK(KB.DDB% + 30%)                                 &
                ! ..and it's terminal interface flag                    &
        \ PRINT "This keyboard is currently set to ";                   &
        \ PRINT "modem." IF (TTINTF% AND 16384%)                        &
        \ PRINT "local." UNLESS (TTINTF% AND 16384%)                    &
                ! tell the user how it's currently setup as             &

1070    PRINT "Change this line to LOCAL/MODEM <Modem> ";               &
        \ I$ = FNI$ UNLESS ERRFLAG%                                     &
        \ GOTO 1080 UNLESS ERRFLAG%                                     &
        \ ERRFLAG% = 0%                                                 &
        \ GOTO 1040                                                     &

1080    MODEM% = -1%                                                    &
        \ MODEM% = NOT(ASCII(CVT$$(I$,32%)) = 76%) IF LEN(I$)           &
                ! the default is MODEM unless the user requested        &
                ! LOCAL status                                          &
                                                                        &

1090    GOTO 32767 IF MODEM% AND (TTINTF% AND 16384%)                   &
                ! what? the user requested to change to a MODEM status  &
                ! on an already MODEM line - scram                      &
        \ GOTO 32767 IF NOT(MODEM%) AND (TTINTF% AND 16384%) <> 16384%  &
        \ NEW.TTINTF% = TTINTF% OR 16384% IF MODEM%                     &
        \ NEW.TTINTF% = NOT(NOT(TTINTF%) OR 16384%) IF NOT(MODEM%)      &

2000    POKE$ = CHR$(6%)                ! FIP call code                 &
              + CHR$(-6%)               ! POKE memory                   &
              + CHR$(KB.DDB% + 30%)     ! address to POKE at            &
              + CHR$(SWAP%(KB.DDB%+30%))! swap of the above address     &
              + CHR$(NEW.TTINTF%)       ! value to POKE                 &
              + CHR$(SWAP%(NEW.TTINTF%))! swap of the value             &
                                                                        &

2010    T$ = SYS(POKE$) UNLESS ERRFLAG%                                 &
                ! POKE it                                               &
        \ GOTO 2020 UNLESS ERRFLAG%                                     &
        \ PRINT "You must be in account [1,1]" IF ERRFLAG% = 10%        &
        \ PRINT ERT$(ERRFLAG%) IF ERRFLAG% <> 10%                       &
        \ GOTO 32767 IF ERRFLAG%                                        &
                ! let the user know if an error occurs during the POKE  &
                ! and just scram                                        &

2020    PRINT "Change sucessful."                                       &
        \ GOTO 1040                                                     &
                ! let the user know everything is OK.                   &

19000   ERRFLAG% = ERR                                                  &
        \ ERRLINE% = ERL                                                &
        \ ERRPROG$ = ERN$                                               &
        \ RESUME IF ERRLINE% < 10000% OR ERRLINE% > 28999%              &
        \ RESUME 19010                                                  &

19010   RETURN                                                          &
                !                                                       &
                ! MASTER ERROR ROUTINE                                  &
                ! If the error occurred in the main program just resume &
                ! to the offending line.  If an error occurred within a &
                ! subroutine then return from the subroutine with ERRFLAG% &
                ! set to indicate subroutine failure.                   &
                !                                                       &

20000   DEF FNI$                                                        &
                ! get a line of input from the keyboard                 &
        \ GOSUB 20010                                                   &
        \ FIELD #0%, RECOUNT AS X$                                      &
        \ FNI$ = ""                                                     &
        \ FNI$ = CVT$$(X$,5%) UNLESS ERRFLAG%                           &
        \ GOTO 20020                                                    &

20010   GET #0%                                                         &
        \ RETURN                                                        &

20020   FNEND                                                           &
                                                                        &

32767   END                                                             &
```

FICHE.BAS

By Scott Banks and Dave Mallery

```
1          EXTEND
10       !       FICHE.BAS                                               &
         !                                                               &
         ! GENERALIZED FICHE TAPE GENERATOR                              &
         !                                                               &
         !       THIS PROGRAM WILL TAKE ONE OR MORE ASCII PRINT FILES    &
         ! FOR INPUT AND CREATE AN EBCDIC MAGTAPE SUITABLE FOR FICHE     &
         ! PRODUCTION.                                                   &
         !                                                               &
         !       EACH INPUT FILE IS SEPARATED BY A SINGLE (EOF) TAPE     &
         ! MARK.  THE TAPE IS WRITTEN AT 1600 BPI.                       &
         !                                                               &
         !       THE OUTPUT TAPE UNIT IS REQUESTED AT RUN TIME.          &
         ! FILE NAMES ARE INPUT PARAMETERS ALSO.  SWITCHES ALLOW OUTPUT  &
         ! FORMAT CHANGES.  AN INDIRECT COMMAND FILE MAY BE SPECIFIED.   &
         !                                                               &
         !       SWITCHES MUST APPEAR ON INDIVIDUAL LINES.  THEY REMAIN  &
         ! IN EFFECT UNTIL ANOTHER OCCURS.  THE FOLLOWING ARE ALLOWED,   &
         ! SHOWN WITH THEIR DEFAULT VALUES:                              &
         !                                                               &
         !       /RL:132        PROVIDES 132 COLUMNS (133 W/PRINT CTRL)  &
         !                                                               &
         !       /BL:20         TAPE WRITTEN WITH 20 LOGICAL RCRDS/BLK   &
         !                                                               &
         !       /RW:YES        REWIND-AND-OFFLINE WHEN ALL DONE JOB     &
         !                      (USE 'RW:NO' TO CANCEL THIS)             &
         !                                                               &
         !       /LC:YES        ALLOW LOWER CASE LETTERS                 &
         !                      (USE 'LC:NO' TO FORCE THEM TO UPPER)     &
         !                                                               &
         !       IF AN INDIRECT COMMAND IS TO BE USED, IT IS ENTERED     &
         ! IN RESPONSE TO THE FIRST INPUT FILE REQUEST.  THE FORMAT:     &
         !                                                               &
         !       @<filespec>    AN EXTENSION OF '.CMD' IS THE DEFAULT    &
         !
90         GOSUB 19600
100        DIM   LST.CTL$(127), LST.TXT$(127)                            &
         !
200        F.CMX% = 1%                          ! FOR CMD FILE           &
         \ F.LST% = 2%                                                   &
         \ F.MAG% = 3%                                                   &
         \ F.CMD% = KB%                         ! ASSUME KB:             &
         !
900        PRINT #KB%                                                    &
         \ PRINT #KB%, 'Fiche  V1.2'                                     &
         \ PRINT #KB%                                                    &
         !
1000       GOSUB 2000                           ! OPEN MAGTAPE           &
         !
1020       GOSUB 4000                           ! DEFAULT PARAMETERS     &
         !
1040       GOSUB 5000                           ! SET UP EBCDIC XLATE    &
         !
1200       GOSUB 3000                           ! GET NEXT FILESPEC      &
         \ GOTO 1600 IF E%                                               &
         !
1220       GOSUB 4200                           ! SETUP WORK AREA        &
         !
1240       GOSUB 4400                           ! BUILD TAPE             &
         !
1280       GOTO 1200                                                     &
         !
1600       GOSUB 2800                           ! CLOSE MAGTAPE          &
         !
1880       GOTO 32767                                                    &
         !
2000     !       OPEN MAGTAPE                                            &
         !
2020       PRINT #KB%, 'Magtape? ';                                      &
         \ INPUT #KB%, F.MAG$                                            &
         !
2040     ! DO FILENAME STRING SCAN TO SET FLAG WORD 2                    &
         !                                                               &
           MAG.FL2% = SWAP%(CVT$%(RIGHT(                ! ERR -> 2100    &
                    ! FILENAME SCAN &
                SYS(CHR$(6%)+CHR$(-10%)+F.MAG$),29%)))  ! FLAG WORD 2    &
         \ GOTO 2140 IF MAG.FL2% AND 2047%              ! NO NAMES       &
                   OR MAG.FL2% < 0%                     ! BAD LOGICAL    &
         \ GOTO 2140 UNLESS MAG.FL2% AND 8192%          ! GOOD DEVICE    &
         \ GOTO 2140 UNLESS (STATUS AND 255%) = 14%     !   = MAGTAPE    &
         !
2060       OPEN F.MAG$ FOR INPUT AS FILE F.MAG%,        ! ERR -> 2100    &
                     RECORDSIZE 8192%, MODE 256%                         &
         \ GOTO 2140 UNLESS STATUS AND 256%             ! DEMAND NFS     &
         \ Z% = MAGTAPE(7%,0%,F.MAG%)                   ! MAG STATUS     &
         \ IF    Z% AND (1024% + 512% + 32%)                             &
           THEN  E% = 14%                       ! HUNG OR WRITE LOCKED   &
         \       GOTO 2100                                               &
         !
2064       IF    (Z% AND 256%) = 0%                                      &
           THEN  PRINT #KB%, 'Warning - tape not at load point'          &
         \       PRINT #KB%                                              &
         !
2080       GOTO 2190                                                     &
         !
2100       PRINT #KB%, FNE$(E%); ' - '; F.MAG$          ! HANDLE ERRORS  &
         \ GOTO 32767                                                    &
         !
2140       PRINT #KB%, '?Non-file-structured magtape required'           &
         \ GOTO 32767                                                    &
         !
2190       RETURN                                                        &
```

```
         !
2800     !       CLOSE MAGTAPE                                           &
         !
2820       E% = MAGTAPE(1%,0%,F.MAG%)           ! REWIND AND OFF-LINE    &
                              IF MAG.RWD%                                &
         \ CLOSE F.MAG%                                                  &
         !
2990       RETURN                                                        &
         !
3000     !       GET NEXT INPUT FILESPEC                                 &
         !
3020       E% = 0%                                                       &
         \ PRINT #KB%, 'Input file? '; IF F.CMD% = KB%                   &
         !
3030       INPUT LINE #F.CMD%, Z$               ! EOF -> 3190            &
         \ F.LST$ = CVT$$(Z$,-1%-64%)                                    &
         \ PRINT #KB%, '--> '; F.LST$                                    &
                     UNLESS F.CMD% = KB%                                 &
         !
3040       IF    LEFT(F.LST$,1%) = '@'          ! INDIRECT CMD FILE      &
           THEN  F.CMD$ = RIGHT(F.LST$,2%)                               &
         \       GOSUB 3200                     ! OPEN CMD FILE          &
         \       GOTO 3020                                               &
         !
3050       GOTO 3100 UNLESS LEFT(F.LST$,1%) = '/'       ! A SWITCH?      &
         \ IF    LEFT(F.LST$,4%) = '/RL:'               ! RECORD LENGTH  &
           THEN  LST.LEN% = FNVAL(RIGHT(F.LST$,5%))                      &
         \       GOTO 3090 IF LST.LEN% < 1%                              &
                       OR LST.LEN% > 255%                                &
         \       GOTO 3020                                               &
         !
3060       IF    LEFT(F.LST$,4%) = '/BL:'       ! BLOCKING FACTOR        &
           THEN  LST.BLF% = FNVAL(RIGHT(F.LST$,5%))                      &
         \       GOTO 3090 IF LST.BLF% < 1%                              &
                       OR LST.BLF% > 128%                                &
         \       GOTO 3020                                               &
         !
3070       IF    F.LST$ = '/RW:NO'              ! NO REWIND REQUEST      &
           THEN  MAG.RWD% = 0%                                           &
         \       GOTO 3020                                               &
         !
3072       IF    F.LST$ = '/RW:YES'             ! REWIND WHEN DONE JOB   &
           THEN  MAG.RWD% = -1%                                          &
         \       GOTO 3020                                               &
         !
3080       IF    F.LST$ = '/LC:YES'             ! ALLOW LOWER CASE       &
           THEN  TBL.LOW% = -1%                                          &
         \       GOSUB 5000                     ! BUILD XLATE TABLE      &
         \       GOTO 3020                                               &
         !
3082       IF    F.LST$ = '/LC:NO'              ! FORCE ALL UPPER CASE   &
           THEN  TBL.LOW% = 0%                                           &
         \       GOSUB 5000                     ! BUILD XLATE TABLE      &
         \       GOTO 3020                                               &
         !
3090       PRINT #KB%, '?Illegal switch or value - '; F.LST$             &
         \ GOTO 3020                                                     &
         !
3100       OPEN F.LST$ FOR INPUT AS FILE F.LST%,        ! ERR -> 3180    &
                                    RECORDSIZE 4096%,                    &
                                    MODE 8192%                           &
         !
3160       GOTO 3190                                                     &
         !
3180       PRINT #KB%, FNE$(E%); ' '; F.LST$            ! HANDLE ERRORS  &
         \ GOTO 3020                                                     &
         !
3190       RETURN                                                        &
         !
3200     !       OPEN INDIRECT COMMAND FILE                              &
         !
3220       F.CMD% = F.CMX%                      ! SAY IT'S THERE         &
         \ F.CMD$ = F.CMD$ + '.CMD'             ! DEFAULT EXTENSION      &
                     UNLESS INSTR(1%,F.CMD$,'.')                         &
         !
3240       OPEN F.CMD$ AS FILE F.CMD%, MODE 8192%       ! ERR -> 3300    &
         !
3280       GOTO 3390                                                     &
         !
3300       PRINT #KB%, FNE$(E%); ' - '; F.CMD$          ! HANDLE ERRORS  &
         \ GOTO 32767                                                    &
         !
3390       RETURN                                                        &
         !
4000     !       DEFAULT PARAMETERS                                      &
         !
4020       LST.LEN% = 132%                      ! LINE LENGTH (NOT       &
                                                !   INCL PRINT CTRL)     &
         \ LST.BLF% = 20%                       ! BLOCKING FACTOR        &
         \ MAG.RWD% = -1%                       ! REWIND WHEN DONE       &
         \ TBL.LOW% = -1%                       ! ALLOW LOWER CASE       &
         !
4190       RETURN                                                        &
         !
4200     !       SETUP WORK AREA                                         &
         !
4220       LST.CNT% = (LST.LEN% + 1%) * LST.BLF%                         &
         !
4240       FIELD #F.MAG%,        LST.CNT% AS LST.BUF$                    &
         \ FIELD #F.MAG%,        (LST.LEN% + 1%) * I% AS D$,             &
                                      1% AS LST.CTL$(I%),                &
                                 LST.LEN% AS LST.TXT$(I%)                &
```

```
                                         FOR I% = 0% TO LST.BLF% - 1%    &
        !
4390      RETURN                                                         &
        !
4400    !       BUILD TAPE                                               &
        !
4420      LSET LST.BUF$ = ''                                             &
        \ FOR I.LST% = 0% TO LST.BLF% - 1%                               &
        !
4440      INPUT LINE #F.LST%, LST.LIN$            ! EOF -> 4500          &
        \ LSET LST.CTL$(I.LST%) = '1'                                    &
                IF ASCII(LST.LIN$) = 12%                                 &
        \ LSET LST.TXT$(I.LST%) = LST.LIN$                               &
        !
4460      NEXT I.LST%                                                    &
        !
4480      GOSUB 4600                              ! XLATE, PUT TO MAGTAPE &
        \ GOTO 4420                                                      &
        !
4500      GOSUB 4600 IF I.LST%                    ! XLATE, PUT TO MAGTAPE &
                                                  !   IF PARTIAL BUFFER  &
        !
4520      CLOSE F.LST%                            ! DONE THIS LISTING    &
        !
4580      GOSUB 4800                              ! WRITE 3 EOF MARKS    &
        \ STOP UNLESS MAGTAPE(5%,2%,F.MAG%)=1%    ! BACKSPACE 2 OF 'EM   &
                      FOR I% = 1% TO 2%                                  &
        !
4590      RETURN                                                         &
        !
4600    !       TRANSLATE TO EBCDIC, PUT TO MAGTAPE                      &
        !
4620      LSET LST.BUF$ = XLATE(LST.BUF$,TBL.EBC$)      ! EOT -> 4700    &
        \ PUT #F.MAG%, COUNT LST.CNT%                                    &
        \ GOTO 4790                                                      &
        !
4700      GOSUB 4800                              ! WRITE 3 EOF MARKS    &
        \ PRINT #KB%, 'Mount next reel on '; F.MAG$                      &
        !
4720      E% = MAGTAPE(1%, 0%, F.MAG%)            ! REWIND AND OFFLINE   &
        !
4740      SLEEP 5%                                                       &
        \ Z% = MAGTAPE(7%, 0%, F.MAG%)            ! GET DEVICE STATUS    &
        \ GOTO 4740 IF Z% AND 32%                 !   TILL ONLINE AGAIN  &
        !
4750      IF    Z% AND 1024%                      ! WRITE-LOCKED         &
          THEN  PRINT #KB%, 'Remove write ring on ; F.MAG$               &
        \       GOTO 4720                                                &
        !
4760      IF    (Z% AND 256%) = 0%                ! DEMAND LOAD-POINT    &
          THEN  PRINT #KB%, F.MAG$; ' not at load-point'                 &
        \       GOTO 4720                                                &
        !
4780      PRINT #KB%, 'Tape was successfully mounted on '; F.MAG$        &
        !
4790      RETURN                                                         &
        !
4800    !       WRITE 3 EOF MARKS, EVEN ACROSS PHYSICAL EOT              &
        !                                                                &
          FOR I% = 1% TO 3%                                              &
        !
4840      Z% = MAGTAPE(2%, 0%, F.MAG%)            ! EOT -> 4880          &
        !
4880      NEXT I%                                                        &
        !
4990      RETURN                                                         &
        !
5000    !       SETUP ASCII-EBCDIC TRANSLATE TABLE                       &
        !
5020      TBL.NUM$ = ''                                                  &
        \ TBL.NUM$ = TBL.NUM$ + CHR$(I%)          ! NUMBERS 0-9          &
                      FOR I% = 240% TO 249%                              &
        \ TBL.LET$ = ''                                                  &
        \ TBL.LET$ = TBL.LET$ + CHR$(I%)          ! LETTERS A-I          &
                      FOR I% = 193% TO 201%                              &
        \ TBL.LET$ = TBL.LET$ + CHR$(I%)          ! LETTERS J-R          &
                      FOR I% = 209% TO 217%                              &
        \ TBL.LET$ = TBL.LET$ + CHR$(I%)          ! LETTERS S-Z          &
                      FOR I% = 226% TO 233%                              &
        \ TBL.LOW$ = ''                                                  &
        \ TBL.LOW$ = TBL.LOW$ + CHR$(I%)          ! LOWER CASE A-I       &
                      FOR I%= 129% TO 137%                               &
        \ TBL.LOW$ = TBL.LOW$ + CHR$(I%)          ! LOWER CASE J-R       &
                      FOR I% = 145% TO 153%                              &
        \ TBL.LOW$ = TBL.LOW$ + CHR$(I%)          ! LOWER CASE S-Z       &
                      FOR I% = 162% TO 169%                              &
        \ TBL.LOW$ = TBL.LET$ UNLESS TBL.LOW%     ! FORCE TO UPPER CASE  &
        !
5040      TBL.EBC$ = STRING$(32%,64%)             !   0- 31 CTRL CHARS   &
                   + CHR$( 64%) + CHR$( 90%)      !  32- 33              &
                   + CHR$(127%) + CHR$(123%)      !  34- 35 " #          &
                   + CHR$( 91%) + CHR$(108%)      !  36- 37 $ %          &
                   + CHR$( 80%) + CHR$(125%)      !  38- 39 & '          &
                   + CHR$( 77%) + CHR$( 93%)      !  40- 41 ( )          &
                   + CHR$( 92%) + CHR$( 78%)      !  42- 43 * +          &
                   + CHR$(107%) + CHR$( 96%)      !  44- 45 , -          &
                   + CHR$( 75%) + CHR$( 97%)      !  46- 47 . /          &
                   + TBL.NUM$                     !  48- 57 NUMBERS      &
                   + CHR$(122%) + CHR$( 94%)      !  58- 59 : ;          &
                   + CHR$( 76%) + CHR$(126%)      !  60- 61 < =          &
                   + CHR$(110%) + CHR$(111%)      !  62- 63 > ?          &
                   + CHR$(124%)                   !      64 @            &
        \ TBL.EBC$ = TBL.EBC$ + TBL.LET$          !  65- 90 LETTERS      &
                   + CHR$(173%) + CHR$(224%)      !  91- 92 [ \          &
                   + CHR$(189%) + CHR$( 95%)      !  93- 94 ] ^          &
                   + CHR$( 96%) + CHR$(125%)      !  95- 96 _ `          &
                   + TBL.LOW$                     !  97-122 LOWER CASE   &
                   + CHR$(139%) + CHR$(106%)      ! 123-124 { |          &
                   + CHR$(155%) + CHR$(161%)      ! 125-126 } ~          &
                   + CHR$( 64%)                   !     127 RUBOUT       &
        \ TBL.EBC$ = TBL.EBC$ + TBL.EBC$          ! 128-255 (PARITY BIT) &
        !
5190      RETURN                                                         &
        !
19600   !> ------ S T A N D A R D   I N I T I A L I Z A T I O N ------   &
          ON ERROR GOTO 19800                                            &
        \ KB%=12%                                                        &
        \ OPEN 'KB:' AS FILE KB%, MODE 4%                                &
        \ RETURN                                                         &
        !
19800   !> ------ L O C A L   E R R O R   T R A P S ------------------   &
          E%=ERR                                                         &
        !
19810     RESUME 2100 IF ERL=2040%                     ! MM: SYS() SCAN
19812     RESUME 2100 IF ERL=2060%                     ! MM: OPEN
19820     RESUME 3190 IF ERL=3030% IF E%=11%           ! KB: EOF
19830     RESUME 3180 IF ERL=3100%                     ! LISTING EOF
19840     RESUME 3300 IF ERL=3240%                     ! CMD FILE OPEN
19850     RESUME 4500 IF ERL=4440% IF E%=11%           ! LISTING EOF
19860     RESUME 4700 IF ERL=4620% IF E%=4%            ! MAGTAPE PUT
19870     RESUME 4880 IF ERL=4840% IF E%=4%            ! WRITE MAG EOF
19990     GOTO 32000                                                     &
        !
20000   !> ------ S T A N D A R D   R O U T I N E S ------------------   &
        !
20060   !> FNVAL        COMPUTES VAL(), ERROR IN E%                      &
        !                                                                &
          DEF FNVAL(Z$)                                                  &
        \ E%, FNVAL = 0%                                                 &
        \ FNVAL = VAL(Z$)                          ! ERR => 20068        &
        !
20068     FNEND                                                          &
        !
32000   !> ------ S T A N D A R D   E R R O R   T R A P S ------------   &
        !
32020     RESUME 20068 IF ERL=20060%                   ! FNVAL
32500     ON ERROR GOTO 0                                                &
        \ STOP                                                           &
        !
32560   DEF FNE$(E%)=CVT$$(RIGHT(SYS(CHR$(6%)+CHR$(9%)+CHR$(E%)),3%),5%) &
        !
32767     END
```

RSTS/E MONITOR INTERNALS PART 1

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

For many years RSTS users have been pleading with DEC to produce a manual describing the internal workings of the RSTS/E monitor. However, DEC is in a difficult position. If they published such a manual some people would complain if DEC later made something in the monitor incompatable with the current version.

This is where we in the user community come in. I'm willing to share some of the information I've gathered in my work with RSTS over the past eight years. I hope this information proves useful.

This is the first of four articles that will describe the internals of RSTS/E V7.0. All the major functions will be described, including job control, memory control and file and device handling.

The articles will be broken into four parts as follows:

Part 1: 1. Job Control
- JOBTBL—Job table
- JDB—Job data block one
- JDB2—Job data block two
- IOB—I/O data block
- WRK—Work block
- Fixed memory locations
- JBSTAT, JBWAIT—Job status tables

Part 2: 2. Memory Control
- MCB—Memory control sub-block
- MEMLST, RESLST—Memory control lists
- RTS—Runtime system descriptor block
- NULRTS—Disappearing RSX RTS
- RTSLST—RTS control list
- LIB—Resident library descriptor block
- LIBLST—Library control list
- WDB—Window descriptor block

Part 3: 3. File Control
- SCB—Small file control block
- FCB—Large file control block
- FCBLST—File control list
- WCB—Window control block

4. Device Control
- DDB—Device data block
- DEVNAM—Device name table
- DEVCNT—Device unit count table
- DEVPTR—Device pointer table
- DEVTBL—Device retrieval table
- LOGNAM—Logical device name table
- UNTCNT—Disk status table

Part 4: 5. Send-Receive
- RIB—Receiver ID block
- ERLRIB—ERRLOG RIB block
- SNDLST—Receiver control list
- PMB—Pending message block

6. Concise Command Language
- CCL—CCL definition block
- CCLLST—CCL control list

7. Miscellaneous

Since the information described in these articles is intended primarily for systems programmers it will be presented in the form of a reference manual. My apologies to those of you who are just looking for a general understanding of RSTS and come away bleary-eyed.

If you plan to use this information to access monitor table information from an application program please remember that DEC may change this information at any time, so write your programs accordingly. They've changed things in every release so far. There's no reason to think they won't do it again.

Enough said. Let's get inside RSTS.

1.0 JOB CONTROL

RSTS can support up to 63 simultaneous jobs. Each of these jobs can be either a user at a terminal or a detached program. The job control structures allow RSTS to share resources (such as CPU time) properly among all users. In addition they provide the means to access the information necessary for almost every other service provided by the monitor, such as device and file handling.

For example, the scheduler uses the information in the job descriptor block to determine which job to run next and what run burst to assign it. The memory manager uses the memory control sub-block and the residency quantum to set up the memory mapping registers and perform swapping, if necessary.

The job control structures consist of a combination of tables and blocks. The size of the tables is determined by the number of jobs specified at sysgen time. The size of a block is typically 16 words.

The tables contain one word for each possible job on the system. As new jobs are created and removed, information in the tables is changed but the size of the tables remain the same.

The job control blocks, on the other hand, exist only while a job exists. They are created (from small buffers) when a job is first created and deleted when the job is removed (ie. by logging out).

The location of specific tables can be determined using the GET MONITOR TABLES SYS calls. Once you know where a table starts, the values within the table can be accessed by adding the required offset to the starting address of the table. The following example gets the address of the Job Data Block (JDB) for job 5 (see section 1.1 and 1.2 for information about JOBTBL and JDB):

```
10 JOBTBL% = SWAP%(CVT$%(MID(SYS(CHR$(6%) + CHR$(-3%)),11%,2%)))
                          !Get starting address of job table (JOBTBL)
20 JDBPTR% = PEEK(JOBTBL% + (5%*2%))
                          !Get pointer to JDB for job 5 from JOBTBL
```

1.1 JOBTBL—JOB TABLE

The Job Table is the root of the job control structures. It points to the Job Data Block which, in turn, points to the other job related blocks.

JOBTBL contains one entry for each possible job on the system. The job control information for each job can be accessed by using the job number times two as an offset from the beginning of JOBTBL. The value found at that location will be the address of the Job Data Block (JDB) for that job. If a 0 is found at that location there is currently no job with that job number.

The first word of JOBTBL (at offset 0) is always a 0. This word corresponds to the entry for the null job, job 0. The last word in the table will always be -1 to signify the end of the table. Thus, the total length of JOBTBL, in words, is JOBMAX (the maximum number of jobs, specified at sysgen time) plus 2.

1.2 JDB—JOB DATA BLOCK ONE

The Job Data Block (JDB) contains the most commonly used information about a job. It is pointed to be the entry in JOBTBL corresponding to its job number. The JDB, in turn, points to the three other job control blocks for the job: JDB2, IOB and WRK.

The scheduler uses JDPRI and JDBRST to determine which job to run next. The memory manager uses JDMCTL, JDSIZE, JDSIZM and JDSIZN to set up the memory mapping registers and to schedule a swap-in. The swapper uses JDRESQ, JDSWAP, JDMCTL and JDSIZE to swap jobs in and out of memory. The EMT handler uses JDIOST, JDFLG and JDWRK to process system and I/O calls.

```
Symbol Offset                                          Offset Symbol

             ---------------------------------------
             |            Pointer to IOB            | 0  JDIOB
             |--------------------------------------|
             |      Primary job status flags        | 2  JDFLG
             |--------------------------------------|
JDPOST   5   |  Posting mask    |  IOSTS for job    | 4  JDIOST
             |--------------------------------------|
             | Pointer to job's work block (WRK)    | 6  JDWORK
             |--------------------------------------|
             |       Pointer to job's JDB2          | 8  JDJDB2
             |--------------------------------------|
JDSIZN  11   |  New job size    | Job status flags  | 10 JDFLG2
             |--------------------------------------|
             |  Pointer to job's RTS descriptor     | 12 JDRTS
             |--------------------------------------|
             |         Residency quantum            | 14 JDRESQ
             |--------------------------------------|
             |      Memory control sub-block        | 16 JDMCTL
             |                                      |
             |                                      | 18
             |                   -------------------|
             |                   |     Job size     | 20 JDSIZE
             |                   -------------------|
             |                                      | 22
             |                                      |
             |                                      | 24
             |--------------------------------------|
             |     L3Q bits to set on residency     | 26 JDRESB
             |--------------------------------------|
JDBRST  29   |    Run burst     |     Priority      | 28 JDPRI
             |--------------------------------------|
JDSWAP  31   | Swap slot number |  Maximum memory   | 30 JDSIZM
             ---------------------------------------
```

Offset	Symbol	Description
0	JDIOB	This word contains the address of the I/O data block (IOB) for this job (see section 1.4).
2	JDFLG	This word contains the main job status flags (see section 1.2.1).
4	JDIOST	This byte contains the error code to be returned to the program after the completion of the current monitor call. If the bit JFIOST is set in the job status flags (JDFLG), JDIOST is moved to the location FIRQB+0 in the user program area (accessed by the variable ERR in BASIC-PLUS). A value of 0 indicates that no error occured. (See the System Directives Manual).
5	JDPOST	If the bit JFPOST is set in the job status flags (JDFLAG) this byte is used to specify which information in the job's work block (WRK) is to be posted to the job's FIRQB or XRB. If the value in JDPOST is positive it is used as an index into a table of bit masks for posting to the FIRQB. If it is negative the low order 7 bits are used as a bit mask for posting to the XRB.
6	JDWORK	This word contains the address of the work block (WRK) for this job (see section 1.5).
8	JDJDB2	This word contains the address of the second job data block (JDB2) (see section 1.3).
10	JDFLG2	This byte contains the secondary job status flags (see section 1.2.2).
11	JDSIZN	This byte contains the size (in K words) to make this job the next time it is swapped in. The memory manager uses this location when a job must be swapped out to find additional memory as it attempts to grow in size.

12	JDRTS	This word contains the address of the runtime system descriptor block (RTS) associated with this job. (RTS blocks will be discussed in the second part of this series). Every job has a runtime system associated with it at all times. If the disappearing RSX runtime system is associated with this job, JDRTS will contain the address of the null runtime system descriptor block (NULRTS).
14	JDRESQ	This word contains the current residency quantum for the job. As long as the job's residency quantum is non-zero the job is not eligible to be swapped out. This is used to reduce memory thrashing. When a job is first swapped into memory it is given a residency quantum. Each time the job runs or does disk I/O the residency quantum is reduced in proportion to the amount of time it ran and the estimated time to complete the I/O. When the job goes into terminal input wait the residence quantum is set to zero to allow the job to be swapped if necessary.
16	JDMCTL	These five words are the memory control sub-block (MCB) for the job. The memory control sub-block will be discussed in the second part of this series.
20	JDSIZE	This word (within the memory control sub-block) contains the size of job in K words.
26	JDRESB	This word contains the bit mask to be posted to the level three queue word (L3Q) when the job is made resident. It is used by the monitor to notify itself when the job is made resident so that a function that required the job to be resident may be continued.
28	JDPRI	This byte contains the job's priority. It can range from -128 to + 127. The scheduler uses this byte to determine which job to run next.
29	JDBRST	This byte contains the job's run burst. The run burst is the amount of time (in clock ticks) the job may execute compute-bound before the scheduler is called.
30	JDSIZM	This byte contains the job's private memory size maximum. A value of 255 indicates that the job may use up to the system job size maximum.
31	JDSWAP	If this byte is non-zero it contains the slot number within the swapping files that the job is swapped out to. If it is zero, the job is either resident in memory or has no job image.

1.2.1 JDFLG — Primary Job Status Flags

The job status flags contained in the word JDFLG are defined as follows:

Bit	Symbol	Description
0	JFPOST	The monitor checks this bit before making a job runable. If it is set the information in JDPOST is used as a mask for updating the job's FIRQB or XRB. The information in J2PPTR and J2PCNT in JDB2 (see section 1.3) may also be used to post large amounts of data to a user-defined buffer.
1	JFIOKY	If this bit is set when a job is made runable the job's keyword is copied into location 400(octal) of the job's image and the contents of JDIOST are posted to the job's FIRQB.
2	JFRSX	If this bit is set the monitor resident RSX support is used to post the job information indicated by JFIOKY, rather than using the standard monitor routines.
3	JFCC	This bit is set when a ↑C is typed at the job's terminal. When the job becomes runable the pseudo-vector P.CC (see the System Directives Manual) will be taken unless JF2CC is also set.
4	JF2CC	This bit is set when a ↑C is typed and at least one ↑C has already been typed since the job was last run. When the job becomes runable the pseudo-vector P.2CC will be taken.
5	JFPPT	If this bit is set when the job is made runable the floating point trap pseudo-vector, P.FPP, will be taken.
6	JFGO	If this bit is set when the job is made runable the I/O redo condition specified by JFREDO is ignored. This bit is set if a user types ↑C during an interuptable I/O operation.
7	JFREDO	If this bit is set when a job appears runable (and JFGO is not set) then the job is stalled waiting for an I/O completion and is not really runable.
8	JFSYST	This bit is set if the job can use temporary privileges.
9	JFFPP	If this bit is set the contents of the floating point hardware (if any) are saved and restored along with the job image. This bit is one of the keyword bits (see the System Directives Manual).
10	JFPRIV	This bit is set if the job is logged into a privileged account. It is one of the keyword bits.
11	JFSYS	This bit is set if the job is currently running with temporary privileges. It is one of the keyword bits.
12	JFNOPR	This bit is set if the job is running non-logged in. It is one of the keyword bits.
13	JFBIG	If this bit is set the job can exceed its private memory size (as defined in JDSIZM). It is one of the keyword bits.
14	JFLOCK	This bit is set if the job is locked in memory. It is one of the keyword bits.
15	JFSPCL	This bit is set if some special processing is required before running the job. The flag bits in JDFLG2 specify the special action to be performed.

1.2.2 JDFLG2 — Secondary Job Status Flags

The job status flags contained in the byte JDFLG2 are defined as follows:

Bit	Symbol	Description
0	JFCTXT	This bit is set if the job's context is to be saved.
1	JFPRTY	This bit is set if the special condition shown by JFSPCL is a memory parity error.
2	JFRUN	This bit is set if the special condition shown by JFSPCL is a new program run entry request.
3	JFSWPR	This bit is set if the special condition shown by JFSPCL is a runtime system or resident library load failure.
4	JFSTAK	This bit is set if the special condition shown by JFSPCL is a stack overflow.
5	JFSWPE	This bit is set if the special condition shown by JFSPCL is a swap error.
6	JFKIL2	This bit is set when the job is being killed and the logout phase of killing a job has completed. The second phase of a logout, the removal of job control information, should be done now.
7	JFKILL	This bit is set if the job is to be killed.

If JFSPCL is set in JDFLG but no bits are set in JDFLG2, the current runtime system is entered at its P.STRT or P.CRAS entry point.

1.3 JDB2 — JOB DATA BLOCK TWO

The secondary job data block (JDB2) contains information about the job that is used less often or is less

Improve your communication

with DEC. Just concentrate.

We've earned our reputation by designing and delivering dependable, easy to live with tape and disk controllers for DEC cpus. Products that really showed the industry how to do it.

Now we've cleaned up the communications I/O morass, with a new DH-11 type multiplexer for PDP-11 and VAX systems. It duplicates all the features of DEC's DH-11. And it goes well beyond, delivering improved line handling capabilities in a far smaller package and at significantly lower cost.

You might say we're now doing for communications controllers what we've already done for tape and disk controllers – applying the latest microprocessor technology and some sexy design to give our customers the performance and reliability they deserve.

Our newest achievement proves the point. To wit: just one hex size board, occupying a single Unibus slot, provides up to 64 lines for terminals, printers or other communications I/O. That's eight times more efficient than the DEC approach.

And our unique system organization means maximum flexibility. You can daisy-chain up to four remote distribution panels, expanding in 8 or 16 line increments all the way to 64 lines. You can configure any 8 line group for an RS-232 interface, with modem control a standard feature. Or you have the option of selecting a current loop interface. And because all your interface variations are made at the distribution panels, no additional hardware or cpu restructuring is needed when you add channels.

You get everything you got from DH-11, such as DMA on transmit operation and line format flexibility. Naturally, we're fully software transparent and run existing DEC diagnostics.

But we like to do everything we do just a little bit better than the other guy. So we boosted performance to 19.2 K baud per line. For peak traffic, we can "double fifo" to 128 characters per 16 lines. Multiple controllers can be added for increased capacity. You get full 16-bit word transfers on DMA operation – not just byte transfers. And, of course, we've taken advantage of our system's high speed bipolar microprocessor to build-in extensive controller and channel self-test and fault indicators.

We're concentrating on DEC, making the world's best cpus even better with some ingenious controllers. We've done it for tape and disk interfacing. Now for communications I/O. We'd like to send you all the details we didn't have room to include here. So write or call Emulex Corporation. Or, if you just can't wait, contact the Emulex representative, dealer or distributor nearest you.

For immediate, off the shelf delivery contact our national distributor: First Computer Corporation; Corporate Square, 825 North Cass Avenue; Westmont, IL 60559; 312/920-1050.

The new Emulex CS-11/H really concentrates your communications I/O. Just one hex size board multiplexes up to 64 lines.

EMULEX

EMULEX CORPORATION
2001 E. Deere Ave.
Santa Ana, CA 92705
714/557-7580
TWX 910-595-2521

time critical than the information in JDB. Its primary use is for accounting and directory information.

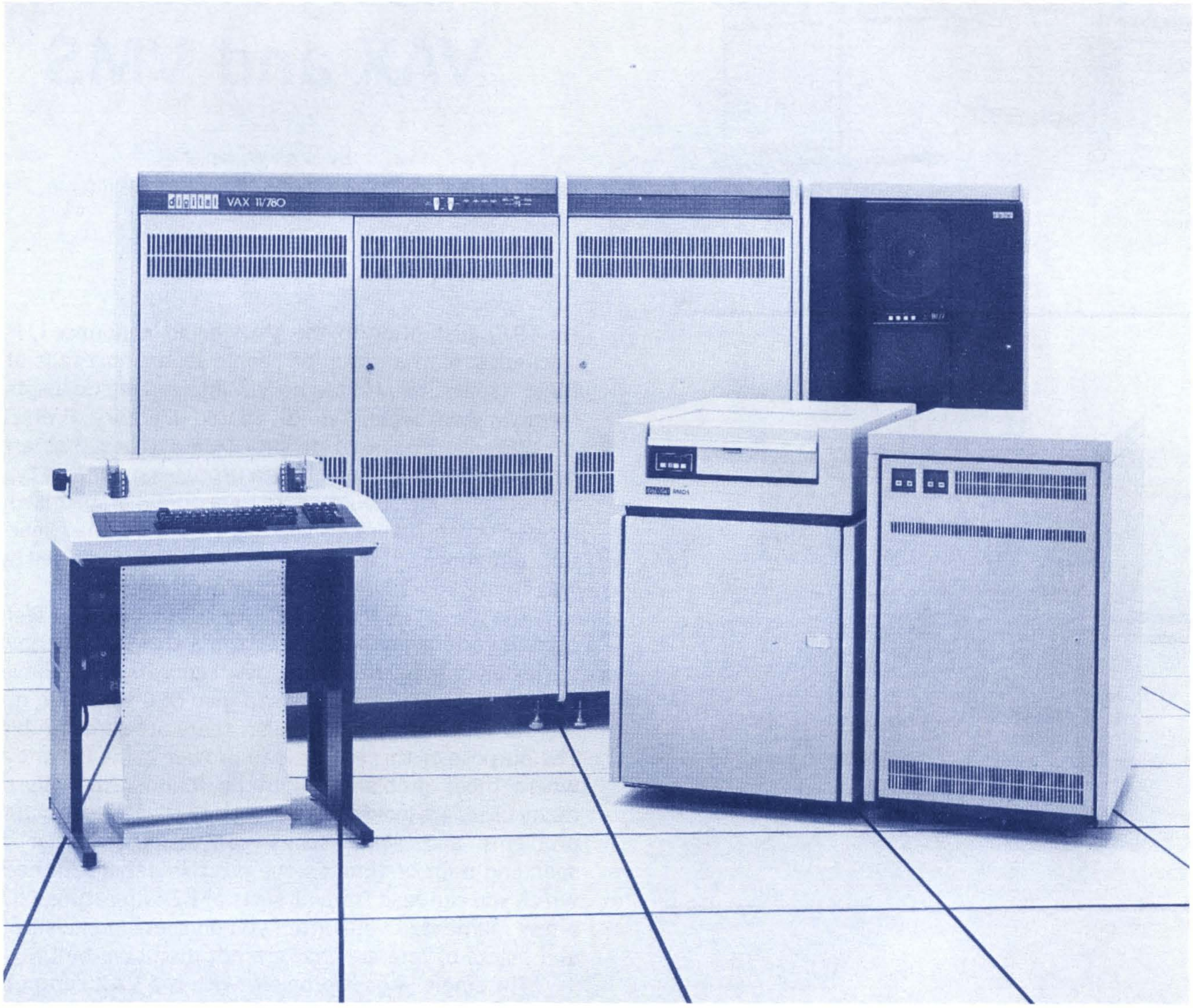
```
Symbol Offset                                      Offset Symbol

                ------------------------------------
               |        Unposted clock ticks        |  0  J2TICK
               |------------------------------------|
               |             CPU time               |  2  J2CPU
               |------------------------------------|
               |           Connect time             |  4  J2CON
               |------------------------------------|
               |         Kilo-core-ticks            |  6  J2KCT
               |------------------------------------|
               |           Device time              |  8  J2DEV
               |------------------------------------|
J2CPUM 11      |  CPU time MSB   |      KCT MSB     | 10  J2KCTM
               |------------------------------------|
               |                                    | 12  J2NAME
               |           Program name             |
               |                                    | 14
               |------------------------------------|
               |  Default runtime system pointer    | 16  J2DRTS
               |------------------------------------|
               |     Receiver ID block pointer      | 18  J2MPTR
               |------------------------------------|
               |     Large data posting pointer     | 20  J2PPTR
               |------------------------------------|
               |     Large data posting count       | 22  J2PCNT
               |------------------------------------|
               |     Project-Programmer number      | 24  J2PPN
               |------------------------------------|
               |      DCN of first UFD block        | 26  J2UFDR
               |------------------------------------|
               | Pointer to window descriptor block | 28  J2WPTR
               |------------------------------------|
               |           ^T CPU time              | 30  J2CPUI
                ------------------------------------
```

Offset	Symbol	Description
0	J2TICK	This word is incremented at each clock interrupt when the job is executing. When the job is descheduled this value is converted to the equivalent number of 1/10th seconds and added to J2CPU. Any amount less than 1/10th second is left in this word. The units of this word depend on how fast the clock is interrupting. For a KW11L clock, running at 60 hertz, the units are 1/60th seconds.
2	J2CPU	This word contains the low order 16 bits of the total CPU time used by this job through the last time J2TICK was posted. The units of this value are 1/10th seconds.
4	J2CON	This word contains the total connect time, in minutes, for this job. Connect time is only computed while a job is logged in.
6	J2KCT	This word contains the low order 16 bits of the job's kilo-core-ticks. One kilo core tick is the use of 1 K-word of memory while executing for 1/10th second. Using 2 K-words for 1/10th second is two kilo-core-ticks.
8	J2DEV	This word contains the total device time for this job in device-minutes. A device-minute is the use of one device for one minute. Using two devices for one minute is two device-minutes, etc.
10	J2KCTM	This byte is the high order 8 bits of the kilo-core-ticks (see J2KCT above).
11	J2CPUM	This byte is the high order 8 bits of the CPU time (see J2CPU above).
12	J2NAME	These two words contain the program name in RAD50. The program name is specified using the .NAME system call. All the standard run-time systems issue this call to post the program name each time a program is run. The contents of this word are for information only, and are ignored by RSTS
16	J2DRTS	This word is the address of the run-time system descriptor block (RTS) of the default RTS for this job. When a running program exits, control returns to this default RTS. If the default RTS is no longer installed when the program exits the system default RTS is used instead. RTS blocks will be described in the second article of this series.
18	J2MPTR	If this job is a message receiver, this word contains the address of its receiver ID block (RIB). If this job is not a receiver this word is 0. RIB blocks will be described in the fourth article of this series.
20	J2PPTR	This word is used as a pointer to a monitor buffer to be used to transfer information to or from a user program buffer. It is normally used for large message send/receive buffer transfers. If the lower 5 bits of the pointer are 0 the pointer is an address into the small buffer area. If the lower 5 bits are not 0 the pointer is an address into the large buffer area that has been rotated left 7 bits.
22	J2PCNT	This word specifies the number of bytes to transfer to or from the buffer specified by J2PPTR.
24	J2PPN	This word contains the job's PPN. The low byte contains the group number and the high byte contains the user number. If the job is not logged in this word will be 0.
26	J2UFDR	This word contains the device cluster number (DCN) of the first cluster on SY0: of the UFD specified in J2PPN, above. The value of this word is undefined if the job is not logged in.
28	J2WPTR	If the job is attached to any resident libraries this word contain the address of the job's first window

. . . continued on page 58

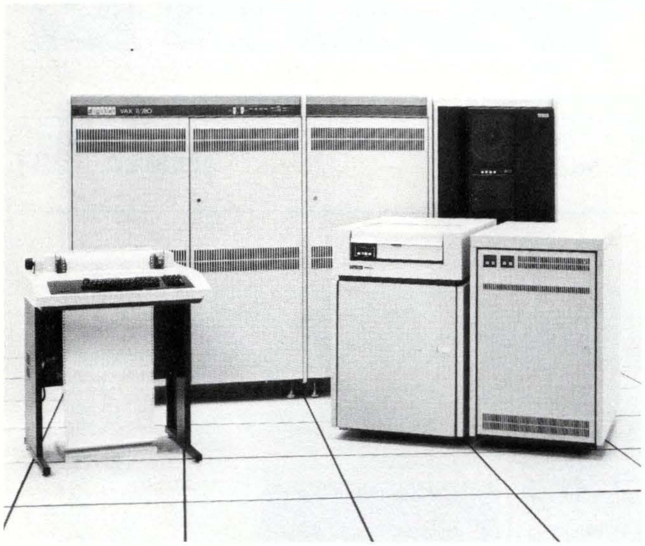
The VAX-SCENE

Number 2 (RSTS PROFESSIONAL, Vol. 3, No. 1) March 1981



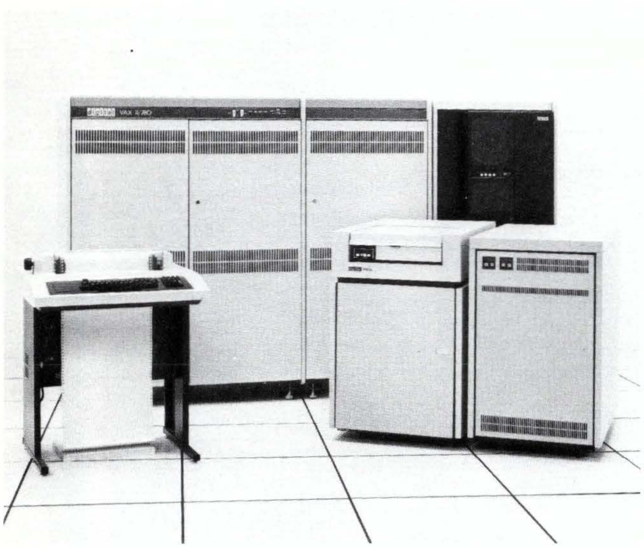
INSIDE:

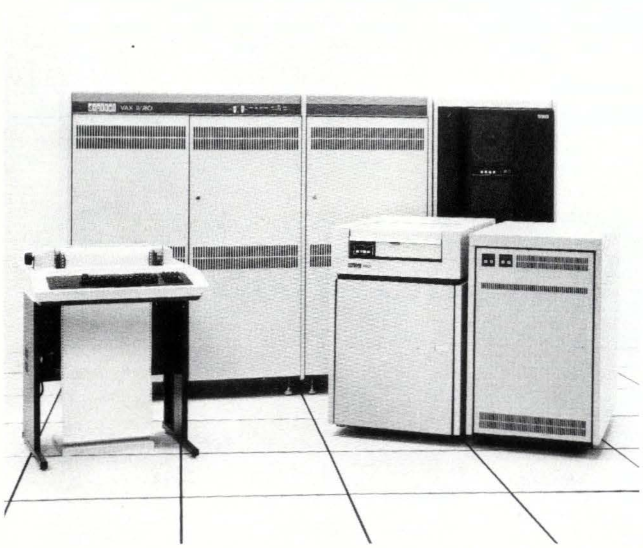
☐ The Other Story About VAX and VMS



The Other Story About VAX and VMS

By Ross W. Miller
President, Online Data Processing, Inc.





In 1977 just prior to the VAX being announced, I was discussing with a group of people some short-falls of the PDP11 computer. (1) Memory Management Concepts; (2) Memory Addressability; (3) Shared Memory Protection.

We also discussed different data types that are required to do a good job of data processing and the type of instructions one should have. I was greatly surprised and pleased when I went to the Fall DECUS of that year and DEC announced the VAX computer and described what was to me an almost perfect hardware design.

The design of the VAX is one of the best I've seen for current technology and it will give a considerable amount of flexibility in incorporating new concepts which have yet to be developed. Overall I would give DEC very high marks for the VAX machine. However, there are some problems. The purpose of this article is to provide some insight as to where those problems might be found, especially since many users are looking to the VAX to alleviate through-put problems and avoid costly conversions. Rather than spending a lot of time on the positive details of the VAX which you can read from all sorts of DEC literature, I'll take a few moments to enlighten you on some of the surprises and points of interest that are not discussed by DEC.

The single largest problem with the VAX computer is the fact that the DEC commercial product line people didn't want to become involved in the early design of the VAX computer and its operating system. I am sure there are going to be some people jumping up and down and saying, "Miller you are all wrong!. . . we were there." but unfortunately, they weren't there when they needed to be. Many things were designed into the VAX computer attempting to be sensitive to commercial interest in areas such as the ability to handle strings more efficiently, binary coded decimal data types and formatting instructions. However, DEC missed the boat in designing the operating system for commercial applications in such areas as operator interface, facilities for handling printers, record management efficiencies and what I would consider some significant oversights in the primary migration language for DEC's current customer base. This being VAX BASIC.

VAX BASIC — There are some good things and some bad things about it. First the good things. . . they have tried to maintain the interactive mode of programming which is a very positive feature of BASIC PLUS. Now the bad things... they didn't go far enough with the interactive features; secondly, it suffers greatly in the area of performance; and thirdly, by trying to give the impression that you can make minor modifications to your BASIC PLUS code and begin executing it in VAX BASIC, you could be misled into doing things that will be detrimental in the areas of performance and effectiveness use of the VAX machine.

Unwittingly, many of us who have been exposed to BASIC PLUS programming have developed several short-cuts and concepts on resolving data processing problems by automatically taking into account the restrictions that BASIC PLUS imposes. By using the same approach to develop your VAX BASIC program as you used with the BASIC PLUS program, you could be hurting yourself; i.e. structuring your program, data types used and, segmenting functions that should not be segmented. However, you automatically do it to circumvent BASIC PLUS restrictions.

With regard to VAX BASIC performance, we found that it is indeed fast in most bench marks, and specifically, in doing computational type operations. My point is: In a commercial application, a very small portion of your application program is actually involved in doing computation. Commercial applications deal mostly with string manipulation and record management functions; searching through a file, doing compares, changing data, updating information in records, etc. Therefore, it is my opinion that most bench marks, that have been run, do not reflect the true commercial environment. We have found that when applications written in VAX BASIC are compared to BASIC PLUS application of the same type (same concepts used) running on the 11/70, surprisingly the CPU time is comparable and this is disappointing.

In trying to determine exactly where the problems occur in VAX BASIC performance, we have traced through listings and micro-fiche. Many of the specific commercial routines that do string manipulation, formatting of data, and record management will do a call to a routine to perform a function which could have been executed in one to three machine instructions. As you trace these calls into the run time library, you find that these routines do calls to other routines, which do calls to other routines, etc. until finally, it gets down to the routine which actually performs the function and does the two or three instructions. It then starts to do the returns and unwinds. Naturally, all this is overhead and can increase the possibility of page faults as you wander your way through the mazes of twisty little passages, all alike, in the run time library.

It appears that a PDP11 programmer wrote the VAX BASIC programs and got carried away with the idea of modular programming forgetting that the VAX had built in instructions to accomplish what you want in machine instructions. It's not as fast as advertised, but it could be much faster.

DATA BASE MANAGEMENT — The VAX machine is the right machine to implement a good DATA BASE management system. As of this time, I have not seen a good DATA BASE management system for the VAX but nothing in the design of the operating system prevents it from being an excellent DATA BASE system. One of the major requirements for DATA BASE management is to have a large address space and do a reasonable job of cacheing frequently used sections of the DATA BASE. A 16 bit machine does not have the address space necessary and can cause you to do excessive thrashing of the disk by using DATA BASE system. A good DATA BASE system should reduce disk access for you.

RMS — Another significant point, is the use of RMS on the VAX computer. RMS is built into the operating system and this has its good points and its bad points. The positive aspect is they have reduced the overhead for RMS significantly from the PDP11 implementation. Those who are familiar with RMS on the 11 will be pleased with the VAX. However, there is still a significant amount of overhead in RMS due to the conservative implementation. An example of this is that whenever you do a put of a record which is 50 or 60 characters long, it will cause a write of the entire block back to the disk. To get better performance, it is preferrable to keep mass storage I/O to an absolute minimum since that is typically the most time consuming operation on a computer. Because of this conservative design, it is easy to get into I/O bottlenecks with RMS under VAX.

I am confident that performance improvements are being made to the RMS facilities. My biggest concern is that the VAX BASIC people do not seem to be concerned about performance issues; nor are they interested in providing additional features which will allow the user to control performance issues such as the many things that RMS will allow you to do but VAX BASIC doesn't provide a clean way of accessing.

VMS — Furthermore, there are some things in the operating system which definitely need to be improved.

1. If you are connected to a dial up line and you lose carrier on that line, your job gets killed. There is no way of preventing it. This isn't very convenient when you are updating multiple files simultaneously.
2. There isn't any way of doing a detach of a job and attaching to a job.
3. There is no way to do a force to a terminal for the system manager to help naive users or to get an X-offed LA 180 started again.
4. It is not possible to mount a tape under program control. The only way you can mount a tape, is at VMS DCL level.

If you can imagine a situation in a commercial environment where you need to process for several hours and then mount a tape, do further processing, change to a different tape to continue processing, you will definitely have problems doing this currently under VMS.

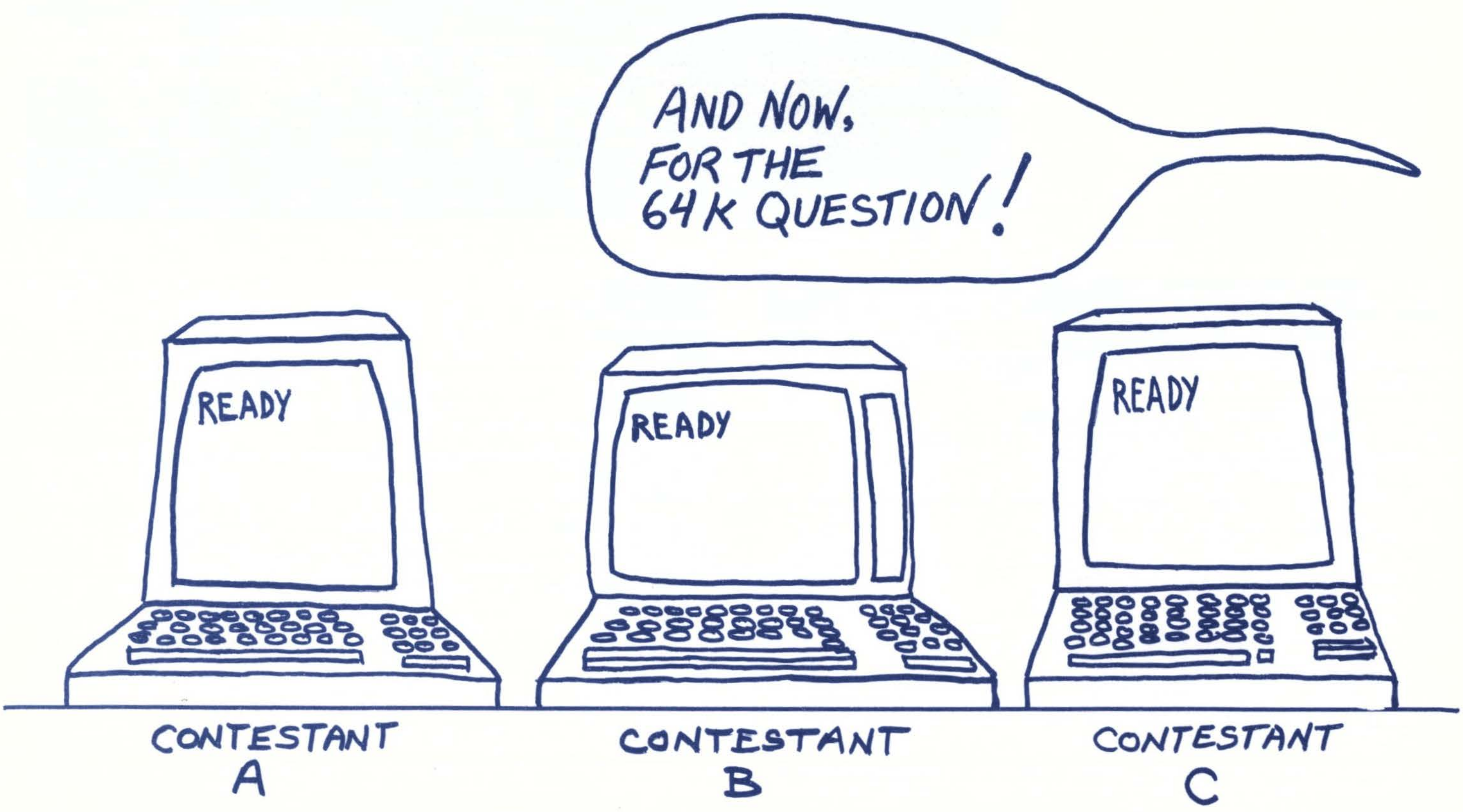
On the positive side, spooled printers and terminal are great. There needs to be additional improvements in forms management and controlling who can be an operator for a specific printer, such as a remote printer, but these again can be resolved.

Things such as being able to call routines written in any language from any other language is a beneficial feature. You can optimize certain functions that are not in the native language that you are using but are executed frequently. Even though I discourage mixing macro code and other such routines in business applications, we have used a few macro coded routines very effectively in business applications for forms management and screen handling routines which we developed prior to DEC having anything available.

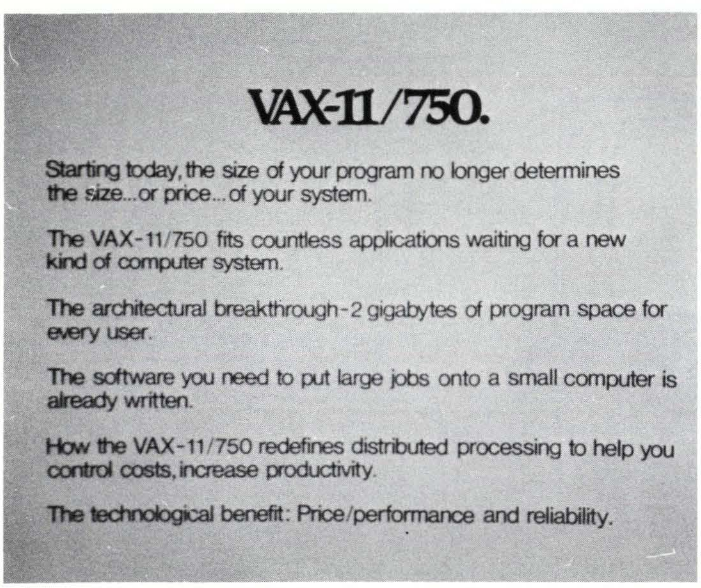
In addition, we found there were some DCL commands that were not provided so it was very easy for us to write the routines the way we wanted them and provide them for our clients as though they were part of the standard DCL set.

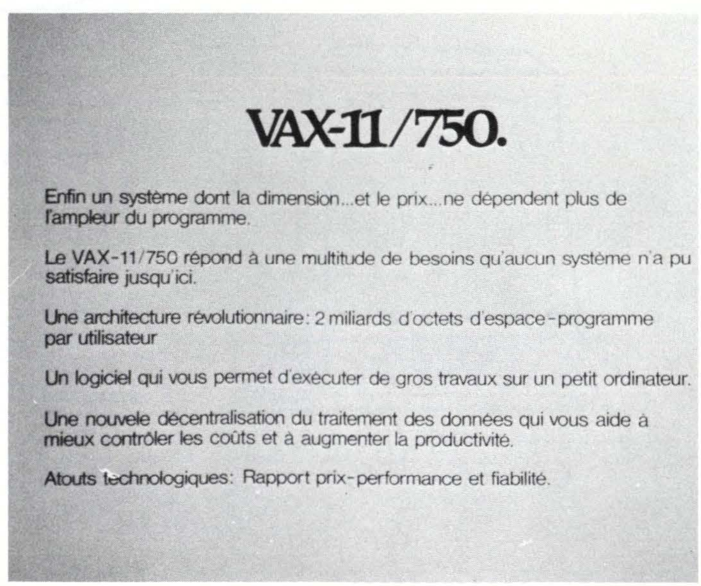
While you may think this has been a negative report, I wish to assure you that this is not my intent. However, I do want you to be aware of the things that are not advertised or widely known. Most of these problems are a matter of presenting them to the proper individuals to get them fixed or modified. Currently, DEC definitely needs strong input from the commercial world. DEC's turnaround time however, is 1½ to 2 years. That means that if you put in a request right now for a new feature or a change and DEC accepts it, you won't see it for 1½ to 2 years. That is the time it takes to implement, test and distribute the new feature or change. So . . . if you wait, you can calculate for yourself when it will be available.

In conclusion, I'm 100% behind VAX/VMS; but I would caution you to be extremely careful about assuming anything about the VAX system, and especially how to get performance out of it. We have been bit the hardest by using functions that worked extremely well on the PDP 11 but when we implemented them on the VAX we were actually working against VMS and thereby increasing system overhead.


AND NOW, FOR THE 64K QUESTION!
READY
READY
READY
CONTESTANT A
CONTESTANT B
CONTESTANT C

SCENES FROM CANADIAN DECUS 1981





RSTS/E Monitor Internals — Part 1

. . . continued from page 50

		descriptor block (WDB). If the job is not attached to any resident libraries this word is 0. WDB blocks will be described in the second article of this series.
30	J2CPUI	This word contains the CPU time used by this job as of the last time a ↑T mini-SYSTAT was taken.

1.4 IOB — I/O BLOCK

The I/O block is used to access information about each open channel. It contains one entry for each of the possible 16 channels. (Note: BASIC-PLUS only allows 12 channels because channel 0 is the user's terminal and channels 13-15 are used internally by the BASIC-PLUS interpreter).

The IOB is 16 words long. Each word is 0 if the corresponding channel is closed and non-zero if it is open. If the channel is open this word contains the address of the device data block (DDB) for non-disk devices, the window control block (WCB) for disk files on large-file systems or the small file control block (SCB) for disk files on small-file systems.

One exception is the entry for channel 0. This entry corresponds to the job's terminal. If the job is running detached this entry may still point to a terminal DDB. However the ownership byte within the DDB (DDJBNO) will not contain this job's job number.

```
Symbol Offset                                            Offset Symbol

          ---------------------------------------
          |  DDB ptr  for  channel  0            |  0
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 1   |  2
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 2   |  4
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 3   |  6
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 4   |  8
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 5   |  10
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 6   |  12
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 7   |  14
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 8   |  16
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 9   |  18
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 10  |  20
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 11  |  22
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 12  |  24
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 13  |  26
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 14  |  28
          |--------------------------------------|
          |  DDB, WCB or SCB ptr for channel 15  |  30
          ---------------------------------------
```

1.5 WRK — WORK BLOCK

The work block is a scratch work area used to hold information that is normally contained only in the user program area. This allows the job to be swappable during certain long monitor calls. The work block normally is used to store the FIRQB during FIP calls and the XRB during I/O transfers.

1.6 FIXED MEMORY LOCATIONS

Several fixed locations in low memory within the monitor are used to store information about the currently executing job. This information provides a shortcut for accessing the most used job control information. The information is as follows:

Address	Symbol	Description
518	JOB	This byte contains the job number (times 2) of the currently executing job. If this byte is 0 the null job is running. In that case the words that follow have special meanings, as noted. This byte is accessed in BASIC-PLUS by PEEK(518%) AND 255%.
519	NEXT	If this byte is non-zero it contains the job number (times 2) of the job that was scheduled to be the current job but is not yet swapped into memory. This job will start execution immediately upon gaining memory residency. In this case the job shown in JOB is "sub-scheduled" to make use of the available CPU time while waiting for the next job to swap in. This byte is accessed in BASIC-PLUS by SWAP%(PEEK(518%)) AND 255%.
520	JOBDA	This word contains the address of the job data block (JDB) for the currently executing job. If JOB contains a 0 this word will be 0.
522	JOBF	This word contains the address of the flag word, JDFLG, in the current job's JDB. If JOB contains a 0 this word will contain the address of a dummy value of 0.
524	IOSTS	This word contains the address of the JDIOST and JDPOST bytes in the current job's JDB. If JOB contains a 0 this word will contain the address of a dummy value of 0.
526	JOBWRK	This word contains the address of the work block (WRK) for the current job. If JOB contains a 0 this word will be 0.
528	JOBJD2	This word contains a pointer to the second job data block (JDB2) for the current job. If JOB contains a 0 this word will contain the address of a scratch location to be used for J2TICK.

530	JOBRTS	This word contains the address of the runtime system block (RTS) for the current job. If the current job is using the disappearing RSX runtime system this word will contain the address of the null RTS descriptor, NULRTS. If JOB contains a 0 this word will be 0.
532	CPUTIM	This word is the address of the value J2CPU in the second job data block (JDB2) for the current job.
534	JOBWDB	This word contains the address of the first window descriptor block (WDB) used by the current job. If the current job is not attached to any resident libraries, or if JOB contains a 0 this word will be 0.

1.7 JBSTAT, JBWAIT — JOB STATUS TABLES

Two tables, JBSTAT and JBWAIT, are used to determine if a job is runable. These tables each have one word for every possible job on the system, including the null job (job 0). They are accessed by using the job number times two as an offset.

When the scheduler wants to determine whether a job is runable it performs a logical AND of the bits in the job's JBSTAT entry with the job's JBWAIT entry. If the result is non-zero the job is runable. If the result is 0 the job is stalled waiting on something and cannot be run.

When a job stalls for any reason (such as doing I/O) the JBSTAT entry corresponding to its job number is set to 0 and a bit is set in the corresponding JBWAIT entry to show what the job is waiting for.

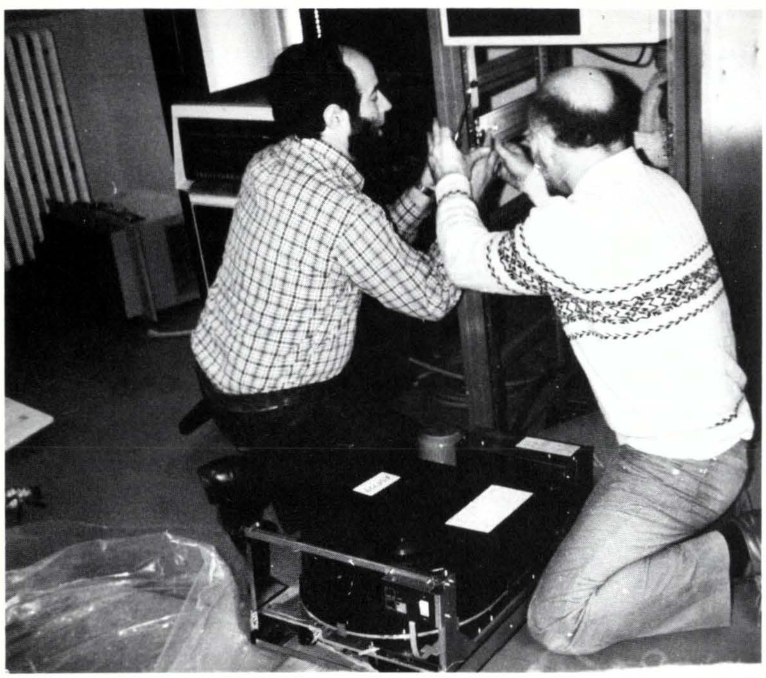
Stalling a program for hybernation clears both the JBSTAT and the JBWAIT entry. When the job is later reattached to a terminal JBWAIT will be updated to some non-zero value.

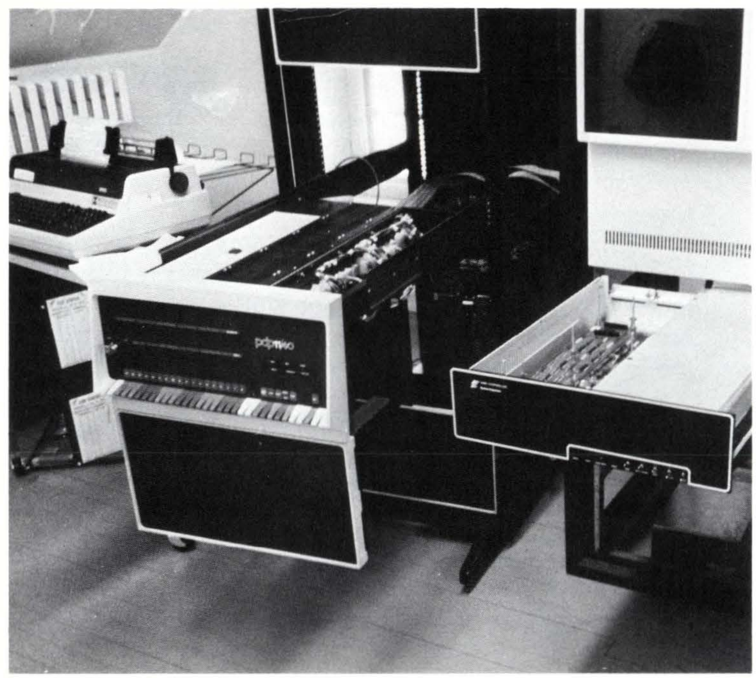
When an I/O completes or some asynchronous event occurs, such as a message receive or sleep timeout, the monitor sets an appropriate bit in the job's JBSTAT word. Depending on what the program was stalled for, this may or may not make the job runable. However, sooner or later the event the job is waiting for will complete and the job will be made runable again.

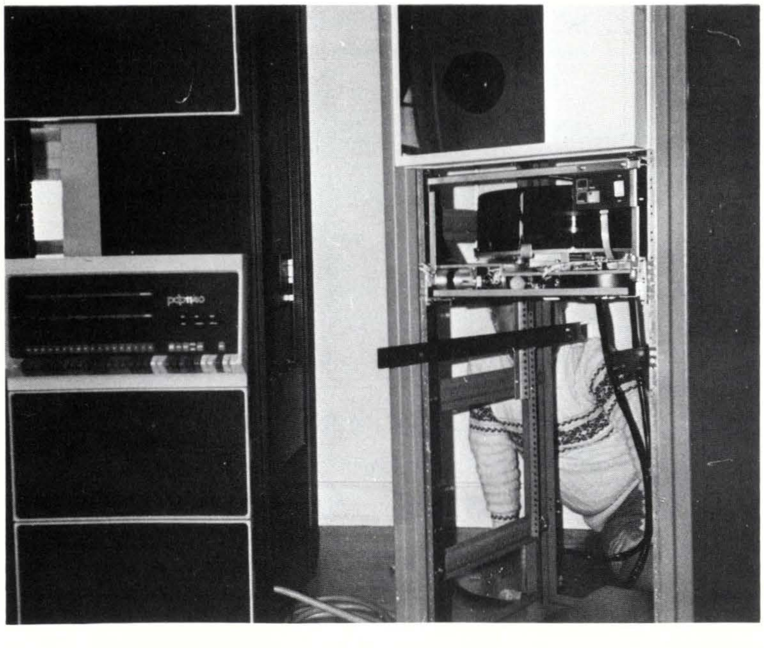
The bit values within the JBSTAT and JBWAIT entries have the following values:

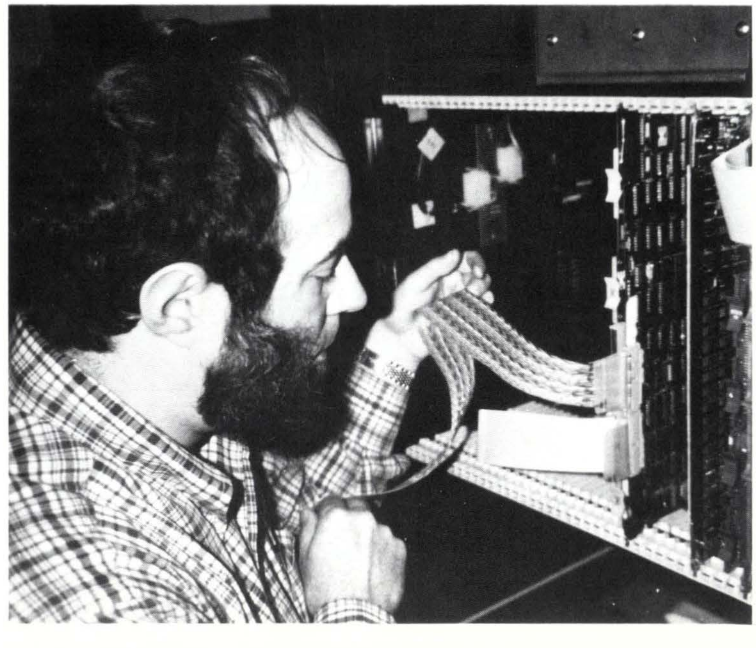
Bit	Symbol	Description
0	JS.SY	This bit is used for I/O on all synchronous devices. This includes disk, floppy disk, magtape, etc. but does not include terminals.
1	JS.KB	This bit is used for terminal input.
2		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
3		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
4		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
5		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
6		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
7		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
8		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
9		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
10		This bit is assigned at sysgen time for devices (such as LP:) that can stall and de-stall a job while still processing its I/O.
11	JSTEL	This bit is used for terminal output.
12	JSFIP	This bit is used for FIP (SYS call) waits.
13	JSTIM	This bit is used for timeouts from various time restricted events, such as .SLEEP, terminal input timeout, message receive timeout, etc.
14	JSBUF	This bit is used to wait for small buffers when no buffer space is available for I/O buffers.
15		This bit is not currently used but is reserved for future use.

In the next issue: MEMORY CONTROL









All Things BRIGHT and Beautiful

By Michael J. D. Mowat, B.Sc., M.B.C.S.
Department of Agriculture and Fisheries for Scotland

Creating a 'Proper' RSTS Disk with Contiguous Pre-extended Directories

If you do a weekly backup copy of your system disk, you could start the week with a restructured disk to get the most out of your system.

The advantages of having a well constructed system disk on a RSTS system are great. The features required on such a disk are that the directories should be contiguous and pre-extended and should be placed near the centre of the disk along with the SATT.SYS and the swapfiles, and that all other files should be physically contiguous as far as possible.

Such a disk is very fragile because user files which are not logically contiguous will fragment as they are modified, and if the system requires that accounts be removed or added the contiguity of the directories will gradually deteriorate. Also if a directory is zeroed its contiguity will be lost. It is possible to build a disk with contiguous directories by starting with a newly initialized disk and creating all the accounts, then prextending all the directories by opening null files and closing them.

With a system with a large number of accounts, this is an extremely time consuming process and it is not reasonable to repeat the process if accounts are created or deleted. System disks do however require to have back up copies made and if a faster method of producing a disk with a proper structure were available it would be possible to rebuild the disk at every backup session, taking account of any alteration in the account structure. Provided that the system can be taken off time sharing for the time necessary to copy the files using PIP the program BRIGHT.BAS described here can build the skeleton 'proper' disk very quickly (about 10 minutes for an 80 megabyte disk with 100 accounts).

The old disk is kept as a backup volume and it can be remounted in the event of a disaster. Recovery of a single file from the old volume is very quick and easy using PIP.

The prerequistes are:—

1. An initialized system disk with the system directories pre-extended and with swapfiles included. This disk must contain the program BRIGHT.BAS and may in addition contain a startup command file (using ATPK) to make the operation automatic. In this case it must also contain all cusp programs and command files used before account [1,2] is copied to the new disk.
2. The previous system disk with all changes made to the structure of the accounts.

The new disk is initalized as a system disk with the directories pre-extended and the SATT, swapfiles and the directory of [1.2] are placed. The program BRIGHT as it stands assumes that these are placed contiguously from cluster 20000 with a pack cluster size of 4. The program must be changed for other settings. When the disk has been initialized and any necessary files added it may be saved to another medium, or an image copy may be made to use the next time the system is rebuilt, to save having to go through the initialization dialougue again. Typically the skeleton disk would have the following.

Bootstrap
INIT.SYS
A SIL
BASIC.RTS
RT11.RTS
SWAP files
CRASH.SYS
INIT.BAC
PIP.SAV
UTILTY.BAC
BRIGHT.CTL (Command file to build the disk)
ERRINT.BAC
ATPK.BAC
LOGIN.BAC
LOGOUT.BAC
BRIGHT.BAS (or BAC)

The new pack is booted on DB0: and the old pack is mounted (physically and logically as a private pack) on DB1:. The program BRIGHT.BAS is then run. The first thing this program does is to build an MFD containing all the accounts from the old system disk. It then opens a contiguous file into which to build the directories and then builds a contiguous pre-extended directory for each file. Each directory will contain a null file called LOCK.UFD with a protection code of <63>. This is to prevent the directory being lost if the account is cleared with PIP *.*/DE from a non-priviliged user. PIP/ZE will still destroy the contiguous directory and should not be used. When all the directories have been built the file into which they were written is removed from the directory (without resetting the SATT). A preliminary trial on a disk bound job with the first disk built by this method showed a speed improvement of about 50% over a disk where most of the files were more or less contiguous but the directories were scattered.

When the directories have been built the files are copied from DB1: to DB0: using PIP.SAV. If your system uses a new files first directory structure, the disk should be initialized as NOT new files first so that a PIP *.* will not reverse the order. The program INVERT.BAS is then run to change to new files first. This program must run in account [1.1] as it must tell the monitor that the change has happened (since the disk descriptor is read only on mount).

The listings of BRIGHT.BAS and INVERT.BAS are as follows:—

BRIGHT.BAS

```
 1 EXTEND
 100 !THIS PROGRAM WILL RUN ON A SYSTEM DISK AND WILL CREATE THE SAME  &
     !ACCOUNTS AS ON AN OLD SYSTEM DISK WHICH MUST BE MOUNTED AS A     &
     !PRIVATE DISK ON ANOTHER DRIVE.
 110 !THE PROGRAM IS WRITTEN TO HANDLE THE NEW DISK ON DB0: AND THE    &
     !OLD DISK ON DB1:
 120 !THE PROGRAM WILL EXTEND THE DIRECTORIES OF ALL ACCOUNTS TO       &
     !SEVEN CLUSTERS AND MAKE THEM CONTIGUOUS ABOVE THE SATT.SYS       &
     !WHICH SHOULD BE PLACED AT BLOCK 80000 (CLUSTER 20000).
 130 !THE SWAP FILES MAY BE PLACED HERE TOO.
 140 !THE PROGRAM ASSUMES A PACK CLUSTER SIZE OF 4.
 200 PAKCLU%=4% !THE PACK CLUSTER SIZE
 220 UFDEXT%=7% !THE NUMBER OF CLUSTERS REQUIRED FOR EACH UFD     &
     !MUST HAVE VALUE BETWEEN 2 AND 7.
 2000 GOSUB 5000 !CREATE ACCOUNTS
 2020 GOSUB 7000 !SET UP DIRECTORY SPACE
 2040 GOSUB 12000 !REMOVE UFDFIL.SYS
 2050 GO TO 32767
 5000 REM SUBROUTINE TO CREATE ACCOUNTS
 5010 REM ACCOUNTS CREATED ON DB0: SAME AS THOSE ON DB1:
 5020 N%=3% !ACCOUNT INDEX
 5030 ON ERROR GO TO 6000 !ERR=5 IF ACCOUNTS ARE EXHAUSTED
 5040 S$=SYS(CHR$(6%)+CHR$(14%)+CHR$(N%)+CHR$(SWAP%(N%))+STRING$(18%,0%)
           +CHR$(68%)+CHR$(66%)+CHR$(1%)+CHR$(255%)) !FIND ACCOUNT
 5041 !NEW CLUSTER SIZE CODED AS QUOTA 4096=4 8192=8 16384=16
 5042 S1$=MID(S$,27%,1%)\S2$=MID(S$,28%,1%)\S3$=MID(S$,29%,2%)
 5043 IF S1$<>CHR$(0%) GO TO 5050
 5044 S2%=ASCII(S2$)
 5045 IF S2%<>16% AND S2%<>32% AND S2%<>64% GO TO 5050
 5046 S3$=CHR$(S2%/4%)+CHR$(0%)
 5047 S2$=CHR$(0%)
 5050 ON ERROR GO TO 6200 !ERR=16 IF ACCOUNT EXISTS
 5060 S$=CHR$(6%)+STRING$(5%,0%)+MID(S$,7%,6%)+S1$+S2$+STRING$(8%,0%)    &
        +CHR$(68%)+CHR$(66%)+CHR$(0%)+CHR$(255%)+S3$
 5080 S$=SYS(S$)
 5090 GO TO 6450
 6000 IF ERR<>5 GO TO 25000
 6010 RESUME 6020
 6020 RETURN !ACCOUNTS CREATED
 6200 IF ERR<>16 GO TO 25000
 6210 RESUME 6450 !DO NEXT ACCOUNT
 6450 N%=N%+1%
 6460 GO TO 5030
 7000 B%=0%
 7005 !THE FIRST PART OF THIS SUBROUTINE FINDS THE TOTAL NUMBER          &
      !OF BLOCKS REQUIRED BY THE EXTENDED UFDS FOR ALL ACCOUNTS          &
      !EXCEPT [0,1],[1,1] AND [1,2]
 7010 ON ERROR GO TO 25000
 7020 OPEN '[1,1]' FOR INPUT AS FILE #1%, MODE 16384%
 7040 DIM #1%,U%(3583,7)
 7060 CLU%=U%(31%,0%)
 7080 !WALK THROUGH DIRECTORY
 7100 PTR%=FNLINK%(U%(0%,0%))
 7120 GO TO 7340 UNLESS PTR%
 7140 GOSUB 7200
 7160 PTR%=FNLINK%(U%(PTR%,0%))
 7180 GO TO 7120
 7200 REM FIND NUMBER OF BLOCKS REQUIRED
 7220 IF (U%(PTR%,4%) AND 64%)=0% GO TO 7320
 7225 T%=U%(PTR%,1%) !DONT COUNT [0,1],[1,1] OR [1,2]
 7230 IF T%=1% OR T%=257% OR T%=258% GO TO 7320
 7240 AB%=FNLINK%(U%(PTR%,6%))
 7260 UFDCLU%=U%(AB%,7%)
 7280 B%=B%+UFDCLU%*UFDEXT%+(UFDCLU%/5%)*4% !ALLOW SPACE TO ALLIGN UFD CLUSTERS
 7320 RETURN
 7340 !TOTAL SIZE FOR UFDS IS B%
 7350 !OPEN A FILE WHICH WILL CONTAIN THE UFDS. EACH UFD WILL CONTAIN     &
      !AN ENTRY FOR A NULL FILE 'LOCK.UFD'
 7370 !THIS INSTRUCTION MAY BE CHANGED IF THE DIRECTORIES ARE      &
      !NOT TO BE PLACED AT THIS LOCATION.
 7380 OPEN '[0,1]UFDFIL.SYS/PO:20000' FOR OUTPUT AS FILE #2%,FILESIZE B%, MODE 16%
 7400 GO TO 9000
 7420 DEF FNLINK%(L%)                    &
       =(((L% AND 3584%)/512%)*CLU%      &
       +(SWAP%(L% AND -4096%)/16%))*32%  &
       +((L% AND 496%)/16%)
 8999 !ROOT% IS THE START OF UFDFIL.SYS IN CLUSTERS
 9000 CSY$=CHR$(6%)+CHR$(-26%)+CHR$(0%)+CHR$(16%)+CHR$(1%)+CHR$(0%)        &
          +CHR$(52%)+CHR$(132%)+CHR$(244%)+CHR$(38%)+CHR$(187%)+CHR$(122%) &
          +CHR$(1%)+STRING$(9%,0%)+CHR$(68%)+CHR$(66%)+CHR$(0%)+CHR$(255%)
 9020 ROOT%=SWAP%(CVT$%(MID(SYS(CSY$),3%,2%)))
 9040 IF ROOT%<0 THEN ROOT%=ROOT%+65536
 9060 DISP%=0%
 10000 !THE UFD ENTRY IS MADE DIRECTLY INTO THE FILE 'UFDFIL.SYS'
       !AND THE APPROPRIATE DCN VALUES ARE INSERTED INTO THE MFD
 10080 CLU%=U%(31%,0%)
 10100 !WALK THROUGH DIRECTORY
 10120 PTR%=FNLINK%(U%(0%,0%))
 10140 GO TO 10480 UNLESS PTR%
 10160 GOSUB 10220
 10180 PTR%=FNLINK%(U%(PTR%,0%))
 10200 GO TO 10140
 10210 !GET PPN
 10220 N%=U%(PTR%,1%)
 10230 !SKIP IF NOT AN MFD ENTRY
 10240 IF (U%(PTR%,4%) AND 64%)=0% GO TO 10440
 10300 DCN=U%(PTR%,7%)
 10310 GO TO 10440 IF DCN<>0
 10320 !UFD ALREADY EXISTS
 10340 AB%=FNLINK%(U%(PTR%,6%))
 10360 UFDCLU%=U%(AB%,7%)
 10380 GOSUB 11000 !MAKE THE ENTRY
 10440 RETURN
 10480 CLOSE #1%\CLOSE #2%
 10500 RETURN
 11000 PL%=ROOT%+DISP%  !DCN OF A UFD
 11005 IF UFDCLU%>PAKCLU% GO TO 11350
 11010 STORE%=DISP%*PAKCLU%+1% !CORRESPONDING BLOCK OF UFDFIL.SYS
 11020 U%(PTR%,7%)=PL%
 11030 NUMBER%=UFDEXT%*UFDCLU% !NUMBER OF BLOCKS IN UFD
 11040 DISP%=DISP%+NUMBER%/PAKCLU% !GET NEXT DISPLACEMENT (CLUSTERS)
 11050 !BUILD A DUMMY UFD.
 11060 FDCM$=CHR$(UFDCLU%)+CHR$(0%)
 11080 FOR I1%=0% TO UFDEXT%-1%  !BUILD THE FDCM
 11100 Z%=PL%+I1%*UFDCLU%/PAKCLU%
 11120 FDCM$=FDCM$+CHR$(Z%)+CHR$(SWAP%(Z%))
 11140 NEXT I1%
 11180 REC1$=CHR$(32%)+CHR$(0%)+STRING$(2%,255%)+STRING$(8%,0%)
 11200 REC1$=REC1$+CHR$(N%)+CHR$(SWAP%(N%))+CHR$(52%)+CHR$(132%)
 11220 REC1$=REC1$+CHR$(1%)+CHR$(0%)
 11225 REC1$=REC1$+STRING$(12%,0%)+CHR$(4%)+CHR$(0%)
 11230 REC1$=REC1$+STRING$(2%,0%)
 11240 REC1$=REC1$+CHR$(91%)+CHR$(77%)+CHR$(192%)+CHR$(68%)+CHR$(52%)+CHR$(132%)
 11246 REC1$=REC1$+CHR$(16%)+CHR$(63%)+STRING$(2%,0%)+CHR$(16%)+STRING$(3%,0%)
 11248 REC1$=REC1$+STRING$(448%,0%)+FDCM$
 11250 RECS$=STRING$(496%,0%)+FDCM$
 11252 FIELD #2%,512% AS Q$
 11253 LSET Q$=REC1$
 11254 PUT #2%,RECORD STORE%
 11256 FOR I1%=1% TO NUMBER%-1%
 11258 FIELD #2%,512% AS Q$
 11259 LSET Q$=RECS$
 11260 PUT #2%,RECORD STORE%+I1%
 11280 NEXT I1%
 11300 !ACCOUNT UFD PRE-EXTENDED
 11320 RETURN
 11340 !TEST ALLIGNMENT OF UFD CLUSTERS
 11350 IF UFDCLU%=8% AND (PL% AND 1%)=1% GO TO 11010
 11360 IF UFDCLU%=16% AND (PL% AND 3%)=1% GO TO 11010
 11370 DISP%=DISP%+1%\GO TO 11000 !TRY AGAIN
 12000 OPEN '[0,1]'FOR INPUT AS FILE #1%,MODE 16384%
 12005 !KILL THE MFD ENTRY FOR THE FILE 'UFDFIL.SYS' WITHOUT
       !CLEARING THE SATT FOR THE UFD AREA. LEAVE THE GARBAGE.
 12010 PTR%=FNLINK%(U%(0%,0%))
 12020 GO TO 12150 UNLESS PTR%
 12100 IF RAD$(U%(PTR%,1%))<>'UFD' GO TO 12300
 12110 IF RAD$(U%(PTR%,2%))<>'FIL' GO TO 12300
 12120 IF RAD$(U%(PTR%,3%))<>'SYS' GO TO 12300
 12130 U%(LAST%,0%)=U%(PTR%,0%)
 12140 CLOSE #1%
 12150 RETURN
 12300 LAST%=PTR%
 12310 PTR%=FNLINK%(U%(PTR%,0%))
 12320 GO TO 12020
 25000 ON ERROR GO TO 0
 32767 END
```

INVERT.BAS

```
 5 !PROGRAM TO CHANGE DISK TO NEW FILES FIRST. USE WITH CARE.
 6 !OPERATES ON DB0: ONLY
 7 !MUST BE RUN ON [1,1]
 9 !TELL THE MONITOR
 10 M$=MID(SYS(CHR$(6%)+CHR$(-3%)),19%,2%)
 20 M%=SWAP%(CVT$%(M$))
 30 S%=PEEK(M%)
 40 S%=S% AND -513%
 50 S%=S%+512%
 60 S$=CHR$(S%)+CHR$(SWAP%(S%))
 70 !POKE
 80 S$=SYS(CHR$(6%)+CHR$(-6%)+M$+S$)
 999 !ALTER THE DISK
 1000 DIM #1%,A%(7)
 1010 OPEN '[1,1]' FOR INPUT AS FILE #1%,MODE 16384%
 1020 A%(5%)=A%(5%) AND -513%
 1030 A%(5%)=A%(5%) + 512%
 1040 CLOSE #1%
 32767 END
```

A typical start up command file for use with ATPK is as follows. You can start it and leave it. The system will start up from the new disk when it is finished. The time taken will depend on the complexity of your accounts. For a system with 120 accounts and 120000 blocks in 7000 files the time was about two hours on an RP04. (considerably longer the first time from the rubbishy old disk).

```
RUN [1,2]UTILTY
SEND KB0: Building Directories
ADD RT11
MOUNT DB1:RSTS7/PRIVATE
UNLOCK DB1:
EXIT
RUN [1,2]BRIGHT
RUN [1,2]UTILTY
SEND KB0: Transferring files.
EXIT
RUN [1,2]PIP.SAV
DB0:[0,1]*.*/PO:20000=[0,1]*.RTS/PR:NOWARN
! FREQUENTLY ACCESSED FILES MAY BE PLACED BY INSERTING
! COMMANDS HERE
^Z
! THIS OPTIONAL FILE HOLDS SPACE NEAR THE CENTRE FOR NEW FILES
! AND WORK FILES.
OPEN 'FILLER.TEM/PO:18000' AS FILE #1%,FILESIZE 16384%
CLOSE #1%
RUN [1,2]PIP.SAV
DB1:[1,2]*.WRK/DE
DB1:[1,2]*.TMP/DE
DB1:[1,2]*.COM/DE
DB0:[*,*]*.*=DB1:[*,*]*.*/PR:NOWARN
DB0:[1,2]BRIGHT.BAS/DE
FILLER.TEM/DE
^Z
RUN [1,2]UTILTY
SEND KB0:File transfer complete.
REMOVE RT11
LOCK DB1:
DISMOUNT DB1:
EXIT
HELLO 1,1;SYSTEM
RUN $INVERT
HELLO 1,2;FILES
$@[1,2]RTS.CMD
$@[1,2]SPOOL.CMD
$@[1,2]CCL.CMD
$@[1,2]TTY.CMD
$@[1,2]PLOT.CMD
RUN $UTILTY
REMOVE LOGICAL LB
ADD LOGICAL SY0:[1,1]LB
ENABLE CACHE
LOGINS
SEND ALL RSTS/E V07.0 NOW UP AND RUNNING
SEND ALL Please say HELLO
EXIT
DIR BRIGHT.DIR=[*,*]*.*/S
QUE LP0:/FO:NARROW/NH/DE=BRIGHT.LOG,BRIGHT.DIR
```

The procedure to initialize the disk, and to position the swap files, and run time systems should be similar to the following description.

```
Instructions for the Weekly Disk Copy.

1)      Run shutup
2)      Mount a scratch disk on DB1:
3)      DSKINT DB1: with the following dialogue
Option: DS
DD-MMM-YY? Date or LF
00:00 ? Time or LF
Disk? DB1
A description of the disk may be output.
Pack ID? RSTS7
Pack cluster size?  4
SATT.SYS base?  80000
MFD password?  SYSTEM
MFD cluster size? 16
Pre-extend directories <no>? Y
PUB, PRI, or SYS? SYS
Library password? FILES
Library UFD cluster size? 8
Library account base? 84492
Date last modified <yes>? N
New files first <no>? N
Use previous bad block info <yes>? Y (Not asked if not a RSTS disk)
Format <no>? N  (unless a new disk)
Patterns? 1 (but use 4 at least if not a RSTS disk)
Proceed? Y (if happy with dialogue)
4) Continue as follows when the disk is initialized.
Option: COPY
DD-MMM-YY?
HH:MM?
To which disk? DB1
Enabling only console, disks, and tapes.
RSTS V7.0 (DB1)
Option: HARDWR
  HARDWR suboption? HERTZ 50
  HARDWR suboption? CSR
    Controller with non-standard address? DZ0
    New controller address? 160040
  HARDWR suboption? VEC
    Controller with non-standard vector? DZ0
    New vector address? 310
  HARDWR suboption? CSR
    Controller with non-standard address? DZ1
    New controller address? 160050
  HARDWR suboption? vec
    Controller with non-standard vector? DZ1
    New vector address? 320
  HARDWR suboption? LIST
(CHECK THAT THE HARDWARE LISTING IS CORRECT)
  HARDWR suboption? LF
  3 changes being made

Rebooting . . .

RSTS V7.0 (DB1)

Option: INSTALL
Sil? RSTSV7


Option: BOOT DB0

RSTS V7.0-07 Marine Lab (DB0)

Option? ST
.............
.............
.............
Command File Name? PROPER.CTL(This copies files required to run BRIGHT)
............
............
When system shuts down put old disk on DB1: and new disk on DB0:
Then start with command file BRIGHT.CTL
(note that DEFAULTS must first be set using BASIC as the RTS)
The system will come up (without DB1:) when the disk copy is complete.
```

A similar process may be used with a disk used as a non system disk but of course automatic start up would not be possible unless you have more than two drives, in which case the process could be done on line. If you have only two the non system disk would ahve to have a system on it in order to run BRIGHT and do the copying.

```
              CLEN  CLEN
              CLEN  CLEN
              CLEN  CLEN
               LENUPCLE  PCLE   CL  UP   EN PC
                ENUPCL  UP  EN  CL  UP   ENUPCL
                 NUPC   UP  EN  CL  UP   EN
                 NUPC   UP  EN  CL  UP   EN
                 NUPC   UP  EN  CL  UP   EN
                 NUPC    PCLE    LENU    EN

                          UPCLENUP  UN
                         NUPCLENUP  UN              CL
                        ENUPC       UN
                        ENUP        UN PC    PCLE   CL   PCLE    LENU
                        ENUP        UNUPCL  UP  EN  CL  UP  EN  CL  UP
                        ENUP        UN  CL  UP  EN  CL  UP      CLENUP
                        ENUPC       UN  CL  UP  EN  CL  UP      CL
                         NUPCLENUP  UN  CL  UP  EN  CL  UP  EN  CL  UP
                          UPCLENUP  UN  CL   PCLE   CL   PCLE    LENU
```

```
1!THIS SHORT PROGRAM DEMONSTRATES CLENUP ON A SMALL BUBBLE SORT
2DIML(20%):L(I%)=RNDFORI%=1%TO20%!BUILD UP SORT LOOP
3&'STARTING SORT AT';TIME$(0%)
4Z1%=0%:Z%=20%:WHILEZ%:FORJ%=1%TOZ%-1%:IFL(J%)<=L(J%+1%)THEN5ELSEL=L(J%):
L(J%)=L(J%+1%):L(J%+1%)=L:Z1%=J%!WE NEEDED TO FLIP HERE
5NEXTJ%:Z%=Z1%:Z1%=0%\NEXT:&'SORTING ENDED AT';TIME$(0%)
6STOP
```

```
                         PCLE   EN PC
                        UP  EN  ENUPCL
                        UP  EN  EN
                        UP  EN  EN
                        UP  EN  EN
                         PCLE   EN
```

```
 00001    !THIS SHORT PROGRAM DEMONSTRATES CLENUP ON A SMALL BUBBLE SORT
 00002    DIM L(20%)\
          L(I%) = RND
                  FOR I% = 1% TO 20%
                                  !BUILD UP SORT LOOP
 00003     PRINT 'STARTING SORT AT'; TIME$(0%)
 00004    Z1% = 0%\
          Z% = 20%\
          WHILE Z%\
                  FOR J% = 1% TO Z% - 1%\
                          IF L(J%) <= L(J% + 1%) THEN 5
                          ELSE L = L(J%)\
                          L(J%) = L(J% + 1%)\
                          L(J% + 1%) = L\
                          Z1% = J%
                                  !WE NEEDED TO FLIP HERE
 00005            NEXT J%\
                  Z% = Z1%\
                  Z1% = 0%\
          NEXT\
           PRINT 'SORTING ENDED AT'; TIME$(0%)
 00006    STOP
```

TECHNICAL NOTES

By The RSTS "Pro" Staff

An understanding of what your computer is doing at the basic level is important in evaluating performance (or lack of it) and deciding what can be done to improve it.

This will be the first in a continuing series to help you understand the physiology of a PDP 11. I am not a hardware expert, lots of people think I am not a software expert, but I am willing to stick my neck out and let you have the ideas as I understand them. As an editor, I will start and keep the ball rolling. I will, when offered, submit other comments and articles to the newsletter. I ask that you send them to me so that I may exercise my editor's perogative and put them in some order.

This first article will deal with input and output. All PDP 11 devices have associated with them a set of registers which may be accessed as if these registers were in the main memory. Each device has associated with it at least two registers: a) a control and status register and, b) a data buffer register. Some devices have more than two registers. For example, one of the simplest devices attached to the PDP 11 may be the console terminal. It has four addresses or registers associated with it. They are: 1) printer data, 2) printer status, 3) keyboard data, and 4) keyboard status. You will have by now noticed that the console is really two devices: a reader and a printer. To print a character on the console printer, all that is necessary is that you deposit via a move or similar instruction the seven bit ASCII code for the character you wish to print into the printer's data register and it will be printed. The printer status register will contain a bit that will say done or ready. Since the CPU runs faster than the printer can print, it is necessary after printing a character to check to make sure that the character has been printed before depositing the next character in the data buffer register to be printed. In an analogous fashion, when a character is typed at the keyboard, it can be 'read' by reading the data register of the keyboard. After 'reading,' a bit in the status register is set to indicate that no data is currently available in the data register. This means, of course, that you now have read the data that was in there. Note that reading a character from the keyboard does not echo it on the terminal printer. This must be done by putting the data that has been read into the printer data register thus echoing the character read.

Notice that there are no true I/O instructions; only the transfer from memory location to memory location is necessary in order to do input or output.

Although this type of input/output programming is simple and straightforward, it would be inefficient in the case of trying to control a multi-terminal system. For instance, a system that had 16 keyboards attached to it would necessitate that the controlling program periodically scan the data or control registers of the keyboard to see if anything had been input. If nothing had been input on that keyboard, it could continue to the next keyboard to see if there was any data there to be input. This technique is known as polling. That is, the controlling program polls or asks each terminal if there is data that it has to transmit. If not, the CPU continues around checking each terminal or device to see if there is any action to be taken. Thus, it becomes a totally CPU-bound job to keep checking your devices to see if there is any input to be done. A second and, as you can see, more efficient class of I/O programming is the so-called interrupt directed input and output. In the control and status register of our console terminal, there is a bit that enables an interrupt. That is, when data becomes available in the data register in the case of input or output, it causes the program to interrupt. What happens when the interrupt occurs from the console terminal? At every CPU cycle the PDP 11, via hardware, is checking to see if an interrupt has occurred. The PDP 11 utilizes a priority vectored interrupt scheme. Let's take them one at a time. The priority of an interrupt directs that if there is, in fact, more than one interrupt occurring at the same time which interrupt will be serviced first. If interrupt processing is in progress and a higher priority interrupt occurs, the lower order interrupt service routine will be interrupted by the higher priority interrupt.

The vectored interrupt scheme allows interrupt processing via hardware to occur directed to particular memory locations. For instance, an interrupt caused from the console terminal keyboard causes the interrupt to trap to location 60 while an interrupt from the console terminal printer causes a like interrupt to location 64. At these locations then, the user, in his program, may specify a particular address where the interrupt service routine is then to begin. This allows hardware to trap the particular location and relieve software of the job of determining which device interrupted and now what to do about it. Thus, when a character is typed on the console keyboard, if the interrupt enable bit has been set in the control and status register for the keyboard, an interrupt will occur. The interrupt will then trap to location 60 where the user could specify a service routine to handle an input type on the console terminal. In a similar fashion, when a power fail is detected by the system, a hardware interrupt is generated with a vector interrupt to location 24. Location 24 directs the processor to the proper service routine to save registers and necessary memory on the power fail sequence. We have now discovered two important concepts about input/output programming on the PDP 11: those of interrupt, priority interrupt and vector direction of the particular interrupt. These are not unique to the PDP 11 and can be found throughout the computer industry.

You will have noticed that it requires the central processor to be involved with every character transmitted to our console terminal. That is, if we wish to transmit to the console terminal data that is located in ten contiguous memory locations, it is necessary for the CPU to be involved with at least ten instructions, and in real fact, many more. The CPU must deposit or move from one real memory location to the data buffer or data register of the particular device each character separately for the transfer. This is all right for a slow device such as the DEC writer, but can cause problems for devices that go quite a bit more rapidly. When you consider that an LA 36 transmits 30 characters a second while a TU 10 magnetic tape transmits 36,000 characters a second, the overhead involved with transmitting characters through a magnetic tape would greatly exceed that of an LA 36. There is another type of I/O transfer available within the PDP 11 hardware. This is the so-called NPR transfer. The PDP 8 refers to a data break facility and a generalized computer term for this is devices that work on a cycle stealing basis. (Non Processor Request) Devices that operate on a cycle stealing basis steal an occasional cycle from the CPU in order to do their input and output. This means that the devices themselves have the ability to do data transfers from memory to the device or from the device to memory without processor intervention. These "smart" devices need only be told by the processor what location in memory to begin transfers from, how many locations to transfer, where on the peripheral device to begin putting the data and then these devices begin doing it by themselves. When they are finished, they automatically interrupt the CPU. Take the situation of transferring a file from disk to memory. The device, in this case the disk, needs to be told at what memory locations to begin putting the data, how many words (Bytes) to transfer, and where on the disk to begin getting the data.

Once these registers are loaded, the I/O begins to take place in the same fashion as a character was printed on the console terminal when the printer register was loaded with the data to be printed. When the number of words that we specified to be transferred has been transferred, the device automatically interrupts the CPU for more information. This is the so-called stealing of a cycle from the CPU. It would be possible to transfer many thousands of Bytes, or words, of information between peripheral devices only utilizing a very few cycles of the CPU. While this transfer is going on, the CPU is free for other instructions.

In the next article, we will deal with more specific high-speed, large data capacity devices, particularly RPO 2, 3, 4, 5 and 6 disk drives. We will also attempt to talk about the Unibus structure, how data is transferred over the Unibus and the Unibus extensions: the Fastbus and Massbus.

For more information about this article and future articles, I suggest the following reading. In 1968 or 1969, DEC published a book called, **Introduction to Programming.** Although this book is based on the PDP8 series of computers, many of the concepts have carried over to the PDP 11. Next, the **PDP 11 Peripherals Handbook** contains in its introduction and beginning chapters useful information on programming peripheral devices. The addresses and use of different control and status registers for different devices can be found in this particular book. I have a 1973 version of the **Peripherals Handbook** as well as the 1976 version and they are somewhat different. If you can get both of them, you are that much better off. In addition to the aforementioned, it is also useful to have the **Processor Handbook** for the particular PDP 11 that you happen to have. It doesn't hurt to have processor handbooks for PDP 11s that you don't have, also. There are differences you will find between the LSI 11 and the PDP 11/70. If you take the time to read and look through some of these books, you will find all of the information that we have talked about here contained in them as well as many examples which space precludes us from putting in the magazine.

I hope this first article has been interesting to all, useful to some and boring only to a few. I apologize for its inaccuracies and look forward to your comments.

BLDCTL: Control File Processor

By David Spencer

The concept of job control files is far from new. On RSTS/E we have as control file processors both BATCH and ATPK [and now QUE.11, ed.]. These programs accept stream input files with lines similar to typing at the terminal. All this is well and good, provided that every control stream only needs to do one thing, and do it the same way every time.

BLDCTL is a TECO program written to pre-process job control files. The most unique feature of BLDCTL is its ability to ask questions during the process phase, and in doing so, mold the output stream according to the responses.

General Concepts

BLDCTL was written in TECO mainly because TECO has a rich set of text processing commands already defined within it. However, any other language would have done the job just as well.

The commands to BLDCTL are found within the source control file itself. Text processing is triggered by a set of special characters found at the beginning of the text line.

; Internal comment, will not appear in the output stream.
& Displays line to TTY, becomes a "!" in output.
?

Full format: ?↑ input name[help test]default*min*max*code

Where:

"?"	Specifies accept user input.
"↑"	Indicates start of name for user input.
"input name"	Is replaced with user input
"["	Ends input name, starts help text.
"help text"	Displayed when user types "?".
"]"	Ends help text, starts default text.
"default"	Default input when flagged.
"*"	End of default value.
"min"	Minimum length of user input.
"*"	Separator.
"max"	Maximum length of user input.
"*"	Separator.
"code"	BLDCTL user input edit code.

To apply default, make "min" a zero. If the user enters a "?", then the help text will be displayed.

. Conditional command

.IF text = constant [or] .IF text < > constant:

.END

When found, the conditional in the ".IF" statement will be tested. If the test is TRUE, then the lines between the ".IF" and the ".END" will be INCLUDED in the output stream. If the conditional is NOT TRUE, then the lines between the ".IF" and the ".END" will be EXCLUDED from the output stream.

";" operator

The semi-colon character defines an internal comment. This allows lines that are useful to see in the source (such as modification histories) to be omitted from the output. I have seen BATCH chew up many minutes processing simple comments.

"&" operator

Use of a line beginning with an ampersand will print the remaining text to the terminal. This is useful for identification of streams, warning messages, etc.

"?" operator

This command is the heart of BLDCTL. It allows questions to be asked of the user at file build time. After the input is accepted, everywhere in the control file where the prompt name is found bracketed by up-arrows is replaced by the user response. This allows dates, numbers, Y or N answers, and all sorts of input to be inserted into the control stream. In addition, there is an unimplemented parameter called code. This number can be used in modified version of BLDCTL to edit user input.

".IF, .END"

The ".IF" conditional allows blocks of code to be inserted into or deleted from the output stream. There are two forms of the conditional. The first form is a compare by a "=", the second with a "< >". These tests act just as they look in BASIC. If the condition is false (Y = N), then the code from that point to the closing ".END" will be DELETED from the output stream. If the conditional is true, then the text remains. Please note

FINALLY, A TERMINAL FOR
DEC PROGRAMMERS

THE PROTERM 80* REDUCES:

Program development time............80%
Disc accesses.......................90% #
CPU time............................66%

PUT THE TEXT EDITOR IN YOUR TERMINAL...
NOT IN YOUR CPU !

PROGRAMMING MODE:

Escape key emits # in programming mode. These keys are lower case: % $! & ( ) + [ ]
60 variable length functions that can type to host, display, or type to host and display.
Samples of single key functions: LISTNH ** OPEN ~ FOR INPUT AS FILE # ** , CLUSTERSIZE
PRINT ASCII(MID(A$,I%,1%)) FOR I%=1% TO LEN(A$) ** GET #~%, BLOCK ** FOR I%=1% to ~%
PROTERM 80 enters fill in the blank mode at the ~ character. Control codes may be embedded.
Save/Send favorite phrase. Ex. RUN DB7:[213,159]VT5DPY`J10`%`T`Sn`D` (` =embedded <CR> code)
Editing features include insert/delete line/character, transmit line/page, copy line.
?Fatal and %Warning messages are trapped and displayed on the bottom line of the display.
The PROTERM 80 marks the line (not line #) that causes BASIC to generate a ?Fatal error.

If you don't like these load your own favorites. The PROTERM 80 is a PERSONAL terminal.

The PROTERM 80 has been used with DEC BASIC's, ROSS/V, DATA BOSS / 2, WS-11 and FINAR. #

ECHO CONTROL MODE:

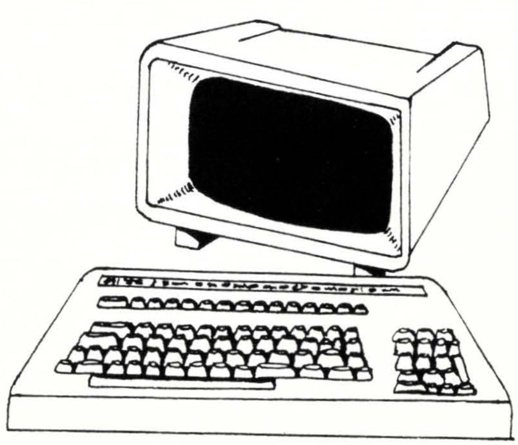
All the features DEC forgot. Field attributes include new/old, alpha, numeric, must fill.
Up to 64 downloadable functions. Type ahead with data compression. Operates on channel 0.
Local editing keys ^U, rubout, cursor left/right, delete/insert character. "Peek a boo".
If you can use INPUT LINE and remote cursor position ALL the rest is done by the PROTERM 80.
Provides a RSTS terminal environment on ANY host, <LF>, <FF>, <ESC>, etc. can be delimiters.

ANSI MODE:

If you want just a VT-100 type terminal, push a key and operate in ANSI mode, VT-52 mode, OR use 1-156 column widths with full editing, format protect, block mode and other features.

Reductions may be less for those that write perfect code the first time.

*
PROTERM 80 is the trademark of DTI Data Terminals Corporation, Medfield, Massachusetts
DEC VT-100 VT-52 are trademarks of Digital Equipment Corporation, Maynard, Massachusetts
ROSS/V is the trademark of Online Data Processing, Inc., Spokane, Washington
DATA BOSS / 2 is the trademark of Florida Computer, Inc., Miami, Florida
WS-11 is the trademark of Northwest Digital Sales, West Covina, California
FINAR is the trademark of Finar Systems Limited, New York, New York



FOR MORE INFORMATION:
telex #
924-325
call
(617) 359-4188
or write
45 WEST ST.
MEDFIELD MASS.
02052

dti DATA TERMINALS CORPORATION

that spaces in the conditional text are meaningful.

Another item worth noting is that ".IF" blocks can be nested within other ".IF" blocks. Of course, any inner ".IF" blocks will be deleted if the outer block is false.

Running BLDCTL

BLDCTL is designed to be entered by either normal run or CCL entry. The default extention on input is ".PCF" (Pseudo Control File). The output default is the common ".CTL".

On run entry, both the input file name and output file name are accepted from the keyboard. There is no default for the input file. The output file will, however, default to the input file name with the proper default extention.

For CCL entry, any CCL name will do. If only the CCL is given, then BLDCTL continues as if it were entered through RUN entry. If a filename is given, it is assumed to be the input file name and generates a default output file. Input and output file can be specified by an equals character, in the form of "out=in".

Final Notes

Although BLDCTL is a complete, functioning program the current version is not intended to be the last. I openly encourage modification to the primary program. Anyone with a reasonable knowledge of TECO should be able to make changes to the code.

The examples should reasonably demonstrate in short form the capabilities of BLDCTL. About the source listing: the only control characters in the source are the two escapes at the very end of the program. These have have been underlined to indicate them as such. Please don't be fooled by the alignment of the comments and don't use tabs to move out to the margin.

Any correspondence concerning BLDCTL can be sent to:

David Spencer
2901 South Sepulveda Suite 305
Los Angeles, Calif. 90064


```
! ************************************** !
!                                        !
!              BLDCTL.TEC                !
!                                        !
! Author: David Spencer                  !
!                                        !
! Audit trail                            !
!                                        !
! dd-mmm-yy  Who  Why                    !
! ---------  ---  ---------------------  !
! 04-Nov-80  DJS  First version          !
!                                        !
! Q-Registers                            !
! A$     Parse input file                !
! A%     Scratch                         !
! B$     Parse output file               !
! B%     Scratch                         !
! C$     Evaluate input by dispatch code !
! C%     Dispatch code                   !
! D$     Default input                   !
! E$     String equate macro             !
! E%     String equate result            !
! F$     Holds input file spec           !
! G$     Get input from user             !
! H$     Help text                       !
! I%     ".IF" depth counter             !
! J$     Left string argument            !
! K$     Right string argument           !
! L%     Line counter                    !
! M%     Minimum length of input         !
! N%     Maximum length of input         !
! P$     Input prompt text               !
! Q%     String equate operator (=, <>)  !
! R$     Input response from user        !
! S$     Scratch                         !
! T$     Delete ".IF" ballon             !
! X$     Open input file spec            !
! Y$     Open output file spec           !
!                                        !
! ************************************** !

@EI//                                         ! Alternate stream off  !

! ************************************** !
!                                        !
! Phase one: load macros                 !
!                                        !
! ************************************** !

! Load open input file macro             !

@^UA$
J @I/:ER/ ZL 27@I// 0XS 0K MSUS              ! Do full spec, eat #   !
G* R 0A-^^."= D | C '                         ! Insert spec, drop "." !
.US :@-S/./"S QSJ                             ! Found, go back to file!
                 |
                 ,QSJ @I/.PCF/                 ! Append default ext    !
                 '
0XF 0K                                        ! Save filespec         !
@^A/Opening / :GF @^A/ for input.../ 13^T 10^T
@I/:ER/ GF 27@I//
@I/"U @^A%?Can't find file or account, aborting...
% HK EK EX '/ -2XS -2K MS                     ! Check file, abort none!
@I/ER/ GF 27@I// 0XX 0K                       ! Open for real now     !
$

! Load open output file macro            !

@^UB$
Z"= GF :@-S/./"S R K ' '                      ! Use input for default !
J @I/:ER/ ZL 27@I// 0XS 0K MSUS              ! Do full spec, eat #   !
G* R 0A-^^."= D | C '                         ! Insert spec, drop "." !
.US :@-S/./"S QSJ                             ! Found, go back to file!
                 |
                 ,QSJ @I/.CTL/                 ! Append default ext    !
                 '
0XY 0K                                        ! Save filespec         !
@^A/Opening / :GY @^A/ for output.../
@I/:ER/ GY 27@I//
@I/"U 13^T 10^T '/ -2XS -2K MS                ! Check file, fix margin!
@I/EW/ GY 27@I// 0XY 0K                       ! Open for real now     !
$

! Load "dispatch code" processor         !

@^UC$
<
  MG                                          ! Do the input from user!
  0:QR-^^?"= :QR-1"= @^A/Help text is as follows:/ 13^T 10^T
                     :GH 13^T 10^T F<         ! Give user help        !
                    '
            '
  :QR"= QM"= .UA GD QA,.XR QA,.D 0;           ! Pass default if set   !
             |
             ,@^A/%No default allowed, type "?" for help text./ 13^T 10^T F<
             '                                ! Otherwise an error    !
      '
  :QR-QM"L @^A/%Input must be at least / QM:= @^A/ character/
           QM-1"N ^^s^T ' @^A/ long./
           13^T 10^T F< '                     ! Too short             !
  :QR-QN"G @^A/%Input must be no more than / QN:= @^A/ character/
           QN-1"N ^^s^T ' @^A/ long./
           13^T 10^T F< '                     ! Too long              !
  0;                                          ! Input must be good    !
>
$

! Load string equate macro               !

@^UE$
QIUS %I ^[ 0K @^US/<=/ -1UE -1UQ              ! Bump count, init scan !
:@S/^EGS/"S -D 0A-^^>"= D 0UQ '               ! Find equate, set flags!
0XJ 0K L 2R 0XK 0L K                          ! Save left, right args !
.UA J :QJ-:QK"N 0UE                           ! Test lengths          !
               |
               GJ J @I/::S/ GK 27@I// 0XK 0K  ! Leave first arg, build!
                                              ! macro to test with 2nd!
               MKUE J :QJD                    ! Test and save result, !
               '                              ! clean up afterward    !
QAJ (QE+QQ+1)"= MT ' '                        ! Delete text if false  !
$
```

```
! Load user input get macro             !

@^UG$
ET#32768ET .UA J :GP
<
  ^TUB
  QB-  3"= HK EK EX              '
  QB-  4"= 0K 13^T 10^T :GP F< '
  QB- 10"= 13^T 0;               '
  QB- 12"= 13^T 0;               '
  QB- 13"= ^TUB 0;               '
  QB- 18"= 13^T 10^T :GP 0T F< '
  QB- 21"= 0K 13^T 10^T :GP F< '
  QB- 26"= 13^T 10^T HK EK EX   '
  QB- 27"= 13^T 10^T 0;         '
  QB-127"= 0^Q"= 7^T
                  |
                  R ET&2"= 0A^T
                           |
                           0A-32"L 2US
                                   |
                                   1US '
                           QS< 8^T 32^T 8^T > '
                  D '
           F< '
  QB@I//
>
0XR 0K QAJ
$

! Load text delete macro                !

@^UT$
< ::@S/.IF /"S 0L %I ^[ '
  ::@S/.END/"S (QI-1)UI '
  0L K QS-QI"N .-Z; >   '
$

! ************************************ !
!                                      !
! Phase two: parse CCL command         !
!                                      !
! ************************************ !

:@S/ /"S 0K | HK '                                  ! Remove CCL name        !
Z"N                                                 ! CCL found              !
   :@S/=/"U ZJ 0XS J GS @I/=/ '                     ! Insure specs           !
   -D 0XY 0K ZJ MA                                  ! Open input file        !
   GY MB                                            ! Open output file       !
  |
   @^A/BLDCTL   V01.00/ 2< 13^T 10^T >              ! Display header         !
   @^UP/File for input? /  MG GR :QR"= HK EK EX '   MA
                                                    ! Get input file         !
   @^UP/File for output? / MG GR MB                 ! Get output file        !
  '
MX MY                                               ! Now open files up      !


! ************************************ !
!                                      !
! Phase three: Main routine            !
!                                      !
! ************************************ !

13^T 10^T                                           ! Space a line           !

< ^N"= A > '                                        ! Pull in all the file   !

< 0A-^^;"= K | L ' .-Z; >                           ! Remove internal remark!

J 0UL                                               ! Zero line counter      !
<
  0A-^^&"= D @I/!/ T '                              ! Display line command   !
  0A-^^?"=
          13^T 10^T                                 ! Blank line before get !
          D @I/!/ .UA :@S/[/"U @O!ERROR! '
          R .UB QA,QBXS 94@:^US//                   ! Get replace text       !
          QAJ C @^UP/Enter "/ .,QB:XP @:^UP/" : /
                                                    ! Save input prompt      !
          QBJ C .UB :@S/]/"U @O!ERROR! '
          QB,.-1XH                                  ! Save help text         !
          .UB :@S/*/"U @O!ERROR! '
          QB,.-1XD                                  ! Save default text      !
          \UM C \UN C \UC                           ! Get min, max, code     !
          MC
          < @S/^EQS/; ^SD GR >                      ! Make all replacements !
         '
  @O!SKIP!

  !ERROR!
  @^A/Ignoring invalid format line:/
  13^T 10^T QAJ T                                   ! Display error message !

  !SKIP!
  ^[                                                ! Eat any free numbers   !
  %L                                                ! Bump count             !
  ^[                                                ! Eat any free numbers   !
  J QLL .-Z;                                        ! Drop to next, end?     !
>

J 0UI

<
  ::@S/.IF /"S ME -L '                              ! If found, do equate    !
  ::@S/.END/"S (QI-1)UI 0L K -L '                   ! Close ".IF" ballon     !
  L .-Z;
>

EX                                                  ! Mission complete, the !
                                                    ! world is again safe    !
                                                    ! for democracy.         !
$$

PIP EXMPL1.PCF
$JOB/CCL/NOLIMIT
!
! Batch stream name: TEST
!
;
; Audit trail
; Date         Who  Why
; 22-Nov-80    DJS  Create stream
```

```
;
!
! Print display text
!
&This batch does daily batches and can close the week
!
! Get responses to questions
!
?^Do week-end processing[Enter "Y" if the week is closing]N*0*1*0
!
! Batch stream code
!
! Phase 1 -- Process all checks
!
$RUN CHECKS
.IF ^DO WEEK-END PROCESSING^=Y
!
! Phase 2 -- Close the week
!
$RUN CLOSER
.END
$EOJ

Ready

PIP EXMPL2.PCF
;
; Stream to compile programs
;
; Print display text
;
;
; Get responses to questions
;
?^compile with BASIC+2[Enter "Y" if compiling with BP2]Y*0*1*0
;
; Do compiles
;
.IF ^compile with BASIC+2^=Y
RUN $BASIC2
LOCK/TSK/LINE/CHAIN/DOUBLE
.END
OLD SOURCE:BMP100.BAS
COMPILE
OLD SOURCE:BMP200.BAS
COMPILE
OLD SOURCE:BMP250.BAS
COMPILE
OLD SOURCE:BMP900.BAS
COMPILE
.IF ^compile with BASIC+2^=Y
EXIT
PIP
TASK:.BAC<232>=BMP100.TSK,BMP200.TSK,BMP250.TSK,BMP900.TSK
BMP100.TSK,BMP200.TSK,BMP250.TSK,BMP900.TSK/DE
^Z
.END
.IF ^compile with BASIC+2^<>Y
PIP
TASK:<232>=BMP100.BAC,BMP200.BAC,BMP250.BAC,BMP900.BAC
BMP100.BAC,BMP200.BAC,BMP250.BAC,BMP900.BAC/DE
^Z
.END

Ready

RUN BLDCTL
BLDCTL  V01.00

File for input? EXMPL1
Opening EXMPL1.PCF for input...
File for output? KB:
Opening KB:.CTL for output...
This batch does daily batches and can close the week

Enter "Do week-end processing" : Y
$JOB/CCL/NOLIMIT
!
! Batch stream name: TEST
!
!
! Print display text
!
!This batch does daily batches and can close the week
!
! Get responses to questions
!
!^Do week-end processing[Enter "Y" if the week is closing]N*0*1*0
!
! Batch stream code
!
! Phase 1 -- Process all checks
!
$RUN CHECKS
!
! Phase 2 -- Close the week
```

AUTHORS!!

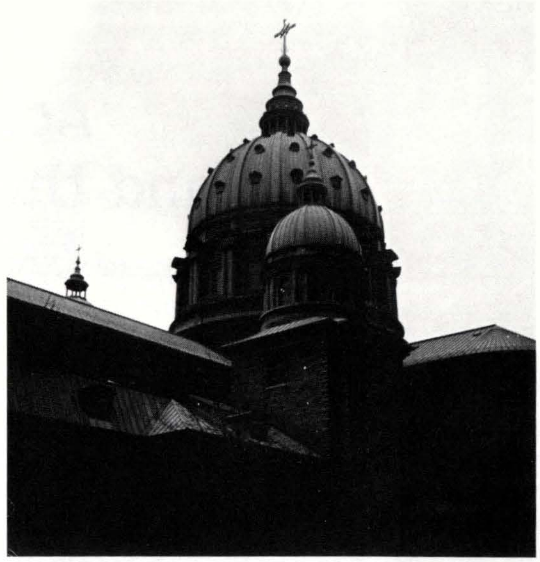
SEND YOUR ARTICLES TO THE RSTS PROFESSIONAL ON MAG TAPE, IN EITHER RNO, PIP OR WORD-11 FORMAT.

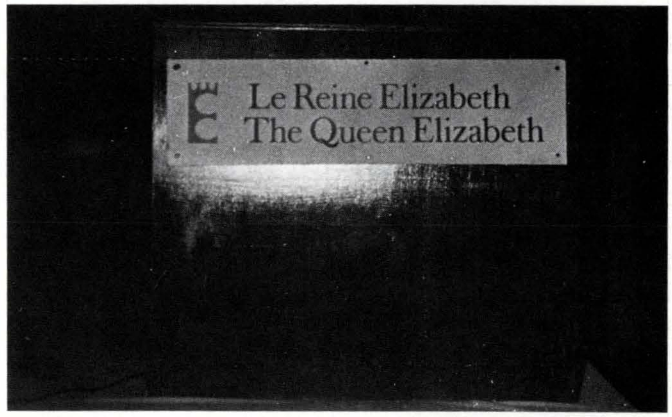
Eighty percent of this issue of the RSTS Professional was transmitted via telecommunications from author's mag tapes to phototypesetting equipment and was not retyped.

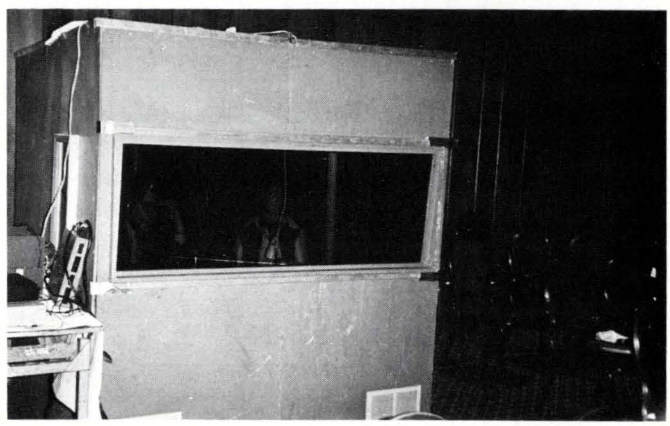
```
!
$RUN CLOSER
$EOJ

Ready

RUN BLDCTL
BLDCTL  V01.00

File for input? EXMPL1
Opening EXMPL1.PCF for input...
File for output? KB:
Opening KB:.CTL for output...
This batch does daily batches and can close the week

Enter "Do week-end processing" : ?
Help text is as follows:
Enter "Y" if the week is closing
Enter "Do week-end processing" : N
$JOB/CCL/NOLIMIT
!
! Batch stream name: TEST
!
!
! Print display text
!
!This batch does daily batches and can close the week
!
! Get responses to questions
!
!^Do week-end processing[Enter "Y" if the week is closing]N*0*1*0
!
! Batch stream code
!
! Phase 1 -- Process all checks
!
$RUN CHECKS
$EOJ

Ready

RUN BLDCTL
BLDCTL  V01.00

File for input? EXMPL2
Opening EXMPL2.PCF for input...
File for output? KB:
Opening KB:.CTL for output...

Enter "compile with BASIC+2" : YES
%Input must be no more than 1 character long.
Enter "compile with BASIC+2" : Y
!^compile with BASIC+2[Enter "Y" if compiling with BP2]Y*0*1*0
RUN $BASIC2
LOCK/TSK/LINE/CHAIN/DOUBLE
OLD SOURCE:BMP100.BAS
COMPILE
OLD SOURCE:BMP200.BAS
COMPILE
OLD SOURCE:BMP250.BAS
COMPILE
OLD SOURCE:BMP900.BAS
COMPILE
EXIT
PIP
TASK:.BAC<232>=BMP100.TSK,BMP200.TSK,BMP250.TSK,BMP900.TSK
BMP100.TSK,BMP200.TSK,BMP250.TSK,BMP900.TSK/DE
^Z

Ready

RUN BLDCTL
BLDCTL  V01.00

File for input? EXMPL2
Opening EXMPL2.PCF for input...
File for output? KB:
Opening KB:.CTL for output...

Enter "compile with BASIC+2" : N
!^compile with BASIC+2[Enter "Y" if compiling with BP2]Y*0*1*0
OLD SOURCE:BMP100.BAS
COMPILE
OLD SOURCE:BMP200.BAS
COMPILE
OLD SOURCE:BMP250.BAS
COMPILE
OLD SOURCE:BMP900.BAS
COMPILE
PIP
TASK:<232>=BMP100.BAC,BMP200.BAC,BMP250.BAC,BMP900.BAC
BMP100.BAC,BMP200.BAC,BMP250.BAC,BMP900.BAC/DE
^Z

Ready
```

DECUS CANADA

By Carl Marbach







Your intrepid Editor winged his way North again this year to visit Montreal, P.Q. the site of this year's Canadian Decus Symposium. I met many of the friends I had seen for the first time last year, and wondered privately why I only see them at the Canadian Symposium; but then I don't see a lot of Americans at these meetings either. Decus is Decus whether its north or south of the border and I must confess the northern branch is much friendlier than its southern neighbor.

Montreal is a delightful city filled with big city noises and smells, but small town friendliness. Bonjour is the greeting of the day and I admit to feeling somewhat inadequate if bi-lingual means I speak COBOL, BASIC, DIBOL, RPG, and FORTRAN but no French. Customs is no problem (What are these magazines????) and the proximity to the States argues for some more attendance from the south. The restaurants are. . . but on to the technical end of this meeting.

Most sessions were translated from English to French but not the other way around, translators were stationed in little booths in each of the rooms of the Queen Elizabeth (LE REINE ELIZABETH) Hotel. The sessions were held from Wednesday through Friday with Tuesday saved for seminars (tutorials etc). I can't get to everything I want to hear, there is just too much. When are we going to get a timely, fully prepared proceedings and/or the availability of cassette tapes of the sessions? Here then is a distillation of my highlights of what I could hear.

A rehash of the intricate Terminal handler for RSTS. Isn't this going to be rewritten soon anyhow? It is enlightening to see how complicated it can be just handling people typing away at keyboards, sort of reminds me of my first PDP-8 program: Accept a character and make it echo on the keyboard. Took me a whole day! Now do it using interrupts. Another day! Echo control?? Forget it.

How about VAX performance. An 11/750 peaks out at about 80 terminals due to memory limitations while an 11/780 (4MB) kept going to at least 128. If you are not CPU bound, an 11/70 did almost as well, although I don't know where it got enough small buffers. The summation with lots of caveats was 30-50 users for the 11/750 more for the 11/780 while a RSTS system (11/70) was good if you were not CPU bound. I asked when a second Unibus would be a good idea and when a massbus would be a good idea and when a second massbus would be a good idea. They didn't know. Didn't make much difference they thought. I don't believe it. A massbus has a bandwidth much larger than the unibus, and I know that my disks could saturate the unibus and I have an 11/70, the VAX must be able to swamp the unibus in a COMMERCIAL environment. I'd like to bet that there is a huge difference between a unibus VAX and a massbus one, and that by judiciously choosing when to add more you can fully utilize the CPU and the disks. Channels for disks is an IBM type word but it starts to mean something for VAX.

There were many more sessions dealing with specific RSTS applications and how professional people were supporting their customers. We ought to know by now that in order to maintain software you must have standards, and lots of other conventions that you live by. This is where you can see and hear what people have done that have made it past a few years. It is always great to be able to hear from these COMMERCIAL users. The atmosphere in Canada seems to support them better than we do in the Lower 48.

I came, I learned and I went home knowing more and feeling that the trip to Canada and Decus was one that I hope I won't miss in the future. Good Job Decus people. Keep up the good work. ♥

Access Control and Utilization Monitor

By C.M. Battistel and Ed Giovanella
Open Learning Institute, 7671 Alderbridge Way, Richmond, BC, Canada, V6X 1Z9

Many installations find that the access control and protection provisions provided by RSTS are often inadequate or are too inflexible to meet their needs. For example, consider the situation where we have ten users, each of whom must have access to a different combination of five programs. Using RSTS protection codes and account placement of programs and files, it is impossible to implement controlled access to these programs. Further, if the combinations of programs to be accessed by an individual were to change, the whole account, protection code structure would have to be changed. Clearly, although RSTS may provide a reasonable level of access control where there is little overlap of program access requirements, it is not adequate in most commercial environments.

Another inadequacy of RSTS exists in terms of utilization monitoring. The information provided by MONEY while useful, does not provide enough detail for many installations. It is, for example, useful to know not only who is using the system, but what programs account for the biggest load. This is especially true in an environment where charge-back and priority setting is important.

There exist instances when a program should not be run concurrently with specific other programs. An example of this is running a General Ledger Close program while another user is running a G/L data entry program. Again, we have found that RSTS and RMS do not handle this situation satifactorily. The typical RSTS solution would be to make one program open the file ALLOW READ. However, if this is done when a program is run against the same file that opens the file ACCESS READ and ALLOW MODIFY, an error is generated. Thus we are left with having to do complex error trapping to try and control this situation.

To resolve these problems we have developed a system which not only controls access, but also monitors utilization.

ACUM is an acronym for Access Control and Utilization Monitor. ACUM consists of a series of programs which restrict access to all applications programs to authorized users and monitor application program utilization rates. ACUM also includes a number of programs which provide information useful for system management. All applications programs are accessed through and return to ACUM. All applications programs also maintain utilization data during execution.

ACUM Data Base

Four files are required to accomplish the control and utilization monitoring functions of ACUM, the Access Control File, The Utilization Monitoring File, the Active Users File, and the Program Conflict File.

The Access Control File contains all of the data needed to ensure that only authorized users can gain access to the application programs and to restrict users to a specific set of programs within the entire application package.

There are 3 record types within the Access Control File. User Records (containing user I.D. number, user name and access code) define authorized users to the system. Each valid program within the applications package is defined in a Program Record (containing program code, program name and the name of the program's .TSK file). The third record type within the file is the User Clearance Record. Each of these records "link" a user to a program, thereby clearing a user for that specific program. Users may be cleared for "ALL" programs or may have several Clearance Records to authorize them for several programs.

The Utilization Monitoring File contains one record for each user-program and user-file combination. Each user-program record contains User Number, program name, CPU time, connect time, number of pages printed, and number of copies produced. The user-file records contain User Number, file name, number of records created, changed, deleted, output, screened for changes, and screened for output.

The Active Users File contains one record for each active applications job. Each record contains the User Number, the program being run, the start CPU and connect time, the keyboard number, and the RSTS job number.

The Program Conflict File contains one record for each applications program which should not be run while another specific program is active. Each record contains the name of the program which cannot be run and the name of the program it cannot be run with.

Program Access Through "ACUM"

Each user of the system is assigned a unique user I.D. number and an access code or password. This data is maintained in the Access Control File. In addition, the Access Control File also specifies the programs which a user is authorized to access. In this way, an individual user can be authorized to use only selected programs from the total applications package.

Application program access through ACUM is accomplished in the following manner. The system will automatically run the Access Control program when a non-privileged user logs onto the system. This program prompts the user to enter

their I.D. number and access code (password). The Access Control File is checked to verify that the user has entered a valid combination. An error message is displayed if the combination is invalid and the user is re-prompted. If the user fails to enter a valid combination after three attempts, the user is logged off the system and a warning message is displayed on the system console.

If the user enters a valid I.D. number and access code, he or she is at ACUM command level and may issue commands or program requests to the Access Control program when prompted to do so. Valid user commands are "LIST" which displays all programs for which the current user is authorized, and "END" which terminates the job and logs the user off the system. If the user enters a program request, the Access Control File is checked to verify that the program exists and that the current user is authorized to use it. If either edit fails, an error message is displayed and the user is re-prompted. If the user enters a valid program request, the Access Control program places the user I.D. number in core common, updates the Active User File to reflect the status of the user, checks for conflicts with currently active programs and if no conflict exists, CHAINS to the appropriate application program. When the user is finished with the application program, they are CHAINED back to the Access Control program and the Active User File is updated to reflect their current status. Users are then re-prompted for a command or program request. If command "END" is entered, the Active User File record corresponding to the user and job is deleted, and the user is logged off the system. It can be seen from the above description that during a job session, a user is always in communication with an application program and at no time can they gain access to RSTS. For the convenience of application programmers, it is possible to clear a user to run a program which allows access to RSTS.

Two other points of interest concerning ACUM must also be mentioned. The first statement of each application program is a CALL to a subroutine which immediately logs the user off the system. The Access Control program "CHAINS" to application programs at a point past the subroutine CALL. It is, therefore, impossible to access any application program directly without going through the Access Control program. The second point of interest is the passing of the user I.D. number into core common. Each application program can then obtain the user I.D. from core common and thereby maintain utilization statistics for that user.

All programs must chain to the Access Control program upon termination. This enables users to select additional application programs for execution or issue the "END" command which automatically logs the user off. In cases where a job detaches and a new job is created, LOGIN has been modified to chain to the Access Control program. The detached job upon termination kills itself. LOGIN chains to the Access Control program by running a program called START in the user's account. If there is no START program in the user's account the user is logged into RSTS command level. This feature is used to enable programming staff immediate access to RSTS without using the Access Control program.

Utilization Monitoring

The Utilization Monitoring Function of ACUM is accomplished through the tracking of user system utilization within each application program. The utilization data must be maintained from the time that the Access Control program chains to an applications program, to the time that the application chains back to the Access Control program or, in the case of a job which detaches, until the job kills itself. Although entry to applications can only be made through the Access Control program, exit from applications programs can take a number of forms.

1. Use of ESC key during data entry
2. KB wait exhausted during data entry
3. 'normal' exit from mainline by chaining to the Access Control program
4. exit from mainline by chaining to QUE

The method of collecting utilization data remains constant regardless of exit mode; however, the method of updating utilization data stored in ACUM files will vary.

In order to properly maintain utilization data while programs are executing the following functions must be performed at the beginning of each applications program.

1. Declare a common area with the following data:
   a. ACUM user number (the Access Control Program places the user number in core common)
   b. number of data files
   c. current CPU time
   d. current connect time
   e. 6 character program name
   f. 6 character file name of data files used in program

2. During program execution the program must maintain in the common area the following data:
   a. number of pages of hardcopy output (handled by subroutines)
   b. number of copies of hardcopy output (handled by subroutine)
   c. number of file records created in each file
   d. number of file records changed in each file
   e. number of file records deleted in each file
   f. number of file records output from each file
   g. number of file records read and reviewed from each file in an attempt to find a specific record which is to be changed
   h. number of records read from each file in order to find a specific sub-set of the file which is then output.

The utilization data maintained by each applications program is written to the Utilization Monitoring File by a subroutine prior to program termination. The method used for update varies with the manner in which the program ended.

Program Conflict Control

Since ACUM knows at all times which application programs are running, it can control the execution of conflicting programs. To accomplish this ACUM maintains a table containing the names of all programs which conflict with each other. This table is checked by the Access Control program prior to chaining to a user selected applications program.

Prior to implementing an applications program which conflicts with other application programs, programmers create the appropriate conflict records in the Program Conflict File as follows:

USER-SELECTED-PROGRAM	CONFLICTING-PROGRAM
PROGA	MPROG
PROGA	PROGER
PROGA	TSTPRG

If the conflict works both ways for all these programs the following additional records would be required:

MPROG	PROGA
PROGER	PROGA
TSTPRG	PROGA

When a user selects a program, the Access Control program checks the Program Conflict File for a matching USER-SELECTED-PROGRAM. For each one found, the corresponding CONFLICTING-PROGRAM field is used to search the Active Users File. If a matching record is found in the latter file an error message is displayed to the user, and he or she is prevented from executing the USER-SELECTED-PROGRAM.

ACUM Management Reports

Two sets of reports are available to inform management as to the current status of application programs, users and computer utilization. One set of reports is printed from the Access Control File and lists all application programs and authorized users. The other set is produced from the Utilization Monitoring File and provides statistics on data file activity and computer resource utilization. These reports are described in detail below.

Access Control File Listings

There are three simple listings that can be produced from the Access Control File. These reports show the current status of users, user clearances, and application programs. The reports can be produced either at a keyboard or on the line printer.

a. Application Programs: lists each application program available through ACUM and shows the program code, program description, and the name of the program's .TSK file.
b. Authorized Users: lists each user cleared to access ACUM and shows user I.D. number, name and access code or password.
c. Authorized Users and Clearances: lists each user with access to ACUM and the application programs for which the user is cleared. The report shows user I.D. number, user name, password, program code and program description. Appropriate messages are printed if the user is cleared for all programs or not cleared for any programs.

Computer Resource Utilization Summaries

There are four summary reports that can be provided from the Utilization Monitoring File. These reports contain a variety of utilization statistics including file activity, CPU and connect time usage and volume of hardcopy output. They are designed to assist management in monitoring the utilization of application data files and computer resources and can be a valuable aid in detailing utilization by individual users, defining through-put "bottle-neck" areas, planning for future hardware and/or software enhancements and monitoring file activity to determine file volatility and storage requirements.

The reports can be displayed at a terminal or printed on the printer.

a. File Activity By User: This report analyzes file activity by individual user. Six activity categories (number of records created, changed, deleted, displayed, screened for change and screened for display) and a total activity count is printed for each user of the data file. The report also shows the percentage of each activity category as it relates to the total activity by each user. A total line is also printed for each file showing the total activity and related percentages for all users of the file. The program that produces the report can be instructed to print data for all files or one file.
b. Computer Utilization By User: This, and the next two reports display the computer resource utilization statistics in a variety of ways. This report organizes the data by user number and displays connect time, CPU time, number of pages and copies printed and total paper used. The program also calculates and displays, for each user, the percentage of each utilization category as it relates to the total of that category.
c. Computer Utilization By Program: This report is identical in content to the one "By User" but is presented in sequence by Program Name.
d. Computer Utilization By User and Program: This report organizes the connect and CPU time statistics by user and program. For each user, the user number and user name is printed. For each of the user's programs, the program name, description, connect time and CPU time is displayed. The percentage of each program's connect and CPU times (as it relates to the total times for the user) is calculated and printed.

CONCLUSIONS

The above description represents the results of our attempts to provide access control and utilization monitoring capabilities beyond those provided by RSTS. We have found the system extremely adaptable and easy to use and maintain. ♥

PDP/11 Systems — UTILITIES

By M. A. Jackson, A.C. Nielsen Co. Ltd., Oxford, UK

INTEGER TO DECIMAL STRING CONVERSION SUBROUTINE

Invocation

CALL NUMASK (A$, B%)

Processing

This module right-aligns in A$ the decimal string representation of B%. Only as many characters as are necessary to hold the value are changed; other characters in A$ are unchanged. If B% equals zero, no action is taken, i.e. the routine will not right-align a single zero character into a blank string.

Checks

1. That two parameters are passed. — No action taken.
2. That B% is not negative. — No action taken.
3. That A$ is long enough to hold the string value of B%. — No action taken.

Critical considerations

Size 108 bytes

Speed <1ms

```
        .TITLE  NUMASK
        .IDENT  /007/
;
; CALL NUMASK(A$,B%)
;
; THIS SUBROUTINE IS CALLED FROM BASIC-PLUS-2. IT RIGHT-ALIGNS THE
; DECIMAL REPRESENTATION OF B% IN A GIVEN MASK STRING A$. IT IS A NO-OP
; IF B% IS NEGATIVE OR ITS DECIMAL REPRESENTATION WILL NOT FIT INTO A$.
;
NUMASK::
        CMPB    #2,(R5)         ; 2 ARGUMENTS ?
        BNE     RETRN           ; NO - GIVE UP
        MOV     @4(R5),R4       ; R4 = B%
        BLT     RETRN           ; NEGATIVE - GIVE UP
        MOV     2(R5),R0        ; R0 = ADDR OF A$ HEADER
        MOV     2(R0),R1        ; R1 = LENGTH OF A$
        BEQ     RETRN           ; A$ NULL - GIVE UP
        MOV     #POWERS,R2      ; START R2 AT 10000
CLR:    CLR     R3              ; INIT COUNTER
SUBT:   SUB     (R2),R4         ; SUBTRACT POWER OF 10
        BLT     NEG             ; OVERDONE IT
        INC     R3              ; INCREMENT CNTR
        BR      SUBT            ; GO SUBTRACT AGAIN
NEG:    ADD     (R2),R4         ; ADD IT BACK IN
        CMP     2(R0),R1        ; FIRST CHAR ?
        BNE     PUSH            ; NO - NEED IT WHATEVER
        TST     R3              ; IS IT ZERO ?
        BEQ     NEXT            ; YES - BYPASS PUSH
PUSH:   MOV     R3,-(SP)        ; PUSH DIGIT ONTO STACK
        DEC     R1              ; COUNT DOWN NUMBER OF CHARS
        BLT     OVERFL          ; TOO MANY CHARS
NEXT:   TST     (R2)+           ; POINT R2 AT NEXT (LOWER) POWER
        TST     (R2)            ; CHECK END OF POWERS
        BNE     CLR             ; MORE TO SUBTRACT
        MOV     (R0),R2         ; R2 = ADDR OF A$
        ADD     2(R0),R2        ; R2 = ADDR OF END OF A$ + 1
FILL:   CMP     2(R0),R1        ; ALL DIGITS POPPED ?
        BEQ     RETRN           ; YES - GO HOME
        BIS     #000060,(SP)    ; CVT TO ASCII DECIMAL
        MOVB    (SP)+,-(R2)     ; MOVE DIGIT TO STRING
        INC     R1              ; COUNT CHARS POPPED
        BR      FILL            ; GO CHECK FINISHED
OVERFL: CMP     2(R0),R1        ; ALL DIGITS POPPED ?
        BEQ     RETRN           ; YES - GO HOME
        TST     (SP)+           ; 'POP A DIGIT' (TO NOWHERE)
        INC     R1              ; COUNT CHARS POPPED
        BR      OVERFL          ; GO CHECK FINISHED
RETRN:  RETURN
POWERS: .WORD   10000.,1000.,100.,10.,1,0

        .END
```

VERIFY SUBROUTINE

Invocation

CALL VERIFY (A$, B$, C%)

Processing

This module returns in C% the position of the first character in B$ which is not in A$, or zero if all characters in B$ are in A$. It also returns zero if B$ is null, even if A$ is also null. This is logical (there are no characters in B$ which are not in A$) and consistent with Basic-plus-2 functions.

Checks

1. That three parameters are passed. — No action taken.

Critical Considerations

Size 70 bytes

Speed Typically <1mls; dependent on the string lengths.

A$ is searched sequentially from the left; characters expected to be most frequently present in B$ should therefore be at the left of A$.

```
        .TITLE  VERIFY
        .IDENT  /00/
;
; CALL VERIFY(A$,B$,C%)
;
; SUBROUTINE TO RETURN THE POSITION OF THE FIRST CHARACTER IN B$
; WHICH IS NOT IN A$. RETURNS 0% IF ALL BYTES ARE PRESENT.
;
VERIFY::
        CMPB    #3,(R5)         ; 3 ARGUMENTS ?
        BNE     RETRN           ; NO - GIVE UP
        CLR     @6(R5)          ; INIT C% = 0%
        MOV     4(R5),R0        ; R0 = ADDR OF B$ HEADER
        MOV     2(R0),R1        ; R1 = LENGTH OF B$
        BEQ     RETRN           ; B$ NULL - RETURN 0%
TESTA:  MOV     2(R5),R2        ; R2 = ADDR OF A$ HEADER
        MOV     2(R2),R3        ; R3 = LENGTH OF A$
        BNE     START           ; NOT ZERO LENGTH
        INC     @6(R5)          ; C% = 1%
        BR      RETRN           ; A$ NULL - MUST RETURN 1%
START:  MOV     (R0),R0         ; R0 = ADDR OF B$
        ADD     R0,R1           ; R1 = ADDR OF END OF B$ + 1
        ADD     (R2),R3         ; R3 = DITTO            A$
NEXTA:  MOV     (R2),R4         ; ADDR OF A$
        INC     @6(R5)          ; INCREMENT C%
TESTB:  CMPB    (R0),(R4)+      ; COMPARE BYTE OF B$ WITH BYTE OF A$
        BEQ     NEXTB           ; IT'S THERE, GO GET NEXT OF B$
        CMP     R4,R3           ; PAST ALL OF A$ ?
        BNE     TESTB           ; NO - TEST NEXT
        BR      RETRN           ; YES - C% ALREADY SET - RETURN
NEXTB:  INC     R0              ; POINT TO NEXT OF B$
        CMP     R0,R1           ; PAST ALL OF B$ ?
        BNE     NEXTA           ; NO - TEST NEXT OF B$
        CLR     @6(R5)          ; YES - SET C% = 0%
RETRN:  RETURN
        .END
```

INTEGER TO OCTAL STRING CONVERSION SUBROUTINE

Invocation

CALL OCTAL (A$, B%)

Processing

This module returns in the six-byte string A$ the octal string representation of B%.

Checks

1. That two parameters are passed. — No action taken.
2. That A$ is six bytes long. — No action taken.

Critical considerations

Size 62 bytes

Speed <1 ms

```
        .TITLE  OCTAL
        .IDENT  /01/
;
; CALL  OCTAL(A$,B%)
;
; THIS SUBROUTINE PUTS THE SIX CHARACTER OCTAL REPRESENTATION OF B%
; INTO THE (SIX-BYTE) STRING A$
;
OCTAL::
        CMPB    (R5),#2         ; 2 ARGUMENTS ?
        BNE     ERR             ; NO - ERROR
        MOV     2(R5),R0        ; R0 = ADDR OF A$ HEADER
        CMP     2(R0),#6        ; A$ SIX BYTES ?
        BNE     ERR             ; NO - ERROR
        MOV     @4(R5),R1       ; R1 = B%
        MOV     (R0),R2         ; R2 = ADDR OF A$
        ADD     #6,R2           ; R2 = ADDR OF END OF A$ + 1
        MOV     #5,R3           ; LOOP COUNT
LOOP:   MOV     R1,-(SP)        ; WORD ONTO STACK
        BIC     #177770,(SP)    ; 'ONE OCTAL VALUE'
        ADD     #'0,(SP)                ; CONVERT TO ASCII
        MOVB    (SP)+,-(R2)     ; STORE IN A$
        ASH     #-3,R1          ; SHIFT RIGHT THREE
        SOB     R3,LOOP         ; TEST FOR FIVE DONE
        BIC     #177776,R1      ; GET LAST BIT
        ADD     #'0,R1          ; CONVERT TO ASCII
        MOVB    R1,-(R2)        ; STORE IN A$
ERR:    RETURN
        .END
```

SUBSTRING PSEUDOVARIABLE EMULATION SUBROUTINE

Invocation

CALL INSERT (A$, B$, C%)

Processing

This module overwrites string B$ into string A$ starting a position C%.

Checks

1. That three parameters are passed. — No action taken.
2. That C% is greater than zero. — No action taken.
3. That B$ will fit into A$ from that position. — No action taken.

Critical consideration

Size 50 bytes

Speed <1ms

```
        .TITLE  INSERT
        .IDENT  /00/
;
; CALL INSERT(A$,B$,C%)
;
; SUBROUTINE TO OVERWRITE B$ INTO A$ STARTING AT POSITION C%
;
INSERT::
        CMPB    #3,(R5)         ; 3 ARGUMENTS ?
        BNE     RETRN           ; NO - GIVE UP
        MOV     @6(R5),R2       ; R2 = C%
        BLE     RETRN           ; C% <= 0% - GIVE UP
        MOV     2(R5),R0        ; R0 = ADDR OF A$ HEADER
        MOV     4(R5),R1        ; R1 = ADDR OF B$ HEADER
        MOV     2(R1),R4        ; R4 = LENGTH OF B$
        BEQ     RETRN           ; B$ NULL - NOTHING TO DO
        DEC     R2              ; R2 = C% - 1%
        MOV     R2,R3           ; R3 = C% - 1%
        ADD     R4,R3           ; R3 = C% - 1% + LEN(B$)
        CMP     R3,2(R0)        ; GREATER THAN LEN(A$) ?
        BGT     RETRN           ; YES - B$ WON'T FIT - GIVE UP
        MOV     (R1),R1         ; R1 = ADDR OF B$
        MOV     (R0),R0         ; R0 = ADDR OF A$
        ADD     R2,R0           ; R0 = ADDR OF 1ST CHAR TO OVERWRITE
MOVONE: MOVB    (R1)+,(R0)+     ; MOVE A CHAR FROM B$ INTO A$
        SOB     R4,MOVONE       ; GO MOVE ANOTHER IF NOT FINISHED
RETRN:  RETURN
        .END
```

STRING INSERTION SUBROUTINE

Invocation

CALL PUSHIN (A$, B$, C%)

Processing

The module writes string B$ into string A$ starting at position C%. Characters in A$ from position C% onwards will be moved up to follow the inserted characters.

Checks

1. That three parameters are passed.
2. That C% is greater than zero
3. That B$ will fit into A$ from that position.
4. That sufficient blanks or nulls are present at the end of A$ to be "pushed off the end". — No action taken if any fails.

Critical Considerations

Size 80 bytes

Speed <1ms

```
         .TITLE   PUSHIN
         .IDENT   /00/
;
; CALL PUSHIN(A$,B$,C%)
;
; THIS SUBROUTINE INSERTS B$ INTO A$ STARTING AT POSITION C%,
; PUSHING THE REMAINDER OF A$ TO APPEAR AFTER B$.
; A$ MUST HAVE SUFFICIENT BLANKS OR NULLS AT THE END TO ACCOMMODATE B$.
;
PUSHIN::
         CMPB     (R5),#3           ; 3 PARMS ?
         BNE      RETRN             ; NO - GIVE UP
         MOV      @6(R5),R0         ; R0 = C%
         BLE      RETRN             ; C% <= 0% - GIVE UP
         MOV      4(R5),R1          ; R1 = ADDR OF B$ HEADER
         MOV      2(R1),R2          ; R2 = LEN(B$)
         BEQ      RETRN             ; B$ NULL - NOTHING TO DO
         MOV      2(R5),R3          ; R3 = ADDR OF A$ HEADER
         MOV      2(R3),R4          ; R4 = LEN(A$)
         ADD      R2,R0             ; POSN PAST END OF INSERT
         DEC      R0                ; POSN OF LAST INSERTED CHAR
         CMP      R0,R4             ; IS THIS PAST END OF A$ ?
         BGT      RETRN             ; YES - GIVE UP
         ADD      (R3),R4           ; R4 = ADDR OF BYTE PAST A$
         MOV      R4,R0             ; STORE DITTO IN R0
TEST:    BITB     -(R4),#137        ; IS IT BLANK OR NULL ?
         BNE      RETRN             ; NO - GIVE UP
         SOB      R2,TEST           ; TEST AS MANY BYTES AS IN B$
         MOV      2(R3),R2          ; R2 = LEN(A$)
         SUB      @6(R5),R2         ; R2 = LEN(A$) - C%
         INC      R2                ; NO. OF BYTES TO BE PUSHED UP
PUSH:    MOVB     -(R4),-(R0)       ; MOVE BYTES OF A$ AFTER INSERT
         SOB      R2,PUSH           ;
         MOV      2(R1),R2          ; R2 = LEN(B$)
         MOV      (R1),R4           ; R4 = ADDR OF B$
         ADD      R2,R4             ; R4 = ADDR OF BYTE PAST B$
         ADD      R2,R0             ; R0 = ADDR PAST LAST INSERT
INSERT:  MOVB     -(R4),-(R0)       ; MOVE BYTES OF B$
         SOB      R2,INSERT         ;
RETRN:   RETURN
         .END
```

STRING INSERTION SUBROUTINE

Invocation

CALL PUSHON (A$, B$, C%)

Processing

The module writes string B$ into string A$ starting at position C%. The character in A$ at position C% is overwritten; characters from position C% + 1% onwards will be moved up to follow the inserted characters.

Checks

1. That three parameters are passed.
2. That C% is greater than zero.
3. That B$ is not null.
4. That B$ will fit into A$ from that position.
5. That sufficient blanks or nulls are present at the end of A$ to be "pushed off the end".

No action taken if any fails.

Notes

1. Check 3 implies that a character cannot simply be "lost" from A$ by specifying replacement of the character at position C% with a null string.

2. That routine may be used simply to overwrite a single character in A$ (similar to INSERT subroutine) in which case check 5 obviously does not apply.

Critical Considerations

Size 82 bytes

Speed <1ms.

```
        .TITLE  PUSHON
        .IDENT  /00/
;
; CALL PUSHON(A$,B$,C%)
;
; THIS SUBROUTINE INSERTS B$ INTO A$ STARTING AT POSITION C%,
; OVERWRITING THE CHARACTER AT POSITION C% AND
; PUSHING THE REMAINDER OF A$ TO APPEAR AFTER B$.
; A$ MUST HAVE SUFFICIENT BLANKS OR NULLS AT THE END TO ACCOMMODATE B$.
;
PUSHON::
        CMPB    (R5)+,#3        ; 3 PARMS ?
        BNE     RETRN           ; NO - GIVE UP
        TSTB    (R5)+           ; R5 = ADDR OF 1ST PARM
        MOV     (R5)+,R3        ; R3 = ADDR OF A$ HEADER
        MOV     (R5)+,R1        ; R1 = ADDR OF B$ HEADER
        MOV     2(R1),R2        ; R2 = LEN(B$)
        BEQ     RETRN           ; B$ NULL - NOTHING TO DO
        MOV     2(R3),R4        ; R4 = LEN(A$)
        MOV     (R5),R5         ; R5 = ADDR OF C%
        MOV     (R5),R0         ; R0 = C%
        BLE     RETRN           ; C% <= 0% - GIVE UP
        ADD     R2,R0           ; POSN PAST END OF INSERT
        DEC     R0              ; POSN OF LAST INSERTED CHAR
        CMP     R0,R4           ; IS THIS PAST END OF A$ ?
        BGT     RETRN           ; YES - GIVE UP
        MOV     R4,R0           ; STORE LEN(B$) IN R0
        ADD     (R3),R4         ; R4 = ADDR OF BYTE PAST A$
        DEC     R2              ; ALLOW ONE CHAR OVERWRITE
        BEQ     NOTEST          ; ONLY ONE BYTE - NEED NO SPACE
TEST:   BITB    -(R4),#137      ; IS IT BLANK OR NULL ?
        BNE     RETRN           ; NO - GIVE UP
        SOB     R2,TEST         ; TEST AS MANY BYTES AS IN B$ MINUS ONE
NOTEST: MOV     R0,R2           ; R2 = LEN(A$)
        ADD     (R3),R0         ; R0 = ADDR OF BYTE PAST A$
        MOV     2(R1),R3        ; R3 = LEN(B$)
        SUB     (R5),R2         ; R2 = LEN(A$) - C%
        SUB     R3,R2           ; R2 = LEN(A$) - C% - LEN(B$)
        INC     R2              ; NO. OF BYTES TO BE PUSHED UP
        BEQ     NOPUSH          ; NONE TO MOVE
PUSH:   MOVB    -(R4),-(R0)     ; MOVE BYTES OF A$ AFTER INSERT
        SOB     R2,PUSH         ;
NOPUSH: MOV     (R1),R4         ; R4 = ADDR OF B$
        ADD     R3,R4           ; R4 = ADDR OF BYTE PAST B$
INSERT: MOVB    -(R4),-(R0)     ; MOVE BYTES OF B$
        SOB     R3,INSERT       ;
RETRN:  RETURN
        .END
```

A Beginners Guide to MACRO-11 . . .

. . . continued from page 33

Chapter 5

Programming Examples.

This section contains a runthrough of routines to read and write to a device. The development is historical: one day I read an article by Bob Meyer in the May/June issue (1980) of the RSTS Professional; the results are before you. At the end is an example of a subroutine which creates a file, to show how the important directive CALFIP is used.

5.1 PRINT Routine — Version 1.

The original program in the article I read looked like this:

```
        .TITLE  PRINT   PRINT DEMO
        .IDENT  /V01.1/

                                        ; PRINT.MAC --- PRINTS TO THE KB.
                                        ;
                                        ; WRITTEN BY BOB MEYER

                                        ; DEFINE DIRECTIVES, XRB
        XRB     = 000442
        .WRITE  = 104004
        .EXIT   = 104046

                                        ; DATA SPACE
        .ENABL LC
OBUFF:  .ASCII  /These are the voyages .../ <cr><lf>
        .EVEN
        MSGLEN = . - OBUFF              ; message length = this location -
                                        ; start of buffer location

                                        ; main code
PRINT:  MOV     #XRB, R0                ; pointer to XRB
        MOV     #MSGLEN, (R0)+          ; move message length to XRB
        MOV     #MSGLEN, (R0)+          ; twice
        MOV     MSG, (R0)+              ; start address
        CLR     (R0)+                   ; channel # * 2 to print
        CLR     (R0)+                   ; block # to print
        CLR     (R0)+                   ; wait time for input
        CLR     (R0)+                   ; optional modifiers

        .WRITE                          ; go do it
        .EXIT                           ; hello runtime
        .END PRINT
```

As you can see, this program loads up the XRB with the information it needs, and goes and does its .WRITE, and then goes home.

5.2 PRINT Routine — Version 2.

What might we reasonably want this PRINT routine to do? First, we should want to re-use it, so it has to go into subroutine form. Second, everything in version 1 is specified too exactly — we have to operate on channel 0, and print block 0, and make it whatever is in OBUFF, etc. So the second program should access memory locations which contain all this information. We could, of course, have it access registers, but if we wrote all our subroutines with such gay abandon, we would never be satisfied with a paltry 8 registers at all. A re-write of version 1 might well look like this:

```
;
; PRINT subroutine, emulates BASIC+
; PRINT statement.
;
; parameters:
;
; MSGLEN --- contains length of buffer
;            to move.
; CHNNL  --- channel * 2 to print to.
; BLOCK  --- block # to print to.
; R5     --- contains start address
;            of output buffer.
;

        .TITLE  PRINT   the .WRITE call
        .IDENT  /V01.2/

                                        ;
                                        ; globals follow
                                        ;
        .GLOBL  .EXIT, .WRITE
        .GLOBL  XRB, XRLEN, XRBC, XRLOC, XRCI, XRBLK, XRMOD
        .GLOBL  MSGLEN, CHNNL, BLOCK

                                        ; routine code follows
PRINT:: MOV     MSGLEN, XRB+XRLEN       ; move size of buffer into XRLEN
        MOV     MSGLEN, XRB+XRBC        ; again
        MOV     (SP), XRB+XRLOC         ; start address was in R5
        MOV     CHNNL, XRB+XRCI         ; channel # * 2
        MOV     BLOCK, XRB+XRBLK        ; block # to print
        CLR     XRB+XRMOD               ; only for RECORD option

        .WRITE                          ; go to it.

        RTS     R5                      ; go home

        .END                            ; end of source file
```

5.3 Assembling and Running.

Version 1 is easy to assemble and run. The commands

macro print, print = print
tkb print, print = print
run print

ought to do it. Version 2 however, cannot even be run alone, since it is just a subroutine. Indeed, that is its chief advantage, since you can now build it with any program that needs a print routine. Suppose we had a program which looked like:

```
                                        ; this program runs the PRINT
                                        ; subroutine once.
                                        ;
        .TITLE TEST the PRINT subroutine.
        .IDENT /V01.1/

                                        ; globals follow
        .GLOBL PRINT

                                        ; data follows.
        .ENABL LC
OBUFF:  .ASCIZ  /These are the voyages .../<cr><lf>
        .EVEN
BLOCK:: .ASCII <nul><nul>                               ; reserve 2 bytes of 0
CHNNL:: .ASCII <nul><nul>
MSGLEN::.ASCII <nul><nul>

                                        ; main program follows
TEST:   MOV     #OBUFF, R5              ; move start address into register
LOOP:   TSTB    (R5)+                   ; is the byte nul? (end of message)
        BNE     LOOP                    ; no, try next byte.
        SUB     #OBUFF+1, R5            ; get message length
        MOV     R5, MSGLEN              ; move into message length
        MOV     #OBUFF, R5              ; put start address back
        JSR     R5, PRINT               ; do the write
        .EXIT                           ; say good-bye
        .END    TEST                    ; end of source, program.
```

To assemble and run this you would command

macro print, print = print
macro test, test = common, test
tkb test, test = test, print
run test

[Note: COMMON.MAC can be found on the SYSGEN tape.]

5.4 Creating a File.

This last example shows a subroutine which creates a file. There are a few new ideas and instructions used, but everything is pretty straightforward:

1. Macro procedures — it becomes repetitive to write a similar piece of code several times, so macro procedures are employed. When the assembler finds a statement calling a macro procedure, it insets the procedure code in place of the call, and replaces the dummy variables in

the definition with the passed parameters in the call. Dummy variables preceded by a question mark have special meaning: if the parameter is not supplied, they become a unique local symbol between 64$ and 127$.

2. .PSECT — this instruction controls linking. In this case it is used to tell TKB that this piece of code is pure instructions.

3. The stack — to use registers freely inside a subroutine, without destroying their previous contents, they are stored by "pushing" them onto the stack. At the end of the procedure, they are "popped" off, returning to them the values they held upon entry into the procedure.

4. Local symbols — these are of the form <number>$. They are local to the .PSECT of code they appear in, and insure that separately assembled pieces of code do not conflict.

To open a file for output, you have to load up the firqb with the file name in RAD50 format, plus info like the disk and ppn, mode, filesize, etc. This is done by making the file string scan call. It automatically sets up most of the information to do the open. The code looks like this:

```
        .title  create a file (if possible)
        .ident  /002trc/

;
;       This module is a subroutine to creat a file. It is called by
;
;       JSR     pc, create
;
;       It expects to see the address of the file name in r0 and the channel
;       to open in r1. It returns errors in location FIRQB.
;

                                              ; macro definitions
        .macro  clear   loc, len, reg1, reg2, ?a; clear len bytes of memory
        mov     #loc, reg1                    ; starting at loc.
        mov     #len, reg2
a:      clr     (reg1)+
        sob     reg2, a

        .endm   clear                         ; end of macro

        .psect  create, i                     ; these are instructions only

create::
        mov     r0, -(sp)                     ; save registers
        mov     r1, -(sp)
        mov     r2, -(sp)
        mov     r3, -(sp)

        clear   firqb, 40/2, r2, r3           ; clear FIRQB and XRB
        clear   xrb, 20/2, r2, r3

        mov     r0, xrb+xrloc                 ; file name start location

        clr     r2                            ; find length of file name
1$:     tstb    (r0)+                         ; is this byte 0?
        beq     2$                            ; If so, drop out
        incb    r2                            ; else add 1 to the length
        br      1$                            ; and try on the next byte

2$:     mov     r2, xrb+xrlen                 ; move in length
        mov     r2, xrb+xrbc                  ; twice

        .fss                                  ; and see if name is legal

        tstb    firqb                         ; test for a returned error
        bne     3$                            ; and cut out if you find one

        clear   xrb, 20/2, r2, r3             ; the xrb got dirty

        movb    #crefq, firqb+fqfun           ; move in create flag
        ash     #1, r1                        ; multiply channel # by 2
        mov     r1, firqb+fqfil               ; and move it in

        calfip                                ; try to create the file

3$:     mov     (sp)+, r3                     ; restore registers
        mov     (sp)+, r2
        mov     (sp)+, r1
        mov     (sp)+, r0

        rts     pc                            ; and go home

        .end    create                        ; end of text
```

What is going on is really pretty simple. We save the registers, find the length of the file name string, set up the xrb, and see if the name is legal. If it is, we set up the firqb and open the file. The only thing which should seem strange is the ASH instruction. It stands for Arithmetic SHift. This one means shift all the bits in r1 left by 1. Bit 15 goes into the C bit. As the procedure dies, it restores the values of the registers it changed to their original contents. The calling program decides if the file was actually created by testing location FIRQB.

5.5 Last comments. People tend to show off what they know (after all, why else write articles?). In the case of assembly language programming, DON'T DO IT! There are only a few times you will really need to program something in MACRO-11: some i/o, or a piece of code that has to be really fast or small. Remember that the cost for this is that development time will skyrocket, and the resulting code will be difficult to maintain—so don't plan on writing an accounts receivable package in assembly language, for you will still be at it when the company folds.

A final note on tools: there is a very good debugging tool called ODT. It allows you to establish breakpoints in your code, look at memory locations, change them and continue onward. Many people also do the same sorts of things from a higher level language, like Pascal or BASIC Plus 2. I think which approach you use is mostly a matter of temperment, but that before too long, you'll find you will need some tool to work with.

Appendix A

COMMON.MAC

```
                                ;
                                ; COMMON --- THIS FILE CONTAINS THE
                                ; DEFINITIONS TO DO RSTS/E VERSION 7
                                ; MONITOR CALLS
                                ;
.TITLE  COMMON  RSTS COMMON DEFINITIONS
.IDENT  /V07.01/

.PAGE
                                ;
                                ; ASCII CHARACTER DEFINITIONS
                                ;
NUL     == 000
SOH     == 001
STX     == 002
ETX     == 003
EOT     == 004
ENQ     == 005
ACK     == 006
BEL     == 007

BS      == 010
HT      == 011
LF      == 012
VT      == 013
FF      == 014
CR      == 015
SO      == 016
SI      == 017

DLE     == 020
DC1     == 021
DC2     == 022
DC3     == 023
DC4     == 024
NAK     == 025
SYN     == 026
ETB     == 027

CAN     == 030
EM      == 031
SUB     == 032
ESC     == 033
FS      == 034
GS      == 035
RS      == 036
US      == 037

SPACE   == 040

ULINE   == 137

AGRAVE  == 140
LC.A    == 141
LC.B    == 142
LC.C    == 143
LC.D    == 144
```

```
LC.E    == 145
LC.F    == 146
LC.G    == 147

LC.H    == 150
LC.I    == 151
LC.J    == 152
LC.K    == 153
LC.L    == 154
LC.M    == 155
LC.N    == 156
LC.O    == 157

LC.P    == 160
LC.Q    == 161
LC.R    == 162
LC.S    == 163
LC.T    == 164
LC.U    == 165
LC.V    == 166
LC.W    == 167

LC.X    == 170
LC.Y    == 171
LC.Z    == 172
LBRACE  == 173
VERBAR  == 174
RBRACE  == 175
TILDE   == 176
DEL     == 177

.PAGE

                        ;
                        ; CHANNEL DEFINITIONS
                        ;
CHAN0   == 0            ; CHANNEL #0.
CHAN1   == 2            ; CHANNEL #1.
CHAN2   == 4            ; CHANNEL #2
CHAN3   == 6            ; CHANNEL #3

CHAN4   == 10           ; CHANNEL #4
CHAN5   == 12           ; CHANNEL #5
CHAN6   == 14           ; CHANNEL #6
CHAN7   == 16           ; CHANNEL #7

CHAN8   == 20           ; CHANNEL #8
CHAN9   == 22           ; CHANNEL #9
CHAN10  == 24           ; CHANNEL #10
CHAN11  == 26           ; CHANNEL #11

CHAN12  == 30           ; CHANNEL #12
CHAN13  == 32           ; CHANNEL #13
CHAN14  == 34           ; CHANNEL #14
CHAN15  == 36           ; CHANNEL #15

.PAGE

                        ;
                        ; TRANSFER CONTROL BLOCK DEFINITIONS
                        ; (XRB).
                        ;
XRLEN   == 0            ; LENGTH OF I/O BUFFER IN BYTES
XRBC    == 2            ; BYTE COUNT FOR TRANSFER
XRLOC   == 4            ; POINTER TO START OF I/O BUFFER
XRCI    == 6            ; CHANNEL # * 2 FOR ACCESS
XRBLKM  == 7            ; RANDOM ACCESS BLOCK NUMBER (MSB)

XRBLK   == 10           ; RANDOM ACCESS BLOCK NUMBER (LSB)
XRTIME  == 12           ; WAIT TIME FOR TERMINAL INPUT
XRMOD   == 14           ; DEVICE MODIFIERS
XRBSIZ  == 16           ; SIZE OF XRB IN BYTES

.PAGE

                        ;
                        ; FILE REQUEST QUEUE BLOCK DEFINITIONS
                        ; (FIROB)
                        ;
FQJOB   == 2            ; JOB NUMBER * 2
FQFUN   == 3            ; FIP/UUO FUNCTION REQUESTED
FQERNO  == 4            ; ERROR MESSAGE CODE AND TEXT BEGIN
FQFIL   == 4            ; CHANNEL # * 2
FQSIZM  == 5            ; FILE SIZE IN BLOCKS (MSB).
FQPPN   == 6            ; PPN OF USER ISSUING REQUEST

FQNAM1  == 10           ; 2 WORD FILE NAME IN RADIX-50 FORMAT.
FQEXT   == 14           ; FILE NAME EXTENSION IN RADIX-50.
FQSIZ   == 16           ; FILE SIZE IN BLOCKS (LSB).

FQNAM2  == 20           ; 3 WORD NEW FILENAME AND EXTENSION
                        ; IN RADIX-50 FORMAT.
FQBUFL  == 20           ; DEFAULT BUFFER LENGTH.
FQMODE  == 22           ; MODE INDICATOR.
FQFLAG  == 24           ; LOW BYTE -- OPENNED FILE'S FLAG WORD
                        ; AS RETURNED.
FQPROT  == 27           ; NEW PROTECTION CODE

FQDEV   == 30           ; 2 BYTE DEVICE NAME IN ASCII
FQDEVN  == 32           ; LOW BYTE --- DEVICE UNIT NUMBER.
                        ; HIGH BYTE -- DEVICE UNIT NUMBER FLAG.
FQCLUS  == 34           ; FILE CLUSTER SIZE FOR FILE CREATIONS.
FQNENT  == 36           ; NUMBER OF ENTRIES ON DIRECTORY LOOKUP
                        ; OR ENTRY PARAMETER.

FQBSIZ  == 40           ; SIZE OF FIROB IN BYTES.

.PAGE

                        ;
                        ; MONITOR CALLS (EMT'S)
                        ;
CALFIP  == 104000       ; CALL FIP
.READ   == 104002       ; READ FROM A DEVICE
.WRITE  == 104004       ; WRITE TO A DEVICE
.CORE   == 104006       ; CHANGE JOB IMAGE SIZE

.SLEEP  == 104010       ; SUSPEND A JOB
.PEEK   == 104012       ; PEEK AT MONITOR MEMORY
.SPEC   == 104014       ; DO DEVICE SPECIAL FUNCTION
.TTAPE  == 104016       ; ENABLE TAPE MODE

.TTECH  == 104020       ; ENABLE ECHO
.TTNCH  == 104022       ; DISABLE ECHO
.TTDDT  == 104024       ; DISABLE FULL LINE BUFFERING(ODT MODE)
.TTRST  == 104026       ; CANCEL O EFFECT.

.TIME   == 104030       ; RETURN JOB TIMING INFORMATION
.POSTN  == 104032       ; RETURN DEVICE HORIZONTAL POSITION
.DATE   == 104034       ; GET CURRENT DATE, TIME, JOB INFO
.SET    == 104036       ; SET KEY WORD BITS

.STAT   == 104040       ; RETURN JOB STATISTIOCS
.RUN    == 104042       ; RUN A PROGRAM
.NAME   == 104044       ; CHANGE PROGRAM NAME
.EXIT   == 104046       ; EXIT TO SYSTEM DEFAULT RTS.

.RTS    == 104050       ; CHANGE TO A NEW RTS.
.ERLOG  == 104052       ; LOG AN ERROR FROM THE RTS.
.LOGS   == 104054       ; CHECK FOR LOGICAL DEVICE NAME
.CLEAR  == 104056       ; CLEAR BITS IN KEY WORD

.MESAG  == 104060       ; MESSAGE SEND/RECEIVE
.CCL    == 104062       ; CHECK FOR AND EXECUTE A CCL COMMAND
.FSS    == 104064       ; TERMINATING FILE NAME STRING SCANNER
.UUO    == 104066       ; UUO HOOK

.CHAIN  == 104070       ; CHAIN TO A NEW PROGRAM
.PAGE

                        ;
                        ; FIP FUNCTION CALLS
                        ;
CLSFQ   == 0            ; CLOSE A CHANNEL
OPNFQ   == 2            ; OPEN AN EXISTING FILE ON A CHANNEL
CREFQ   == 4            ; CREATE A FILE
DLNFQ   == 6            ; DELETE A FILE

RENFQ   == 10           ; RENAME A FILE ON DISK OR DECTAPE.
DIRFQ   == 12           ; RETURN DIRECTORY INFORMATION.
UUOFQ   == 14           ; PROCESS UUO.
ERRFQ   == 16           ; RETURN ERROR MESSAGE TEXT

RSTFQ   == 20           ; RESET CHANNEL(S).
LOKFQ   == 22           ; LOOKUP A FILE
ASSFQ   == 24           ; ASSIGN A DEVICE
DEAFQ   == 26           ; DEASSIGN A DEVICE

DALFQ   == 30           ; DEASSIGN ALL DEVICES
CRTFQ   == 32           ; CREATE A TEMPORARY FILE
CRBFQ   == 34           ; CREATE A COMPILED FILE

.PAGE

                        ;
                        ; .UUO/UUOFQ SUBFUNCTION CODES DEFINED
                        ;

UU.ATR  == 177747       ; READ/WRITE ATTRIBUTES DEFINED

UU.CCL  == 177750       ; ADD/DELETE CCL COMMAND
UU.LOG  == 177755       ; SET NUMBER OF ALLOWED LOGINS
UU.RTS  == 177756       ; RUN-TIME SYSTEM CONTROL
UU.NAM  == 177757       ; SET FILE'S RTS NAME

UU.DIE  == 177760       ; SPECIAL SHUTUP LOGOUT
UU.ACT  == 177761       ; ACCOUNTING INFORMATION DUMP
UU.DAT  == 177762       ; DATE/TIME CHANGER
UU.PRI  == 177763       ; PRIORITY, RUN BURST, JOB MAX CHANGER
UU.TB2  == 177764       ; GET MONITOR TABLES, PART II
UU.BCK  == 177765       ; CHANGE FILE BACKUP STATISTICS
UU.HNG  == 177767       ; HANGUP/ENABLE A DATASET

UU.FCB  == 177770       ; RETURN FCB/DDB INFORMATION
UU.POK  == 177772       ; POKE MONITOR MEMORY
UU.TB1  == 177775       ; GET MONITOR TABLES, PART I
UU.NLG  == 177776       ; SET NUMBER OF ALLOWED LOGINS TO 1.
UU.YLG  == 177777       ; SET NUMBER OF ALLOWED LOGINS TO MAX

UU.PAS  == 000000       ; CREATE AN ACCOUNT
UU.DLU  == 000001       ; DELETE AN ACCOUNT
UU.CLN  == 000002       ; CLEAN A DISK PACK
UU.MNT  == 000003       ; MOUNT/DISMOUNT A DISK PACK
UU.LIN  == 000004       ; LOGIN
UU.BYE  == 000005       ; LOGOUT
UU.ATT  == 000006       ; ATTACH
UU.DET  == 000007       ; DETACH

UU.CHU  == 000010       ; CHANGE PASSWORD/QUOTA
UU.ERR  == 000011       ; GET ERROR MESSAGE TEXT
UU.ASS  == 000012       ; ASSIGN A DEVICE
UU.DEA  == 000013       ; DEASSIGN A DEVICE
UU.DAL  == 000014       ; DEASSIGN ALL DEVICES
UU.ZER  == 000015       ; ZERO A DEVICE
UU.RAD  == 000016       ; READ ACCOUNTING INFORMATION
UU.DIR  == 000017       ; RETURN DIRECTORY INFORMATION

UU.TRM  == 000020       ; SET TERMINAL CHARACTERISTICS
UU.LOK  == 000021       ; WILDCARD DIRECTORY LOOKUP
UU.CHE  == 000023       ; ENABLE/DISABLE CACHE
UU.CNV  == 000024       ; CONVERT DATE/TIME TO ASCII
UU.SLN  == 000025       ; SET/CLEAR SYSTEM WIDE LOGICAL NAMES
UU.SWP  == 000027       ; ADD/REMOVE SWAP, OVERLY OR ERROR
                        ; MESSAGE FILES

UU.JOB  == 000030       ; JOB CREATION
.PAGE

                        ;
                        ; MONITOR LOW MEMORY AREA DEFINED
                        ;
DATE    == 1000         ; CURRENT DATE IN INTERNAL FORM
TIME    == 1002         ; CURRENT TIME IN INTERNAL FORM
TIMSEC  == 1004         ; SECONDS TO NEXT MINUTE
TIMCLK  == 1005         ; TICKS TO NEXT SECOND
JOB     == 1006         ; JOB CURRENTLY RUNNING
NEXT    == 1007         ; NEXT JOB TO RUN

JOBDA   == 1010         ; POINTER TO CURRENT JOB DATA BLOCK
JOBF    == 1012         ; POINTER TO CURRENT FLAGS
IOSTS   == 1014         ; POINTER TO CURRENT I/O STATUS
JOBWRK  == 1016         ; POINTER TO JOB'S WORK BLOCK

JOBJD2  == 1020         ; POINTER TO JOB'S 2ND JOB DATA BLOCK
JOBRTS  == 1022         ; POINTER TO JOB'S RTS BLOCK
CPUTIM  == 1024         ; POINTER TO CURRENT CPU TIME BUCKET
.PAGE
```

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.)

We need space and time! Need 1170 RSTS time and office space in Southern California evening and weekend use! Will pay % of your lease costs. Call Allen (213) 394-3857.

RSTS/E BASIC+, DIBOL Programmers call or send resume to Scientific Placement, Inc., P.O. Box 19949, Houston, TX 77024; (713) 496-6100.

MEDIA TRANSFER SERVICES - We can transfer your data between any of the following media: RK05, RK06, RK07, RX01, and 800 BPI 9 Track MT. We handle most DIGITAL directory structures on a fast turn around basis and also offer programming services for any special transfer problems you may have. Call (201) 584-4026 Advanced Digital Data, 354 Route 206, Flanders, New Jersey 07836. Authorized DIGITAL DISTRIBUTOR.

FORTRAN Software Design and implementation at reasonable cost. RSTS, RT-11, and RSX are our specialty. Golden Triangle Computers, Box 1866, Orange, TX 77630, (713) 883-4588.

DOUBLE YOUR LA36
DECWRITER'S SPEED TO 60 CPS

The **ACCELEWRITER** converter enables you to convert the LA36 Decwriter from 10/15/30 to 10/30/60 CPS.

- Utilizes Decwriter's own built-in 60 CPS speed
- Plugs into logic board
- Compatible with Decwriter option boards
- Decwriter reconverts to "stock" in seconds
- Thousands sold
- In use for over 3 years
- Full one-year warranty
- $115 includes shipping

Brochure sent promptly at your request. Write or call us.

LARKS ELECTRONICS & DATA
P.O. Box 22, Skokie, IL 60077
(312) 677-6080

---

11 Blue 7 DEC LS1-11 Transduction, 1645-11 Sismet Road, Mississauga (416) 625-1907.

For competitive software needs write PÀR, 417 Tulsiani, Nariman Point, Bombay, India.

EDUCATORS: RSTS/E software needed. Contact Steve Dudley, Brunswick School, Greenwich, CT. 06830.

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

Now paying $5050 for used DEC RK07 drives. Call Alvin at (714) 732-0782.

Australian DEC specialists. Call (02)4766820.

Gary Oppenheimer-Consulting (9 years experience with RSTS & RSTS/E). N.Y.C. area (212) 787-2416.

'Art in RSTS/E' — Optimization, Sysgens, and other RSTS/E consulting. Jeffrey R. Harrow (404) 294-1999.

For Sale: Systems Industries Model 9500 SMD Controller complete with cables, manual $3,500 30 day warranty. Consolidted Funding Corporation, 2361 W. Devon, Elk Grove, IL 60007 (312) 766-1400.

PDP-11:RSX,RT, VAX:VMS Is there a VAX in your future? Do you need an APPLICATION MIGRATION EXPERT? Application and system consulting, analysis, design, configuration. Twenty years DEC/DP experience. Give me your tough problems. Carl E. Friedber, PH.D., In House Systems, P.O. Box 1063, New York, NY 10007 (212) 233-5470.

RSTS/E freelance work in San Diego, call Gary (714) 263-7321.

How-to COPYRIGHT SOFTWARE. $9.50 to Sofprotex, Box 271-P, Belmont, CA 94002.

Protect your PDP11 from damage with Nassau Systems over temperature protection unit, Box 19329, Cincinnati, OH 45219.

$200 RSTS/E*
WORD PROCESSOR
CBEDIT.BAS

Single Basic-Plus* program with CRT input, window edit and file save. Add, locate, change, replace, delete, block move & copy, standard paragraph append, etc. VT* series and Hazeltine terminal drivers. Others easy to add.

Fully formatted output (margins, justify, center, underscore, headers, page numbers, etc.) to terminal, disk or line printer. Bi-directional driver for Diablo. We use it daily for all our secretarial work.

9-Track $200 RK05 $260 ppd

T. F. Hudgins & Assoc., Inc.
P.O. Box 10946 Houston TX 77018
Woods Martin 713/682-3651

*TM Digital Equipment Corporation

H&R for electronics, 401 E. Erie Avenue, Philadelphia, PA 19134, request catalog.

Connect your office to the future with cableshare's inter-office communication system. Electronic Mail, Word Processing, Electronic Filing, Message Switching, and more for RSTS users.

DEC systems and components - buy and sell, Western Data Sales, Inc., 23520 Telo St. #12, Torrance, CA 90503, (213) 539-8917.

Systems Analysis, Design and Programming. RSTS and many other systems. Phone OIC, (213) 506-8700.

RSTS/E Medical Billing Programs, Pete Johnson (505) 255-3777.

---

3 11/34 MM11DP Core Boards, Craig (916) 666-1691.

RSTS BASIC and BASIC-PLUS-II Programming Service Company.

RSTS/E and BASIC-PLUS Programming Training Courses. Conducted at your site. Edge Data Corporation, 100 Franklin Street, Boston, MA 02110 (617) 451-1919.

BT-100AA $1595.00, LA-34DA $1095.00, DataGuard Corporation (312) 985-4615.

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
On Track Systems, Inc.
and a well known (and read)
Disk Directory expert.
CALL 24 HOURS
215—542-7008

```
                ;
                ; JOB UNIQUE LOW MEMORY LAYOUT
                ;

USERSP  == 400  ; DEFAULT STACK AREA
KEY     == 400  ; KEY WORD OF JOB'S CURRENT STATUS
FIRQB   == 402  ; FILE REQUEST QUEUE BLOCK
XRB     == 442  ; TRANSFER REQUEST QUEUE BLOCK
CORCMN  == 460  ; CORE COMMON
USRPPN  == 734  ; USER'S ASSIGNABLE PPN
USRPRT  == 736  ; USER'S ASSIGNABLE PROTECTION CODE

USRLOG  == 740  ; USER'S LOGICAL DEVICE TABLE

NSTORG  == 1000 ; END OF LOW MEMORY FIXED LAYOUT

.PAGE
JOBF    == 1012 ; POINTER TO CURRENT FLAGS
IOSTS   == 1014 ; POINTER TO CURRENT I/O STATUS
JOBWRK  == 1016 ; POINTER TO JOB'S WORK BLOCK

JOBJD2  == 1020 ; POINTER TO JOB'S 2ND JOB DATA BLOCK
JOBRTS  == 1022 ; POINTER TO JOB'S RTS BLOCK
CPUTIM  == 1024 ; POINTER TO CURRENT CPU TIME BUCKET

.PAGE
                ;
                ; JOB UNIQUE LOW MEMORY LAYOUT
                ;

USERSP  == 400  ; DEFAULT STACK AREA
KEY     == 400  ; KEY WORD OF JOB'S CURRENT STATUS
FIRQB   == 402  ; FILE REQUEST QUEUE BLOCK
XRB     == 442  ; TRANSFER REQUEST QUEUE BLOCK
CORCMN  == 460  ; CORE COMMON
USRPPN  == 734  ; USER'S ASSIGNABLE PPN
USRPRT  == 736  ; USER'S ASSIGNABLE PROTECTION CODE

USRLOG  == 740  ; USER'S LOGICAL DEVICE TABLE

NSTORG  == 1000 ; END OF LOW MEMORY FIXED LAYOUT

.PAGE
                ;
                ; JOB DATA BLOCK #1 DEFINED
                ; (JDB)
                ;

JDIOB   == 0    ; POINTER TO THE I/O BLOCK (IOB)
JDFLG   == 2    ; JOB STATUS FLAGS
JDIOST  == 4    ; I/O STATUS OF JOB
JDPOST  == 5    ; INDEX FOR BIT PATTERN TO POST
JDWORK  == 6    ; POINTER TO THE JOB WORK BLOCK (WRK)

JDJDB2  == 10   ; POINTER TO JOB DATA BLOCK #2
JDFLG2  == 12   ; MORE JOB STATUS FLAGS
JDSIZN  == 13   ; SIZE FOR JOB TO BE NEXT RESIDENCY
JDRTS   == 14   ; POINTER TO RUN-TIME SYSTEM BLOCK(RTS)
JDRESQ  == 16   ; JOB'S RESIDENCY QUANTUM

JDMCTL  == 20   ; JOB'S MEMORY CONTROL AREA
JDSIZE  == 26   ; JOB'S CURRENT SIZE IN K.

JDRESO  == 32   ; LB QUEUE DIST TO SET ON RESIDENCY
JDPRI   == 34   ; JOB'S PRIORITY
JDBRST  == 35   ; JOBS RUN BURST
JDMSIZ  == 36   ; JOB'S PRIVATE MEMORY MAXIMUM
JDSWAP  == 37   ; JOB'S SWAP PARAMETERS

                ; JOB FLAG (JDFLAG) BIT ASSIGNMENTS
JFPOST  == 000001 ; CHECK 'JDPOST' FOR POSTING INFO
JFIOKY  == 000002 ; POST THE I/O STATUS AND KEY WORD
JFCC    == 000010 ; C TYPED BY USER
JF2CC   == 000020 ; 2 QUICK C'S TYPED BY USER
JFFPP   == 000040 ; FPP EXCEPTION AST NEEDED
JFGO    == 000100 ; FORCING JOB (NO I/O REDO)
JFREDO  == 000200 ; RE-DO THE I/O CONDITION
JFSYST  == 000400 ; TEMPORARY PRIVILEGES POSSIBLE
JFFPP   == 001000 ; SAVE/RESTORE ASYNCHRONOUS FPP CONTEXT
JFPRIV  == 002000 ; JOB HAS PERMANENT PRIVILEGES
JFSYS   == 004000 ; JOB HAS TEMPORARY PRIVILEGES ACTIVE
JFNOPR  == 010000 ; JOB IS NOT YET LOGGED IN
JFBIG   == 020000 ; JOB CAN EXCEED ITS PRIVATE MEMORY MAX
JFLOCK  == 040000 ; JOB IS NOT TO BE SWAPPED
JFSPCL  == 100000 ; SPECIAL CONDITION (CHECK JDFLG2)

                ; JDFLG2 BIT ASSIGNMENTS IF JFSPCL
JFCEXT  == 001  ; PRESERVE THE JOB'S CONTEXT
JFPRTY  == 002  ; PARITY FAULT
JFRUN   == 004  ; RUN ENTRY
JFSWPR  == 010  ; RUN-TIME SWAP (LOAD) ERROR
JFSTAK  == 020  ; SP STACK OVERFLOW
JFSWPE  == 040  ; SWAP ERROR FOR JOB
JFKIL2  == 100  ; KILL THIS JOB IMMEADIATELY
JFKILL  == 200  ; KILL THIS JOB AFTER CLEANUP

.PAGE
                ;
                ; JOB DATA BLOCK #2 (JDB2) DEFINED
                ;

J2TICK  == 0    ; NUMBER OF CLOCK TICKS UNCONVERTED
J2CPU   == 2    ; CPU TIME COLLECTED FOR JOB
J2CON   == 4    ; CONNECT TIME COLLECTED FOR JOB
J2KCT   == 6    ; KCT COLLECTED FOR JOB

J2DEV   == 10   ; DEVICE TIME COLLECTED FOR JOB
J2KCTM  == 12   ; KCT COLLECTED FOR JOB (MSB)
J2CPUM  == 13   ; CPU TIME COLLECTED FOR JOB (MSB)
J2NAME  == 14   ; 2 WORD JOB'S PROGRAM NAME IN RAD-50

J2DRTS  == 20   ; POINTER TO DEFAULT RTS FOR JOB
J2MPTR  == 22   ; POINTER TO JOB'S RECEIVER ID BLOCK
J2PPTR  == 24   ; LARGE DATA POSTING POINTER
J2PCNT  == 26   ; LARGE DATA POSTING COUNT

J2PPN   == 30   ; JOB'S PPN
J2UFDR  == 32   ; RETRIEVAL POINTER TO JOB'S SY: UFD

.PAGE
```

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 photos or manuscripts. Send news releases to: RSTS NEWS RELEASE, P.O. Box 361, Fort Washington, PA 19034.

October 27, 1980

QUODATA ANNOUNCES LOW COST PDP-11 WORD PROCESSING/DATA MANAGEMENT SYSTEMS

Hartford, Connecticut — QUODATA Corporation announces its new series of low cost QW systems which provide simultaneous word processing and data management. The systems are built around the PDP-11 family of computers from Digital Equipment Corporation. Besides word processing, features include large file capacities, comprehensive report generation, and flexible sorting and selection capabilities.

QW systems, says QUODATA spokesman, "embody a different concept than that employed by most of the giants. Unfortunately, they use the approach of overlaying data management on otherwise excellent word processing software. The result is that file sizes, sorting limitations, and other restrictions preclude proper data management.QUODATA, takes the reverse tack. We start with the most established and popular data processing timeshared operating system available on low cost computers: RSTS/E. Then we add the QWORD word processing software to the already proven QDMS data management system."

Inexpensive Timesharing With Power

At the low end, users can start with a system which costs as little as $39,900. The QW620 consists of DEC's new, popular PDP—11/23 with 192K bytes of memory, four port multiplexor, two disk drives with 20 megabytes of removable storage, VT100 video console, QDMS, QWORD, and RSTS/E software, DEC will install and maintain the hardware under a service contract.

Users can obtain a processor as large as the new DEC PDP-11/44 or even the 11/70, capable of handling well over fifty users simultaneously. More than a billion bytes of disk storage can be added to these systems, as well as almost any combination of terminals, letter quality printers, and high speed draft or data processing printers.

Word Processing and Data Management Simultaneously

The QW series allows simultaneous use by a number of people for both word processing and data management. On the one hand, you can prepare single documents or mass mailings. On the other hand, you can perform data processing functions such as interactively update files and access data, sort and select, run application programs or generate instant reports.

The list processing option is an example of one function that combines both word and data management. It allows you to produce the same kind of output over and over again — a letter, standardized form, row in a table such as telephone directory; even with certain data varying from one piece of output to the other. Specific data may be taken from records in the data base and merged into the document which is then reproduced a number of times. "Fields" within the finished documents will contain data taken from individual records within the data base.

The QWORD software is 'menu driven' and therefore very versatile. Users can add, delete, transpose or duplicate text. Words, sentences, paragraphs, pages or even entire documents can be manipulated easily. Special color-coded keys exist to facilitate such tasks as setting margins and tabs, centering, and deleting or transposing parts of documents. QWORD is functionally similar to DEC's WPS/8 word processing system for the PDP8.

In addition to color-coded keys, the system prompts the operator to provide next-step choices as well as error messages. The system can also store commonly used abbreviations and paragraphs. Stored text saves time and reduces error by allowing insertion of commonly used phrases, company or product names, paragraphs or other information by simply using a few key strokes.

Instant Reports

Reporting is a strong suit. Two variations of the report writer exist. One is called PDQ, which permits non-technical staff to generate instant reports under computer guidance. A more sophisticated variation of the report writer can handle virtually any format desired. Control breaks, extensive calculations, subtotals and many other features are standard. Sort and selection options are easy to use. More than a million records may be sorted, and over 200 levels of selection critera exist.

Proven Components Integrated

The PDP11 is the world's most popular minicomputer family. RSTS/E is recognized as the easiest to use and most popular interactive timesharing system available on a minicomputer. It is versatile, sophisticated, and proven operating software, boasting some 20,000 installations. RSTS/E supports concurrent, interactive and batch processing, BASIC-plus, BASIC-plus II, COBOL, FORTRAN, PASCAL, APL, RPGII, PL/I, assembler and more programming languages. This same RSTS/E can run on 11/23, 11/34, 11/44, 11/60 and 11/70 computers with as many as 63 independent user jobs and connections to 127 terminals. QUODATA, not DEC, supports RSTS/E on the 11/23.

The QDMS Data Management Systems has received high marks from users for its exentsive report generation facilities, and ease of use by non-technical personnel. Users can build data files quickly, and obtain reports from several files simultaneously. QDMS saves programming time and lets even first-time users create and run reports themselves.

Additional inquiries should be addressed to Barbara Westley, Marketing.

February 6, 1981

RABBIT-2 PERFORMANCE ANALYSIS FOR VAX/VMS AND RSTS/E USERS RELEASE 2.0

West Palm Beach, Florida — RAXCO Inc. announces Release 2.0 of RABBIT-2, a computer performance analysis system for VAX/VMS and RSTS/E environments.

New features, enhancements, and options increase system efficiencies, extend capabilities and add new analytical techniques under the new release.

New commands include PROFILE, which will quickly provide a statistical analysis of user resource demand; RATIO displays resource consumption per connect hour or per second; RANK will list users in order of demand; WHO will provide user lists for relevant time period; HELP will list over 30 commands and associated formats.

One new option available under RSTS is the ability to determine the percentage of time users spend in various states, such as RUN, SLEEP, TERMINAL INPUT/OUTPUT etc.

Another option for both VMS and RSTS is program efficiency analysis whereby individual program statistics are generated. The program is identified, and frequency plus quantity of usage are developed.

RABBIT-2 is a command driven analytical tool for investigating computer resource demand, bottlenecks, and throughput over specified time periods. The system data may be viewed in the aggregate or split into batch, interactive and detatched segments for analysis.

RABBIT-2 is priced from $99/month rental or $2495 permanent license and is available through RAXCO and Distributors in the U.S., U.K. and Canada.

RAXCO provides a wide range of operational software for DEC computers ranging from JOB ACCOUNTING, RESOURCE AUDITING AND BILLING, FILE SECURITY, DATA MANAGEMENT and FINANCIAL PLANNING.

For more information contact: RAXCO Inc., 3336 N. Flagler Drive, West Palm Beach, Florida 33407, U.S.A. Phone: (305) 842-2115

February 1981

FINAR INTRODUCES "FINESS" HIGH-SPEED FINANCIAL MODELING SOFTWARE FOR PDP-11

Denver, Co. — Mr. Tony Kobine, Marketing Director at Finar Systems, Ltd. announced the availability of a new, high-speed version of the successful FINAR, Financial Analysis and Reporting language, for the DEC PDP-11.

Called FINESS — "FINAR Extra Speed", this new version takes advantage of the overlay features of Basic-Plus-2. The result is a product that runs several times faster than the Basic-Plus version, and has the same appearance to both the user and the internal file structures.

FINESS will provide financial managers and planners with the same easy-to-use modeling system as FINAR for budgeting, forecasting, consolidation, investment evaluation and similar activities, at higher speeds — especially important when analyzing "what if?" alternatives.

Prices for FINAR start at $18,000. For further information call or write: Tony Kobine, Finar Systems, Ltd., 6000 E. Evans, Suite 2-300, Denver, CO 80222 (303) 758-7561.

February 1981

FINAR — FINANCIAL ANALYSIS AND REPORTING SOFTWARE IS NOW AVAILABLE FOR DEC VAX-11

Denver, CO. — Mr. Tony Kobine, Marketing Director at Finar Systems Ltd. announced the availability of FINAR on the DEC VAX-11, running in "native-mode" under VMS.

FINAR, the Financial Analysis and Reporting system has been available for four years on the PDP-11, and has provided managers and planners with an easy-to-use modeling system for budgeting, forecasting, consolidation, investment evaluation and similar activities.

"The VAX version of FINAR is a powerful addition to the software," said Mr. Kobine. "It will enable users of this system to carry out high-quality, high-speed modeling — especially important when evaluating alternatives in 'What if ?' analysis."

Current FINAR users who follow the natural migration from PDP-11 to VAX can continue with no external change in function, and a simple transfer of data and program files.

Prices for FINAR start at $18,000. For further information, call or write: Tony Kobine, Finar Systems, Ltd., 6000 E. Evans, Suite 2-300, Denver, CO. 80222, (303) 758-7561.

January 30, 1981

COSAP STATISTICAL PACKAGE FROM LAWRENCE UNIVERSITY

Appleton, Wisconsin — Lawrence University has announced Version 2B of COSAP, a Conversationally Oriented Statistical Analysis Package for RSTS/E.

Version 2B of COSAP brings RSTS/E users a coherent and varsatile set of routines featuring: convenient terminal data entry; flexible commands for editing or transforming data; many types of analysis for discrete and continuous data; a powerful select mode; a uniform data format; a conversational environment; ample capacity (up to 32767 observations on up to 4095 double-precision variables or 8191 single-precision variables); only 8 KW of memory per user.

The licensed package consists of the BASIC-PLUS source code for all modules, internal table files, and test data files. Printed documentation includes detailed instructions for installation, codebooks for test data files, and the comprehensive User's Guide to COSAP.

Although Version 2B is a proprietary licensed product, a spokesperson from Lawrence noted that the 1973 version of COSAP would remain in the DECUS program library because it has reputedly been one of the most frequently requested items in the entire library.

The software product announcement for COSAP Version 2B may be requested from: Computer Center Publications, Lawrence University, Box 599, Appleton, Wisconsin, 54912.

———————

List of Advertisers

LETTERS to the RSTS Pro . . .

. . . continued from page 6

Here's what our new photo contests inspired. (RSTS Professional, Vol. 2, #4, p. 83, Dave Mallery, "Best Caption"; p. 91, William Shakespere at Stratford upon Avon. The question concerning Stonehenge, (Salisbury Plain, Wiltshire, England), p. 19, was not intended as a contest, but simply a rhetorical statement. However, because we received a few comments we decided to include the photo as part of this months feature.)

Dear Dave and Carl,

I'd like to enter the following caption in your contest.

"You mean I can't play DUNGEON on a VAX?!!"

I enjoy you magazine very much and look forward to the next issue.

Sincerely, Ronald Charbonneau
Pittsfield School District
Pittsfield, MA

—

Dear RSTS Protessional:

It's anyone's guess as to what Dave Mallery was thinking or about to say in the picture, but he certainly looks surprised to me.

After San Diego, I would offer a caption of either:

"You're running 63 jobs with 96 ports on an 11/70 under Version 7.0 and NO SMALL BUFFER problems!! Come on!!!"

OR

"Did you just say that you know how to solve the SMALL BUFFER problem once and for all???"

I really enjoy reading your magazine and anxiously await its arrival. I think that many of the articles contain information that is useful to many people at all levels. I wish that we could see the RSTS Professional distributed more often.

Very truly yours,
Arnold V. Fish, President
Integrated Computer Systems, Inc.

—

"Oh-h-h-h No-o-o-o! I don't speak his language."

Ed Hamaker
Amos Press Inc., Sidney, OH

—

Carl & Dave

Concerning your photo caption contest, how about—

"You mean someone has solved the small buffer problem?!!!!".

I'm still working on the last issue's TECO problem. A few months ago I was driving East on I-70 just out of Columbus with my wife and saw a similar truck. I said "Hey, TECO!" and my wife just stared at me and said "Huh?". Some people just don't understand.

I really enjoyed your show at San Diego, but I thought you had said that your long awaited "How to build a well structured disk" article was to appear in the December issue. I guess things get a bit blurry after midnight.

Bob Ainsley
National Revenue Corp.
Columbus, OH

Blurried and Hurried, Bob. See "Disk Structure Notes", this issue.

—

Dear RSTS Professional:

Below is a caption I feel suitable:

"Small Buffers?? — Oh, I've got 2 of those."

Sincerely, Kim Branch
Daniel International Corp.

Gentlemen:

Your current photo (p. 19) is of Stonehenge. Many believe that this is indeed a computer, used to calculate astronomical data such as the date of the equinox which was of great importance to prehistoric farmers and hunters. I expect it was reliable, or at least minimal downtime!

Everyone involved with timesharing at our installation really enjoys your magazine; keep up the good work.

Donald E. East
First International Services Corp.

—

Dear RSTS Pro:

In reference to the photo contest (p. 19), Some people think that it was a start. Actually the object in the photo is called a Dolmen. A group of these placed in a circle is called a Cromlech. The most famous Cromlech is Stonehedge. It is believed to have been built around 2000 B.C. and seems to have been a kind of astronomical observatory. How's that!!!!!!!

Sincerely, Joseph M. Nyikos
Union-Tribune Publishing Co.

THAT was just great!!!!!!!

—

Here's someone who has included a little something for everyone.

Gentlemen . . .

Page 19 — Yes. Stonehenge.

Page 83 — "Hey you guys . . . give me a break will you."

Page 91 — Bill Shakespere at Stratford upon Avon.

Please remind Users that during a single user Sysgen one should allocate all except 2 job slots to XBUF. (Free available memory that was.)

How does one reset the statistics withot the switch register - i.e., after RDC have nicked it?

Bye Y for now
Peter Dick
Silver Programs, London

OK, Peter, you get a T-shirt for at least one of the above.

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

LOOK

at the "tear-out" cards in this issue.

There's a "HOT to COLD" form for rating our articles. Let us know what is most and least interesting to you.

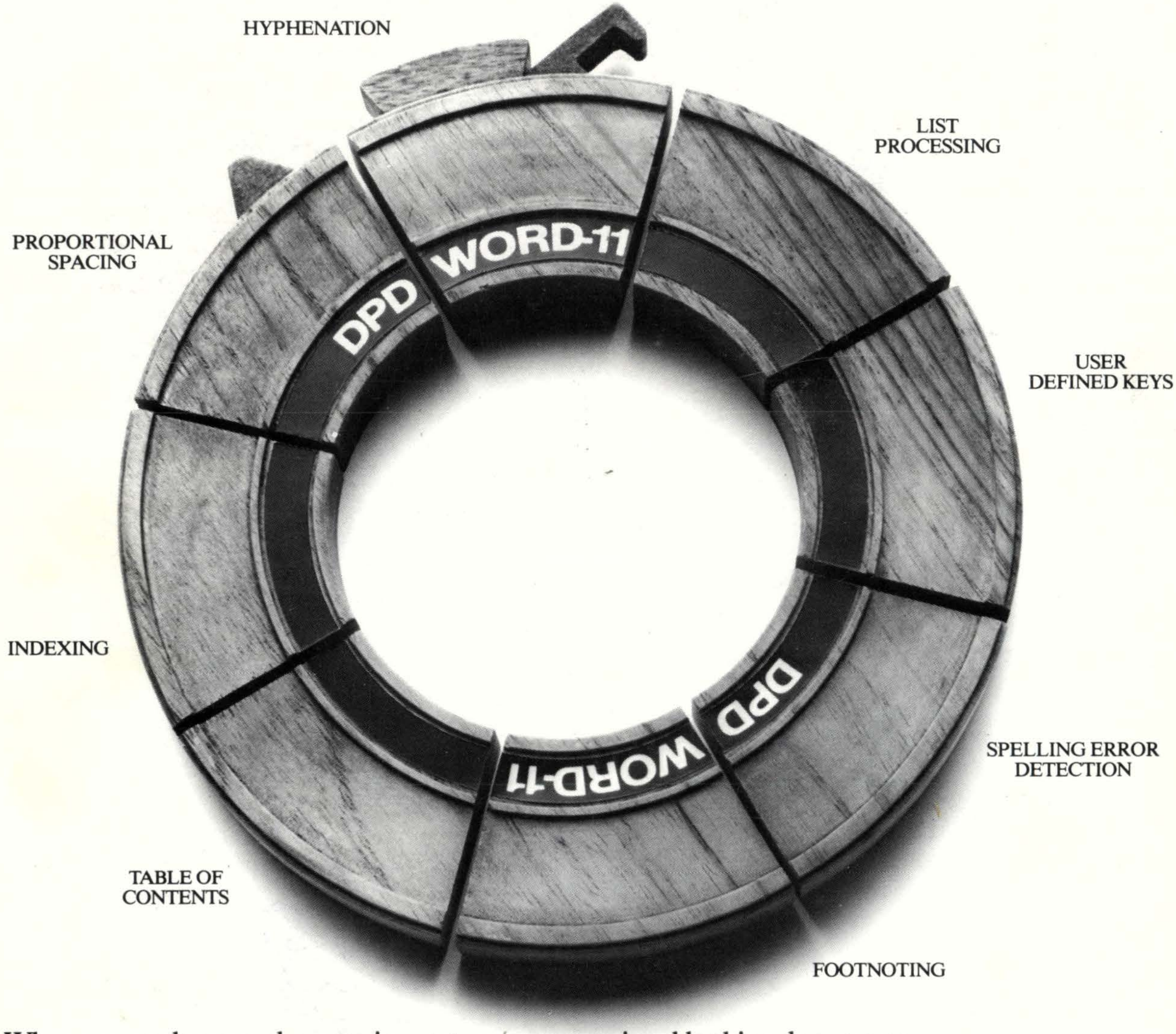
There's a FREE gift for you. Bring in new subscribers and collect rewards. See "BOUNTY HUNTERS" card.

RSTS Professional Rate Increase

Effective May 1, 1981

U.S. (3rd Class)	$25
Canada & U.S. (1st Class)	$35
Foreign (Air Mail)	$40

The Whole Truth About PDP-11 Word Processing.



When you explore word processing systems, you'll find a number of systems that offer part of the package. And frankly, if you're only looking for text editing, almost any system will do.

But if you're looking for full word processing capabilities, you should insist on a system that can handle all of the tough tasks you'll encounter.

For starters, a good system will have list processing, the vital function that generates correspondence, reports, even statistical analyses. It should have user defined keys to simplify repetitive operations.

And a good system can handle the details. Like spelling error detection, automatic renumbering of footnotes, tables of contents and indexes. And proportional spacing and hyphenation for professional looking documents.

A good system can be shared. It should be able to support up to fifty terminals. And it must be backed up by successful installations and a strong service team.

After you've examined the options, we think you'll select WORD-11, the only system with all the sophisticated features you could want.

For the details, please call Data Processing Design Inc., Corporate Office; 181 W. Orangethorpe Ave., Suite F, Placentia, CA 92670. (714) 993-4160. Telex 182-278. New York Office: 420 Lexington, Suite 647, New York, NY 10170. (212) 687-0104.

Data Processing Design, Inc.

AUTHORIZED digital COMPUTER DISTRIBUTOR

Overseas Distributors: *Management Information Services PTY. LTD. Melbourne, Australia* · *Jenson LTD. Bristol, England* · *Systime LTD. Leeds, England* · *Network Computer Services PTY. LTD. Sydney, Australia*