BASIC SIG

August 1983 Issue

This is the FIRST Subscription Service Issue

Copyright © Digital Equipment Corporation 1983
All Rights Reserved

It is assumed that all articles submitted to the editor of this newsletter are with the authors' permission to publish in any DECUS publication. The articles are the responsibility of the authors and, therefore, DECUS, Digital Equipment Corporation, and the editor assume no responsibility or liability for articles or information appearing in the document. The views herein expressed are those of the authors and do not necessarily express the views of DECUS or Digital Equipment Corporation.

Printed in the U.S.A.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	PDT
DECnet	Digital Logo	RSTS
DECsystem-10	EduSystem	RSX
DECSYSTEM-20	IAS	UNIBUS
DECUS	MASSBUS	VAX
DECwriter	PDP	VMS
		VT

UNIX is a trademark of Western Electric Corporation

Converting BASIC-PLUS-2 V1.6 Tasks to run on RSTS/E V8.0

by
William Tabor
Computer Products, Inc.
1400 N.W. 70th St.
Fort Lauderdale, Fla.

At the spring DECUS symposium in St. Louis Missouri, users of BASIC-PLUS II
learned the following:

* DEC will be releasing BASIC-PLUS II Version 2.1 shortly.

* RSTS/E Version 7.2 will not support BASIC-PLUS II Version 2.1.

* RSTS/E Version 8.0 will not support BASIC-PLUS II Version 2.0 or Version 1.6.

That means that all of the BASIC-PLUS II Version 1.6 programs that are now on your system will not run when you bring up RSTS/E Version 8.0 and that you can not compile any program now in BASIC-PLUS II Version 2.1.

Therefore the BASIC-PLUS II user is faced with the situation that all BASIC-PLUS II programs MUST be compiled as soon as RSTS/E Version 8.0 is installed before they will run.

A temporary workaround can be used so that you may be able to use your BASIC-PLUS II Version 1.6 programs on your RSTS/E Version 8.0 system. You will have to recompile all of your programs at a more convient date so that they will be compiled in BASIC-PLUS II Version 2.0.

The following four step method will allow you to move your BASIC-PLUS II

Version 1.6 tasks from your RSTS/E Version 7.2 system to your RSTS/E Version 8.0 system.

1. Copy all tasks onto your system.

2. Copy your BASIC-PLUS II Version 1.6 resident library to the new system.

3. Copy your RMS Version 1.8 resident libraries to the new system making sure that you rename the RMS libraries to

(continued page 2)

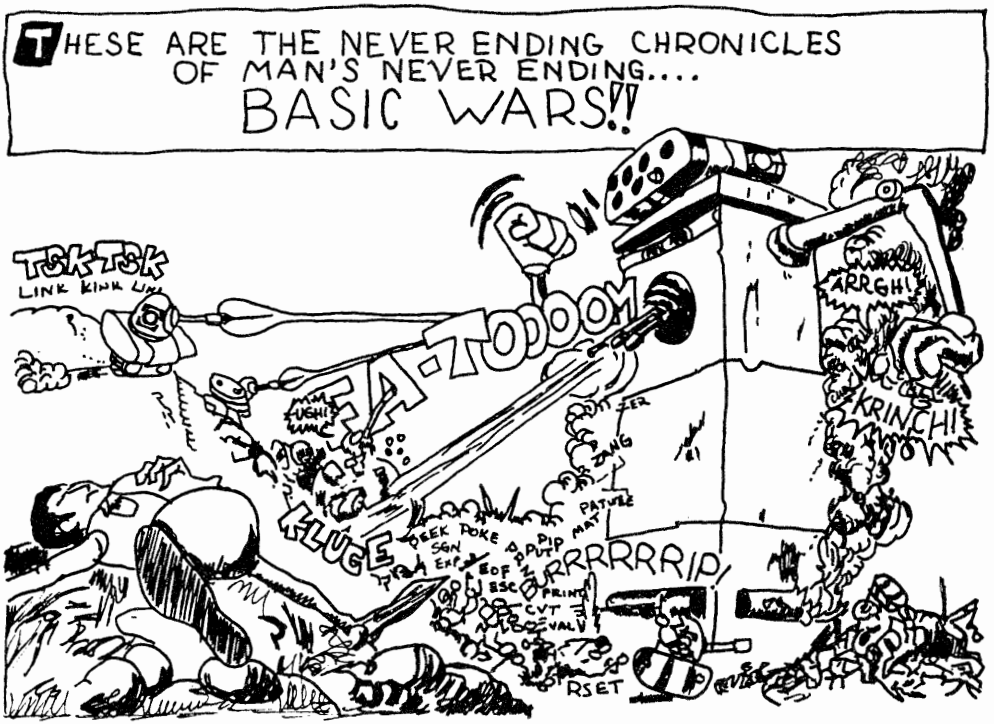
of the new name of RMSOLD

4. Run ONLPAT to change the name of the resident library in the task header to access the new library name

```
RUN $ONLPAT
COMMAND FILE ? <CR>
FILE TO PATCH ? <YOUR TASK NAME> <CR>
BASE ADDRESS ? 0 <CR>
OFFSET ADDRESS ? 40 <CR>          Octal address of first
                                  resident library name
                                  in the task header.

000000 0000040 071233  ? <LF>     Note: This is the first
                                  three letters of RMSRES
                                  in RAD50.

000000 0000042 070533  ? %OLD     Enter the new "old" to
                                  change the RMS library
                                  name to RMSOLD.

0000000 0000000 0000000 ? ^Z      To terminate the patch.
```

If the value at address 40 does not match for RAD50 of RMS then you need to go to the next address for library names in the task header. To get there add an octal 34 to the previous octal address.

For further information on the layout of the resident library names contained in the task header please refer to the TASK BUILDER Manual.



Baking Peach Pies

by Shel Jones

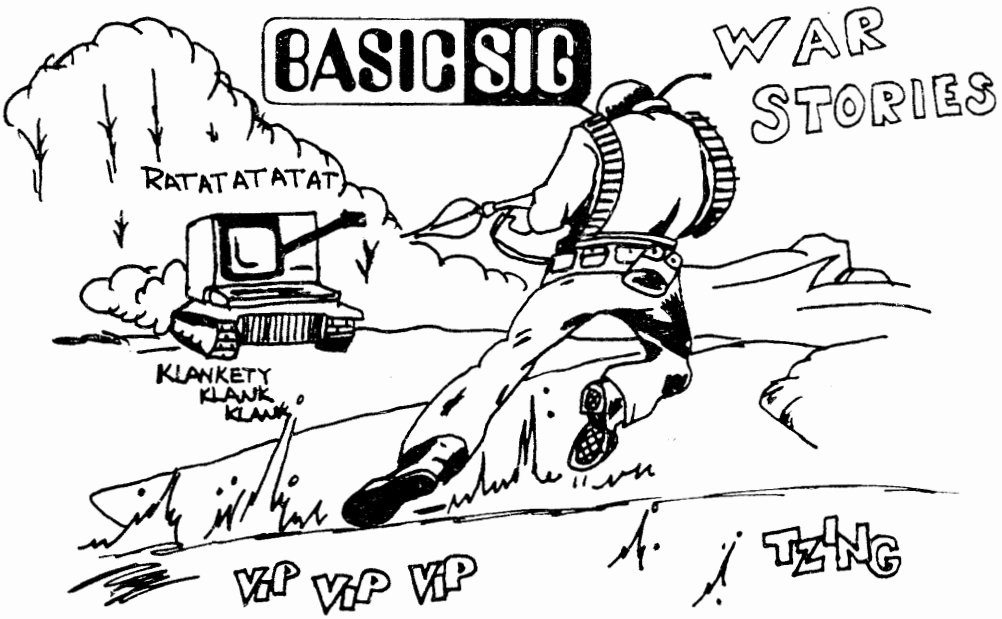
NCA
388 Oakmead Parkway
Sunnyvale, CA 94086-5486

The Oakland Tribune (The Peach) used a Harris 2200 automated typesetter with a PDP-11/05 as its heart. Everyone who has any knowledge of computers knows that they need to "keep their cool". To ensure this, the thermostat in the computer room was equipted with an alarm which rang in the security office.

While making his rounds a security guard found that the computer room air conditioner had shut down and it was hot enough to bake peach pies.

He opened all the doors to cool down the room and called the repairman to fix the air conditioning unit.

After bringing the unit back on line the repairman suggested that the heat alarm would work better if the switch was left in the "ON" position.



BASIC RULES

Any program, when running, is obsolete.

If a program is useful, it needs to be changed. If it is useless it needs to be documented.

Every program expands to fill all avaiable memory.

Program complesity grows until it exceeds the capability of the programmer who must maintain it.

Every program costs more and runs longer.

Make it possible for programmers to write BASIC programs in English, and you will find that programmers can not write basic English.

If at first you don't succeed, change you data.

Adding programmers to a late software project makes it later.

Any time you wish to demonstrate a program, the number of faults is proportional to the number of viewers.

Each problem solved introduces a new unsolved problem.

What the New DECUS Will Be

Clair Goldsmith
University of Texas
San Antonio, TX

Clair Goldsmith, Session Chairperson
University of Texas
San Antonio, TX

Reported by Micheal Kintz, DECUS Scribe Service

The Organizational Development Task Force established in 1981 functions "...to evaluate and recommend methods and practices which will benefit DECUS leadersnip and users in the more effective delivery of DECUS services." The Task force is developing a project designed to give DECUS a new look.

The Organizational Development project stems from the 1981 and 1982 Leadership Interning which indicated the need for:

1. Clarification of DECUS's purpose.
2. Reevaluation of DECUS's structure, control mechanisms and interrelationships.
3. An executive board of managers and policy makers.
4. Emphasis on improved communications.
5. Career development for leadership.
6. More leadership development.

In order to achieve a new look for DECUS, the project developed a strategic plan which consists of several phases: mission, goals, and action plans; organizational structures; transition plans; and human resources plans for volunteers. Clair Goldsmith stated that the function of the strategic plan is "to promote the exchange of information processing related information among users of Digital Equipment Corporation products."

The goals of the OD project are to actively represent the interests of members, the establishment of activities to promote

information exchange, the design and implementation of strategies to encourage active membership, effective chapter management, maintenance of the special DECUS relationship with Digital Equipment Corporation, and support of communications between suppliers of products compatible with DIGITAL equipment and users of DIGITAL equipment.

Action plans will be enacted to implement goals of the strategic plan and will become basic activities necessary to continued project progress in the following year.

The organizational structure proposed on May 20, 1983 will not be rigid, but will let elements come and go as appropriate. The currently planned elements of the organizational structure are:

1. An Executive Board with 9 to 11 members (1 DIGITAL representative, 6 voted in by members-at-large, and 2 from the Management Council.) The Executive Board will be responsible for long term planning.

2. A Management Council with 13 or 14 members to include delegates from Functional Groups, SIGs, and LUGs. The Management Council will manage the day-to-day activity of the organization.

3. A Chief of Staff who will manage the DECUS professional staff. The Chief of Staff will serve on the Executive Board as a non-voting member.

4. A five member Recruitment Committee. The Recruitment Committee will head new leadership development.

5. Placement of SIGs and LUGs at the national level of the organization.

6. Several Functional Committees chartered by the Executive Board, to include: Library, Symposium, Publications, Standards, and Special Projects groups.

The transition plan will consist of reviewing and incorporating the data input from the Spring U.S. Symposium, Saint Louis, regarding the June meeting of the Task Force. Goldsmith said that members should watch for feedback in DECUSCOPE, the Pageswapper, and SUGgestions.

After reviewing and incorporating the data input, the Task Force will design a transition plan, revise Bylaws, and submit the revised Bylaws to the membership for approval. Goldsmith said the Task Force would like to have the Transition Plan completed by June 1984.

The final phase of the OD project is the human resources plan for volunteers, which will consist of leadership and career development and volunteer recognition and rewards.

THE
BEAR
FACTS

DECUS Scribe Service Launched at Spring Symposium

By Ted A. Bear
BASIC SIG newsletter Editor

You might notice that there are more articles in this newsletter than ever before. That is because Ralph Stammerjohn put together the eight wonder of the world. His brain child is the DECUS Scribe Service became a reality in St. Louis. He recruited college students from the St. Louis area, armed them with pad, pencils and DEC PC 350's and sent them out to cover the sessions. You can judge some of the fruits of their combined labors for yourself. About the only thing that the DECUS Scribe Service can not do is come up with BASIC War Stories, that is still up to you.

BASIC-PLUS-2 Status & Directions

Joe Mulvey

BASIC Development Manager

Since the release of BASIC-PLUS-2 V2.0, we have been working on a series of update releases. V2.1 is now released; it fixed all V2.0 bugs known at the time it was submitted. It also provides some incremental performance improvements for RSTS/E systems. Subsequent update releases will address larger improvements in compile-time performance as well as fixing bugs. Because there has been confusion in the field with respect to BASIC-PLUS-2, we hope that the following information will be of use to you.

1. Supported Versions of BASIC-PLUS-2

   - V1.6 is currently supported on RSTS/E V7.2 and RSX-11M/M-PLUS V4.0/2.0
   - This support had been planned to be discontinued after July, i.e., six months after V2.0 FCS. As a result of the feedback we have been getting and requests from customers at Spring 1983 US DECUS, we have decided to continue support of V1.6 through November 1983. This date coincides with the termination of support of RSTS/E V7.2 but V1.6 will be supported on both RSTS/E and RSX during this time.
   - V1.6 is not supported on RSTS/E V8.0 or RSX-11M/M-PLUS V4.1/2.1 because it does not run with RMS V2.0.
   - V2.0 is supported on RSTS/E V7.2 and RSX-11M/M-PLUS V4.0/2.0. We will support V2.0 through November 1983.
   - V2.1 is now released and is supported on RSTS/E V8.0 and RSX-11M/M-PLUS V4.1/2.1. V2.1 will not run on RSTS/E V7.2 because of its dependency on RMS V2.0.

   V2.1 will run on RSX-11M/M-PLUS V4.0/2.0 but with some restrictions:

   - you cannot use the /CLUSTER qualifier
   - you cannot use the NAME AS statement

   These restrictions along with other pertinent information are detailed in the cover letter sent with the V2.1 kit to RSX users.

2. Transition from BASIC-PLUS-2 V1.6 to V2.0

- We believe that extending the support of V1.6 beyond July will help ease the transition to BASIC-PLUS-2 V2.1 and the newer versions of the operating systems.

- In response to requests received at DECUS, we agreed that we would also investigate what it would take to get BASIC-PLUS-2 V1.6 to run on RSTS/E V8.0. We have done that investigation and as a result, we will be publishing a Small Buffer article detailing the steps that must be taken in order to do this. This article will cover moving files, renaming files, changing task header names, and patching the V1.6 code. As soon as we complete the testing now underway and verify that these procedures are successful, the article will be published.

- We emphasize that this is strictly an interim mechanism to be used only in extreme situations. We will support V1.6 only until November 1983.

3. Bugs reported in V2.0

- We always strive to have zero bugs in our products. We had a six-month field test period (longer than usual) because of all the new functionality in V2.0. We polled sites several times both during and after field test. Prior to SDC submission, we corrected all V2.0 problems reported via QAR and detected in internal tests. Except for the areas in V2.0 where we tightened the rules and syntax from V1.6 (which were documented as part of the documentation set), we believed most V1.6 applications would run without problems. Obviously our field and internal testing did not detect all the problems that have been reported since V2.0 was released.

- In addition, information disseminated to the field by Digital (but not by Engineering) about V2.0 was sometimes inaccurate and misleading. We have attempted to solve this since V2.0 was released, but have not been completely successful. The development group was very much surprised at DECUS by some of the information that the users had received from Digital. Most surprising was information that said that development did not intend to fix any bugs. In V2.1, we have corrected all known problems in V2.0. We have worked very closely with both the CSC and US Area support groups to insure that known problems were corrected and that the product validated with existing test packages and applications.

- We will continue to fix all known problems and will use the update mechanisms available to us from the operating systems. We solicit your assistance in this problem

solving process.

4. Compile Time Performance

- In planning V2.0, as a result of direct user input, one of the major goals that we addressed was programmer productivity. We feel that programmer productivity means features that minimize the overall cost of writing software. These features are improved error detection and reporting, structured programming constructs, I/O performance using RMS V2.0, and new data types to support additional processing and integration. These new features would, of course, affect compilation speed. Because compiler speed is generally a small part of overall program development and because we felt that time spent compiling would become proportionally smaller with these new programmer productivity features, we felt that some compiler performance could be sacrificed. Our field test sites and other users to whom we talked concurred with this.

- We established metrics for our (limited but representative) benchmark programs that were met prior to submission.

- In this area also, we polled all of our field test sites during and at the conclusion of field test to get their feedback on performance. All reported that it was acceptable given the new features tradeoff and that the product could be submitted.

- Now there appears to be a wide disparity between our compile-time benchmark programs and those encountered at several customer sites. We have not been able to determine a common factor causing some sites to experience longer compile times than others. We have, however, been able to improve some of the problem sites' performance by suggesting some system tuning measures. For example, placing the compiler on a different disk from its work files, and, for RSTS/E systems, enabling data caching.

- During our diagnosis of performance problems, we did notice that some sites were compiling their programs on V2.0 differently than they were on V1.6. In particular, some were compiling on V2.0 using the new V2.0 features /LIST and /CROSSREFERENCE. These features indeed impact compiler speed, as was explained in the Release Notes.

- In V2.1 we have provided some improvement in compiler speed on RSTS/E systems. This has included taking advantage of the new 32Kw job limit now available in RSTS/E V8.0.

- As we told users at DECUS, work is now underway to continue improving compile time performance. We have begun planning some V2.n releases whose main purpose is performance

improvement (and, of course, to fix known bugs). At this point we hope to have the next point release, V2.2, available sometime during Q2FY84 and hope to have a follow-on release providing even further performance improvements available during Q4FY84.

- In doing any performance improvements, benchmark programs are critical. We again ask for your assistance in obtaining them. If you do have any programs that you feel may be appropriate, representative of user code, or beneficial for measurement, please forward those programs to the BASIC development group. If any of your users have such programs, we urge you to have them submit them to the BASIC SIG, C/O DECUS office in Marlboro. These will then be forwarded to us.

BASIC-PLUS-2 is a key layered product in the PDP-11 space and we are committed to ensuring a high level of support and active response to field needs. Please let us know of your suggestions, comments, concerns, and benchmarks.

BASIC Version 2

Edward Vogel
Digital Equipment Corporation
Nashua, NH

Version 2 of BASIC has been submitted to the SDC and will be available in the first quarter of calendar year 1983. The new features in BASIC V2 have already received much attention, so I would like to take this opportunity to write about the new documentation set, and what you can do to help us improve it in the future.

In the summer of 1980, the BASIC documentation group circulated a questionnaire to all BASIC (BASIC-PLUS,BP2 and VAX-11 BASIC) licensees, and to LISTNH. We received more than 130 replies, and used the results in the design of the Version 2 documentation set.

The transformation of questionnaire results to doc set design was not easy. Complaints about the documentation were rather evenly divided between "Too technical -- not for beginners" and "Not technical enough -- no help for advanced programmers." Of course, the replies were liberally sprinkled with pleas for better indexes and "real-life" programming examples.

To make a long story short, we came up with the following documentation set:

Beginner's Books:
- Intro to BASIC (rewritten for Version 2)
- BASIC for Beginners
- More BASIC for Beginners

Core Documentation:
- BASIC Reference Manual
- BASIC User's Guide
- BASIC Pocket Reference Guide

System-specific Manuals:
- BASIC on VAX/VMS Systems
- BASIC on RSX-11M/M-PLUS Systems
- BASIC on RSTS/E Systems
- Installation Guide and Release Notes (One each for RSTS/E,RSX-11M/M-PLUS and VAX/VMS)

Each manual contains a section labeled "To The Reader" that explains the
structure of the documentation set, conventions used, and so on.

YOU SHOULD READ THIS SECTION BEFORE USING THE DOCUMENTATION.

However, here's a sneak preview of the documentation structure:

The "core" documentation applies to Version 2 of both PDP-11 BASIC-PLUS-2

and VAX-11 BASIC. The BASIC Reference Manual is the place to find anything you ever wanted to know about BASIC, but it does NOT make for light reading. It starts with a consise overview of the BASIC language, followed by sections on compiler commands, compiler directives, statements, functions and BP2 debugger commands. Each of these sections contain keywords in alphabetical order. This format is familiar to users of VAX-11 BASIC V1, but may come as a surprise to BP2 users. The BASIC User's Guide contains new and improved information, in the "normal" Digital User's Guide format. The Pocket Reference Guide contains the same syntax boxes as the reference manual, plus a short description of each language element.

This "core" documentation should make it much easier to write transportable code. The Reference Manual clearly labels system specific language elements, and also calls out any syntax differences if the language element can be used on more than one system.

The system-specific manuals perform two functions. First, each of the BASIC on RSTS, RSX and VMS manuals tells a beginner how to log onto the system and use compiler commands in the BASIC environment. Second, these manuals show how to use features that are available on only one system For example, BASIC on VAX/VMS Systems explains the VAX-11 BASIC RECORD statement and how to extract record definitions from the VAX-11 Common Data Dictionary. BASIC on RSTS/E Systems and BASIC on RSX-11M/M-PLUS Systems explain how to use the BASIC-PLUS-2 Debugger and how to make the most efficient use of resident libraries.

Finally, the three Installation Guide and Release Notes manuals tell you how to install the product; they explain the tradeoffs involved in choosing installation options. These manuals also contain both software and documentation release notes.

Because one section of the release notes has a list of technical changes in BASIC Version 2, these notes contain information that the average programmer might find very useful.

In the past two years we've worked very hard to improve BASIC documentation. We will continue to make it better only if you, our audience, take the time to tell us where improvement is needed. To do this, you have only to fill out the postage-paid Reader's Comment form and send it in to us.

Please note that you are not limited to reporting complaints and documentation 'bugs.' We would be delighted to receive suggestions for new or improved figures and tables and for any subject area that you feel requires better treatment.

We can make no promises for other Digital documentation sets,but we guarantee an acknowledgement of any BASIC Reader's Comment Form you send in.

BASIC SIG Wishlist

Joe Mulvey
BASIC Development Group
Digital Equipment Corporation
110 Spit Brook Road
Nashua, New Hampshire 03062

Enclosed for publication is a printout of our Wishlist of data from previous DECUS Symposia. The information in the attachment reflects the state of the database prior to the Spring 1983 DECUS meeting. Included are all Digital's responses to the Wishlist items. After publication, we will purge out those items that no longer apply, e.g., "Feature is in V2", add in the items collected in St. Louis with Digital's response, and then submit it to you for publication. At that time, the information published in the newsletter will accurately reflect the status of our database and will be on-line at the start of the Fall DECUS 1983 DECUS Symposium in Las Vegas.

BASIC SIG WISHLIST - PRINTED ON 31-May-83 AT 02:58 PM

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00001	04/23/80	04/24/80	Y

WISH:
INTERPRETED CODE MIXED WITH COMPILED CODE IN A CHECKOUT SYSTEM. 3 LEVELS OF BASIC: INTERPRETED, FAST COMPILE, AND OPTIMIZED COMPILE ALL WORKING TOGETHER.

DEC RESPONSE:
AGREE WITH THE GOAL OF EASY CHECKOUT PLUS GOOD PERFORMANCE, THIS IS A GOAL OF DEC BASIC. (LEAVE COMPILE/INTERPRET ISSUE TO DEC).

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00002	04/23/80	04/24/80	Y

WISH:
INLINE CODE FOR BP2 FOR INTEGER OPERATIONS AT LEAST.

DEC RESPONSE:
POSSIBLE. THINKING ABOUT IT.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00003	04/23/80	04/24/80	Y

WISH:
IMPLEMENT ARCCOS/ARCSIN RADIAN FUNCTIONS.

DEC RESPONSE:
WILL CONSIDER IF ENOUGH REQUESTS ARE MADE FOR IT
(FUNCTION AVAILABLE VIA *CALL* ON VAX BASIC)

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00004	04/23/80	04/24/80	Y

WISH:
PRINT USING FORMAT THAT ALLOWS ZERO SUPPRESSION FROM THE
RIGHT. I.E. 1.3600 = TO 1.36 BUT 1.3645 = 1.3645

DEC RESPONSE:
WILL CONSIDER
SET TO LOW PRIORITY

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00005	04/23/80	04/24/80	Y

WISH:
MORE DATE CONVERSION ROUTINES, I.E. SYSTEMDATE AS
MMM. DD, 19YY

DEC RESPONSE:
EASY TO DO WITH USER_DEFINED FUNCTIONS, DON'T SEE NEED IN PRESENT
BASIC. NEED ARTICLE IN NEWSLETTER CONTAINING FUNCTIONS TO DO
THIS

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00006	04/23/80	04/24/80	Y

WISH:
ALLOW MULTIBLE SUBPROGRAMS TO BE COMPILED INTO A SINGLE
OBJECT FILE FROM A SINGLE SOURCE FILE

DEC RESPONSE:
AVAILABLE IN VAX-11 BASIC. WILL CONSIDER FOR PDP-11's (LOW PRIORITY)

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00007	04/23/80	04/24/80	Y

WISH:
NEW BIT FOR CVT$$ FUNCTION 512 - CONVERTS "(" AND ")"
TO "[" AND "]". THE 64 BIT WOULD TAKE PRECEDENCE OF THE
512 BIT SO THAT CVT$$(A$,-1%) WILL WORK AS IT DID BEFORE

DEC RESPONSE:
REQUIRES MORE THOUGHT ON OUR PART TO MAKE SURE NO PROBLEMS FOLLOW.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00008	04/23/80	04/24/80	Y

WISH:
B2XREF TO ACCESS THE MCR/CCL ENTRY LINE SO
>B2X OUT/SOU/KEY/FUN
WOULD WORK

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00009	04/23/80	04/15/83	Y

WISH:
BRANCH ON NON-EXISTENT FILE (IN OPEN) AND OTHER
ERRORS

DEC RESPONSE:
Will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00010	04/23/80	04/24/80	Y

WISH:
BETTER DOCUMENTATION ON THE ACTION USED BY THE COMPILER
IN HANDLING STEP SIZES THAT ARE NOT EXACT BINARY FRACTIONS

DEC RESPONSE:
WILL CONSIDER A NEWSLETTER ARTICLE FOR THIS.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00011	04/23/80	10/13/82	Y

WISH:
RE-ENTRANT CODE GENERATED BY BASIC-PLUS-2 COMPILER.

DEC RESPONSE:
BP2 CODE ($CODE, BP20TS) SECTIONS FOR V2 WILL BE RE-ENTRANT IN THAT THEY ARE PURE CODE AND CAN BE INCLUDED IN A RESIDENT LIBRARY.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00012	04/23/80	04/15/83	Y

WISH:
MID ON LEFT SIDE OF THE = EG. MID(A$,5,3)="CAT"

DEC RESPONSE:
AGREE WITH INTENT. MAY CONSIDER.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00013	04/23/80	10/13/82	Y

WISH:
USER FUZZ FACTOR FOR APPROX = COMPARE

DEC RESPONSE:
WILL CONSIDER. VIEWED AS LOW PRIORITY.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00014	04/23/80	04/24/80	Y

WISH:
A CLOSE ALL STATEMENT TO CLOSE ALL CHANNELS

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00015	04/23/80	04/24/80	Y

WISH:
AN END OF FILE DETECTION FUNCTION E.G. I%=EOF(CHANNEL%)

DEC RESPONSE:
WILL CONSIDER.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00016	04/23/80	04/24/80	Y

WISH:
AN END OF DATA FUNCTION EG. I%=EOD

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00017	04/23/80	04/24/80	Y

WISH:
OPTIONAL ARGUMENT FOR THE RND FUNCTION

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00018	04/23/80	04/24/80	Y

WISH:
STATIC PRESET (DATA INITIALIZATION) AT COMPILE TIME

DEC RESPONSE:
WILL CONSIDER. USE MACRO ROUTINES THAT DECLARE AND DEFINE
COMMON/MAP PSECT CONTENTS, THEN LINK INTO TASK.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00019	04/23/80	04/15/83	Y

WISH:
A FUNCTION TO RAISE THE ERROR FLAG

DEC RESPONSE:
CALL BAS$ERROR ON VAX FOR NOW, MAY CONSIDER THIS ENHANCEMENT
FOR THE FUTURE.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00020	04/23/80	04/15/83	Y

WISH:
RESTRICT CHANGE TO A ONE DIMENSIONAL ARRAY

DEC RESPONSE:
THIS MAY BREAK PROGRAMS.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00021	04/23/80	04/24/80	Y

WISH:
MAT OPERATIONS IN PLACE

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00022	04/23/80	04/24/80	Y

WISH:
FUNCTION FOR DETERMINANT

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00023	12/09/81	10/13/82	Y

WISH:
THE BASIC PLUS 2 EDITOR USES VALUABLE LINES ON THE CRT SCREEN WITH BLANK LINES AND OTHER MESSAGES WHEN TRYING TO LIST A RANGE OF PROGRAM LINES. LISTNH HELPS SOME, BUT JUST LISTING THE LINES WITH NO EXTRA BLANK LINES ADDED AT THE END WOULD BE A BIG HELP. IF I WANT TO KNOW WHAT PROGRAM I AM LISTING I COULD ALWAYS DO SOMETHING LIKE LISTH.

DEC RESPONSE:
THIS IS A FEATURE.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00024	12/09/81	09/27/82	Y

WISH:

Someone once said "MORE IS NOT ALWAYS BETTER". Basis Plus 2 is a good product as it is. This is not to say improvements would not be nice, but a language can become too rich. Please consider new features carefully.

DEC RESPONSE:
We think V2 shows sensitivity to the most needed requirements of BASIC users, in both PDP-11 and VAX environments. We're trying to be responsive without overendowing the language. Let us know if you think V2 succeeds in this goal.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00025	12/09/81	10/13/82	Y

WISH:
Wish:

Add support in the RSTS/E Basic-Plus-2 product for some of the n
ewer
SYSCALL's in RSTS.
Particularly the Spawn SYSCALL and especially the spool SYSCALL.

Reason:

These are particularly useful SYSCALL's and support really shoul
d be
forth coming.

DEC RESPONSE:

We agree. Supported RSTS SYS() calls should be supported by BP2 also.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00026	12/10/81	10/13/82	Y

WISH:
Support the spool sys call in Basic-Plus 2 as well as in
Basic-Plus

DEC RESPONSE:
Should be supported. in V2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00027	12/10/81	10/13/82	Y

WISH:

HOW ABOUT "RESTORE <LINE>", READ <LINE>, OR RETURN <LINE> CAPABILITIES.
(WHERE <LINE> SPECIFIES A LINE NUMBER.)

DEC RESPONSE:
Not planned. Not consistent with the direction of "line-less"
programming.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00028	12/10/81	10/25/82	Y

WISH:
Matrix functions should work PROPERLY on arrays DIMensioned like DIM A$(0). "Properly" means as documented; i.e. MAT A=ZER for an array DIM A(0) should not affect A(0). Also, support re-dimensioning like DIM A(100) \ MAT A=ZER(10,10) ! Can be done now, but not supported

DEC RESPONSE:
WORKS OK FOR V2. HOWEVER CHANGING THE DIMENSIONALITY OF THE ARRAY WILL NEVER BE SUPPORTED.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00029	12/10/81	10/13/82	Y

WISH:
Matrix operations like A=A*A (where A is a matrix) should work the same as A=B*C where A,B, and C are all identical matices. (If this works okay for scalars, why not for mats?)

DEC RESPONSE:
BP2 implementation of some MAT functions does use the array for intermediate storage. We will look into modifying this situation in the future.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00030	12/10/81	04/15/83	Y

WISH:
PLEASE ALLOW THE SCALE COMMAND IN PROGRAM CODE (LIKE EXTEND).
REASON: SCALE AFFECTS CVTF$ DATA AND PRECISION AND ROUNDING.
SINCE THE PROGRAM WILL NOT DO THE SAME THING UNDER DIFFERENT SCALE FACTORS, IT SHOULD BE UNDER PROGRAMMER CONTROL TO OVERRIDE THE SYSTEM DEFAULT.

DEC RESPONSE:
WILL BE PERMITTED IN V2 USING "OPTION" STATEMENT:

```
100 OPTION SCALE = 2
```

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00031	12/10/81	10/25/82	Y

WISH:
Give VAX BASIC immediate mode full functionality. For example, allow the call of LIB$ routines that reside in STARLET.OLB only.

DEC RESPONSE:
VAX-11 BASIC cannot "LOAD" object modules other than BASIC object modules. This is fundamentally why most LIB$ routines are not available. We may consider removing this limitation in some future version.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00032	05/11/82	08/27/82	Y

WISH:

I would like to see quadword data types since this is the way in which VAX stores dates when using the RTL routines.

DEC RESPONSE:

Not planned specifically, but you can define a quadword data type using the RECORD facility:

```
RECORD QUADWORD
      WORD Q(3)
END RECORD
...
DECLARE QUADWORD A,F00(10)
```

and can similarly define quadword-type functions to manipulate the data in
a quadword data type instance appropriately.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00033	05/11/82	10/25/82	Y

WISH:
I would like to see a bit data type. This would avoid unecessary masking and would make the code much more readable.

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00034	05/11/82	09/02/82	Y

WISH:
Please add RESUME with a label.

Reason: Now that Basic has labels, line numbers will be used much less, so RESUMing to a line number will have little use.

DEC RESPONSE:
Will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00035	05/11/82	09/27/82	Y

WISH:
Please add the ONECHR function to VAX BASIC
this will make my current applications upward compatible.

DEC RESPONSE:
NOT PLANNED.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00036	05/11/82	09/27/82	Y

WISH:
Please add the functionality to be able to chain with
line to all versions of BASIC

DEC RESPONSE:
Will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00037	05/11/82	09/27/82	Y

WISH:
allow a user to invoke an error exception
or document how it can be done

DEC RESPONSE:
Will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00038	05/11/82	09/27/82	Y

WISH:
allow a the ability to add an error abort

DEC RESPONSE:
If you mean an error signaling capability, will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00039	05/11/82	10/13/82	Y

WISH:
enhance the debugger to include
a) showing a variable as it changes
b) break on conditionals (ie IF UNTIL WHILE ....)

DEC RESPONSE:
Not planned for V2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00040	05/11/82	10/13/82	Y

WISH:
enhance the debugger to place " around strings when printed

DEC RESPONSE:
Wouldn't this be a hindrance if you're PRINTing terminal escape sequences?

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00041	05/11/82	10/13/82	Y

WISH:
allow for a mat print in the debugger

DEC RESPONSE:
Not planned for V2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00042	05/12/82	09/27/82	Y

WISH:
Allow MACROs. This would really enhance the BASIC+2 product and would seem to fit nicely with conditional assemblies.

DEC RESPONSE:
Will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00043	05/13/82	08/27/82	Y

WISH:
Provide a cross compiler on VMS that will allow users to develop and task build code on the VAX for their RSX-11M system.

DEC RESPONSE:
Will consider. In VAX BASIC V2, there is a /FLAG:BP2 switch that will direct the compiler to recognize only the BP2 language subset. This will permit to some extent host system compilation. Generation of PDP-11 OBJ's from the VAX BASIC code generator does not exist (yet).

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00044	05/13/82	10/13/82	Y

WISH:
We need a reasonably fast inkey$ function or something like it for single character input without a line terminator.

DEC RESPONSE:
Will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00045	05/13/82	10/25/82	Y

WISH:
It would be helpful to have the ability to specify the output width of the cross reference listing.

Since our standard size paper is 80 column, it is a hassle to change paper.

Thanks so much.

We love DEC!!

DEC RESPONSE:
WILL CONSIDER

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00046	05/13/82	04/15/83	Y

WISH:
Wish that basic + II would implement an ERL$ variable to trap errors at a label. This goes along with the idea that one should be able to resume to a label. Let's assume that you say IF ERL$="GET.REC" and you do not allow IF ERL$=LABEL$ so that you can resolve referrences at compile time.

DEC RESPONSE:
NOT PLANNED.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00047	12/06/82	04/14/83	Y

WISH:
i wish that there was an edt interface to VAX-11 BASIC. This would allow users to access the language help facilities and have a decent editor.

DEC RESPONSE:
This is provided in V2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00048	12/07/82	04/15/83	Y

WISH:

Why not allow the lower bound of an array to be defined as in FORTRAN?

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00049	12/07/82	04/14/83	Y

WISH:

There should be a way from MACRO subroutines, to return an error to the calling program. We should be able to set the ERR variable, and then return to the ON ERROR GOTO section of the calling routine. The current technique is to use the first argument as an integer, which contains the error code.

However, sloppy programmers don't always (or even usually) check this variable.

DEC RESPONSE:
MACRO operations are not formally supported in BP2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00050	12/07/82	04/15/83	Y

WISH:
I would like to add considerable support to wishes 25.

DEC RESPONSE:
V2 does support newer SYS calls on RSTS/E.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00051	12/07/82	04/14/83	Y

WISH:

Renumber BASIC lines while in BASIC instead of using a system utility outside of BASIC.

DEC RESPONSE:
VAX-11 BASIC provides this capability. BP2 does not because of space restrictions.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00052	12/07/82	04/14/83	Y

WISH:
Provide a method to allow chaining in RSX-11M Basic without having to install.

DEC RESPONSE:
This is currently provided in BP2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00053	12/07/82	04/14/83	Y

WISH:

Provide capabilility to chain to line numbers in RSX-11M Basic Plus 2

DEC RESPONSE:
May consider

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00054	12/07/82	04/15/83	Y

WISH:
Need a single charater input without CR LF.

DEC RESPONSE:
Use ONECHR or QIO. May consider an INKEY$ type function in the future.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00055	12/07/82	04/14/83	Y

WISH:
Line labels should be allowed to begin at the left margin.

DEC RESPONSE:
We reported this as a "will consider" item. Upon further investigation, it has been determined that this is a "can't be done".

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00056	12/08/82	04/15/83	Y

WISH:
I like FIELDs better than MAPs!!!

DEC RESPONSE:
Glad you like FIELD; it's documented. Dynamic mapping in V2 now is much more powerful.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00057	12/08/82	04/14/83	Y

WISH:

Pointer (Pascal,PL/1 types) data types please? A compiler that does not support immediate mode but exists only for compiling without other enhancements?
User control of impure areas (to support pointer types)?
Generating in-line (no threads) for very fast subprograms?
thank you!

DEC RESPONSE:

It is unlikely that we will provide a pointer data type. We don't feel it's in the spirit of the language.

As for a compiler that doesn't support immediate mode, we may consider a subset compiler.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00058	12/08/82	04/14/83	Y

WISH:
COMPILED NATIVE MODE BASIC
INTRINSIC BIT MANIPULATION VIA BIT ARRAY DECLARATION

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00059	12/08/82	04/15/83	Y

WISH:
It would be nice if the VAX-11 BASIC compiler had the option of multiple pass optimizations like some other languages do. It would help the performance and the reputation of the product.

DEC RESPONSE:
We agree in principle; will consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00060	12/08/82	04/14/83	Y

WISH:
When is VAX basic V2 due to be shipped to customers?

DEC RESPONSE:
VAX-11 BASIC V2 is the current version of the product, and is now available.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00061	12/08/82	04/14/83	Y

WISH:
NEED QUADWORD DATATYPE.
REASON: PARTICULARLY FOR USE IN MANIPULATING VAX/VMS DATE TYPE AND ACCOUNTING INFORMATION. SINCE BASIC SUPPOSEDLY IS TO PROVIDE FULL RMS COMPATIBILITY, HAVING A QUADWORD DATATYPE IS ONLY LOGICAL. ALSO, IT WOULD PROVIDE GREATER COMPATIBILITY WITH DATATRIEVE AND CDD.

DEC RESPONSE:
This can be accomplished with RECORD's in VAX-11 BASIC V2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00062	12/08/82	04/15/83	Y

WISH:
Ability to use $ and % variable suffix in a declared variable
Example:

```
MAP (TEST) STRING TEST1$ = 3
         , TEST2$ = 4
         , LONG TEST3%
         , TEST4%
```

DEC RESPONSE:
We have had some feedback on this and may look into doing it. In the meantime, we recommend that you don't use these suffixes.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00063	12/08/82	04/15/83	Y

WISH:
Integer or floating point fields in a MAP should be aligned automatically by the compile. The programmer should not have to add in a FILL = 1 to align the field.

DEC RESPONSE:
BP2 does this for you. It is not needed in VAX-BASIC.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00064	12/08/82	04/14/83	Y

WISH:
VAX BASIC should support RMS records with variable length records with fixed length headers.

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00065	12/08/82	04/14/83	Y

WISH:
THE CURRENT METHOD OF USING MAPS BY REQUIRING THE NAMES TO BE HARD-CODED IS VERY INCONVENIENT. I WOULD LIKE THE CAPABILITY OF CHANGING THE DEFAULT MAP SPECIFIED BY THE OPEN STATEMENT AT THE TIME OF THE GET; POSSIBLY WITH THE FOLLOWING SYNTAX:

```
    GET #CHAN%, MAP MAP_NAME$
```

WHERE MAP_NAME$ CONTAINS THE MAP NAME TO CONTAIN THE DATA OBTAINED WITH THE GET

DEC RESPONSE:
Not planned.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00066	12/08/82	04/14/83	Y

WISH:
Supply a task-builder for programs too big for SLOTKB.

For instance: The BP2 compiler will soon be too big for SLOTKB. Supply a program which is even bigger (call it: VSLTKB or GNIGHT).

I also have programs (nearly) this big, and need a bigger task-builder.

DEC RESPONSE:
Thank you for your suggestion. This item should be brought to the attention of the RSX Development Group.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00067	12/09/82	04/14/83	Y

WISH:

Provide a non-privileged function in BP2 to return the date in internal format. We should not have to be privileged to avoid the string and code overhead of converting 31-Dec-82 to 12365

DEC RESPONSE:
This is a good candidate for a BASIC SIG Library Function.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00068	12/09/82	04/14/83	Y

WISH:

Put the BASIC-PLUS-2 compiler on the PC (the cross-development is ugly)

DEC RESPONSE:
We agree. We will work this issue with the PROFESSIONAL Group.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00069	12/09/82	04/14/83	Y

WISH:
Put that example converter program on the V2 kit. I like it, but its gonna be a pain to type it in

DEC RESPONSE:
Thank you for your input.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00070	12/09/82	04/14/83	Y

WISH:
Play around with this in BASIC-PLUS:

```
10 EXTEND \
   DIM SCREEN%(127,47) \ ON ERROR GOTO 32000 \ X=65. \ Y=-64. \
   U%=FNSET%(INT(X),INT(Y))
11000 DEF* FNSET%(X%,Y%) \ Screen%(X%,Y%)=1% \ FNEND
32000 If erl=11000% and ERR=55% then resume 11040 else print 'err'
32767 end\
```

DEC RESPONSE:
Don't understand this. If it's a problem, please submit an SPR.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00071	12/09/82	04/14/83	Y

WISH:

Supply BASIC-PLUS-2 V2.0 with a /VAX switch to catch those references

(we have found two so far) which work correctly on BP2 but not on VAX basic.

DEC RESPONSE:
We agree in principle and will probably do something in the longer term to do this.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00072	12/09/82	04/14/83	Y

WISH:
PLEASE -
Please check immediate RUN of an INPUT prog line in V2 -
i.e. 1 input integer_variable
RUN does NOT work !!!!

DEC RESPONSE:
Fixed in V2.0; sorry.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00073	12/09/82	04/14/83	Y

WISH:
Wish 220

Please, I know this is a repeat & it may not pertain to BASIC, but PLEASE make ERR & ERL examinable under DEBUG!!

Thank You

DEC RESPONSE:
For BP2, it's there. For VAX-BASIC, we'll discuss this with the DEBUG people.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00074	12/09/82	04/14/83	Y

WISH:
When opening a unit record device (ie TI: or LP:) if the allow clause is ALLOW NONE please attach to the device so that two basic programs do not over write each other.
example: Program A open LP: allow none and program B opens the LP: and both start printing you will get intermixed i/o.

DEC RESPONSE:

For TI:, we will consider. For LP:, the RMS group is currently looking into this issue.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00075	12/09/82	04/14/83	Y

WISH:
Allow the redirected terminal to be specified at task build time so that when running a program that has a sub program in debug mode go to another terminal.

DEC RESPONSE:
It is unlikely we'll do anything on this item.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00076	12/09/82	04/14/83	Y

WISH:
add the begin and end statement to basic that way I can run my old ALGOL programs will work

DEC RESPONSE:
Not planned.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00077	12/09/82	04/14/83	Y

WISH:
Allow COMPILE/LABEL to keep label names for an ERL$ variable. Actually, we can do this ourselves and test the label name in the error trap (IF ERR = 11% AND ERL$ = 'FOO' THEN ...) but it would be handy to have the system help.

DEC RESPONSE:
It is unlikely we will do this.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00078	12/09/82	04/14/83	Y

WISH:
Allow the use of longwords as array subscripts.
I need virtual arrays with dimensions larger than 32767.
...the multi-paging capabilities of virtual arrays is great!!

DEC RESPONSE:
This is provided in VAX-BASIC. In BP2, we may consider this item, but it may increase array header size.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00079	12/09/82	04/14/83	Y

WISH:
PLEASE GIVE THE FUNCTIONALITY OF
COBOL TO A PRINT-USING REPLACEMENT!

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00080	12/09/82	04/14/83	Y

WISH:
PLEASE GIVE MORE USER SESSIONS IN ST-LOUIS !!!

DEC RESPONSE:
We emphasize USER sessions. Bring this to the attention of the BASIC SIG.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00081	12/09/82	04/14/83	Y

WISH:
Please have the SIG help define USEFUL
benchmarks for BASIC. I'm really tired
of hearing about Whetstones

DEC RESPONSE:
We agree; we really need something like that. We believe it is underway with some volunteers at the SIG meeting. We need to have them - you are the ones to get them to us.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00082	12/09/82	04/14/83	Y

WISH:
ONECHR on VAX BASIC ... QIO is a HACK !!

DEC RESPONSE:
We agree - it should be there (As one of the developers responded: Don't be a baby, use QIO's).

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00083	12/06/82	04/14/83	Y

WISH:

When a syntax error is found, please print out the line and statements that precede the error.

DEC RESPONSE:
This is provided in V2.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00084	12/06/82	04/14/83	Y

WISH:
Please allow the presence of line numbers in INCLUDE or some other syntax thereof to allow include to work with routines which handle errors using RESUME. ALTERNATIVE: Allow the RESUME statement to use a statement label instead of require a line number.

DEC RESPONSE:
More likely to see RESUME to a label.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00085	12/06/82	04/14/83	Y

WISH:

Allow some subset (because of what you describe as a space problem) of the RECORD declaration on the PDP-11. Possibly to allow definition

of some simple datatypes (COMPLEX, non-repeating record format, etc.)

this would allow user expansions without requiring the contorted code now needed.

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00086	12/07/82	04/14/83	Y

WISH:
Support a sys call for BASIC-PLUS on RSTS/E to dump to a disk file.

DEC RESPONSE:
No plans.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00087	12/07/82	04/14/83	Y

WISH:
Allow substring assignment (MID on LHS).

DEC RESPONSE:
Good idea, may consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00088	12/08/82	04/14/83	Y

WISH:
Please consider making labels local to, for example, function definitions, and other appropriate locations. Perhaps a local/global designation for labels would be a solution.

Reason: How many variations of OOPS: can I be expected to come up with?

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00089	12/08/82	04/14/83	Y

WISH:
PROBLEM: IMPLEMENTATION OF MAILBOX COMMUNICATION
DEFINED: AT THE HIGHEST LEVELS OF BASIC CONSTRUCTS THERE EXIST FEW

SYNTAX TOOLS TO FULLY UTILIZE THE FULL SCOPE OF MBX COMMUNICATI
ON
WITHOUT RESORTING TO SYSTEM SERVICE CALLS (i.e. QIO...)
SOLVED: ADD SYNTAX CAPABILITES TO PROVIDE FOR A MORE HIGHER LEVEL
IMPLEMENTATION OF MBX COMMUNICATION FEATURES

H

DEC RESPONSE:
See next wish.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00090	12/08/82	04/14/83	Y

WISH:
PREVIOUS WISH CONTINUED.....

EXAMPLE: TO PROVIDE FOR SYNC/ASYNC CAPABILITY SOMETHING SUCH AS:

```
00010       PUT #MBX.CHANNEL%, ASYNC
```

THIS WOULD ALLOW A MBX MESSAGE TO BE SENT ASYNCHRONOUSLY AND NOT REQUIRE MESSAGE RECEIPT BEFORE PROGRAM EXECUTION RESUMED.....

DEC RESPONSE:
We would like to have the SIG look at this and make some recommendations.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00091	12/08/82	04/14/83	Y

WISH:

Allow RESTORE <channel#> on native-mode sequential (ASCII stream) files on PDP-11 to avoid overhead of closing and re-opening for each scan.

DEC RESPONSE:
This is provided in V1.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00092	12/09/82	04/14/83	Y

WISH:
PROBLEM: BASIC PERFORMANCE WITH DBMS CALLABLE INTERFACE

SOLVED: ADDITION OF IMBEDDED DML CAPABILITY IN THE VAX 11 BASIC COMPILE R

SO AS TO REAP THE 20%-30% PERFORMANCE IMPROVEMENT REALIZED BY THOSE LANGUAGES WHERE DML IS IMBEDDED.....

DEC RESPONSE:
This has been on the wishlist before. We are still looking into it.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00093	12/09/82	04/14/83	Y

WISH:
PLEASE PROVIDE SOME DOCUMENTATION TO ALLOW MACRO ROUTINES TO CREATE AND ALTER STRING LENGHTS. THE MACRO ROUTINE CAN VERIFY THE LENGHT OF A STING IN V1.6 ON THE 11'S. HOWEVER IF THE STRING IS TO SHORT, THE MACRO ROUTINE CAN ONLY RETURN AN ERROR. IF THE MACRO ROUTINE COULD MAKE A LIMITED CALL TO BASIC RTS TO PROVIDE A STRING OF PROPER LENGHT IT WOULD HELP US A LOT.

DEC RESPONSE:
Rumor has it that there will be s SIG newsletter article stating how to do it.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00094	12/09/82	04/14/83	Y

WISH:
I WISH YOU WOULD NOT (PERMANENTLY) NUMBER THE WISHES. I AM TIRED OF SUCH SETANCES AS:

> "I WISH TO ADD MY SUPPORT TO #17X. THE PROBLEM WITH #34 IS THAT IT WON'T WORK IF YOU INCLUDE #99, SO #17 SHOULD BE INCLUDED WITH THE #66JL IDEA. AND THANK YOU FOR YOUR ANSWER IN #27, #33, AND #88."

SEE WHAT I MEAN?

DEC RESPONSE:
We agree.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00095	12/09/82	04/14/83	Y

WISH:

Support of MACRO from BP2. Several times I have heard people complain about problems from MACRO, and DEC's response has been "MACRO IS NOT SUPPORTED." This is news to me, and seems strange because:

- o MACRO-11 is only language universal across all PDP-11 Op. Systems.
- o Calling MACRO-11 is extensively documented
- o DEC sometimes recommends using MACRO subroutines

...continued

DEC RESPONSE:
See next wish.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00096	12/09/82	04/14/83	Y

WISH:
...continued
- o Can offer both size and speed improvements
- o CALL BY REF only works with MACRO subprograms

Solution is simple: SUPPORT MACRO!

P.S. Thanks for version 2 -- it looks fantastic!

DEC RESPONSE:
Thank you for your thoughts. It's unlikely we'll change our position.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00097	12/07/82	04/14/83	Y

WISH:

Would like end_of_file condition handled by INPUT/GET stmts rather than be

treated as an error condition. For example: GET/END_OF_FILE=label ... s o
that control would be transfered to that label on EOF.
Reason: Cleaner code.

DEC RESPONSE:
May consider.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00098	12/07/82	04/15/83	Y

WISH:
Please stop flagging FIELD as a declining feature. We like the diagnostics when compiling, but must use FIELDing for generalized I/O routines when we do not know at compile time which buffer we will be using. Dynamic mapping does not work for this!!!--it's too limiting: each time we bring a block in, we'd have to move the buffer to the map, and we may need 6 or 7 different ones.

DEC RESPONSE:

The declining feature flagger can be disabled with the /FLAG qualifier.

```
WISH NUMBER   DATE OF WISH     DATE OF RESPONSE    RESOLVED
00099         12/07/82         04/14/83            Y

    WISH:
    Instead of making FIELD a declining feature, enhance it by
    adding other data types to it, like:
      FIELD #DATA.CH%, 2% AS Z%(1%), 8% AS Z(A% + B%)
    or even:
      FIELD #DATA.CH%, ARRAY Z%(32%)

    DEC RESPONSE:
    This is inconsistent with the direction BASIC is going.
-------------------------------------------------------


WISH NUMBER   DATE OF WISH     DATE OF RESPONSE    RESOLVED
00100         12/07/82         04/14/83            Y

    WISH:
    Provide a way at run-time to change the channel of a
    virtual array.

    DEC RESPONSE:
    May consider.
-------------------------------------------------------


WISH NUMBER   DATE OF WISH     DATE OF RESPONSE    RESOLVED
00101         12/07/82         04/14/83            Y

    WISH:
    Provide a way for variable modes of file opens in Vax Basic,
    similar to the RSTS MODE clause.  That is:

   OPEN DATA.FILE$ FOR INPUT AS FILE DATA.CH%, ORGANIZATION DATA.ORG$
        ALLOW DATA.ALLOW$
    like the RSTS
       OPEN DATA.FILE$ FOR INPUT AS FILE DATA.CH%, MODE DATA.MODE%
    Reason:  Generalized file open routines and modules.

    DEC RESPONSE:

Unlikely; would drastically slow down OPEN, and cause a size problem.
-------------------------------------------------------


WISH NUMBER   DATE OF WISH     DATE OF RESPONSE    RESOLVED
00102         12/07/82         04/14/83            Y

    WISH:
    LONG integer type for array subscripts.
    Reason: Larger virtual arrays, e.g. DIM  ARDATA%(80000)

    DEC RESPONSE:
    May consider.
-------------------------------------------------------
```

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00103	12/08/82	04/14/83	Y

WISH:
Inter-node file access from my VAX to my 70 and VAX-VAX, 70-70, etc., should be as transparent as
OPEN "NODE1::DEV1:[ACCT]FILNAM.EXT" FOR INPUT. It could also be as nice for starting processes accross nodes.

DEC RESPONSE:
Planned for RMS V2.0 when released.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00104	12/08/82	04/14/83	Y

WISH:
I realize that BASIC-PLUS is a stable product, but please don't leave all of us that use it out in the cold!!!! BP2 is too slow for 100 beginning programming students!!!!!

DEC RESPONSE:
We really do agree with you. We do NOT intend to decommit support of BASIC-PLUS at this time.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00105	12/08/82	04/14/83	Y

WISH:

There is a problem in BASIC+ that is almost more of a bug than a new feature! LEFT, RIGHT, and MID should work as LEFT$, MID$, RIGHT$. I realize you can't make LEFT, RIGHT and MID illegal -- but please make the other version legal. It is hard to show students what a string function is and then have to explain LEFT, MID and RIGHT.
INSTR%, etc.

DEC RESPONSE:
You're right. We're not sure we can fit in it, but we will definitely look at it.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00106	12/08/82	04/14/83	Y

WISH:
Let me code MAP (MYMAP) A$ = 1, B$ = 2, B1$ = 5 REDEFINES B$, C$ = 3

so I don't have to use 2 maps to specify that B1 starts where B does.
This would simplify maintenance as well as program readability.
Thanks!

DEC RESPONSE:
Unlikely. RECORD's in VAX-BASIC provide this via VARIANT's.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00107	12/08/82	04/14/83	Y

WISH:
MAKE CALLS TO CDD AT RUN TIME
TO MAKE UNIVERSAL EDITORS ... POSSABLE

DEC RESPONSE:
This is inconsistent with what CDD provides. CDD creates variable names at COMPILE time. To do an INCLUDE of CDD at runtime is too late - your code has referenced all the variables it ever will. DBMS provide Data Base Mgmt. at runtime.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00108	12/08/82	04/14/83	Y

WISH:

I Would like to add my plea to make the Basic-plus 2 compiler generate as much in-line code as possible. V2 has done excellent things with the Basic language. Now lets do something about its speed.
Thank you

DEC RESPONSE:

We agree in principle, but we do not want to blow away any application programs. There may be some inline code we can do. There are of course definite benefits to threaded code.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00109	12/09/82	04/14/83	Y

WISH:
When opening a Virtual type file with a large Recordsize (i.e. 4096)

separate disk accesses are made every time a different block is referenc
ed,
even though the referenced block is already in the buffer previously called in. In effect this means no speed gain from accessing Virtual files with large buffers, even though buffer space is allocated.

This problem has been mentioned to the V2

DEC RESPONSE:
This is a known problem that we are looking into fixing. We suggest you submit an SPR.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00110	12/09/82	04/14/83	Y

WISH:
De-stablize Basic-Plus!! Unless you're going to makre
BP2 have a REAL immediate mode someday.

DEC RESPONSE:
Unlikely.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00111	12/09/82	04/15/83	Y

WISH:
Please document call to BP2 to get space (GSA) used by RMS.
I would like to be able to use the call myself.

DEC RESPONSE:
No possibility. You can use Dynamic MAP - it will do
AMAZING things for you.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00112	12/09/82	04/15/83	Y

WISH:

MAKE PDP 11/CPU=(ALL,INCLUDING=PC3**) BASIC COMPILER GENERATE PIC, RE-ENTERANT, AND SHAREABLE CODE
INCLUDE A %MACRO MACRO_NAM, [ARG1]... DIRECTIVE
HAVE %INCLUDE AND %MACRO TALK TO CMS

HAVE ODL BUILDER FOR TOOL KIT BUILD MEMORY RESIDENT OVERLAYS BASIC/MEMORY_USE=NOTSTUPID

DEC RESPONSE:
The first suggestion is unlikely.
%INCLUDE A %MACRO is unlikely, it just won't fit.
CMS is language independent.

For the last suggestion, we suggest you talk to the PROFESSIONAL folks.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00113	12/06/82	04/14/83	Y

WISH:
WISH THAT $BPCREF WOULD CORRECTLY PRINT INFO ABOUT REAL VARS I, L, M, R, S, AND T. (IT DON'T IN VERSION 7.0). PLEASE REFER TO CORRECT PEOPLE IF NECESSARY.

DEC RESPONSE:
Yes, we will refer this to the correct people.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00114	12/06/82	04/14/83	Y

WISH:
RMS: It should have a 'Fast-forward/reverse' FIND capability. Currently the FIND starts looking from the beginning for the key... allow for searching forwards/backwards from current position in the files. This is mainly for people who use duplicate keys. FIND should also work for alternate/secondary keys.

DEC RESPONSE:
We will forward to RMS.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00115	12/06/82	04/14/83	Y

WISH:
#27 could be more meaningful with the word label substituted for line. (How's that for line-less preogramming.) The point is the functionality, not the line numbers.

DEC RESPONSE:
Doubtful.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00116	12/09/82	04/14/83	Y

WISH:
PLEASE MAKE BP/2 PROPERLY UPDATE THE EOF POINTER IN A BLOCK I/O FILE. ie OPEN "X.X" FOR INPUT AS FILE 1%, VIRTUAL
PUT #1, RECORD 9
GET #1, RECORD 6
CLOSE #1
DO A DIRECTORY X.X IS FLAG AS 6./9.

DEC RESPONSE:
An SPR has been submitted; we'll fix it.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00117	12/09/82	04/14/83	Y

WISH:
PLEASE DO NOT DUPLICATE CODE IN YOUR OBJECT LIBRARY.
THERE ARE MANY CASES IN WHICH BP/2 AND RMS HAVE THE SAVE ROUTINES BUT DIFERENT ENTRY POINT NAMES
IF BP/2 COULD USE THE SAME ONES AS RMS THIS WOULD REDUCE THE SIZE OF MANY TASKS.

DEC RESPONSE:
Point well taken; we certainly will look into this and take advantage of any savings that can be done.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00118	12/09/82	04/14/83	Y

WISH:
PLEASE DOCUMENT OR SEND TO THE SIG NEWSLETTER THE FORMAT OF YOUR .OBJ FILES SO THAT USERS CAN WRITE ROUTINES THAT IN MACRO THAT CAN BE LOADED.

DEC RESPONSE:
Unlikely we'll do this; may imply defacto MACRO support which we do not intend to do.

WISH NUMBER	DATE OF WISH	DATE OF RESPONSE	RESOLVED
00119	04/14/83	04/14/83	Y

WISH:
Modify the wishlist program to:
    Allow longer wishes
    Use Decnet device name in file open
    Make wishlist program multi-user

DEC RESPONSE:
Fixed at next DECUS.

Introduction to BASIC

Joe Mulvey
Digital Equipment Corporation
Nashua, NH

Charles Mustain, Session Chairperson
Stark County Local School System
Louisville, OH

Reported by Marty Olevitch, DECUS Scribe Service

Joe Mulvey, BASIC Language Development Planner for Digital, gave an overview of the two versions of DEC BASIC used for commercial purposes (VAX BASIC and BASIC-PLUS-2). In general, DEC BASIC is an extended BASIC, derived from BASIC-PLUS-2 (which is, in turn, derived from Dartmouth BASIC). It has advanced I/O and string-handling capabilities; floating point, integer, and character variables. It is available under RSTS/E, RSX, and VMS, with a high degree of source compatibility. The BASIC Development Group believes that it is very user-oriented.

In the latest version of DEC BASIC, several changes have been made. Line numbers are now required only on the very first statement. The '&' is used only for a statement continuation from one line to another. Labels may be used to precede directives or statements and be used as the target of a GOTO statement. Finally, the symbols '%' and '$' are required for undeclared variables only.

A special feature of the new versions is its improved error-handling capabilities. Compile-time errors are displayed in terms of the error's severity (I - informational, W - warning, E -trappable, and F - fatal). The error is identified by number, location, and a mnemonic. The text of the error is given. During run-time, BASIC will diagnose and display to allow the user facility to test the error condition and take action. With an informational/warning error execution will continue. A trappable error will, at the user's discretion, terminate BASIC. A fatal error causes execution to terminate.

Another special feature is COMMON DATA feature. It allows independent programs to access a named shared area. The area must have the same common name. The program must know the structure of the common module, for its structure is arbitrary. The COMMON option is related to the MAP statement.

The phases of the system are as follows: the EDITOR (containing the BASIC statements and commands), the COMPILER (which process the statements), a RUN phase (for immediately executable code), an .OBJ phase (for linkable output from the compiler), a LINKER (which produces an executable image), and the RUN-TIME system to provide support for execution.

In the edit phase of operation, the commands are: NEW (to establish the name of a new program), OLD (to bring an existing file from disk to one's area of memory), APPEND (to merge an existing file with the current file), RENAME (to change the name of a file), UNSAVE (to delete a file), SAVE (to store a file on disk for the first time), REPLACE (to store subsequently), SEQUENCE and RESEQUENCE (to assign line numbers and references), DELETE (to eliminate lines), LIST (to print the lines), EDIT (to edit lines either by 1) invoking the editor or 2) entering the edit mode).

In the compiling phase, there are the following commands: SET (which allows us to select among the default options), SHOW (which will display the defaults), COMPILE (to create the object module). Some of the qualifiers are /CROSSREFERENCE, /DEBUG, /LIST, /PAGESIZE, and /TYPEDEFAULT.

In preparation for execution (including the obj/link/run phases), the commands are BUILD (to create command and overlay controls), LOAD (to bring in the object module), and RUN (to execute the program). The qualifiers are /BYTE, /DEBUG, /LINE, /TYPEDEFAULT, and /SINGLE (or /DOUBLE).

Commands available during run time are CONTINUE ,which enables one to resume execution after an interruption, and the debugging commands BREAK, ERL, ERR, LET, PRINT, STEP, and TRACE.

The procedure for invoking BASIC is, of course, dependent on the operating system. The other commands on the environment level are HELP/INQUIRE ,which allows the user to request assistance, IDENTIFY, to display a header, SCALE, to control round-off, ASSIGN, which will equate logical names, and EXIT, to leave the BASIC environment. The prefix '$' can be used to execute an operating system command.

In summary, the BASIC Development Group believes that this version of BASIC, with its computing and I/O power, is the state of the art. It has integer and decimal math, single and double precision real arithmetic, GFLOAT, and HFLOAT, and both string and matrix arithmetic. There are virtual arrays, block I/O, and terminal format, sequential, relative, and indexed files. It improves programmer productivity because it is easy to learn, easy to read and maintain, and is a structured language.

Introduction to UNIX

Mark Bartelt
HSC Research Development Corporation
Toronto, Ontario

Dorothy Geiger, Session Chairperson
Cal Poly
San Luis Obispo, CA

Reported by Kevin Carlin, DECUS Scribe Service

UNIX (a trademark of Bell Systems) is, in the DEC environment, a timesharing system for the PDP-11 and VAX processors. Some of the applications to which UNIX is suited include text processing, phototypesetting, communications, networking, and language development. The UNIX environment is designed to be friendly to the programmer, especially, but not exclusively, the systems programmer.

The UNIX Command Line Interpreter (CLI), or "shell," is not a privileged program, any programmer can define or redefine their shell command set at the interpreter level as well as the standard command file levels.

A powerful symbolic debugging tool is available.

The following languages are standard with the UNIX distributed by Digital: C, FORTRAN 77, RATFOR, STRUCT (a FORTRAN 77 to RATFOR utility), BASIC, DC, and BC.

Some optional languages available for the UNIX system are: ALGOL 68, PASCAL, SNOBOL, and APL.

There are certain features UNIX is definitely not designed to support, and these include real time applications, asynchronous I/O, and generalized interprocess communications.

Some of the more interesting tools available from DEC UNIX are:

MAKE — reviews symbolic dependencies between between multiple source files

LINT — program to do all the type checking and sloppy code flagging the C compiler doesn't

YACC/LEX language development tools for syntax definition and analysis

ROFF a RUNOFF ancestor now somewhat obsolete

NROFF a ROFF ancestor which permits macro definition

TROFF a typesetting version of NROFF .

Most of the operating system is in C to promote ease of transportability and modification. Approximately 7000-10000 lines of C code, as opposed to 1000-2000 lines of assembly code, appear in the kernel. The system utilities are virtually 100% C.

The file system is expressed in the following diagram: <<<Insert diagram 1 from my notes -- KEC>>>

A special convention for designating the user's current position in the tree is ".", while ".." is the user's root directory.

Cd is a system command to change the user's current directory. The command

```
cd ..
```

relocates the user's directory to the root of his current directory, popping him up a level.

```
cd ../..
```

relocates the user to the root directory of his current root directory, popping him up two levels.

No internal file structure is defined for UNIX. All file structures are user defined. There is a buffer cache scheme that supports the functions delayed write and advanced read, which permits better efficiency in system I/O. I/O protocols are always the same on UNIX, regardless of whether the interaction is with files, devices, or pipes.

The three standard predefined files employed by UNIX are 0, standard input, 1, standard output, and 2, a standard error file.

The I/O primitives used by C are:

```
fd = open(file,mode)
fd = create(file,prot)
nb = read(fd,buf,nbytes)
nb = write(fd,buf,nbytes)
close(fd)
pipe(fd)
```

```
lseek(fd,position)  (a random access primitive)
fd2 = dup(fd1)
dup2(fd1,fd2)
```

There is a special structure for beginning (or cloning) a sub-process from whatever process is in progress. The function:

```
status = fork();
```

initiates the new process, which will always be identical to the current process. The parent (or original process) and the child (or newly generated process) know who they are in relation to the other by the value returned by fork (yes, execution of the new process begins at the fork function, not at the initial entry into the process). If status equals -1, then the attempt to fork failed. If status equals 0, then the process knows it is the child. If status is greater than 1 then the process is the parent and the value in status can be used to identify the child.

A procedure which renders fork more useful is:

```
exec(file,args);
```

which replaces the current process with a new one (which is entered at its beginning). These are very common among children.

```
exit(status);
```

is a procedure which terminates the current process and returns the user defined value status to the parent process.

Special error routines may be placed below either of these two procedures since, if the process behaved as it should, it would never get there.

There are several commands to redirect standard I/O to user defined files, structures, or pipes.

There is also a facility for having a shell command be dealt with as a background process in the command line which looks as follows:

```
shellcommand &
```

The syntax also allows for the standard output of a process to be included as part of a shell command line:

```
...`shellcommand`...
```

where the dots represent an enveloping command line. Multiple commands may be combined on a command line as follows:

```
shellcommand1; shellcommand2; shellcommand 3;
```

But the command line:

```
shellcommand1; shellcommand2; shellcommand3 &
```

will probably not have the result desired. If the intention was to set up all three processes as a background process, then the syntax should be:

```
(shellcommand1; shellcommand2; shellcommand3;) &
```

to force more than just the last process into the background.

Pipelines are used to make the output of one command the input for the next. The syntax for this is:

```
shellcommand1 | shellcommand2 | shellcommand3
```

Three special shell command operators can be used to redirect the standard I/O devices. These operators are "<", ">" and ">>". The command construction:

```
shellcommand > file
```

redirects the standard output for that shell command to file. The construction:

```
shellcommand >> file
```

redirects the output to that file, appending if the file is already present. And so the redirection of input looks like this:

```
shellcommand < file
```

These redirections can be combined in a single command, a simple example being:

```
shellcommand < ifile > ofile
```

The utilities offered with the UNIX system are generally small and stupid. The small is better philosophy, with a few notable and necessary exceptions, is the universal philosophy of UNIX utilities. Some of these utilities are:

who -- which lists the login name and terminal number of all logged in users.

ls -- which lists the contents of the user's current directory.

wc -- which returns the number of words, lines, and characters in the input file.

sort file -- which sorts the file "file" into sorted text.

grep string file -- searches for occurences of "string" in "file."

hm file -- lists the symbol table elements in an object code file.

ar t lib -- lists the contents of an archive file.

tee file -- copies standard input to standard output.

cat file1 file2 -- copies files to standard output.

tr [options] string1 string2 -- replaces string1 with string2 while copying standard in to standard out.

uniq [options] -- Copies input to output, deleting repeated occurences of lines.

Shell scripts are UNIXs version of indirect command files. If a file is specified for a process which contains a shell command and seems to be a shell script, the shell spawns itself (just as fork permitted a process to spawn a child above) and uses the script as its standard input. Shell scripts may be constructed and used as command language which includes multiple line for, while, and until constructions.

Bibliography:

Bell System Technical Journal
Volume 57, number 6, part 2, July - August 1978

Software Tools
B. Kernighan and P. J. Plauger

The C Programming Language
B. Kernighan and Ritchie

How to Make BASIC Run Faster

Richard Baldwin
North County Computer Services, Inc.
Escondido, CA

Bill Tabor, Session Chairperson
Racal-Milgo
Miami, FL

Reported by Todd Spangler, DECUS Scribe Service

One of the main objects in the computer world is to minimize the amount of time required to perform a job. The problem has almost always rested in the hands of the programmer. It is the job of the programmer to make the code as efficient as possible, both in memory conservation and CPU time. In the presentation by Richard Baldwin of North County Computer Services, Inc., the problem is taken from the hands of the programmer and placed in the hands of the hardware people.

In the BASIC language there is one option to be considered. By placing a floating point processor or floating point accelerator on the CPU, the speed of many commands can be quickened. Using the processor, the math operations and related internal functions such as sin, cos, tan, etc., will be handled much faster. To demonstrate this increase, an experiment using a simple for-next loop was tried.

In this experiment, the for-next loop was run without any additional instructions in the loop and timed by itself. From this, the CPU time calculated with statements in the loop minus the for-next time will be the time used of the instruction. By iterating the for-next loop twenty to thirty thousand times, accurate time calculations can be made and compared with the run times of the same set of instructions using a floating point accelerator. When the figures are compared, one can see that there is almost always a significant difference when the accelerator is used.

There are four variables to be considered which influence CPU time. The type of machine and the type of compiler play an important role. Also, the hardware options and the capabilities of the programmer will be crucial to the operations of the machine. The experiment shows that virtually no matter what machine or type of Basic is used, the program floating point operations will always run faster when using the floating point accelerator. Otherwise, variances will be machine dependent and can be found using similar for-next loops. In general, the floating point accelerator will enhance the operations of the computer.

BASIC Magic (and other tall tales)

Daniel Esbensen
Touch Technologies
Escondido, CA

Bill Tabor
Computer Products
Ft. Lauderdale, FL

Reported by Laura Havlin, DECUS Scribe Service

The main topic of discussion was centered around the idea of creating a standard form of BASIC. The version that is now up for public review by the standards committee, as the group discussed it, seemed to be unacceptable. Although many people had heard about the specific version, very few people had seen a copy of it. Several people even said that they had sent checks with their requests for a copy of the standard version, had received a cancelled check several months ago, but still had not received a copy yet.

As Daniel Esbensen said that the committee cannot allow this standard version of BASIC because: (1) There are no integer data types, (2) There is no bit manipulation, (3) You cannot open files and write or read from them, and (4) You cannot resume a line number. Everyone present agreed that, yes, it would be difficult to program with these restrictions, but the main problem was that this standard version of BASIC could not meet the business needs of the community.

Since BASIC is really used for business and the proposed standard version cannot meet all the business needs, the question was then asked what can be done to prevent this version to become the standard BASIC.

The only solution suggested was to write a substantial amount of objection letters to the committee. These letters should address all aspects of the version, because the committee is required to answer all letters received. If enough letters are sent to the committee, then the proposed version has to go through another public review. It was stressed that all areas of objection must be dealt with before another review, because once it was reviewed for one specific objection, then that objection cannot be brought up again(so that there isn't an infinite loop of reviews). Emphasis was placed on the fact that the letters need to be sent out immediately.

After the discussion of standardizing BASIC was finished, the group told war stories. One topic covered was scale factors and the differences and complications that arise when using sale factors of zero or six. Another topic dealt with was BASIC PLUS and how when doing a listing, every single character generates an I/O call. The next meeting will be held in September in San Francisco.

BASIC-PLUS-2 Installation Tuning

Edward F. Vogel
Digital Equipment Corporation
Nashua, NH

Ray Strackbein,
Session Chairperson
ICS
Palm Desert, CA

Reported by
Todd Spangler,
DECUS Scribe Service

Installing BASIC-PLUS-2 requires many considerations which will effect the efficiency of the system. Edward F. Vogel from the BASIC Development Group, Digital Equipment Corporation explained the importance of the use of resident libraries, overlays, mappings, and the installation of the RUN command. He also explained how each of these features affect the system performance.

Depending on the system and memory available, two resident libraries are the BP2RES and BP2SML. BP2RES will require 18K to 20K of system memory, while disk usage is 150 blocks under RSX and 235 blocks under RSTS/E. BP2RES takes longer to execute and is larger than the BP2SML. BP2RES advantages are that it contains more address space, it is clusterable (depending on the system), it can share code and has a smaller task file. The library also has the capability to link RUN to it.

With BP2SML, 8K of virtual address space is required. The disk usage is 45 blocks under RSX and 100 blocks under RSTS/E. The advantages of smaller library are that it is smaller, it is better than not using a library at all, it is clusterable, has shareable code, and has a smaller task file than a system with no library. The task file here is slightly larger than that of the BP2RES, and is smaller than the task file of a system with no library. Another difference is that the smaller library can be customized.

Memory and run time are trade-offs involved with the choice of libraries. When possible, build both libraries into the system. By default, one should link to the smaller library since it requires less memory and has a small task file. If memory becomes

a problem, the next step is to cluster the smaller library with RMSRES if it is in the system. This will conserve memory and run time. If there are still problems with memory, one can use the larger resident library, BP2RES. This results in slower execution time , although the exact slow down is undetermined. Again, the choice is up to the needs of the users. If at all possible try to avoid clustering as this imposes a large overhead which can slow the system by as much as 30 percent.

The next consideration is whether or not to install RUN, which closely fits in with the use of resident libraries. If a resident library is not used, every user using RUN has his own copy of RUN which will cost 200 blocks per user. Using resident libraries will considerably reduce the memory requirements. Using BP2RES with run increases the execution time, but requires less disk space since there is no disk overlay. This can be done in version 2.0 only. In version 2.1 all resident libraries will again increase execution time and use less disk space. In version 1.6, RUN is available in all systems and is fully functional, including matrix statements, a calculator mode, run/debug and more address space. The increased address space will allow larger programs to be executed under the RUN command. Installing remote file access can also be helpful. It will allow the system to access another DECnet node, but will reduce usable address space.

File placement also affects run time; the proper placement can also reduce compile time. In order to minimize compile times a good choice of devices is important. A fast disk will improve the compile time as will keeping the compiler on a separate disk from the system disk. This also applies to the work files which should be kept separate from system and compile files. If there are not enough disks available it is better to combine the compiler and the system in order to maximize efficiency. Other tips include caching compiler work files and marking the compiler for data cache (this is possible only on RSTS/E).

The system hardware configuration will determine what features work best. If the system has a big disk and large memory, install all libraries, use RUN and remote file access. With a small disk and large memory, use the resident libraries and try not to install RUN. If RUN is needed, link it with the libraries. With a big disk and small memory, use RUN without the resident libraries. With a small disk and small memory, trade-offs must be tailored to the individual system. In all cases, there are no "rules" to follow since each system requirements are unique.

Using New Compiler Directives in BASIC V2.0

Edward Vogel
Digital Equipment Corporation
Nashua, NH

Bob Van Keuren, Session Chairperson
North County Computer Services
Escondido, CA

Reported by Susan Miller, DECUS Scribe Service

Compiler directives in BASIC have not been effective for some users. Speaker Edward Vogel of DIGITAL gave a presentation on "Using New Compiler Directives in BASIC V2.0." Vogel said, "I'm gonna tell you what they're good for, not how to use them." But his presentation did not convince some audience members. These users cited problems they had experienced and problems that they anticipated with compiler directives.

In his speech, Vogel said that DIGITAL tells users that a compiler directive will control listing file, will include code from another file and will conditionally compile programs. But what they really do is provide more maintainable code, provide transportable code, aid in debugging and increase development time. He gave these general rules:

1. Begin with percent sign.
2. Cannot begin in first column.
3. Must be the only text on the line.
4. Cannot be within a quoted string.

To list file directives, Vogel says to use %TITLE, %SBTTL (subtitle) and %PAGE. This provides module information, readability and makes it easier to maintain code.

Another set of directives include %NOLIST, %LIST, %NOCROSS and %CROSS. Use %NOLIST for the old working code and %LIST for the new code. While %CROSS is placed at information needed for future reference.

Vogel said that %IDENT would modify listing and output files, track versions of source, aid in debugging and could be seen in task map.

To include code from another file, the directive is %INCLUDE. This helps data definitions, constant definitions, commonly used functions and subroutines. Then the code will be easier to modify, more reliable and transportable.

In the control compilation, normal integer operations can be used with the lexical constants. The controlling compilation consists of %LET, %VARIANT, %IF, %THEN, %ELSE and %ABORT. The %VARIANT is evaluated when you complete file. If the equation is false, everything between then and else is not in the output. The %ABORT stops program compilation and gives error checking ability.

After Vogel's 30-minute presentation, some audience members were eager to fire questions. Sheila Kelly from Michigan asked, can you on the 11/70, like on the VAX, compile and link and avoid the switch on the BASIC. Vogel said, "Not at this time. That is also a strongly requested item."

Greg Whitier of Kimcorp Computer asked, "What's the change of getting V2.0 compiler to run faster than the taskbuilder?" Vogel said that a V2.0 will be available in a few weeks.

"On RSTS systems you should see some improvement. It is our highest priority. That and fixing bugs to improve compiler performance", Vogel answered.

Jeff Harrolock of Georgia Company gave a personal observation of the compiler directives. "When the system was busy with the disks in heavy use, it wasn't unusual to get 10 to 11 times longer." In another observation, he noted, "In the real world in a large commercial shop, all the points that you make about overall maintenance of the compile software have some merit. You can't take a shop that is using its computer already at a reasonable figure of its maximum ability and quadruple or worse the compilation time and still expect them to be able to get their work done. You can't do it; not in the real world." About half of the audience showed their agreement with this comment by clapping. Vogel said that the point was well taken. "We improve compiler performance by about four times between field test two and field test three. We are looking into it."

Another user complained about the time wasted with the compiler directives. "When I took 15 minutes to compile before and it takes an hour now, that is totally unacceptable. It's just out of the question for us to swallow that kind of thing." Vogel said that the reason performance is too slow is because of added features. These features were added because of requests from people at DECUS sessions.

The conflict between DIGITAL and users could be eased with the realization of tradeoffs. The systems cannot handle all the features and maintain maximum performance. Compiler directives may have advantages, but there are problems that need to be worked out.

Using External MACRO-11 Functions in BASIC-PLUS-2

Brian A. Hetrick
Digital Equipment Corporation
ZKO2-3/KØ7
Spit Brook Road, Nashua NH Ø3Ø62.

This article describes undocumented, unsupported features of PDP-11 BASIC-PLUS-2. It lays bare the secrets of writing MACRO-11 functions that can be invoked from PDP-11 BASIC-PLUS-2. Of course, none of this is supported in any way, shape, or form, by Digital, the author, or anyone else. I repeat: unsupported. Let the reader beware. With that provison, you can be using MACRO-11 functions from BASIC-PLUS-2 within an hour.

If you're a BASIC-PLUS-2 version 2 wizard, then what you need is the second sentence of the first paragraph of "Writing the external function". Otherwise, read on.

Introduction

External functions are supported in PDP-11 BASIC-PLUS-2 version 2. Like SUB subprograms, these are separately compiled modules linked together with the task builder; like DEF functions and BASIC built-in functions, they return values that can be used directly in expressions without going through a temporary variable.

However, the BASIC-PLUS-2 documentation does not describe how to write MACRO-11 functions, and the linkage conventions used are unique to PDP-11 BASIC-PLUS-2. As far as BASIC is concerned, this doesn't matter: MACRO subprograms aren't supported anyway. There is fine line dividing what is and what isn't supported. The semantics of the CALL BY {DESC | REF | VALUE} statement are supported at entry to the subprogram, the parameter list looks a certain way. However, no use of this statement is supported. Is this clear? Good.

But this makes it rather difficult to write external functions in MACRO-11 that are "callable" from BASIC-PLUS-2.

Now, before we get to the good stuff, a note about the terminology used in this article: an ACTUAL PARAMETER is what a calling program sends a subprogram, a FORMAL PARAMETER is what the subprogram receives from the calling program, and a just plain PARAMETER is both at once. The equivalent FORTRAN terms are "actual argument," "dummy argument," and "argument;" (the equivalent COBOL terms are "argument," "argument," and "argument.")

Declaring the external function

In the BASIC-PLUS-2 program, the external function must be declared before it can be used: otherwise, the compiler treats the function invocation as a reference to an element of an array. The syntax for the declaration of an external function is:

```
                                                    {DESC
EXTERNAL resulttype FUNCTION name [BY              {REF    ]
                                                    VALUE}

        [ ( formalparameter [, ...] ) ]
```

"Resulttype" is the data type of the function result. This can be RFA or any numeric type: BYTE, WORD, LONG, SINGLE, or DOUBLE. The INTEGER and REAL keywords can also be used, but this is dangerous, as then the assumed datatype for the function result depends upon the compilation switches for or the OPTION statement in the calling program. Using STRING as a result datatype is VERY dangerous, unless you really know the ins and outs of the BASIC-PLUS-2 OTS;

.....and that has to wait for another article.

"Name" is the name of the external function. Unlike the EXTERNAL SUB statement, the EXTERNAL FUNCTION statement does not permit a quoted string for the name, so this name must be both a legal BASIC-PLUS-2 name and a legal external name. It must be 1 to 6 characters long, start with a letter, and continue with letters, digits, dollar signs, and periods.

The BY {DESC | REF | VALUE} clause, immediately following "name," is optional. If present, it has two effects:

The thread generated at invocations of the function is CBR$, rather than CAL$. This avoids certain context saving action, and results in a quicker running program.

The default parameter passing mechanism is set to BY DESC, BY REF, or BY VALUE, as specified, for all parameters of the function.

If the external function is written in BASIC-PLUS-2, the "BY {DESC | REF | VALUE}" clause must NOT be specified, because BASIC-PLUS-2 external functions can't receive formal parameters by anything but the default passing mechanisms, and the CBR$ thread doesn't save enough context to call a BASIC-PLUS-2 program.

If the external function is written in MACRO-11, the "BY {DESC | REF | VALUE}" clause need not be specified, but the program will run faster if it is specified.

The formal parameter list description as a whole is optional. If present, it specifies the number of formal parameters to the function, and specifies the type, structure, and parameter passing mechanism for each formal parameter. At most eight formal parameters can be described in the formal parameter list description. Each "formalparameter" has the syntax:

```
                                                    {DESC
[datatype] [DIM ([,...])] [= length] [BY           {REF    ]
                                                    VALUE}
```

"Datatype" is the data type of the formal parameter. This can be any BASIC type (BYTE, WORD, LONG, SINGLE, DOUBLE, STRING, or RFA). The INTEGER and REAL keywords can also be used, but this is dangerous, as then the assumed datatype for the function formal parameter depends upon the compilation switches for or the OPTION statement in the calling program. If the datatype keyword is omitted, the datatype of the preceeding formal parameter is assumed. In the description of the first formal parameter, "datatype" must be present.

The "DIM ([,...])" clause is optional. If present, it specifies that the formal parameter is an entire array with a particular number of dimensions. Its absence specifies that the formal parameter is not an array.

The "= length" clause is optional, and may be given only if the data type for the formal parameter is STRING. If present, it specifies that the formal parameter is a static string (i.e., a string from a COMMON or MAP), and is "length" characters long. If the datatype is given as STRING and the "= length" clause is not given, then the actual parameter can be either a static string or a dynamic string.

Finally, the "BY {DESC | REF | VALUE}" clause is optional. If present, it specifies the parameter passing mechanism to be used for the parameter. If absent, it specifies that the "BY {DESC | REF | VALUE}" clause appearing after "name" specifies the parameter passing mechanism. If neither "BY {DESC | REF | VALUE}" clause is present, the passing mechanism to be used for the parameter depends on the datatype and structure of the parameter: this is BY REF for unsubscripted numeric and RFA items, and BY DESC for strings or for entire arrays. The "BY {DESC | REF | VALUE}" clause at the parameter level does NOT affect the thread generated at invocations of the function.

There is one special case that is not covered above. If the formal parameter list is described as () then the function is explicitly declared as having zero parameters. If the formal parameter list description is merely omitted, then nothing is implied about the parameters.

There are both advantages and disadvantages to giving formal parameter descriptions in the EXTERNAL statement. The big advantage is that BASIC-PLUS-2 won't let you mismatch the actual and the formal parameters. It will convert the actual parameter to the type of the formal parameter, if possible (and give an informational message if the conversion means that the actual parameter is no longer modifiable), check the structure of the actual parameter against the formal parameter, and pass the actual parameter by the mechanism specified in the formal parameter description. The big disadvantage is that BASIC-PLUS-2 won't let you mismatch the actual and the formal parameters. If you have an external function that takes a variable number of parameters, or where the datatype of the parameter changes from invocation to invocation, BASIC-PLUS-2 won't let you invoke the function. But, you don't have to describe the formal parameters in the EXTERNAL statement, so there is an escape hatch.

Invoking the external function

The syntax for an invocation of an external function is:

```
                    {variable           {DESC
name [ ( [ expression      [BY  REF    ] ] [, ...] ) ]
         entire array}             VALUE}
```

"Name" is the name of the function as given in the EXTERNAL statement.

Unlike the CALL statement, the function invocation does not permit a "BY {DESC | REF | VALUE}" clause on the function invocation as a whole.

The actual parameter list as a whole is sometimes mandatory, sometimes optional, and sometimes forbidden, depending on the formal parameter list specification in the EXTERNAL statement:

If the formal parameter list description was given as () in the EXTERNAL statement, then the actual parameter list is forbidden.

If the formal parameter list was specified in the EXTERNAL statement as anything but (),then the actual parameter list is mandatory.

If the formal parameter list was not specified in the EXTERNAL statement, then the actual parameter list is optional.

If the parameter list is specified at all, then there is at least one parameter. For example, if () is given as the parameter list, there is one parameter, and it is omitted (see below). This differs from the syntax in the EXTERNAL statement, where () is used to explictly indicate that there are zero parameters.

An entire parameter may be omitted. Omitting a parameter in BASIC-PLUS-2 is the same as specifying '-1'W BY VALUE that is, the word value -1 is passed instead of an address in the parameter list. Even if the parameter was described in the EXTERNAL statement, omitting the parameter is legal.

If a parameter is not omitted, it can be an expression, an unsubscripted variable, or an entire array. [Subscripted variables are essentially expressions.] There's a whole big blurb in the Language Reference Manual about what can be passed how, and whether it is modifiable, so that won't be covered here.

If the parameter list was described in the EXTERNAL statement, then the actual parameters in a function invocation can not contradict the formal parameters in the EXTERNAL statement. However, the actual parameters don't need to exactly match the formal parameters; BASIC-PLUS-2 will perform conversions to get the actual parameter to be the same type as the formal parameter, if possible, and will pass the parameter by the mechanism specified in the EXTERNAL statement. However, if the formal parameter list was not described in the EXTERNAL statement, then the function invocation can do what it pleases, even if it specifies actual parameter lists that differ totally from actual parameter lists in other invocations of the same function, and BASIC-PLUS-2 won't get in the way.

Writing the external function

Writing a MACRO-11 external function to be called by BASIC-PLUS-2 is just like writing a MACRO-11 external subprogram to be called by BASIC-PLUS-2, except, of course, that it's different. The difference is that the function result is the first formal parameter of the function. Here's where the difference between actual parameters and formal parameters can really bite you!!! This trick is called the "hidden first parameter convention," and BASIC-PLUS-2 picked it up from the VAX. It's a little strange, but makes some sense: the function can't return a LONG, SINGLE, DOUBLE, or RFA value in R0, but can return one with a hidden first parameter.

In terms of what it means to the function,the EXTERNAL statement:

```
EXTERNAL datatype FUNCTION name [BY clause] [(parameterlist)]
```

is equivalent to the EXTERNAL statement:

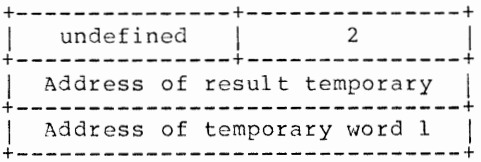
```
EXTERNAL SUB name [BY clause]
     (datatype BY defaultmechanism [, parameterlist])
```

The result is always passed as the first actual parameter, by the default mechanism for that type ("defaultmechanism" above): for RFA and all numeric types, this is BY REF.

Suppose that a BASIC-PLUS-2 program has the declaration:

```
EXTERNAL WORD FUNCTION FOO (WORD)
```

and the function invocation X = FOO (1.5) where X is a WORD variable. The parameter list the function FOO would get would be:



Note that BASIC-PLUS-2 has converted the floating point constant 1.5 to the word integer 1%, put that word 1% into a temporary, and then passed the temporary BY REF (that being the default passing mechanism for WORD values). This is the effect of specifying "WORD" in the formal parameter description of the EXTERNAL statement. Also, the function is essentially a subprogram with two formal parameters: the first formal parameter is the function result temporary, while the second formal parameter is the first parameter at the function invocation site.

The formal parameters received by the function are the same as the formal parameters described by the EXTERNAL statement, with the addition of a first parameter that is the function result.

The external function FOO could be written in BASIC-PLUS-2 with the FUNCTION statement:

```
FUNCTION WORD FOO (WORD Y)
```

Here, Y is the formal parameter, and assignment to FOO within the body of the function would give a value to the function which, in the main program, would be given to X.

Suppose the function's action was to return the value of its parameter. The BASIC function would be:

```
10      FUNCTION WORD FOO (WORD X)
        FOO = X
        END FUNCTION
```

while the equivalent MACRO-11 function would be:

```
FOO::   MOV     @4(R5),@2(R5)
        RTS     PC
        .END
```

How about a function that can't be easily done in BASIC-PLUS-2: a word left rotate? The function declaration would be:

```
EXTERNAL WORD FUNCTION ROTATE (WORD BY VALUE, WORD BY VALUE)
```

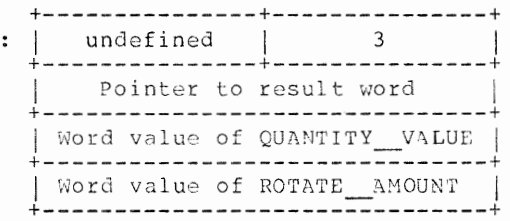
The first parameter is the word to be left rotated, and the second parameter is the number of bits by which the first parameter is to be rotated. The function itself would be:

```
        .GLOBL  $SAVVR
ROTATE::JSR     R2,$SAVVR       ; Save registers 0-2
        MOV     6(R5),R2        ; Get number of bits to rotate
        BIC     #177760,R2       ; Take modulo 16
        SUB     #20,R2           ; Get ASHC count
        MOV     4(R5),R0        ; Get word to rotate
        MOV     R0,R1           ; Copy it to make longword
        ASHC    R2,R0           ; Rotate into R1
        MOV     R1,@2(R5)       ; Store result
        RTS     PC              ; Restore registers and return
        .END
```

An invocation of the function might be:

```
ROTATE (QUANTITY_VALUE, ROTATE_AMOUNT)
```

The parameter list given to the function is:



Although the function has two parameters, they are at 4(R5) and 6(R5) (rather than 2(R5) and 4(R5)) because of the hidden first parameter result convention. The parameters are at 4(R5) and 6(R5) instead of @4(R5) and @6(R5) because they are passed BY VALUE rather than BY REF. Finally, the result has to be put into @2(R5) rather than R0, again because of the hidden first parameter result convention.

Writing the external function "compatibly"

Suppose that the MACRO-11 function is to be usable from both FORTRAN and BASIC-PLUS-2 calling programs. There are several difficulties:

The same object file couldn't be linked with a FORTRAN program and a BASIC-PLUS-2 program, because of the hidden first parameter convention used by BASIC-PLUS-2.

The passing mechanism couldn't be BY DESC or BY VALUE, because FORTRAN passes everything BY REF.

But, if the function accepts all its parameters BY REF and uses conditional assembly, the same source code (at least) could be used, as follows:

```
_.IIF NDF BP2    BP2 = 1          ; Assume BASIC-PLUS-2 linkage
_.IF     EQ      BP2              ; Following for FORTRAN linkage
WORD     =       2                ; Word to shift
SHIFT    =       4                ; Amount by which to shift
_.IFF                             ; Following for BP2 linkage
WORD     =       4                ; Word to shift
SHIFT    =       6                ; Amount by which to shift
_.ENDC                            ; End of linkage conditional
         .GLOBL  .SAVR1
ROTATE::JSR      R5,.SAVR1        ; Save registers 1-5
         MOV     @SHIFT(R5),R2    ; Get number of bits to rotate
         BIC     #177760,R2       ; Take modulo 16
         SUB     #20,R2           ; Get ASHC count
         MOV     @WORD(R5),R0     ; Get word to rotate
         MOV     R0,R1            ; Copy it to make longword
         ASHC    R2,R0            ; Rotate into R1
_.IF     EQ      BP2              ; Following for FORTRAN linkage
         MOV     R1,R0            ; Put result into R0
_.IFF                             ; Following for BP2 linkage
         MOV     R1,@2(R5)        ; Store result
_.ENDC                            ; End of linkage conditional
         RTS     PC               ; Restore registers and return
         .END
```

In order to write a function so that it is callable from both BASIC-PLUS-2 and FORTRAN (although using conditional assembly, and hence with different object files for the two languages), it was necessary to:

Pass all parameters BY REF, as FORTRAN can't pass parameters BY VALUE or BY DESC.

Use symbols for the offsets of the parameters in the parameter list, and conditionalize these symbols based on the linkage convention, as they are different in the FORTRAN and the BASIC-PLUS-2 cases.

Use a different register save co-routine, so that the value can be returned in R0 in the FORTRAN case.

Conditionalize the storing of the return value, putting it into R0 in the FORTRAN case and into @2(R5) in the BASIC-PLUS-2 case.

The function can be declared in BASIC-PLUS-2 with:

```
EXTERNAL WORD FUNCTION ROTATE (WORD, WORD)
```

and in FORTRAN with:

```
INTEGER _* 2 ROTATE
```

Of course, FORTRAN won't check that there are two actual parameters, and that they are words; BASIC-PLUS-2 will.

Summary

The EXTERNAL statement is used to declare an external function. An external function is invoked with syntax that is a cross between a DEF function invocation and an external subprogram invocation. The external function receives a parameter list where the first formal parameter is the function result, and the remaining formal parameters are the actual parameters specified at the function invocation. Finally, conditionalized MACRO-11 source code can be used for external functions callable from both BASIC-PLUS-2 and FORTRAN, although separate object files are required.

MACRO & BASIC-PLUS-2 User Tutorial

Wef Fleischman
Software Techniques Inc.
Los Alamitos, CA

Bill Tabor, Session Chairperson
Racal-Milgo
Miami, FL

Reported by Gene Mitchell, DECUS Scribe Service

"A high-level language like BASIC is almost always the best implementation language for maintenance and transportability reasons. But you should recognize the potential areas for improved performance and enhanced efficiency afforded by MACRO."

Such was the conclusion reached by Wef Fleischman of Software Techniques, Inc., the speaker at a session entitled "MACRO and BP2 V2 Tutorial." Topics of interest were focused into MACRO vs. BP2, likely applications of both, and programming techniques.

A comparison of the two languages showed some of the better features of BASIC:

1. Well organized, logical structure.

2. Protection of programmer from pitfalls. For instance, BP2 requires all variables to be explicitly and correctly spelled. Earlier BASIC systems allowed for misspelled variables, which could have adverse consequences on other sections of a program.

3. Good maintenance characteristics--many people have a knowledge of the language.

4. Transportable code--can migrate to many different machines.

MACRO-11, on the other hand, said Fleischman, is by comparison confusing to many. Drawbacks include:

1. MACRO-11 does not impose a logical structure.

2. It is sometimes difficult to understand. Programmers will occasionally have difficulty reading each other's code.

3. Debugging is an art. Debugging MACRO programs can adversely affect even logically independent parts.

4. MACRO-11 is not very transportable. MACRO is great for getting at "intimate details" of the PDP-11, but not useful on other machines.

But BASIC has some faults that do not arise in MACRO. While BASIC is more logical, it may not be optimally efficient. In addition, BASIC may not allow some access features that MACRO will. Unneeded overhead may be incurred as a result of the overly structured BASIC program.

MACRO allows several things that BASIC may not:

1. Unrestricted access to the underlying machine and operating system.

2. Greater speed and efficiency in many cases.

3. The performance, in certain cases, of special data manipulations.

4. Explicit control of memory is allowed.

Fleischman offered three examples of incorporating MACRO with BASIC. The first was initialization of static data. First the BASIC program was given:

```
1000 DECLARE INTEGER PART
     MAP (PARTS) STRING DESCRIPTION (7)=8

2000 INPUT "PART NUMBER "; PART
     PRINT "DESCRIPTION "; DESCRIPTION (PART)
     GOTO  2000

32767 END
```

Next, the MACRO equivalent was provided:

```
     .TITLE   PARTDF

     .PSECT   PARTS,RW,D,GBL,REL,OVR

     .ASCII   "BREATHER"
     .ASCII   "CABLE   "
     .ASCII   "CHOKE   "
     .ASCII   "COVER   "
     .ASCII   "PISTON  "
```

```
.ASCII    "SOLENOID"
.ASCII    "SPRING  "
.END
```

Problems of sharing data between MACRO and BP2 systems were addressed, with two solutions being offered:

1. Through COMMONs or MAPs. According to Fleischman, the .PSECT directive in MACRO corresponds directly to the BASIC MAP or COMMON statement.

2. Through passed argument lists. An argument list is a contiguous block of memory assigned by the BASIC main program when the CALL statement is executed. Arguments can be passed by reference (REF), descriptor (DESC) or value (VAL).

The second example of incorporating MACRO with BASIC concerned how to pass data to a MACRO subprogram, and system function not supported by BP2, such as (the .NAME directive). It assumes that one has fairly lengthy processing. According to Fleischman, "This ability to change a job's name is not privileged but cannot be performed in BASIC." The MACRO code to do this is as follows:

```
        .TITLE  CHNAME

        .PSECT  CHNAME,RO,D,GBL,REL,OVR

CHNAME::

        MOV     #XRB+XRLOC,R1   ;POINT TO XRB
        MOV     2(R5),RO        ;POINT TO DESCRIPTOR
        MOV     (RO)+,(R1)      ;COPY LOCATION OF STRING
        MOV     (RO),-(R1)      ;COPY LENGTH OF STRING
        MOV     (R1),-(R1)      ;DUPLICATE IN XLREN
        CLR     @#XRB+10        ;NO SPECIAL LOGICAL TABLE
        .FSS                    ;CONVERT TO RAD50
        TST     @#FIRQB         ;ERRORS?
        BNE     10$             ;YES, EXIT WITHOUT EFFECT
        .NAME                   ;CHANGE THE JOB NAME
10$     RETURN                  ;RETURN TO CALLER

        .END
```

Calling a MACRO-11 subprogram, such as the one above, can be done with four steps:

1. Declare the external name.

2. Specify parameter list (if any).

3. Double-check parameters.

4. CALL.

An example:

```
1000      EXTERNAL SUB CHNAME (STRING BY DESC)

2000      CALL CHNAME ("STEP1")
                *
                *
                *
          CALL CHNAME ("STEP2")
                *
                *
                *
          CALL CHNAME ("STEP3")
                *
                *
                *
              (etc.)
```

The third example of incorporating MACRO with BASIC is in demonstrating special RMS file handling. The BASIC program is as follows:

```
1000      MAP (BUF) STRING KEY=10%
2000      OPEN "LOAD.IDX" FOR OUTPUT AS FILE #1     &
                , INDEXED FIXED                     &
                , ACCESS MODIFY, ALLOW NONE         &
                , MAP BUF                           &
                , PRIMARY KEY                       &
                , USEROPEN BKTFIL
          PUT #1%
32767     END
```

The MACRO program is as follows:

```
          .TITLE   BKTFIL
          .PSECT   BKTFIL,RW,I,LCL,REL,OVR

          .MCALL   RABOF$,XABOF$,$STORE,$SET
          .MCALL   $FETCH,$CREATE,$CONNEC
BKTFIL::
          MOV      2(R5),R0          ;GET ADDRESS OF FAB
          MOV      4(R5),R2          ;GET ADDRESS OF RAB
          $FETCH   R1,XAB,R0         ;GET ADDRESS OF XAB
          $STORE   #256.,DFL,R1      ;SET DESIRED DATA FILL
          $STORE   #256.,IFL,R1      ;SET DESIRED INDEX FILL
          $CREATE  R0                ;CREATE THE FILE
          $CONNEC  R2                ;CONNECT THE RAB
          $SET     #RB$LOA,ROP,R2    ;SET "LOAD BY FILL"
          RETURN
          .END
```

Fleischman added that in creating the RMS-11 Bucket fill

control, there are four steps we must observe:

1. Specify fill factor in key XAB.
2. Create file ($CREATE/CONNECT).
3. Set fill control bit in ROP field of RAB.
4. Load records.

The session concluded with some BASIC V1 "unsupported RMS routines" and other applications for MACRO-11. Unsupported RMS routines include:

1. RETRFA - return record file address
2. GETRFA - set record by RFA
3. NULKEY - enable a null alternate key
4. SEGKEY - define a segmented key
5. BKTFIL - define bucket fill factors
6. SETROP/CLRROP - set record options
7. DEFALQ - set default extension size
8. DEFFNA - set default file name
9. SETFOP - set file options

Other applications for MACRO-11 include:

1. Passing variable number of arguments
2. Accessing job "lowcore" area
3. Patching constants via MAKSIL/ONLPAT
4. Observing/modifying BP2 OTS behavior
5. Controlling libraries explicitly (.PLAS)
6. Performing device specific I/O (.SPEC)
7. Creating "executable" files (CRBFQ)

Fleischman concluded by reemphasizing the areas that MACRO can affect towards improving performance. He also attempted to dispel the idea that MACRO is overly sophisticated for the needs of many. It is on the contrary easy to learn and use. Use of MACRO may decrease programming time or reduce the need for new equipment.

Managing FDA Labs With BASIC-PLUS-2

Larry Alber
Food and Drug Administration
Chicago, IL

Tony Seckel, Session Chairperson
McDonnell Douglas Automation
Berkeley, MO

Reported by Scott Howell, DECUS Scribe Service

Tony Seckel introduced the sessions speaker, Larry Alber who began his discussion with an outline of the laboratory management systems structure. The steps are as follows:

1. Sample analysis completion date
2. Analysis home district
3. Operation
4. Hours
5. Position class
6. Employee number
7. Analyzing district
8. Program assignment code

Alber then introduced the symposium coordinator of the Labs SIG, Mac Overton. Alber gave a brief description of the background of the FDA's needs for a computer system. Terminals are scattered throughout the laboratory for easy access. BASIC-PLUS-2 is used to record all tests made on foods and drugs. Included in this record is the employee who conducted the test, the item tested, and its result from the test.

Alber issued a handout with the information that his system employs. When searching for a record the following appears on the screen:

A- Add new records or data to existing records

C- Change data on existing records
D- Delete an existing record or segment
P- Compliance program data entry menu
M- Monthly file maintenance routine
T- Lab designation table maintenance
S- Search for specific LMS records
R- LMS reports menu
E- End your data entry session

There are three keys for accessing records. They are:

1. The sample number

2. The program assignment code

3. The LID code

Each field is edited to insure correct input. For example, if someone entered May 33 as a date the system beeps and requests an alternate input. The entire system took approximately one year to install and be in working condition. Alber concluded his thirty minute discussion with a list of the various reports that his BASIC-PLUS-2 system makes. This report menu is as follows:

1. A weekly report is published. (backlog, inprocess, completes by lab)

2. Program assignment code summary of completed records.

3. Track/archive file special reports.

4. Employee report is published.

5. A narrative report is made.

6. A pesticide listing is made by product code.

7. A pesticide residue level tabulation is done.

8. A violative sample report is furnished.

9. A compliance/LMS sample report is completed.

INTERPRETIVE BUSINESS BASIC WITH RMS-11K

JON COLEMAN & DAVE NASATER
COMPUTER SYSTEMS DEVELOPMENT, INC.
140 MAYHEW WAY, SUITE D700
PLEASANT HILL, CA 94523

HISTORY

Business BASIC as discussed here, was first offered in 1971 by MAI Basic Four Corporation on their small business minicomputers. This was the only language available on their machines. After experience proved Basic Four unable to cope with customer needs for package modifications and custom programs, they actively recruited independent software vendors to provide software development and ongoing support for their customers. In the last 13 years, 400 plus software organizations have provided support for approximately 16,000 Basic Four machine installations worldwide.

One software company, Science Management Corporation (SMC), had developed an application development tool and DBMS written in Business Basic. Their desire to market their system on other hardware prompted them to implement an MAI Business BASIC (TM) look-alike product, first on the IBM Series-1 and then on approximately 30 other 16-bit microcomputers.

Computer Systems Development (CSD) has been involved in Business Basic software development since 1974, and has developed several vertical packages that we wished to market on other hardware, particularly DEC. Discussions between CSD and SMC determined that SMC was not going to put their products on DEC. Experiments with converting Business Basic applications to BASIC-PLUS-2 convinced us that some implementation of Business Basic for DEC equipment would be the best solution.

DESIGN AND IMPLEMENTATION

General desirable characteristics of the language were deemed to be:

1. The language should be interpretive, for ease of developmement and customer support.

2. Business math should be designed in; rounding should be automatic, and type conversions unneeded or rare.

3. Extensive screen handling functions should be part of the language, instead of programmer written or included from a library.

4. Basic input verification ability should be built in.

5. Disk I/O should include keyed (index), relative-record, and sequential file types.

6. String handling capabilities should include all normal business programming needs.

7. Records and files should be able to be locked.

Several design decisions were made for the initial development of the interpreter.

1. Program execution and storage would use a compact pseudo-code.

2. Syntax would be the same as Basic Four with minor exceptions.

3. The compiler would be table-driven for ease of future expansion.

4. The initial versions would be written for machines without floating point or commercial instruction sets.

5. RSTS/E versions would be implemented first because business programs are suited to a synchronous operating system, and because of the larger number of jobs RSTS can handle.

6. RMS-11K was chosen as the file system due to DEC support and the availability of using the same files as other DEC supported languages.

Choosing RMS as the file system affected other design considerations. RMS buffers must come out of the user's space. If the initial design criteria of having the ability to open seven (7) typical files was to be met, at least part of the buffer pool would have to be allocated dynamically. Because of buffer space requirements for RMS, it was decided to forego extended variable names. This had the effect of reducing the pseudo-code space requirements by approximately 50%.

INTERPRETER VERSUS COMPILER

For several years, the larger DEC users have been gently pushed away from interpreters towards compilers. The primary reason for this is supposedly the much higher execution speed of compiled programs. Secondarily, the BASIC-PLUS users needing the functionality of RMS had to switch to BASIC-PLUS 2 to get it.

We believe that the future of COEMs and software developers is directly linked to their ability to produce turnkey solutions at reasonable cost. Software development and maintenance costs are

typically large budget items that are sometimes difficult to relate to direct revenue production. Using an interpreter can cut development costs significantly.

In a test done in 1980, duplicate specifications for an Accounts Receivable package were given to two project teams. One team was to write the system in BASIC-PLUS-2 for a PDP 11/34 under RSTS/E, and the other team was to write the system in MAI Business BASIC. The results were quite different than expected. We expected that it would take two or three times as many man-hours for the compiled versus the interpretive versions, but the actual factor was more than seven to one.

Research using daily task sheets submitted by programmers and interviews revealed several reasons for the larger than expected differential:

1. The BASIC-PLUS-2 (BP2) system had 1.85 times as many lines of code as the Business BASIC version.

2. The BP2 programmers had to keep at least two programs in their minds at a time, in order to be productive during re-compile and task-build. They felt that this divided attention caused oversights.

3. A significant amount of time was spent constructing ODL files.

4. BP2 programmers stated that debugging with ODT was extremely tedious. Programmers claimed they would quickly reach a bug they could not get past, and so would re-compile and task-build, in order to continue.

5. Three of four programmers on the BP2 team said they had a strong tendency to not work on anything else while they were re-compiling and task-building a program in test.

NOTE ON BASIC-PLUS

BASIC-PLUS is a very powerful, easy to use interpreter, however, it has no direct access to keyed RMS-11K files. Most business applications, and virtually all interactive business applications can be more effective using keyed files.

ELEMENTS OF BUSINESS PROGRAMS

Since our object was to design a business oriented interpreter, we tried to determine what differences there are between business and scientific programs. The following list refers to degrees of differences for business as opposed to scientific programs.

1. Virtually all arithmetic, except loop control, is oriented to produce results accurate to two decimal places.

2. Less arithmetic is performed, measured as number of instructions executed.

3. More string manipulation is performed.

4. More input/output operations are performed.

The above elements were kept under constant consideration during implementation.

BUSINESS BASIC STATEMENT FORMAT

Business BASIC programs are made up of collections of ordered statements. The statements have the general form of:

statement #	An integer in the range of 0 to 9999. No statement # or one of 0 indicates an immediate mode statement. Immediate mode statements are not part of a program, but are typed in console mode and are acted on immediately.
directive	The function the statement is to perform. For example, PRINT, INPUT, OPEN, LET ...
options	Some directives require or can have statement options. These options include I/O channel, key to use and error processing.
arguments	The variables, literals, and/or expressions the directive is to act on. Some directives require no arguments.

Examples of Business BASIC statements.

a. PRINT "HI"

   Causes HI to be printed immediately, since no statement # is given.

b. 10 READ (1,END=1000,KEY=A$)R$,R,R1

   When executed will read from channel 1 with a key of the value of A$. The data read in will be put into variables R$, R and R1. If end of file occurs, execution will be tranferred to line 1000.

c. 100 END

When executed, will terminate the program closing all channels. Variables remained assigned.

d. 3050 IOLIST A$,B,C(1),D$(5,6)

Defines an I/O argument list that can be referenced later with an IOL=3050 in the argument list. D$(5,6) refers to bytes 5 through 10 in D$.

e. 200 WRITE RECORD (7,KEY=A$+"A",DOM=1000,ERR=9000)C$

When executed will write a single record of data without type translation to the file opened on channel 7. The key of the value of A$ with the character A concatinated to it will be used. If the key already exists, no record will be written and execution will be transferred to 1000. Any other error will cause execution to be transferred to 9000.

PSEUDO-CODE FORMAT & EXECUTION

Whenever a statement is entered into a program in Business BASIC, it is compiled from the source form into a dense Pseudo-Code (P-Code). The resulting P-Code statement is then inserted into the user's program in the correct statement number sequence. The interpretor executes the user program by evaluating the P-Code instructions or atoms. The user program is always maintained in the P-Code format, including the disk image when saved. Listing is accomplished by "De-Compiling" the P-Code back into source statements.

The pseudo-machine the P-Code is generated for, is a high-level stack machine. It is essentially on the same format as basic statements, but more tightly packed. This accounts for the reason it is listable. Basically, there is one atom for each key word (directive, operator, function, variable, etc.). There are also atoms for control of the execution (start of option list, end of statement, end of argument, list etc.). Expressions (string and numeric) are stored in Reverse Polish Notation (RPN), so there is no need for atoms for paranthesis, etc.

The P-Code statement format follows the same lines as the basic statement. The statement starts with a one-word binary statement number. It is followed by the one byte directive atom. The options, if any, then follow. An option is made up of a value (often a one word statement number) followed by an option atom. After the options, come the arguments, if any. Arguments can either be simple values and variables or more complex expressions. Values are stored as a type atom followed by the value. Variables are stored as a variable-type atom, followed by the type code for the name. Expressions are stored as values and variables followed operator or function atoms, i.e. in standard RPN. Statements are terminated with an end of statement atom.

Here are some examples of statements and expressions in P-Code form:

```
BASIC SOURCE    : A1 + 1
P-CODE (OCTAL)  : 300        001   353           001   136
MEANING         : NUMERIC    A1    SINGLE BYTE   1     +
                  VARIABLE         CONSTANT
                  A-P

                : STR (Y9 + 3)

                : 301        211   353           003   136   200

                : NUMERIC    Y9    SINGLE BYTE   3     +     STR( )
                  VARIABLE         CONSTANT
                  Q-Z

                : 10 PRINT A

                : 0092       045   364      360      302       013   360

                : STATEMENT  PRINT END OF   START    NUMERIC   A     END
                  NUMBER     ATOM  OPTIONS  OF I/O   VARIABLE        OF
                                            LIST     (MAJOR)         STATE-
                                                     A-P             MENT

                : 100 LET A = A + 1

                : 000 144      111   302          013   300          013

                : STATEMENT #  LET   NUMERIC      A     NUMERIC      A
                               ATOM  VARIABLE           VARIABLE
                                     (MAJOR)            (MINOR)
                                     A-P                A-P

                      : 353          001   136   360

                      : SINGLE BYTE  1     +     END OF
                        CONSTANT                 STATEMENT
```

In processing a P-Code statement, the evaluator first saves the directive atom. Then, it processes the options, saving the values in the appropriate locations in the task information table. Arguments are processed in RPN fashion, leaving the resulting value for each on the arguments stack. Each function and operator routine processes a number of values on the stack, replacing them with its output. When the end of statement atom is reached, the routine for the directive is called, which processes the results of the arguments left on the stack. The options processed, control the directive routine and/or tell it how to handle exceptions.

The following illustrates the operation of the evaluator:

```
10 LET A = A + 1
```

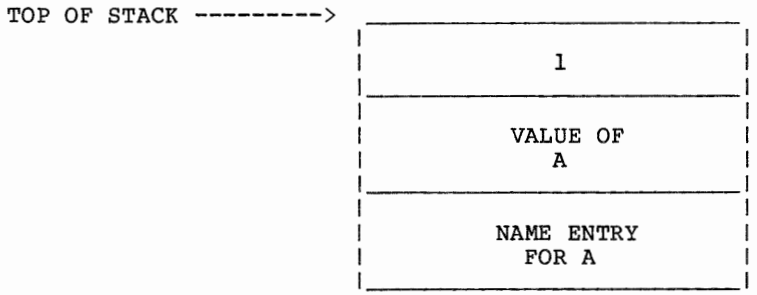
STEP 1: Saves the atom for the directive LET.

STEP 2: Reads in the atoms for the first variable A. Since it is on the left-hand side of the equal sign (major variable), an entry for its name is made on the stack.

STEP 3: Reads in the atoms for the second variable A. Since this one is one the right-hand side of the equal sign (minor variable), the variable get routine is called to get the current value of A. This value is then pushed onto the stack.

STEP 4: The atoms for the constant are read in and the value 1 is pushed on the stack.

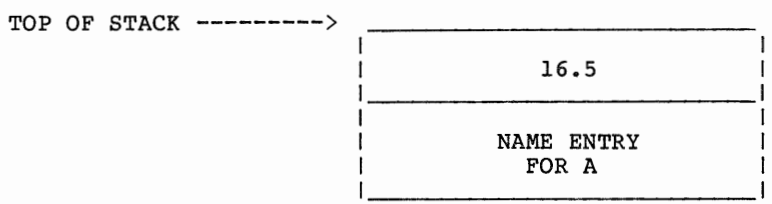
At this point, the stack looks like:



1
VALUE OF A
NAME ENTRY FOR A

STEP 5: The addition atom is read in. The evaluator then calls the addition routine which adds the two top of stack entries and replaces them with the result. Let us say that A was 15.5, so the result is 16.5.

At this point, the stack looks like:



16.5
NAME ENTRY FOR A

STEP 6: At this point, the evaluator reads in the end of statement atom. This causes the directive routine to be called. The variable put routine (LET) takes the value on top of the stack and assigns it to the named variable in the next entry.

STEP 7: The stack is then cleared and the next program statement is found and executed.

INTERACTION WITH RMS

To use RMS in an interpretive environment, several problems had to be dealt with.

RMS macros generate impure code. This means the actual execution macros had to be embedded in the user's space also. This was done by passing an execution control word in a register generated by the disk-driver module to the execution module in the user space. The execution module (RMSCOD) executes a specific macro for each bit turned on in the control word. The macros are executed in sequence as the bits are arranged. In the case of any RMS error, the original control word, the remaining to be executed control word, and the RMS STS and STV return values are posted to the task information table (TIT). Control is always returned to the disk-driver module, which checks for expected and unexpected errors, before returning control to the evaluation modules. RMSCOD always maps to the RMS Reslib with APR 5 and 6, then rewindows to the disk driver module when exiting. RMSCOD only needs three addresses to operate: task information table, current FAB (RMS file access block), and current RAB (record access block). The disk driver module uses the TIT, the current record buffer variable, and the current File Control Table (FCT) pointed to by words in the TIT. The FCT contains all information needed to process a file request: current key, next key, key size, current relative record number, a FAB, RABs, a key XAB and the private I/O buffers required for the file. Detailed information on these RMS control blocks can be found in various DEC RMS manuals. The execution control word passed to RMSCOD consist of Bit-0 through Bit-15, requesting the following RMS operations in order:

```
$INIT,$CREATE,$OPEN,$CONNECT,$FIND,$GET,$PUT
$DELETE,$UPDATE,$EXTEND,$REWIND,$ERASE,$FREE,
$FLUSH,$DISCONNECT,$CLOSE.
```

RMS pool and I/O buffers had to come out of the user's address space. If a buffer pool had been allocated permanently to handle a fixed number of files, it would have severely restricted the program space available. A compromise solution was chosen. Buffer pool for internal RMS control blocks was allocated permanently. I/O areas are allocated dynamically as files are opened, and de-allocated when files are closed. Recovery of de-allocated file areas could be handled by the variable space recovery routine, if I/O areas and RMS control blocks could be moved between operations. After considerable research, we found only three sets of pointers in the RMS control blocks that needed updating. Each set corresponds to each of the three RMS structures stored in the FCT; the FAB, RAB, and I/O Buffer. These pointers appear invariant over each type of RMS file Business BASIC deals with. (See Figure below.) The actual control block fields that require adjustment are:

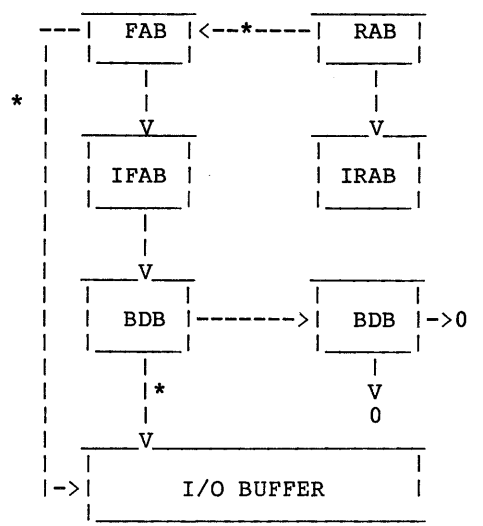
1. FAB (offset = 44 octal) in the RAB, which points to the FAB.

2. BPA (offset = 32 octal) in the FAB, which points to the I/O buffer.

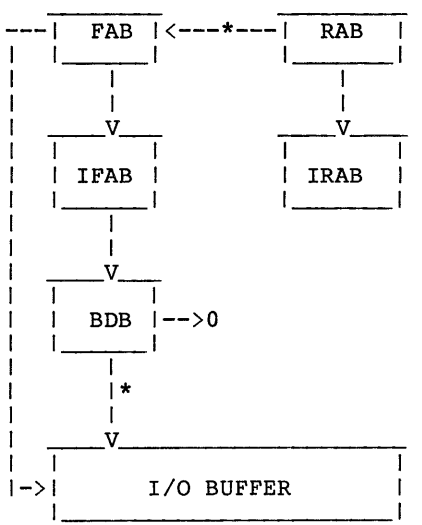
3. Offset 22 octal in each of the buffer descriptor blocks (BDB's) allocated for each I/O buffer (one for sequential and relative files, two for indexed). NOTE: extra BDB's are allocated for indexed and relative files for access stream sharing. Since this feature of RMS cannot be used under RSTS/E, the buffer addresses are zero and need not be adjusted. The BDB's are reached via the IFAB from the FAB.

For more information on RMS internals, refer to the RMS MACRO-11 manual and the section on internals in the Programming with RMS (Vol. 2) manual.

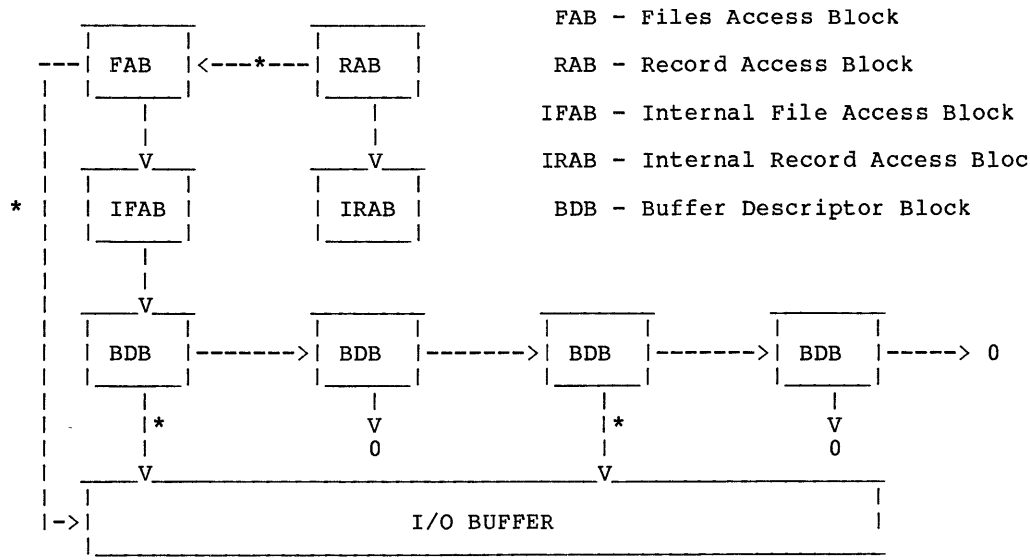
INDEXED (REL)



SERIAL (VAR SEQ) / STREAM ASCII



DIRECT/SORT (IDX)



FAB - Files Access Block

RAB - Record Access Block

IFAB - Internal File Access Block

IRAB - Internal Record Access Block

BDB - Buffer Descriptor Block

* ADJUSTED POINTER

Currently these are some restrictions when using existing RMS files with Business BASIC:

1. Access is by primary key only. An option for specifiying key-of-reference will be added in the near future.

2. The RECORD modifier must be used with I/O statements for files created by non Business BASIC programs. In these cases, the programmer must map variables to and from the record, instead of using the IOLIST statement.

3. Access to a keyed file (IDX) by relative record number is not allowed unless the file was created by a Business BASIC DIRECT or SORT statement.

Record formats are in two basic formats, depending upon whether or not the RECORD modifier was used when writing a record.

Records written with a list of variables by a statement such as WRITE (1,KEY=K$)A$,B$,C,D$ ..... will be written out using the current length of each variable, with left zeros suppressed for numeric variables. Each variable in the record will be followed with a special field separator consisting of a line-feed with the high bit on (octal 212). Any remaining bytes in the record are padded with nulls.

Records written with the RECORD option are written with one variable with a statement of the form WRITE RECORD (1,KEY=K$)R$. No field separator is written, and any remaining bytes are padded with nulls. Records containing binary data should normally be read and written with the RECORD option, as on octal 212 could occur in the binary data.

RECORD AND FILE LOCKING

Keyed and relative-record files are normally opened in write-share mode. If a LOCK is issued for a file, the file is closed and an attempt is made to open it in non-shared mode. An error is signaled if this fails. When normal reads are requested, a $FREE command is executed to release the currently locked blocks. When a record locking read is requested with the EXTRACT directive, the read is issued without the $FREE, leaving all blocks for the current bucket locked. A subsequent input on the same file will be preceded by a $FREE. All write or update operations are followed by a $FLUSH, which unlocks any locked blocks for the file. If a requested bucket is locked by another task, the requested operation is retried ten (10) times at one second intervals before any error exit is taken.

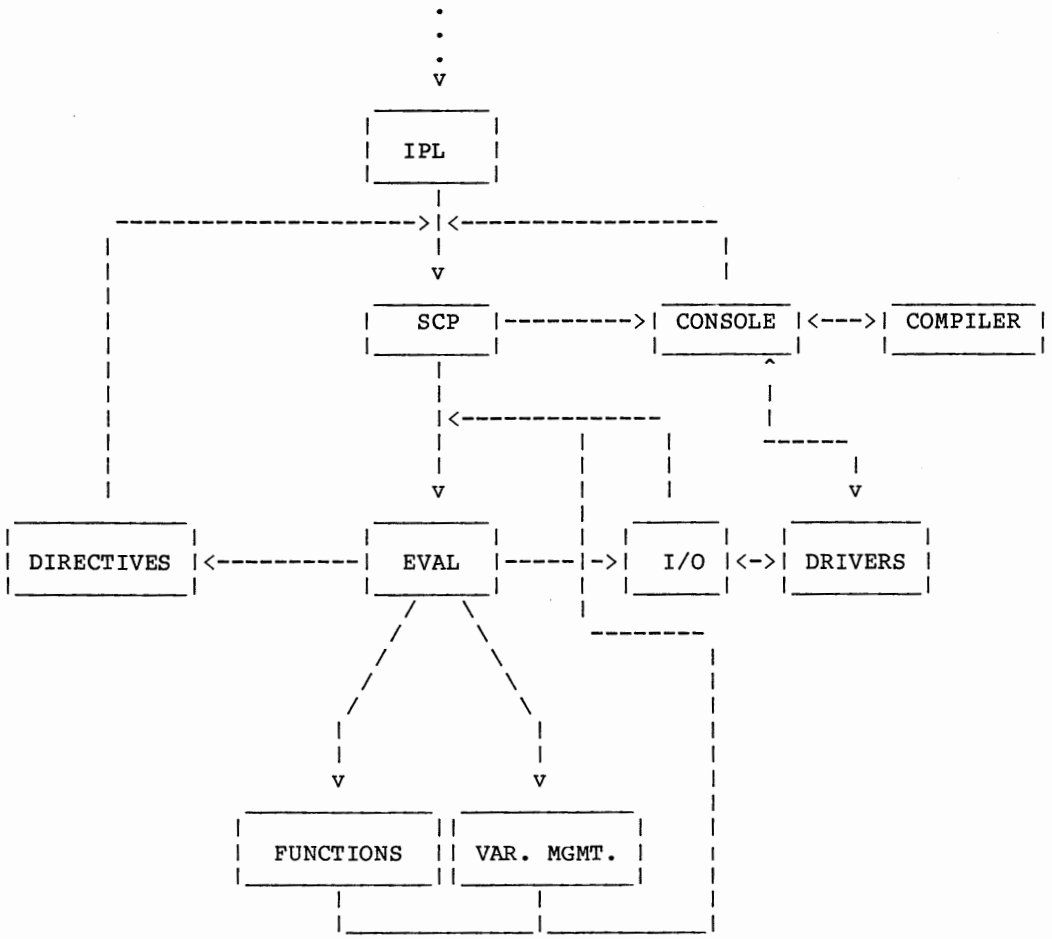
LOGICAL STRUCTURE

When entered, Business BASIC first executes an Initial Program Load (IPL) program. This program reads the configuration file and sets up the internal tables for processing. After initializing the stack, it calls the System Control Program (SCP). From the SCP, either a BASIC program may be executed or control can go to the Console mode program.

The Console mode program will accept statements from the keyboard, either to be entered into the program or to be executed immediately. The compiler is called to convert the statements into P-Code. If an immediate mode statement is entered, the control is returned to the SCP for execution. When completed with the statement, control is returned to the Console mode, unless the immediate statement was a RUN or CALL, in which case control remains in the execution section until an error, a program break point (ESCAPE) or a program END occurs.

The SCP controls the execution of a program. It first finds the next statement to execution and passes control to the statement evaluator (EVAL). EVAL processes the program as described in the section on P-Code execution. When end of statement is reached, the directive routine is executed and control returns to the SCP. If this was an immediate mode statement, the control is returned to Console mode, otherwise the next statement is found and executed.

LOGICAL FLOW OF BUSINESS BASIC



PHYSICAL SYSTEM STRUCTURE

Business Basic is composed of five highly interconnected modules. Together they provide all the functions required by RSTS/E for run-time systems and all the functions required for Business Basic. A sixth module, the RMS-11 resident library, is also accessed. See the RSTS/E RMS User's Guide and RMS Installation Guide for details on this library.

SMCBAS.RTS

SMCBAS.RTS is the run-time system module. It contains the routines to handle RSTS/E synchronous and asynchronous system traps (memory management violations, stack overflows, etc.), common subroutines used throughout the system, routines to handle running programs (non-Business Basic programs), routines to handle job creation, etc. For speed considerations the main statement eveluator for Business Basic is also located in the run-time system.

SMCRES.LIB

SMCRES.LIB is the main Business Basic library of routines. It contains most of the modules required to execute and maintain Business Basic programs. It is always resident in memory and is shared by all Business Basic users on a system.

SMCCPL.LIB

SMCCPL.LIB is the library of routines used to compile a BASIC source statement into the p-code that is executed. Like SMCRES.LIB, it is also resident and shared.

SMCLST.LIB

SMCLST.LIB is the library of routines used to list (decompile) a BASIC p-code statement back into source form. Like the other libraries, it is resident and shared.

CONSOL.EXC

CONSOL.EXC is a machine code program that is executed by the Business Basic when a user enters the system. It sets up the internal tables and variable area for the user. It also reads the IPL configuation file to set up the devices and accounts that will be used. CONSOL.EXC contains the buffers and routines needed by RMS for file processing. One copy of CONSOL.EXC is used by each user in Business Basic since it will contain the user's data and BASIC

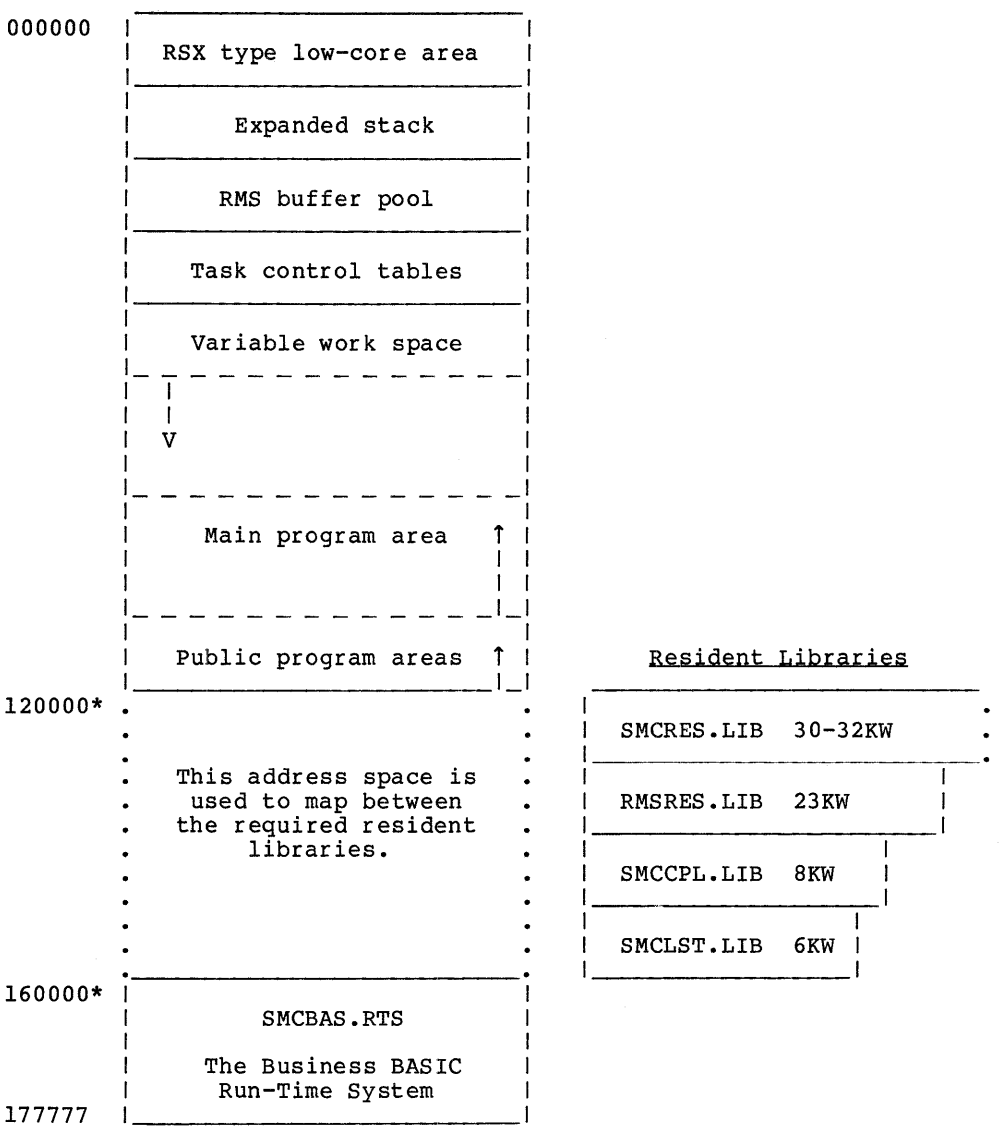
VIRTUAL ADDRESS SPACE

The following figure shows the layout of the virtual address space for each user running under Business Basic. The regions marked with an '*' are shared between all users.

The program and variable areas dynamically grow and shrink as needed. Variables are allocated on chains by letter and type (ie all Ax numerics are chained). Strings are allocated at the end of the variable space and annexed in, unless they can overlay the existing value for the variable. A compress routine is run when deleted space becomes excessive.

Public programs are added into memory on CALLs and ADDRs. They are placed at a higher address than the main program and the main program is moved up, if necessary. If memory becomes too tight to add a public program on a CALL and if the main program is larger, it will be overlaid by the called routine. It is re-loaded when the called routine exits. Nested called routines may also overlay each other.

VIRTUAL ADDRESS SPACE FIGURE



INTERNAL CALLING MECHANISM

When a module is called within the Business BASIC RTS, it is necessary for certain things to be done before entering it. These are:

- save the registers R0 - R7
- allocate workspace for the module
- map, if necessary, to the correct position in the correct resident library

The first two were accomplished by ordinary means. Registers were saved on the stack and workspace was allocated from the stack based upon a size located in the module itself. The third, however, presented a problem, since we were designing the system under RSTS/E V7.1 - no clustered library support. This meant that the calling mechanism was required to do its own mapping directly.

The calling mechanism was implemented by creating a table of all of the modules (MODTAB) that contained:

- the library the module was in
- the module's size
- the module's entry point in the library
- miscellaneous flags

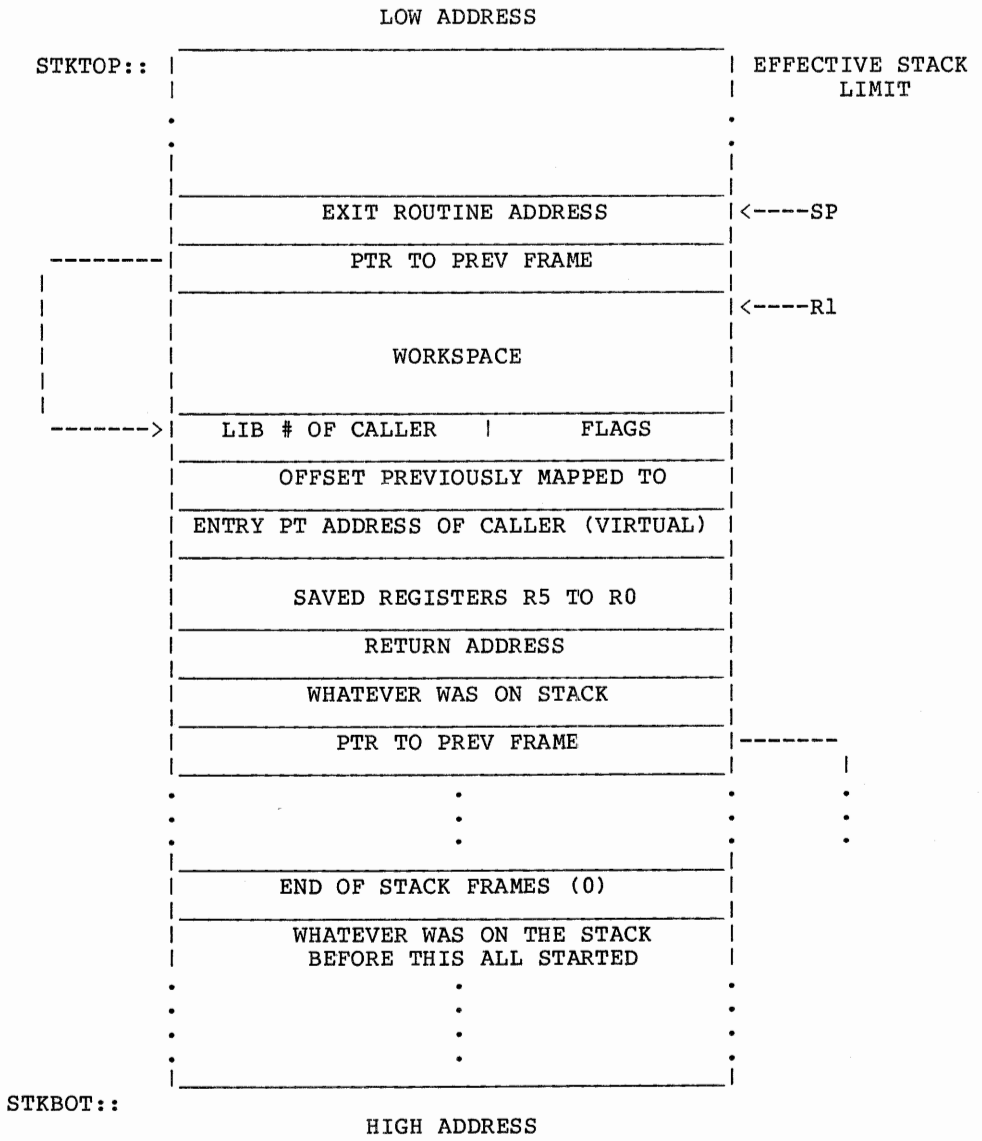
The caller then calls a common mapping routine with the MODTAB address for the module to be called. The routine then uses the table information with the current mapping information to determine whether or not it is necessary to remap. A map is done if:

- the module is not in the current library and not in the RTS
- the module does not start in the current window
- the module is not wholly contained in the window
- the module is flagged to always force a map

Whenever a map is done, the current mapping information is saved on the stack and updated. On exit, if the caller was mapped out, the exit routine remaps it back in from the information on the stack.

A special case occurs when RMSCOD is called, since it will destroy the current window when it maps RMSRES. A flag was set up in the MODTAB entry for a forced remap on exit. When RMSCOD is called, the mapping routine makes it look like it required a map. This way, exit will remap to the caller of RMSCOD even though RMSCOD is contained in the low-core program and no map was necessary.

The stack used in the calling routine looks like:



The advantages of using this method are:

1. Flexibility

   During development, we were able to add modules to the library without worrying about the ODL file and segment sizes.

2. Ease in Testing

   One of the problems in using resident libraries is testing, since it is difficult to set breakpoints in a routine not yet mapped. With this system, we were able to move the routine in question into low core, change its name and MODTAB entry, and re-TKB only CONSOL.EXC. Thus, the routine was no longer mapped and could be modified as it ran.

The disadvantage is:

1. Since RMSRES uses $AUTO with memory mapped overlays, it requires two 4KW windows. We have been only able to use the root window for our mapping, since $AUTO has set flags in segment descriptors for the overlay window. This slows the system down by restricting us to a 4KW window, requiring more frequent mapping. We are currently looking into either using $AUTO on V7.2 or greater system, or changing the flags in the segment descriptors to make $AUTO think RMSRES is no longer mapped.

DEBUGGING FEATURES

Business BASIC has all of the inherent debugging features of other interpreters and a few more. The more obvious are listed below.

CONTROL-C: Interrupts program execution. Prints out the last statement executed. Variables may be displayed and modified. Statements may be inserted, deleted or modified. The execution pointer may be changed. Processing is resumed with a RUN statement.

ESCAPE: This is the embedded breakpoint statement. All notes on CONTROL-C apply.

SETTRACE [(n)]: This statement causes an execution trace listing to be sent to an open device or file on channel n. As a statement is executed, it is de-compiled, and listed in original source format. SETTRACE can be embedded anywhere in a program or used in console mode. SETTRACE is terminated when an ENDTRACE is executed.

DATA REPRESENTATION

Data handled in BASIC occurs in the form of either numeric or alpha-numeric string data. Numeric variables or constants have a range in standard and floating form of:

```
     99999999999999 to -99999999999999
    .99999999999999 E + 141 to -.99999999999999 E - 113
```

String variables or constants can contain any 8-bit character.

Numeric variable names are A through Z9. Numeric array names are kept in a separate variable chain and can have the same name as non-array variables.

String variable names are A$ through Z9$. Substrings are defined as string-var (offset,length). For example, if A$="ABCDEFGH" then A$(4,3) is equal to "DEF". If no length is specified, the substring is evaluated to the end of the parent string. That is, for the above A$, A$(6) is equal to "FGH".

ARITHMETIC OPERATORS	RELATIONAL OPERATORS
- minus	= is set equal to
+ plus	> is greater than
* multiplied by	< is less than
/ divided by	<> or >< is not equal to
^ raised to the power of	>= or => is greater than or equal to
** raised to the power of	<= or =< is less than or equal to

STATEMENT DESCRIPTIONS

ADD	directory cache feature.
ADDR	add a public program and keep resident.
BEGIN	reset all variables, stacks; close all files.
CALL	Call a public program and return in line.
CLOSE	Close a file or device.
DEF	Define a function.
DELETE	Delete statements from the program.
DIM	dimension an array.
DIRECT	Create an RMS keyed (IDX) file.
DISABLE	disable a user disk (logical) to change directory search order.
DROP	Drop a previously ADDRed public program from resident status.
EDIT	Edit a BASIC statement.

ENABLE	Re-establish a logical disk for directory search.
END	Terminate a program, reset control variables, close all files, set execution pointer to head of program. Terminate MERGE.
ENDTRACE	Terminate trace listing.
ENTER	Define parameters passed to a public program.
ERASE	Erase a file if not in use.
ESCAPE	Breakpoint. Switch to console mode.
EXECUTE	Compile and execute a string as a BASIC statement.
EXIT	Define exit point and return code for a public program.
EXITTO	Exit from subroutine or FOR/NEXT loop and clear one entry from return stack.
EXTRACT	Read and lock a record.
FIND	Read a record without updating the next key pointer if not found.
FILE	Create a file using string created by FID function.
FLOATING POINT	Sets PRECISION to 14, except for output and allows calculations with larger numbers in scientific notation.
FOR	Estalish control variable, limits and STEP value for FOR/NEXT loop.
GOSUB	Perform a subroutine.
GOTO	Change execution pointer to specific statement number.
IF/THEN/ELSE	Perform statement based upon truth of a logical expression.
INDEXED	Create an RMS relative-record file.
INPUT	Position cursor, output prompt, input and verify data.
IOLIST	Define a variable list for use in an input/output statement.
LET	assign a value to a variable.
LIST	List de-compiled program statements to a file or device.
LOAD	Obtain a program from disk and set execution pointer to beginning of program.
LOCK	Obtain non-shared write access to an open file.
MERGE	Merge list format program statements into current program from a file or device.
NEXT	Increment control variable specified in FOR.
ON GOSUB	Computed GOSUB.
ON GOTO	Computed GOTO.
OPEN	Open a file or device.
PRECISION	Set results precision for all calculation to a number of decimal places.
PRINT	Output to printer or terminal.
PROGRAM	Define a new program file on disk.
READ	Read a record or variable(s).
RELEASE	Perform END and release task memory (Switch to default RTS).
REM	Remark or comment.
REMOVE	Delete a record from an RMS keyed file.

RESET	Reset system control variables and clear return stack.
RUN	Resume processing from console mode, or LOAD and execute a program from disk.
SAVE	Save the current program to disk.
SERIAL	Create a variable length record, sequential, RMS file.
SETDAY	Change the system date.
SETERR	Force all default error branches to a particular statement number.
SETESC	Set Control-C trap to a particular statement number.
SETTIME	Set system time.
SETTRACE	De-compile and list each executed statement to a file or device. Continue until BEGIN, END or ENDTRACE encountered.
START	Adjust size of user task, and RUN specified program.
SORT	Create a special format of RMS keyed file.
STOP	Perform END, except do not terminate MERGE.
TABLE	Define a mask and conversion table for input or output data.
UNLOCK	Reset file access to shared mode.
WAIT	Wait or sleep for specified seconds.
WRITE	Write a record or variable(s).

INPUT/OUTPUT OPTIONS

I/O options are used to augment the execution of an I/O directive. I/O options are specified within the parentheses immediately following the file number (channel). They can cause branching, set up control to override defaults, specify a record to access, and more.

BLK=expr:	Use to specify buffer size for STREAM ASCII files.
DOM=stno:	Duplicate or missing key. Control is tranferred to the specified statement if record is not found on an input or REMOVE operation, or if the key specified in a WRITE operation is already in the file. If DOM= is not used on a WRITE, and the key already exists, the record is updated.
END=stno:	At end of file, transfer control to specified statement.
ERR=stno:	Transfer control on any error to specified statement.
IND=expr:	Specify relative record number (index) to be accessed.

ISZ=expr:	Temporarily redefine (at OPEN) the record size for a file and set type to relative.
KEY=string expr:	Specify the key of a record to be accessed.
LEN=min,max:	Specify the minimum and maximum length of an input variable. Generates error 48 if variable is out of range.
SEQ=expr:	Used to specify relative file number to open on tape.
SIZ=expr:	Specify maximum characters to be read during an INPUT or statement. Input is terminated if the maximum characters are entered, even if a terminator (CR) was not entered.
TBL=stno:	Specify the statement number of a translation table to be applied to data. See TABLE directive.
IOL-stno:	Specify the number of an IOLIST statement.

FUNCTIONS

Functions are used to manipulate data. They perform a variety of operations, such as converting characters to different forms, checking for data integrity, returning file information, converting from strings to numerics and vice versa, and more. In addition to these pre-defined functions which are a part of Business BASIC, 52 user-defined functions are available for each program through the DEF FNx directive.

ABS (expr [,ERR=stno]) Returns the absolute value of a numeric expressions.

AND (string expr,string expr [ERR=stno]) Returns the string result of the Boolean multiplication of two strings.

ASC (string expr [,ERR=stno]) Return the first byte of a string as a decimal number.

ATH (string expr [,ERR=stno]) Returns an ASCII string from conversion of a hexadecimal string. The source string should contain only the characters (0 - 9) and (A - F).

BIN (expr,length [,ERR=stno]) Returns a string containing the binary value of a numeric expression. The result string is either truncated or padded with nulls on the left to achieve the specified length.

CHR (expr [,ERR=stno]) Returns an ASCII character representing the numeric expression whose value is between 0 and 255.

CPL (string expr [,ERR=stno]) Compiles the string containing a valid BASIC statement into a result string that can be executed. Used in program generators and utility programs.

CRC (string expr) Returns a two-byte string that is a 16 bit check sum of the specified string. Can be used with binary synchronous communication protocol by swapping the result bytes.

DEC (string expr [,ERR=stno]) Returns a signed decimal number by converting the binary string expression.

EPT (expr [,ERR=stno]) Returns the exponent of a numeric expression.

FID (fileno [,ERR=]) Returns a string containing detail information about a specified file number. See the reference manual for full details. The result string can be used in a FILE directive to re-create an empty file after it has been erased.

FNx [$](argument list) Returns the result of a programmer-defined function defined with the DEF directive. x is a letter from A - Z. $ is used if the function returns a string.

FPT (expr [,ERR=stno]) Returns the fractional part of a numeric expression, rounded to the current PRECISION.

GAP (string variable or literal) Generates an odd parity copy of the source string, ignoring the high order bit of each byte in the source string.

HSH (string expr) Generates a two-byte hash total of the source string. Used to verify program integrity during LOAD.

HTA (string expr [,ERR=stno]) Returns a string containing the hexadecimal representation in ASCII of the source string. Each byte of the source string is represented by two bytes in the result string.

IND (fileno [,ERR=stno][,END=stno]) Returns the index or relative record number of the next record to be accessed on the specified file.

INT (expr [,ERR=stno]) Returns the integer part of the numeric expression. Any fractional digits are removed, and rounding does not occur.

IOR (string expr, string expr [,ERR=stno]) The inclusive or function returns a string that is the result of combining the bits in the specified strings. When a particular bit position in either source string is a one, that bit position in the result string is set to a one.

KEY (fileno [,ERR=stno][,END=stno][,IND=recno]) Returns a string containing the KEY of the next logical record to be accessed from the file.

LEN (string expr [,ERR=stno]) Returns the length of the string, including any non-printable or fill characters.

LRC (string expr) Returns a one-byte string equivalent to XORing of all bytes of the argument string. If the length of the source string is currently 0, a binary zero is returned.

LST (string expr [,ERR=stno]) Converts a compiled BASIC statement into LIST format.

MOD (expr-b, expr-b [,ERR=]) Returns the remainder of the integer division of expr-a divided by expr-b.

NOT (string expr) Returns a string that is the result of inverting each bit of the source string.

NUM (string expr [,ERR=stno]) Returns the numeric value of the characters in the string. Legal characters in the string are 0-9 + - , . and E.

PGM (stno) Returns the compiled format of the specified statement number. If the specific statement does not exist, the next higher one is used.

POS (scan-string relational-operator target-string [,step value] [,occurance][,ERR=stno]) Where scan-string is the string being searched for, relational operator is a valid comparison symbol, target-string is the string to be searched, step value is the incremental position at which the target-string is examined for each subsequent comparison, occurance is the occurance of satisfying the scan. POS returns a number indicating the byte position in the target-string where the specified parameters were satisfied. A zero is returned if no substring is found that meets the requirements.

PUB (0) Returns a string representing all public programs resident in this task area. Sixteen (16) bytes per program are used to specify location, length, name, type, and disk number.

SGN (expr [,ERR=stno]) Returns -1, 0, or 1 depending upon the sign of the numeric expression.

STR (expr [:mask][,ERR=stno]) Converts a numeric expression to a string using an optional edit/format mask. For example: A=100.236;X$="$##0.00"; A$=STR(A:X$);PRINT A$ $100.24 assuming PRECISION=2.

XOR (string expr, string expr) Returns a string that is a combination of the source strings such that a zero-bit is returned where source bits are equal, and a one-bit is returned where source bits are unequal.

TERMINAL MNEMONICS

The following mnemonics can be used virtually anywhere in an INPUT or PRINT statement:

MNEMONIC	EFFECT
@(x)	Position cursor or print position horizontally on same row.
@(x,y)	Position cursor at column x, row y.
'BE'	Begin echo if echo is currently off.
'BI'	Begin input transparency. Passes all data through the terminal driver without interception.
'BO'	Begin output transparency. Passes all data and control sequences through to target device (except for 'EO').
'BS'	Moves cursor back one space, erasing prev. character.
'BT'	Begins type ahead if previously cancelled with 'ET'.
'CE'	Clears screen from current cursor position to end of screen.
'CF'	Clear foreground. On VT131 clears bold to spaces. See also 'SB' and 'SF'.
'CH'	Cursor Home. Set cursor to (0,0) and sets foreground mode.
'CI'	Clear input. Clears all data currently in the input buffers for this terminal.
'CL'	Clear line. Clears to blank from cursor to end of line.
'CR'	Carriage return, plus line feed on terminal.
'CS'	Clear screen, including protected areas.
'DC'	Delete character. Deletes the character at the current cursor position, and shifts the remainder of the line one to the left.
'EE'	End Echo. Ends echo on channel Ø.
'EI'	Ends input transparency. See also 'BI'.
'EO'	Ends output transparency. See also 'BO'.

'EP' Expanded print. Prints double high characters on some terminals.

'ES' Escape. Sends an ESC character to the device. Used as lead in code for control sequences. Device dependent.

'ET' End type-ahead. Cancels input buffering.

'IC' Insert character at cursor position and shifts the balance of the line one to the right. Sets mode to foreground. See 'SF'.

'LD' Line delete. Removes the line at the cursor position, rolls all lines below it up one line, inserts a blank line at the bottom of the screen, and sets the mode to foreground. See 'SF'.

'LI' Line insert. Inserts a blank line at the cursor position, rolls all lines below it down one line, deletes the bottom line, and sets foreground mode. See 'SF'.

'PE' Protection end. Cancels 'PS' protection mode.

'PG' Page mode. Sends screen data to local printer port from (0,0) to cursor position. (VT132)

'PS' Protect start. Begins protected display and stops scrolling.

'RB' Ring Bell. Causes beep on terminal.

'RC' Read Cursor. Provide current cursor position. Used with or followed by an INPUT directive.

'SB' Start background mode. Sets normal intensity on DEC terminals, half-intensity on others. Marks background characters as protectable, but does not begin protection.

'SF' Start foreground mode. Sets bold on DEC terminals, normal intensity on others. Data printed or entered in this mode can be cleared with the 'CF' without clearing data printed in background mode. 'SF', 'SB', 'CF' are used to design "fill in the blank" type of screens for repetitive data entry or display.

'TL' Transmit line. The line of screen data at the current cursor position is transmitted to the next input variable.

'VT' Vertical Tab. Used to support printing of reports to the screen instead of a printer.

SYSTEM VARIABLES

CTL indicates the last field terminator input, as follows:

VALUE	LAST TERMINATOR INPUT	
0	CR	carriage return
1	PF-1	Program function key 1
2	PF-2	Program function key 2
3	PF-3	Program function key 3
4	PF-4	Program function key 4

DAY contains the system date in the form MM/DD/YR.

DSZ returns the number of available bytes in the task area.

ERR contains the occurring error number.

PSZ returns task information: TCB(3) returns last system error code, TCB(4) returns the current statement number, TCB(5) returns the statement number to retry, TCB(6) returns the statement number that SETESC is currently set to, TCB(7) returns the statement number that SETERR is currently set to.

TIM returns the current system time.

TSK(0) returns devices configures for task.

RSTS System Performance Optimization

Michael Mayfield
Northwest Digital Software, Inc.
Newport, WA

Ed McKay, Session Chairperson
Galveston College
Galveston, TX

Reported by Todd Spangler, DECUS Scribe Service

System optimization is always important in the business world. Michael Mayfield of Northwest Digital Software, Inc. presented several ways in which a system can be optimized.

In system optimization, the ideal is to maximize all features with respect to the others. This might sound a bit awkward, but at no time will all of the features work at 100% operating capacity with no waste. Some goals to achieve are:

1. Less than 5% CPU idle time (time that the CPU is not in use).
2. 0% of CPU time lost due to insufficient memory.
3. More than 66% CPU time used for user jobs.
4. Less than 15% monitor overhead charged to a job.
5. Less than 55% monitor overhead not charged to a job.
6. Minimum number of characters output to the terminal without affecting user performance significantly.
7. Less than 10% amount of time FIP is in use or waiting.
8. Less than 7% amount of time FIP is in use.
9. Less than 15% monitor overhead for interrupt processing.
10. More than 15% of small buffers not in use.
11. Less than 80% of maximum number of accesses per second for disk type.
12. Less than 30% of maximum transfer rate per second for disk type.

When optimizing the system, it is best to have a low monitor overhead time when compared to the actual user time involved, as little swapping as possible (a little swapping is not bad), and reasonable memory management. Disk access is also important. In order to optimize the disk, one needs to decrease the usage of the disk and also the amount of seek distance that the head must travel. To do this there are disk kits available in which the most frequently accessed files are grouped together in one area on the disk. Secondly, the files can be grouped further on the disk with respect to the number of times that the file is actually opened and accessed the most. In other words, group together according to UFDs. The UFDs can also be cut down by maximizing the cluster size. A formula for this is:

CLUSTERSIZE= -(2INT(LOG(FILESIZE/7)/LOG(2) + .9999))

The cluster size should be adjusted to -256 if the formula yields a number equal to or smaller than -256. Also one should minimize the overhead related to clustersize during random I/O by using proper cluster sizes. Cache hit ratio verses cache age also is important because if a cache buffer is required and the proper data is not there, then there is considerable time lost during the search for the proper data. Cache size relates to the exact same principle, since if the data is not there then the system must search for the proper data. FIP is crucial. The most time consuming process is FIP. In order to optimize this, one can optimize directory structures, create contiguous files, and maximize clusters. This will make a drastic improvement in the system response. Small buffers needed by the system can cause an interesting problem. If there are less than 25% of free small buffers available, the system goes into a first level panic. This means that the number of small buffers allocated to separate devices is limited and can cause allocation to be denied to the device. If the percentage of free small buffers goes below 20%, then a second level panic exists. At this time, many more devices will be denied access to the small buffers. If there is less than 10% of small buffers free, the system halts. During each of these phases, the run time decreases, the CPU time optimization is good, overhead time increases, so overall it is wise to maintain enough small buffers.

In order to find the statistics for an individual system, use the RPM (RSTS Performance Moniter). For individual systems, needs will be different, therefore the system optimization will require that different things be adjusted. When proceeding with optimization, change only one thing at a time since changing more than one thing may result in unreadable results or two good things may cancel each other out giving misinformation. The final analysis will be based on the system needs.

For further information, contact:

Michael Mayfield Northwest Digital Software, Inc. Box 2-743 Spring Valley Road Newport, WA 99156

New Users of RSTS/E Hints & Tips

Carl B. Marbach and Dave Mallery
RSTS Professional Magazine
Fort Washington, PA

Thomas W. Robbins, Session Chairperson
Seattle Pacific University
Seattle, WA

Reported by Susan Miller, DECUS Scribe Service

Are you a RSTS Guru? If so, you probably know of the Carl and Dave show. Carl B. Marbach and Dave Mallery of the RSTS Professional Magazine brought their show to the DECUS Symposium. Their presentation was "New Users of RSTS/E-Hints and Tips."

Marbach began with an editorial on the restriction of the exhibitors at the symposium. "Commercialism is not allowed at DECUS. You are not allowed to have a hospitality suite. I am not allowed to sell you things on the DECUS site. I am not allowed to advertise my product and say that you can find me at DECUS. . . . Only certain exhibitors were invited to be here. That is to say only certain exhibitors were given an exception from the rule. Selective application of the law is the most dangerous thing that can happen in any user community." The audience applauded, and no one accepted his offer for rebuttal.

The audience received a handout, entitled "New User's Manual For RSTS/E." The handout explained some of the hardware of the computer such as:

> UNIBUS--"The UNIBUS (DEC trademark) forms the backbone of any PDP-11 computer system. It constitutes a 56 wire party line on which any device of the system can talk to any other device. The ribbon cables interconnect each device from one to the next. Inside each device, the UNIBUS takes the form of a backplane into which the various circuit boards are plugged. Eighteen of the bus lines carry address information; another 16 lines are for the data. The rest of the lines are used for synchronization, handshaking, interrupts, and initialization signals."
>
> CONTROLLERS--"The controllers are devices which exist between a physical device (what you see) and the interface inside the computer. They are found on mass storage

devices that need lots of preprocessing of data. For instance, a disk drive reads and writes bits on your disk pack, but the controller groups them into words, counts the words as they are moved, checks the parity, and even corrects the data if it can. It also contains all the control and status registers that you see in an ERRDIS printout. Controllers are also called formatters. Simpler devices, such as terminal interfaces, have the registers right on the interface card and do not process data.

Marbach then gave some information on different processors offered to users of the PDP-11. The terms were explained in their numerical order. Marbach has not found any logical reason for the names of the terminology. But he did list some differences:

1. The 11/20 is the first PDP-11 and also the first one to support time sharing.

2. The Micro-11 supports up to 4 megabytes of memory. It uses a Q-Bus system.

3. The 11/23 uses Q-Bus and is packaged a little differently so that additional disks can be attached.

4. The 11/24 is functionally similar to 11/23 but uses a UNIBUS system.

5. The 11/34 is a UNIBUS which only does 18-byte addressing.

6. The 11/44 has cache memory, which makes the memory operate faster, and supports up to four megabytes of memory.

7. The 11/45 is a fast processor and limited in memory.

8. The 11/60 is also limited in memory but allows a user to write her own instructions. This is good for scientific but not commercial use.

9. The 11/70 has 22-bit addressing. It has UNIBUS and MASSBUS, which puts it through faster.

Performance information can be obtained on the RSTS. Marbach suggests contacting your local salesman and asking for the Performance Handbook. "If he doesn't know what a performance handbook is, tell him to call the home office and ask for Al Saloky." He's responsible for most of the performance monitoring. His group provides documented information on performance of all Digital processors.

Hints were also given to configure a computer system. Try to establish what you're buying. How many people will use it?

Decide what kind of processing and memory you need. And configure statistics information into your system.

The need to have a backup was strongly urged. The backup should be kept in a different place from the work area. If the building burned, everything would be gone. Mallery had his backup stored in the back seat of a car until he could afford a vault in a fireproof warehouse. Usually backups are needed most because of a human's error, such as a wrong key stroke that happens in a fraction of a second. They recommended having a backup of the previous four days work. Then a weekly backup. And of course a monthly backup for as many months or years that you can keep it.

Mallery gave tips on how to structure disks. This is the single biggest factor in the RSTS system. The user file directory, or UFD, should be contiguous, centered and about one-third of the way on the disk.

The most amusing hint and who knows--maybe the most useful--was to never use a newly released product until the Patch Kit B has arrived.

New Frontiers in Training Technology

Del Lippert
Digital Equipment Corporation
Bedford, MA

Adam Zavitski, Session Chairperson
Seismongraph Service Corporation
Tulsa, OK

Reported by J. Rick Mihalevich, DECUS Scribe Service

Del Lippert of DIGITAL stated that the next ten years promise to be a real challenge both in education and in industry. Rapid technological change and the increasing intensity of foreign competition will test our businesses. If we react boldly, we can provide the kind of training and retraining environments that will keep our organizations and our employees competitive.

He went on to describe some of the factors which are leading us to choose educational technology as part of our training solution in industry. He also described how DIGITAL applies educational technology in an industrial training environment, since their training needs are similar to those of many industrial organizations. Finally, the he discussed some exciting new developments in educational technology.

Why do we need to apply technological solutions to our training needs in industry? The answer to this begins with one word: change. As a recent report form the Office of Technology Assessment of the U.S. Congress states,

"A key element in all of these educational needs is that they will constantly change. In a rapidly advancing technological society, it is unlikely that the skills and information base needed for initial employment will be those needed for the same job a few years later."

The people hired today will not be performing the same job in five years, or even two years in many cases. The problem is that the working lifetime of our employees is now very much greater than the development and support lifetimes of new technologies.

This means that most employees will need to be retrained

more than once during their careers in order to remain productive. For example, journeyman electricians in the automobile industry must now be retrained in electronics, robotics, and computers. The old model of work in a trade or profession is a long apprenticeship leading to an indefinite and stable journeyman status. This model based on the medieval guilds is still accepted by many people today. Change is breeding change and we can no longer expect that the obsolescence of skills facing our country today will be a unique event.

Right now a typical employee can look forward to a career that consists of cycles with alternating periods of training and productive contribution. It used to be that the productive period took up most of a person's working life. Childhood schooling and sometimes college were sufficient preparation for a lifetime of work; this is no longer the case.

The typical engineering graduate in this country is 22-23 years old. Engineers themselves state that they face their greatest challenge around age 27, and they categorize themselves at technologically obsolete by age 32. That is an interesting number, because in 1982 the median age of the United States' population was 32. Compare this with the situation projected for 1990, when the median age will be 40.

The challenge facing us is to keep those who will soon amount to most of our workforce productive in a time of great technological change. This is more than a problem of higher management. While good technical people can overcome poor management, the reverse is not generally true.

Compounding this problem is the fact that many of the qualified applicants to engineering schools were rejected in 1980 due to lack of capacity in these schools. Because it will become increasingly difficult during the next decade to find qualified engineering graduates, ongoing retraining must be maintained as a source of productive technical work force.

The concept of a learning cycle that keeps people both challenged and productive is one answer to these problems. The learning cycle begins with formal schooling, perhaps college, and sometimes an advanced degree, all provided by traditional academic institutions. The cycle is maintained by continued retraining to meet both job needs and the need for personal growth. This continually changing part of the learning cycle is increasingly the task of industry itself or of industry in partnership with educational institutions.

The reliance on the learning cycle is based partly on the faith in the adaptability of adults and partly on recent psychological studies. Adults, like children, continue their intellectual development beyond the age of formal schooling, and that people's receptiveness to change is very intense at several times during adulthood. If we can take advantage of this aspect

of human adaptability we will be meeting our own human as well as business needs.

What employers can look forward to from the learning cycle is continued productivity and protection of their investment. The investment of employees from age 22 to age 65 is an investment of between four and five million dollars to a company. The reason that educational technology is being considered as a way to implement the learning cycle concept is simply, "How do we protect and maintain that investment as an asset to the company in the most cost-effective way."

It has been estimated that corporations already spend over $30 billion dollars per year on training, and employ 75,000 people full-time and another 75,000 part-time. The importance of that level of investment is clear and its efficiency will have to increased to meet projected demands.

To be more specific, the following are the five changes now occurring in industry that are relevant to the presentation:

1. The amount and complexity of job-specific training needed to work in modern industry has increased.

2. The need for continual upgrading of work skills has increased.

3. The number of people per year needing training has increased.

4. The geographical dispersion of a modern company's workforce has increased.

5. The cost of delivering training via lectures is becoming prohibitive.

None of these changes alone points to a need for educational technology. But taken together, they argue strongly for decentralized training that is individualized and can meet the continuing needs of adult learners.

One solution to achieving decentralization is to employ the power of today's educational technologies. Experience at DIGITAL has shown that these technologies can be used to deliver instruction cost effectively at a large number of widely dispersed locations, even if only a few people need to be trained at each location.

The issues forced on us by the changing nature of American industry are sufficiently urgent, and the use of educational technology sufficiently promising, that industry will be in the forefront of the widespread use of educational technology. Industry has a unique combination of resources and motivation to apply technological solutions to teaching on a large scale. Our

traditional close partnership with colleges and universities will be of value to both higher education and industry as we continue to make great strides in the application of technology to education.

Industry is in the midst of decentralizing its workforce while traditional educational institutions remain largely centralized. A major difference today is that industry itself must provide much of the training to its employees because of increased specialization and technological change. And the median age of the population is no longer 16, but rather 32. This means that educational changes that influence large percentages of the population will have to occur beyond the years of traditional schooling.

Thus industry is in a position of needing a decentralized approach to training and also having to provide that approach itself. In many cases education and industry together can cooperatively address this lifelong learning challenge. Opportunities exist for using the resources of both institutions to address one of our major challenges - employee obsolescence. DIGITAL trains people which have diverse backgrounds and learning goals. Some join DIGITAL with significant experience while others have no job-specific experience. Some study courses to obtain detailed job skills, while others look only for generalities. Adaptive training programs should handle each learner slightly differently to maximize each person's performance against their own unique goals. In addition to traditional classroom instruction and self-paced printed materials, DIGITAL presently use three classes of technology to achieve their training goals: linear video, computer assisted instruction, and interactive video.

Linear video is a very useful and easily decentralized training technology. It is helpful in sales training, in pre-sales customer training, and in conveying general concepts. This type of media presentation is supplemented with printed material. However, the type of presentation is limited and the level of interaction between the student and the technology is low. In many training situations we need a way to deliver decentralized training that is more adaptive.

Much of this adaptability can be provided by computer assisted instruction, which can be highly interactive and tailored to a student's individual needs. In particular, a student using computer using computer assisted instruction can:

1. Receive instruction at a time convenient for the student.

2. Control the pace and order of the instruction.

3. Ask for help, advise or the definition of an unfamiliar term.

4. Get feedback on what parts of a course to review.

Computer Assisted Instruction has an important role in a technical training environment although with some limitations. The graphic quality of today's computer assisted instruction is excellent, and for many training applications this is sufficient. However, standard computer assisted instruction technology cannot reproduce photographs or live action sequences with the clarity of video presentation.

Much of the material presented at DIGITAL requires video quality images. To satisfy this need while maintaining the level of adaptability provided by computer assisted instruction DIGITAL has begun to produce courses that take advantage of their own interactive video technology.

DIGITAL has developed an interactive video system that combines the best qualities of both computer generated and video displays. This system fully integrates video and computer assisted instruction by allowing both images to be displayed on the same screen. In addition to allowing two separate "windows" into the course, this allows video images to be overlaid with computer generated graphics and text. When the video source used is a videodisc player capable of randomly accessing 54,000 separate images the adaptability of this system far exceeds that of either linear video or computer assisted instruction on its own.

A student might be introduced to a new piece of equipment with a video sequence. If later evaluation shows that the student does not understand how a particular component works, part of the video sequence. If later evaluation shows that the student does not understand how a particular component works, part of the video sequence might be repeated with the component highlighted via computer graphics overlays. Or a student can be taught the assembly and disassembly of a complicated piece of equipment by allowing them to simulate the steps interactively, thus eliminating some of the need to maintain costly equipment at training sites. These techniques and many other variations allow interactive video systems to be extremely adaptable to a large number of student backgrounds, abilities, and learning styles.

An example of the advantages of decentralized training using the adaptability of interactive video system is illustrated by one of DIGITAL's major groups. The total cost of training for this group will rise by over 40% during the next five years if that training is delivered in the traditional way, by lectures and self paced courses at training centers. These costs include not only the cost of producing courses but also the travel and time costs associated with training centers that are remote from many work sites. Using a model where much of this training is provided by interactive video systems installed in the group's field offices, almost all of that increased

training cost can be saved. This results is consistent with the general trend toward higher costs for traditional educational delivery, which is labor intensive, versus the use of educational technology, which continues to benefit from decreased costs, especially hardware.

In addition to the cost savings, studies at DIGITAL of the initial use of interactive video courses have shown that properly designed interactive video courses can be as effective as lectures or self-paced courses for specific kinds of training. We have also found that the interactive video courses are excellent reference sources that allow employees to refresh their skills in particular areas just when they will be needed.

With all the benefits there are some limitations. The primary limitation with the creation of new hardware technologies the limited availability of software support. Many of the problems found with using educational technology are raised by the technology itself. In fact, educational creativity is outpacing the ability to implement instructional designs that are exceedingly complex. One of the promising developments in this regard is the use of artificial intelligence techniques in educational technology. Resent developments in both hardware and software have now made artificial intelligence techniques practical as an instructional tool. The promise of using artificial intelligence is that teaching systems can be built with all the technical knowledge of subject matter experts and all the teaching knowledge of psychologists and educators.

In summary, the major problem facing industry is change; employees last longer than the need for their present skills. The concept of the learning cycle in which a person's working lifetime consists of alternating periods of productive work and retraining, using some examples of specific changes in industry that required training to be decentralized and individualized. This combination of requirements makes educational technology a viable solution to training problems. Industry has a significant need to adopt educational technology on a large scale. The traditionally close partnership between industry and education in this country will be of great importance.

Maxi In Your Mini

Christopher R. Johnson
North Shore Sanitary District
Gurnee, Illinois

ABSTRACT

Does your program require RMS indexed file support? Does your boss insist that you include Resident Libraries? Does the program specification call for more than five files to be open at once? Then the system is probably telling you that the maximum memory has been exceeded. This paper will give the inexperienced programmer insight into some of the techniques that can be used to permit the necessary files to be open, and processing to be completed. Included in this discussion will be techniques such as program segmentation, the efficient use of maps, and overlay structures. When these methods are implemented, even the largest programs will be able to execute.

INTRODUCTION

Does your program require RMS indexed file support? Does your boss insist that you include Resident libraries? Does the program specification call for more than five files to be open at once? If you answered "yes" to all these questions then it is conceivable you could have a problem. Your system may be telling you that you have exceeded the maximum memory allocation for your program. The first time this problem surfaces on one of your programs, it can be quite disturbing. Maximum memory exceeded is many times a problem for those of us using mini-computers like Digital's PDP-11 series. But I have found, with a lot of help from my colleagues at NSSD, that there are ways to get around this problem. This session is designed with the inexperienced programmer in mind. I will try to show some of the ways that I have learned to optimize the code of a program, and get those large programs running with all the necessary components included.

Central to a discussion of making a program "fit" is the job area allocated for each program written on the PDP-11's. The job area for programs is 32K words. This is because the PDP-11's use 16-bit words to manipulate data and instructions. A 16-bit word can access 2 to the 16th, or 65,536 bytes, which is 32,768 words. This area is also referred to as Virtual Address Space, since your job area does not really consist of your program's code itself, but the addresses of where the computer will find that code. Now, if a program was all that went into this 32K work space, there wouldn't be too many problems fitting programs into it. But of course, other things go into this space along with the program. Basically, there are three things that go into the virtual address space: the run-time system, libraries, and the actual program.

RUN-TIME SYSTEMS AND LIBRARIES

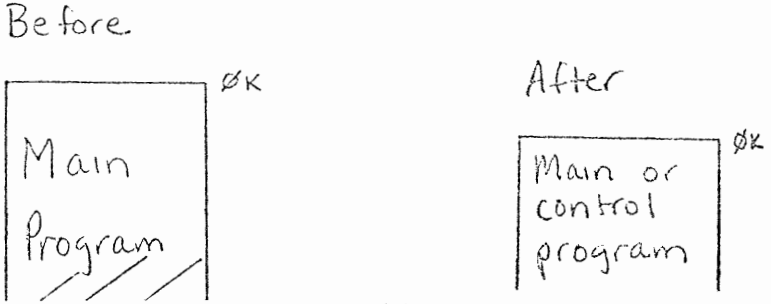
One of the first things to take up some of your job area is the run-time system. Whether you have chosen BASIC2, BP2COM, or RSX, all these take up some space in your job area. BASIC2 takes up 16K words, fully half of your job area, while BP2COM and RSX take only 4K. This space is allocated in the high address part of the job area, i.e. from 28K to 32K for BP2COM and RSX. RSX has the added feature of being a "disappearing" run-time system, if the RSTS/E monitor was installed with "RSX directive emulation". This means that RSX will occupy 4K of the job area while it loads your program, and will "disappear" at execution time, leaving you 31K (you never have all 32K) for your program at execution time. Resident libraries also share the job space with the run-time system and your program. Depending on the files your program tries to access, and other factors, the resident libraries you chose to link to your program take from 4 to 16K, always in increments of 4K. At our installation at the NSSD, when BUILDing our programs, the defaults taken for the run-time system and libraries, take a total of 16K words, leaving us with 16K words for our programs.

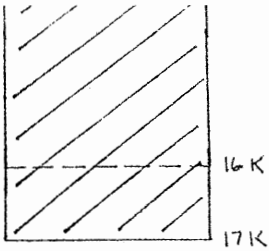
THE ACTUAL PROGRAM

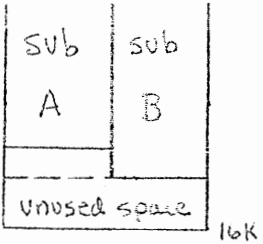
To fit large programs into this 16K word area takes some efficient coding. Although efficient coding is good practice whether the program is large or not, it is critical for the larger ones. Some of the ways that I have found to help make programs fit include program segmentation and overlay structures, the efficient use of maps, and using IF statements and implied IF's in the right places. These three areas alone, when used well, will save valuable space, and allow programs to fit into the allotted area that otherwise would not. All of the methods I am about to describe are things that we at the NSSD have found reduce the size of a program. The figures are by no means exact, but come from experimentation with the different structures to be discussed.

Program segmentation and Overlay structures

The first thing to do when a maximum memory exceeded message comes up, is to think about the ways that you can segment your program into smaller, more manageable pieces. When you have decided what things should go together, they can be put into subprograms. Then all of these pieces can be "overlayed" according to the program's Overlay Description Language file, or ODL file for short. Each of these separate subprograms must be "logically independent", in other words they must not call each other. This is because only one of the programs that are overlayed can be in the job area at any given time. This is better explained using an example.

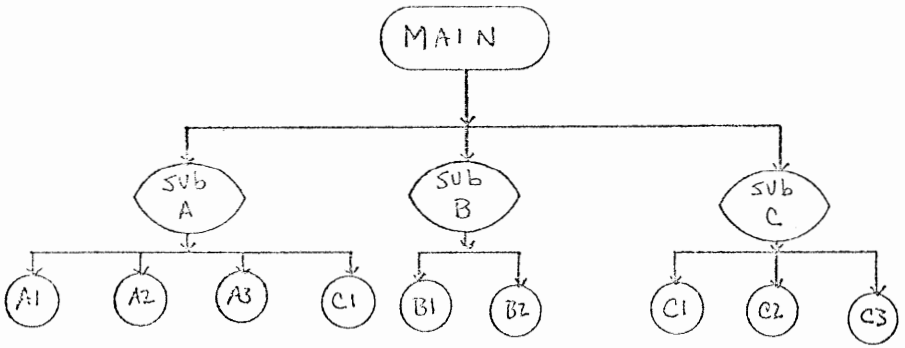






In the diagram above, the program is shown before and after segmentation. In the before stage, the program is self-contained. There are no subprograms to be overlayed. When maximum memory exceeded results, the programmer "cut up" the program into subprograms A and B. Now subprograms A and B can be overlayed in the job area and gain more space for the program to run. This is accomplished by "swapping" the overlayed programs in and out of the job area as they are called by the main or control program. When the main program calls A, the computer will move A into the job area and then proceed to do whatever A calls for. When A is finished running, control returns to the main program, with A remaining in the job area. Should A be called again, before B, this subprogram will still be there. When B is called by the main, A is removed from the job area and replaced by B. The processing called for by B is then done and control ultimately returns to the main program. This swapping is done until the program is completed. As you can see this procedure could save considerable space when a large program is cut into small pieces that can be overlayed.

Programs can be overlayed at several different levels, with one program calling another, but this can also defeat the purpose of having the programs overlayed. For instance, subprogram A could in turn call subprogram A1, A2, and A3. These three subprograms can be overlayed within the structure of subprogram A's overlay. The tree below helps to show what can be done:



The main program is the root, and each subprogram has been overlayed on one level or the other. In this example, subprograms A, B, and C overlay each other, while on the next level subprograms A1, A2, A3, and C1 overlay each other, B1 and B2 overlay each other, and C1, C2, and C3 overlay each other. Subprograms A1, A2, and A3 cannot be called by any program but A, although subprogram A1 could be included in another path of the tree, for example as a subprogram called by B. As you can see, subprogram C1 is included as a subprogram of both A and C. This is a perfectly legal structure even though

it is somewhat self-defeating. If a certain subprogram is included in many different paths of the overlay tree, it may be beneficial to include that subprogram on the first level, with the main program. Then it is always in the program's job space and is available for any other subprogram to call without control actually passing back to the main program. If all of the subprograms are approximately the same size, the maximum space saving is achieved. But if there are one or two subprograms that are quite a bit larger than the rest, adding another level to that subroutine path would probably be of help. In this way, different parts of these large subprograms can be overlayed, thus saving space and making these large subprograms more manageable. One important point must be remembered when you are deciding how much your program needs to be segmented. Setting up the subprogram itself takes up space. You may lose more space by having the subprogram than you gain by the segmentation.

The way to see how effective your program segmentation has been is to generate a "Memory Allocation Map" when you task build your program. The first page of the allocation map is all that concerns us presently and an example of one is shown below:

```
UCR831.TSK   Memory allocation map  TKB Ø7.2Ø4      Page 1
                     2-MAY-83   15:ØØ


Partition name : GEN
Identification : 16RE
Task  UIC      : [2,29]
Stack    limits: ØØ1ØØØ ØØ1777 ØØ1ØØØ ØØ512.
PRG xfr address: Ø21Ø54
Total address windows: 4.
Task  image  size  : 1ØØ16. words
Task address limits: ØØØØØØ Ø47ØØ7
R-W disk blk limits: ØØØØØ2 ØØØØ54 ØØØØ53 ØØØ43.



UCR831.TSK Overlay description:

Base    Top         Length
----    ---         ------
ØØØØØØ  Ø32773  Ø32774  13820.      $ICIO2
Ø32774  Ø35503  ØØ2510  Ø1348.          UCR83A
Ø32774  Ø43657  Ø1Ø664  Ø4532.          UCR83B
Ø43660  Ø45153  ØØ1274  ØØ7ØØ.      R3PUT
Ø45154  Ø47ØØ7  ØØ1634  ØØ924.      R3UPDA
```

The program overlay description is what interests us. There are five columns, the BASE, the TOP, two columns under LENGTH, one in octal and one in decimal, and the program segment names. We are concerned with the LENGTH columns and the name column. The two subprograms, UCR83A and UCR83B are the subprograms that are overlayed in this program. Subprogram UCR83A is 1348 bytes long and subprogram UCR83B is 4532 bytes. This disparity between the sizes of these subprograms was not critical in this instance but it could be. If the main were still too large, the program may run while processing

is controlled by the main program or subprogram A, and then bomb with the maximum memory exceeded message when B is called. Should this happen, it is necessary to examine the allocation map and check the size of the program where the maximum memory was exceeded (the message will tell you what program this happened in). Any programs that are larger than the guilty program should be cut further to avoid more maximum memory exceeded messages on subsequent testing runs. This is the main reason that all the subprograms should be approximately the same size.

Maps

Most of the programs we write at the NSSD include maps. When maps are used, each individual variable name uses approximately four bytes of the job area. If there are three or four maps in your program, with a lot of fields in those maps, this can add up to quite a bit of program space. Unless you use all of the fields of each map in the program, it is not necessary to have all those field names declared in the program. For every variable name that is not used in the program, you can save the four bytes that name would have used. When this is added up among all the saved variable names, it can mean major savings. The way this savings is accomplished is illustrated below.

```
9Ø5     !                                                                &
        ! FD    UCF1:UCIØ5.MST-----------Invoice file map.                &
          MAP   (UCIØ5)                                                  &
                 IN.INVOICE.NO$          =     8 ! Invoice number.       &
                ,IN.ACCT.NO$             =    15 ! NSSD customer acc't number. &
                ,IN.LIEN.NO$             =     7 ! Lien number, if any.  &
                ,IN.INV.DATE$            =     6 ! Date of invoice (YYMMDD). &
                ,IN.DUE.DATE$            =     6 ! Date by which payment is due &
                                                 !  (YYMMDD).           &
                ,IN.PAID.DATE$           =     6 ! Date paid in full (YYMMDD). &
                ,IN.BALANCE             !=     8   Balance due.          &
                ,IN.ORIGINAL.AMT        !=     8   Original invoice amount. &
                ,IN.REFERENCE$           =     3 ! Reference.            &
                ,IN.STATUS$              =     1 ! Status -- open, closed, &
                                                 !  prepaid.            &
                                        !   ---                          &
                                        !    68   Total length           &
                                        !
```

In the file description shown above, all of the field names would take up about forty bytes of the job area. But if the program only used the invoice number and the balance in the program, it would be beneficial to replace and combine the field names that are not used into a FILL$ field as follows. The account number, lien number, invoice date, due date, and paid date would be grouped into one field called FILL$ and with a field length equal to the sum of the field names removed, or forty. The original amount, reference and status could also be grouped together in a similar fashion, also with the field name FILL$. The file description would then look like this:

```
9Ø5     !                                                                &
        ! FD    UCF1:UCIØ5.MST-----------Invoice file map.                &
          MAP   (UCIØ5)                                                  &
                 IN.INVOICE.NO$          =     8 ! Invoice number.       &
```

```
,FILL$                  =    40                                  &
,IN.BALANCE             !=    8    Balance due.                   &
,FILL$                  =    12                                  &
                        !   ---                                  &
                        !    68    Total length                  &
                        !
```

In this way eight field names have been eliminated (FILL$ does not take up any space) and a savings of thirty-two bytes is the result. This is a small file description but ample enough to show the savings that can result. It is important to remember that when subprograms are called, the data in the buffer area for a specific map is passed to the subprogram only if the map is included in both the calling program and the subprogram. But it is not necessary for the maps in each program to have the same fields declared. The main program may get the data from a certain file, but not use it anywhere within the program. If this is the case then it is necessary to include the map in the main program; but it can consist of just one field name, FILL$, having a length equal to the total file length size. When this program calls a subprogram, the data will be passed to the subprogram if there is a corresponding map in the subprogram. Then the subprogram can use what data it needs, by declaring the field names used, and leaving the rest as FILL$ fields. This way space is saved in the main program and in the subprogram as well.

IF's, Implied IF's and Line Numbers

The use of IF statements in our programs is one of the most important parts of any program. Without IF statements, no decisions could be made based on any data available. It is a good thing they are around. In BASIC-plus-2 any regular IF statement must be ended by a line number for the next line. In the new version, BASIC-2, the IF statement can be ended in this way or by the new END IF statement. The problem with the BASIC-plus-2 way is that line numbers take up approximately sixteen bytes of your job area. When fitting a large program into a small space, unnecessary line numbers become an unaffordable luxury. If your site has installed the new version, then the END IF statement should be used to end any IF-THEN or IF-THEN-ELSE blocks of code. When running under the old version, deciding between implied IF's and regular IF-THEN blocks becomes important. If there are regular IF statements in your program that have only one statement in them, it is beneficial to change that to an implied IF and take out the line number to end the IF-THEN block. On the other hand, trying to save too many line numbers by changing multi-statement IF's to a bunch of implied IF's to save line numbers does no good, in fact you lose some space. It is clear that line numbers should only be used where absolutely needed, such as the target lines of GOTO's and GOSUB's, and to end multi-statement IF's. Programs can be broken up in other ways. The use of a blank line is one example. It is much better than using a line number, since a blank line takes up none of the job area.

There are other things that can be eliminated when you are trying to conserve space for your program. One of the most important of these is the use of unmapped constants. For example, if a program compares many strings to "A" and does not use a variable to represent "A", each time that constant "A" appears in the program, sixteen bytes are used. To declare an unmapped variable equal to "A" would take four bytes for the variable name (no matter the size of the variable, i.e. LETTER.A$ takes the same amount of space as

A$), four bytes for the assignment statement, sixteen for the string constant in the assignment statement for a total of twenty-four bytes. There are four bytes used for each additional reference to the variable name, so you can see that if that constant "A" where used ten times in a program, it would result in 160 bytes used. If the variable was used those same ten times, there would be only sixty-four bytes used. Another quick way to save some space is to use GOSUBs instead of functions. In BASIC-plus-2, functions are added to the code wherever they are called, but GOSUB routines are only included once in the program.

CONCLUSION

These are not the only ways to save space in a program that is large. I have arrived at these figures and examples while working with large programs we have written at the NSSD. Program segmentation, maps, IF statements, and unmapped strings are not the only space saving techniques. More advanced measures are available, COTREES being one example. But for the programmer inexperienced in the problems of making programs fit, the topics covered here give him or her a great head start.

RSTS/E Tips

Jeffrey Killeen, Session Chairperson
Information Design and Management
Sherborn, MA

Reported by Joseph Lowery, DECUS Scribe Service

Q. On EMT logging in the switch register, now that you've given us everything on the switch register between 1 and 15, why not let us select 6 or 7, or however many switches are left, so we can turn the subset of the EMT loggers off and on?

A. We're looking at a more generic way of handling that.

Q. What are some hints with regard to what I need to look at in the DDB of a modem controlled line to tell if someone is connected to the line but not logged in?

A. There's a variable offset in the DDB called MODCLOCK, you have to look up that value in your monitor SIL. If the most significant bit is set then the modem line is hung up.

Comment: In terms of turning off and on a subset of the EMT logging you can do it yourself from your basic program by peeking at the switch register location.

Q. When RSTS runs low on small buffers, what starts happening?

A. As it starts to run out, it will quit doing things. Around 75 or so it will logging errors, at 40 it will quit logging in, and as it keeps getting lower it will keep slowing down until it hits zero, when it will stop until things free up. If any interrupts occur and DSQs get freed up from disc IO, then things will gradually come back.

Q. Is there any other reason you can think that would cause the system to keep people from logging in at about 24 users online when there is sufficient swap space and the jobmax is 30?

A. Small buffers, swap space, or if, after you added the swap file, you forgot to reenable logins to force recalculation

of the new maximum.

Q. I have a tip with regard to the HELP program. If you put $HELP.HLP as the first file in the $ directory to make it contiguous, it helps quite a bit. Do you know why this is?

A. It does a sequential search through the directory to find your HELP file. If it's at the end of the directory, it takes a while to get to it. Sequential caching would help as well. Also, depending on the indirect references to other HELP files within the HELP, you could position the more frequently accessed HELP files at the beginning of the directory.

Q. We've had some problems with NPR devices, specifically the DMR-11. It will start hitting the bus with heavy NPR activity, and there's almost no way to notice what's going on. RSTS never seems to get the time to notice that. Is there any way to tell if anything is actually hitting on the NPR?

A. Use the logic analyzer, since the software can't tell what the bus is doing.

Q. If RSTS is keeping count of small buffers, as was earlier mentioned, is there a global peak address I can look at so that I can keep count and broadcast a warning message to appropriate terminals?

A. FREES+2, but that will change in each of your SILs.

Q. What happens when the buffer levels get down to 20% free and 25% free?

A. Those two percentages deal with buffer quotas for character oriented devices. As long as you are over both 20 and 25 percent free, character oriented devices will be allowed to use as many small buffers as they like to buffer their IO, with the exception of a terminal that has been stopped with xoff. When you start to drop below those values, then it will no longer be allowed to exceed its buffer quota. The difference between the 20% and 25% figures is that at 20% it will no longer let it get its quotas, and you'll get caught in a buffer stall in the BF state.

Q. On disabling terminal lines, we often use a DH port, and found that setting speed zero on a DH works fine for shutting it down and trying to run login on both sides. On DZs, which don't have zero speed on the hardware, how can you set zero speed? Is the monitor looking at the speed?

A. If the monitor sees speed zero, then it will disable transmits and ignore receives.

Q. With an 11/70 with four RM03s, two of them set up as DMs and two as DRs, two different controllers, how awful is this? Is the CPU being beaten on by both of those controllers so much that it is not really thinking very much?

A. It's actually a pretty good configuration, since you can not only overlap seek on both drives, but you can overlap the IO operations as well through the two controllers.

Q. What exactly is a missed error?

A. A missed error is when you get below 75 buffers and it can't log it, or if the error copy program has exceeded its message limit and gone into hibernation. This can also happen if you are in the error logger when you get another one; in this case the second error is missed.

Computer Center Relocation

Larry W. Hicks
R.J. Reynolds Tobacco Company
Winston-Salem, NC

Emily Kitchen, Session Chairperson
A.H. Robins Company
Richmond, VA

Reported by Scott Howell, DECUS Scribe Service

After being introduced by the session chair, Emily Kitchen, the speaker, Larry Hicks discussed how he successfully planned and completed the relocation of the computer center of R.J. Reynolds Tobacco Company.

Hicks began the presentation by discussing the background to his particular relocation. He had to move a single DECSYSTEM-2060 supporting three departments. The DECSYSTEM-2060 was in the corporate computer center, while users and technical support were in the tobacco company, a subsidiary.

Hicks described the method to plan the move, the most important factor of which is early preparation. To do this one must:

1. Determine the major activities.

2. Determine the impact of each phase of the move.

3. Determine the lead times and critical events.

4. Break each phase into steps.
   a. identify all steps.
   b. determine step and activities dependent upon them.
   c. determine how to perform each step.
   d. estimate time for each step. (be conservative)

5. Validate information of each step.
   a. check other considerations.
   b. verify time estimations.
   c. check overlapping response.

6. Prepare preliminary time table for move.
   a. establish desired move date.

b. determine initial activity and the earliest starting date.
c. establish earliest move date.
d. set realistic move date.

7. Formalize time-table.
   a. solicit comments on the preliminary time table.
      -from user department
      -from vendors
      -from other parties in move
   b. modify time table and schedule.

8. Prepare for implementation
   a. determine "go/no go" points in schedule.
   b. decide action if "no go".

9. Implement move schedule.
   a. publish implementation or move schedule.
   b. add detail.
   c. request immediate notification of problems and questions.

10. Track completion of each step.
    a. have one person with ultimate authority.
    b. review steps weekly.
    c. check daily during the last two weeks.
    d. monitor "go/no go" points.

11. Publish decisions.

12. Schedule periodical report.

The preceding are all the necessary steps needed in a computer center relocation. Hicks added several things one must do after the project is completed.

1. Evaluate move.

2. Review problems.

3. Analyze slack time.

4. Evaluate accuracy.

5. Determine move's impact.

6. Review your own action.

7. Thank involved parties in the move.

Hicks concluded his session with a summary of how to make a successful transition of a computer center. These are four basic easy steps.

1. Make detailed advanced planning.
2. Allow for a conservative schedule.
3. Make a close monitoring of progress.
4. Publish results.

VAX Security Panel

Stephen Tihor
New York University
New York, NY

Ross Miller
Online Systems
Spokane, WA

Dave Schmidt
Management Science Association
Pittsburgh, PA

Frank Kieltyka
Los Alamos National Laboratory
Los Alamos, NM

Almon Sorrell
Westinghouse Defense and Electronics Center
Baltimore, MD

Andy Goldstein
Digital Equipment Corporation
Nashua, NH

C. Douglas Brown, Session Chairperson
Sandia National Labs
Albuquerque, NM

Reported by J. Rick Mihalevich, DECUS Scribe Service

Dave Schmidt from Management Science Association discussed security from the commercial perspective. Security on large networks is an issue of great concern. Special commercial security concerns are:

1. Client data protection

2. Accounting protection

3. Personnel and planning data protection

4. Simple pitfalls

Addressing the problem of client data protection requires consideration and monitoring of a variety of computing functions. First, a consideration of each client as a unique group must be maintained. Access across clients should be protected to prevent "world access" to a client's directory; In addition to online consideration of data protection, there is an equally pressing issue of offline data protection.

Offline procedures must be monitored to prevent accidental or unauthorized access. Possible areas of security penetration are uncontrolled hardcopy output, backup tapes, communication lines, data tapes, and delivery and shipping mistakes.

One goal of commercial security is to provide an easy way to use tool for controlled access to client directories by support persons. In this case, the SETUIC (fall 1979 SIG tape) was implemented. This allows for installed images, screens UIC access by a control list, and can be modified to log access if desired.

In terms of accounting protection, SNDACC allows entry of transactional or value added billing entries to the system accounting file, however this is not available from DCL. SNDACC does provide a secure record that is difficult to transfer with and can be used with a command procedure and user written program to do royalty record keeping for leased software.

The second approach in accounting protection is to provide a tool for support personal to easily change the VMS account field in the billing records with which they work. In order to accomplish this goal the use of ACCSET (fall 1980 SIG tape) was used as will as the installing of image.

The issue of personnel and planning data protection has some very serious security protection problems. If it is essential that security be maintained with total certainty, then a microcomputer and its processes in a stand-alone environment is the only alternative. It is impossible at this time to protect data from system programmers and managers. Even the use of data encryption is not a viable alternative. A dumb terminal requires plain text data transmission that can be intercepted easily. Also, SDA and other tools allow investigations of a running process's memory space.

It is possible for a system manager to become any user, allowing several methods in which to penetrate the system.

In summary there exist some simple pitfalls. Awareness of these and the implementation measures to prevent them will do much to improve protection. The first suggestion from this

presentation is: do not install AUTHORIZE or INSTALL utilities. Secondly, monitor and require management accountability of all users with highly privileged accounts. This is done knowing that most computer crime is traced to a person in this category or one with access to these accounts. Also, try to actively penetrate your own system and involve senior management in the monitoring and accountability. Thirdly, don't leave terminals logged in and unattended. Fourth, thoroughly check all details of changes to operating procedures and computer room environment. The fifth suggestion given was do not write down and file passwords in a central place for record keeping convenience. The final suggestion given was not to rely on user ignorance of VMS. There are currently computer science majors being graduated with 4 years of VMS experience and growing numbers of other persons are now familiar with VMS.

Ross Miller of Online Systems, another representative from the commercial sector, was the second presenter.

In regard to complex security systems, it was mentioned that one can reduce the number of combinations and permutations with simpler or more manageable security systems.

Another item of concern is manager mobility within a system and their availability to a number of passwords, therefore increasing access and penetration protection.

To combat the problems of system penetration the implementation of the "CHANGE" facility was suggested. This facility removes default devices and default directories. Another feature of this facility is that it monitors logon sessions and creates a record of user/time/location information.

A final precaution is the use of time out screen management techniques. This requires the user to maintain interaction with the system otherwise termination of the job will occur.

Steve Tihor from New York University discussed the problems of security in the academic world. In the academic computing world there are often users which differ greatly, from faculty research and records to administrative functions to student assignments and projects. The challenge of system security is to find the middle ground between too much access to system and too much security. Users are in effect, handcuffed.

Efforts to discourage system penetrations by the tracking of logons are often very effective. This produces a thorough report of all sessions.

The selection of passwords are often too predictable and sometimes result in a problem with penetration within certain system domains.

Frank Keieltyka of Los Alamos National Labs who works in the classified environment addressed the network security and

the use of proxy logins.

In protecting the default DECnet account the best solution was not to have one. Otherwise prevent interactive use, prevent batch jobs, and prevent "task" access. The proxy login allows for selective mapping between systems' user bases. however this is an experimental, unsupported implementation in VMS V3.0. The DECnet access control provides default access. Which is non-selective, insufficient protection, and insufficient accessibility. The explicit access control, on the other hand, is cumbersome, hard to audit, and there is risk of password compromise.

Current vulnerabilities are wire tapping, integrity of remote systems and node impersonation. Future directions will require greater flexibility, possibility of encryption, and the use of non-discretionary security support.

Almon Sorrell of Westinghouse defense and electronics an Department of Defense contractor. This presentation addressed the problem of label marked printouts with classified levels. The implementation of a utility called "CLASSPRIN" and another utility used with great success which is available through Software Services.

Three comments were made regarding general system security. the first is the use of auto logout utilities to protect unattended terminals. Secondly, protecting against field service personnel who use the same password between classified and non-classified routes, and the use of patterned passwords. The final comment is that PFT systems are not suitable for privileged files.

Security Mechanisms for VAX/VMS

Mark Pilant
Digital Equipment Corporation
Nashua, NH

Douglas Brown, Session Chairperson
Sandia National Labs
Albuquerque, NM

Reported by Omar El-Ghazzawy, DECUS Scribe Service

The session on VMS security, presented by Mark Pilant of DIGITAL, discussed various security mechanisms and their implementation in future releases of the VAX/VMS operating system. The security mechanisms covered were access control list (ACL), non-discretionary controls and audit logging.

The term security was applied to the need to protect information from unauthorized distribution and/or modification. The protection services required to achieve these objectives were the ability to grant or deny a request, and the ability to monitor request. Currently, the VAX/VMS (version 3.2) operating system's main security mechanism is the use of UIC to determine access.

In the security environment each request is comprised of three components :

1. object - anything which is protected (i.e. file, disk)

2. agent - anything which requests access to an object (i.e. process, device)

3. identifier - a unique method of identifying the agent. In the VAX/VMS environment the identifier is externally represented by an alphanumeric name and internally by a unique 32-bit number which may be defined at several levels - system wide, group wide and per user. In the access control list (ACL) mechanism an identifier list composed of access control entries (ACE) is constructed that determines what access is to be allowed by an agent to an object. Some specifications on ACL are :

    1. an ACL may contain one or more ACEs

2. an object may have only one ACL

3. for files, ACLs are propagated from the previous version of the file, or the parent directory.

A second mechanism is the use of non-discretionary controls which provide protection by containing access to an object by an agent. In contrast to the previous mechanism the owner has no direct control over this method of protection, all changes in protection require the intervention of the system manager. Two forms were presented :

1. Integrity defined as a measure of trustworthiness.

2. Security defined as a measure of sensitivity.

The integrity level, which ranges from Ø (least trustworthy) to 255 (most trustworthy) is used to control modifications to the objects. It properties are such that an agent cannot read less trustworthy information or write more trustworthy information than he is allowed access to. In addition integrity compartments can be created which are functions of 64-bit mask identifiers. The security level is identical in structure, it also ranges from Ø (least security) to 255 (most security), and is used to control access to the objects. Its properties are that an agent cannot read more sensitive information or write less sensitive information then he is privileged. Similarly, security compartments can be created which are functions of 64-bit mask identifiers.

During the question and answer period the addition of group level bypass privilege to the VAX/VMS operating system, and a mechanism for the encryption of data transmitted via DECnet were mentioned by DIGITAL personnel as possible future additions.

Challenges for the 80's

Information, Resource Management--Challenge for the Eighties

Albert B. Crawford
Digital Equipment Corporation
Concord, MA

Sanford Kruegar, Session Chairperson
AMAX Copper Inc.
Carteret, NJ

Reported by Susan Miller, DECUS Scribe Service

Albert B. Crawford of DIGITAL presented his vision of computer management in the '80s. Businesses need a strategic computing plan because of obsolete techniques in information processing and data management systems. Crawford's speech, entitled "Information, Resource Management--Challenge for the Eighties," was based on his paper about DIGITAL's current and future management plans.

The scope of information resource management connotes breath of function and that data is a resource which needs to be managed. Management should cover data processing, text/image processing, office systems, process control and telecommunications. The technology of the backbone network helps with information management.

DIGITAL's business environment can be described as high technology manufacturing, sustained high growth, expanding competitive market, multinational, and a high rate of change. While the organization/management environment contains persons who act as entrepreneurs, a dependence on matrices, an organized product manufacturing, decentralized decision making, and also a high rate of change.

Technology is influencing business with computer literacy, cheaper computer power, the fifth generation architecture and tools, and the use of the expanding application portfolio. The portfolio has strategic planning, management control and operational control. These layers are also divided by departments. The sales and engineering departments are behind in automation unlike the finance department which has full automation. Crawford described the bottom layer as generic tools. The purpose of the tools is to optimize technology for something like electronic mail.

Changes are evolving throughout the work force and the role of the information resource management director. The work force faces sociological changes and is becoming multi-cultural. Spanish is developing into a second language and about 50 percent of the work force will need to speak it. Employees of a business are becoming more dispersed with persons working out of their homes. Crawford thinks a management director needs these changes: a business orientation, to act more like a staff consultant, define context not content, drive human resource management and facilitate technological transfer.

"A Strategic Computing Vision for the '80s" was created for DIGITAL. This vision embraces the idea of end-user computing. It has to be implemented with wiring up. There would be a dramatic shift in computer use with an information specialist and end user, along with an application portfolio mix.

Crawford wants to readjust his portfolio, though he is not sure how to do it. There would be a shift from the '81 figures to the target for the '86 to '88 period:

	Year '81	(Target) '86-'88
Operational Control	30%	20%
Management Planning/Control	10%	50%
Strategic Planning	05%	15%
Generic Tools	05%	15%

Their strategy for the mid '80s involves more control of data elements. The programs need timeliness and accessibility. End users face certain problems that restrict their accessibility. The languages are too complex. Data elements need to be consistently defined in terminology and levels of summarization. Users need help in navigating through the structure of the multiple files and logical design. And the multiple tools should be simplified.

Crawford said, "The real payback is to go after the knowledge worker." This includes managers, professionals, secretaries and administrators to equal the corporation's goals. DIGITAL doesn't use public terminals. Employees have their individual terminals and about one-third in the electronic mail service have a terminal at home. They can work at evenings or on the weekend.

Computers have caused behavioral changes in people. Workers feel they have more productivity. The quality of their work has improved. This helps the working relationship within a business. As long as business manages their computers and information, it helps if all employees can punch their own system.

Field Service Overview

Field Service Overview---Issues and Innovations

DIGITAL Field Service
Digital Equipment Corporation
Stow, MA

Emily Kitchen, Session Chairperson
A.H. Robins Company
Richmond, VA

Reported by Joseph Lowery, DECUS Scribe Service

The Digital Equipment Corporation is making plans for the future to improve service to its customers. That's the topic that a panel of speakers representing DIGITAL's field service department addressed before a large crowd of conventioneers Tuesday morning.

The speakers, Jay Atlas, Larry Fox, Jeff Holmes, Tom Karpowski, Dan Fatte, and Don Hauger each concentrated on a different aspect of some major innovations that Digital users can expect to see finalized this summer. The innovations were grouped into three major areas: (1) the integration of hardware and software maintenance, (2) new services for network users and (3) a system for improving logistics conditions.

Fox began the session with a presentation of four issues requiring an increased amount of attention. The first is that of problem escalation. Fox noted the importance of Digital service providing timely and prudent solutions to the users problems. With this in mind Fox described the already existent Action Outage Program employed with the goal that eventually all DIGITAL's best resources would always be available for use by the customers. Fox also alluded to two changes that he feels will be made: (1) to more efficiently handle the initial call for service via improved conditions on call handling systems and (2) integration of DIGITAL's customer service functions.

The second issue addressed was the specific handling of intermittent problems, which were then subclassified into the areas of (1) better recognizing the problems through the diagnostic SPEAR program and customer input, (2) taking action on the problem by focusing attention on the customer and employing a systematic approach through a standardizing process targeted to be finalized in June and implemented by July, and

(3) to stand committed to a policy of ensuring success in the areas of service.

Fox's third issue entailed increasing the quality of service in all of its branches by employing data attained through customer feedback. He added that the key metrics in this situation need to be monitoring the levels of customer satisfaction, repeat customers, service time and response time.

The fourth and final issue presented by Fox concerned the realm of servicing non-DIGITAL products, a field that they hope to improve on to meet the customer's needs more successfully.

Holmes followed and spoke on the first of the three innovations presented earlier in the session: the integration of hardware and software service corps. Holmes described the problem in the past of customers having vague ideas on which area a problem might lie, and hence not being sure of which service to call. DIGITAL's goal in this integration program, then, is to perhaps alleviate this problem by requiring the customer to make only one call for service and yet still retain the high quality service that Digital has provided in the past.

The customers would then benefit from the program in two ways. First, in the aspect of service product innovations, systems agreements, network services and guaranteed uptime will all improve. Second, the implementation of service delivery innovations, such as the aforementioned call handling process and the use of a single point of contact for entire networks, will increase both speed and quality of service.

Karpowski then took the podium to introduce the plans for the field of network service. Planned are three major programs: (1) a comprehensive service for all networks, including wide and local area, integrated and mixed vendor networks, (2) Life Cycle Services, to be employed in the areas of the design, installation, operation and further evolution of networks, and (3) a program for customer training, so that certain problems that arise may be resolved without Digital service at all.

Karpowski concluded that this plan, if successful, will prove to provide a complete service program for all of DIGITAL's network customers.

The final speaker, Don Fatte, addressed the problems to be overcome in the realm of logistics. He listed the three main objectives as: (1) to deliver no parts to the service call that arrive inoperative, or "dead on arrival" as he termed them, (2) to keep no repair calls on hold because of the company's inability to locate the needed the part, and (3) to avoid making repeat trips on service calls. Fatte proffered that these goals can be attained by concentrating on three areas. The first was to keep stockrooms well filled and close to the customers. The second was to employ a frequent replenishment of supplies and a planned, as opposed to a random, distribution schedule.

The third, and final, point brought up was done with respect to the repairs themselves. The decentralization of repair shops to field locations, tighter control of engineering processes and shorter lead times were all proposed as being beneficial to faster and higher quality repairs.

PRO300 Communications

Ray Shapiro
Digital Equipment Corporation
Maynard, MA

Jeffrey H. Rudy
Digital Equipment Corporation
Maynard, MA

Stephen G. Finch, Session Chairperson
Emulex Corporation
Costa Mesa, CA

Reported by Gene Mitchell, DECUS Scribe Service

Communication was the subject of a heavily-attended session entitled "Professional 300 series communications." Speaker Ray Shapiro of Digital discussed communications services, telephone management systems, IBM communications and some future plans. Communications services include:

1. End user interface.

2. Callable service routines.

3. File transfer at speeds up to 9600 bytes. This may require an extra memory board. This file transfer also offers, among other options, a new command file transfer, password protection, and command terminal support.

Shapiro added that when the version 1.7 comes out, file transfer will be possible from Professional to Professional. He speculated that by the end of the year, the following might be included in the 1.5 or 1.7:

1. The menu could contain communications set-up, enter terminal-emulation mode, Professional-to-Professional file transfer, and call control services.

2. The 1.5 menu may include set default line characteristics, modify current line characteristics, and restore dafault line characteristics.

The file transfer set-up menu should allow the user to:

1. Accept or reject copying file.

2. Accept or reject incoming file.

3. Supersede or reject new files with previously-used names.

4. Password protection.

The terminal-emulator set-up menu includes both terminal echo and terminal type. The terminals emulated are: The VT52, VT102, VT125, and the Professional. Also available are:

1. Log keyboard input to file.

2. Access communications set-up from terminal menu.

Next on the agenda were Telephone Management Systems (TMS). Features include:

1. Integral modem. The unit has 300/1200 baud integral modem with Bell 103J and 212A asynchronous compatible modes that fit into the Professional box.

2. Four modes of operation, including voice data, serial data and touch-tone frequency.

3. An optional voice unit.

4. Easy programmer interface with capabilities for both simple and complex programming.

5. Supports two telephone lines.

6. Professional phone book: A software program allowing the user to maintain a directory of names and numbers.

Shapiro mentioned some sample applications: Telemarketing, DTMF inquiry and command, voice annotation and dictation (not done through the telephone), and teleconferencing. The TMS driver is full-sequence, with a 4096-byte buffer. It was suggested that the user use a dual-buffer approach for best results.

Shapiro concluded the TMS section by adding that it is "well-engineered," offering:

1. Several timers.

2. An emergency cutoff switch.

3. No pre-empting of telephone use.

4. A suitable dial mode: DTMF, push-button and rotary.

5. Ability to switch to voice code.

The IBM communications section dealt with connecting the Professional Series with IBM host computers. Three devices were used as illustration: The Digital RJE, the 3276-BSC from Advanced System Concepts, and the 3276 SNA from Paramim. The 3276-SNA can work as a 3287 printer emulator. The Virtual Terminal Emulator (VTE) from Advanced System Concepts works as a 3276 emulator. The VTE also includes a computer-based packaged tutor for the ease of the user, who need not go to the manual for instruction.

PRO300 Series - Future Directions

Steve Paavola
Digital Equipment Corporation
Maynard, MA

Steve Finch, Session Chairperson
Emulex Corporation
Costa Mesa, CA

Reported by Nick Szabo, DECUS Scribe Service

The Professional 300 began being a low cost, open market product with simple hardware and crude software tools. Later, it became a computer with low cost hardware and useful applications. It was a PC designed to be used as a businessman's tool, which has since had rapid market growth and a high projected growth. It then took the shape of a PC in an overall MIS structure with growth paths to bigger systems and across networks. The Professional 300 became a marketable product driven by solving users long term computing needs.

Future directions for the Professional 300 include lower price and more performance. It also looks forward to broadened options with high capacity disks and higher resolution displays. The Professional series will track VAX, PDP-11, and networks. PCs will be a mainstream investment focus and they will be a high functionality workstation.

Key trends for the Professional 300 are system integration, which include clusters (networks), text and graphics, and text and voice. Also, the user interface will provide graphics, pointers, and voice.

On the office automation side, the Professional 300 will be an integrated workstation with combined OA and DP functions. There will be compatible functions across DEC systems and networks, with voice being a key function.

The future key design of the Professional 300 will have a hard disk system, with emphasis placed on lower cost and more storage space.

The video aspect of the Professional 300 will include increased screen resolution with half page display and multiple windows. New input devices will be a Mouse and a Tablet. There will also be video disk integration.

Communications will involve Teletex, IBM interconnect, and DECnet.

PC's as Office Network Terminals

Rembert Aranda
Digital Equipment Corporation
Merrimack, NH

Eileen Deal, Session Chairperson
Digital Equipment Corporation
Merrimack, NH

Reported by Joseph Lowery, DECUS Scribe Service

Personal computer systems are providing a revolutionary driving force in the field of office automation, according to Rembert Aranda of Digital Equipment Corporation. This information was offered in analogy to what Aranda described as the three major revolutions that have occurred in the computer industry.

The first of these, the introduction of mainframe systems, occurred from the 1950s through the mid 60s, when minicomputers were developed and initiated the second revolution. In the 1970s the third and presently existing phase, that of the personal computer, began and started a trend that Aranda projects to continue to rise even more dramatically through 1986 than it has in the past few years.

The presentation then began a comparison of the personal computer (PC) and mainframe based data processing terminals (DPs.) The six major advantages of PCs were described as:

1. The Cost---PCs are less expensive in the long run than main-based terminals due to work station costs. PCs may cost more initially, but can perform an estimated 75% of the tasks usually found in an office environment, such as paper filing, word processing and documentation. The reduced time required for performance of these tasks was noted as perhaps the most important feature of utilizing a PC system. Since terminals carry higher costs for labor, hardware/software maintenance and software acquisition/development, PCs were firmly proposed as being a more economical choice.

2. The availability of color and graphics on PCs.

3. The faster response time of the PCs.

4. Users of PCs can install, acquire and develop software applications with a greater level of ease than DP users.

5. PCs possess greater capabilities for user tailoring.

6. PCs have available new kinds of software that do not exist with DP systems. No further elaborations on this topic were offered.

A brief comparison was then drawn with respect to the traditional software of DPs and the "Volksware" programs of the PC. Again most of the areas mentioned favored the PC. Included in these were documentation, modification time, availability of color, the number of users it can service and most strongly, the payback period for the two options: 5-10 years for the traditional software and six months to one year for that of the PC.

Aranda then described some of the disadvantages of the PC systems. There exists a certain lack of flexibility with the system, in that there are some situations where the programs are unable to be modified at all. Other drawbacks of the system are that the memories are often smaller than those of a DP and that the applications requested by the user can sometimes exceed the capabilities of the hardware.

Popular applications of the software for the PC were then discussed. Of these were (1) modeling, (most notably the spread sheet,) (2) data management, (3) personal word processing, and a variety of others which were felt to be of somewhat less importance, such as the utilization of graphics, communications packages, application/report generators and integrated user environments or "shells." The latter, and also the newest, is expected to undergo the most rapid growth over the next 1 1/2 years.

The services provided via the online data base of the system was then addressed, listing many of the specific features that the system is capable of performing. Highlights of these services are an integrated user interface, a private videotex, an electronic filing cabinet, nonprocedural languages, database management systems, compatibility/conversion facilities, communication gateways, and an electronic mail service. Of these, the electronic mail service was described as the most useful, due to five of its primary features. These are:

1. External gateways which allow for private, as opposed to public, transferral of information.

2. The reduced time and cost of document storage, delivery and retrieval.

3. Easier processing of incoming mail.

4. The fact that it widens a given "information community".

5. It allows the user to handle correspondence from home or any remote site.

Aranda concluded his talk by listing four metrics concerning the application space of a PC system which should be considered before purchasing a system, namely the depth, width, variety, and compatability of space of the system, and added that a wide range of capabilities, (the width of the system) is the most critical and desirable of these properties.

Giving Effective Presentations

Wesley E. Mullen
McDonnell Douglas Automation
Berkeley, MO

Adam Zavitski, Session Chairperson
Seismograph Service Corporation
Tulsa, OK

Several years ago, Mr. Mullen attended a conference where he became quite disturbed about the lack of quality in the presentations. He became so disturbed that he started taking notes. That year he attended 4 or 5 conferences and was able to continue taking notes. They were mostly negative things, things that disturbed him, so he started he started to put together a presentation that would be helpful to those called upon to present papers.

Most people have to make a presentation at one time or another. The presentation could be to a large group or to a single person. It could be an informal discussion or, as here at DECUS, a formal talk. We have probably noticed that when we begin to speak, we get weak knees, our hands shake, our voice begins to quiver, and we get this general feeling of inadequacy. The intent of this talk was to help each of you overcome these anxieties and give more effective presentations.

Mr. Mullen noted that here at the Cervantes Convention center, a group of St. Louis businessmen who created the International Speaker's Hall of Fame Award have plaques honoring the recipients. The award is for professional speakers who speak throughout the country. Their criteria for judging are: enthusiastic, entertaining, exciting, interesting, informative, inspirational and motivational. Although you are not expected to be professional speakers, you should definitely think about their criteria when you are preparing and giving a presentation. Following is a discussion of the presentation itself, its preparation, techniques for giving the presentation, and the use of visual aids. If you pick up only one of the ideas presented here, you will be leave a better speaker.

1. BE YOURSELF. It is very important to be yourself when you are giving a talk. If you pretend to be someone else, you won't look natural. Face your audience. Smile. Don't start your talk with an apology; you want your audience on your side. If you are prepared,

they will find out soon enough. Be positive about yourself and your talk.

2. DON'T READ YOUR PAPER. Nothing is more boring than to listen to someone read word for word from a prepared text. There are very few people who can do this and still keep their talk interesting. If you can't keep your talk interesting, you will put the audience to sleep. Use index cards if it will help. Use an outline or use your visual aids as a guide. Ad lib. Add any thoughts that may come to you as you are giving your talk and make your talk different every time. Keep the content flexible. For instance, Mr. Mullen slipped the information on the Professional Speaker Awards into his talk.

3. ESTABLISH AUDIENCE CONTACT. Make the audience a part of your presentation. Establish eye contact. Find out who they are. What do you have in common with them? What language do they speak? What do they want?

4. INVOLVE YOUR AUDIENCE. Use the words "we" and "you". Establish objectives for them but keep them realistic. Don't have too many objectives. It is better to have one objective and present it well than to have too many of them presented poorly. Make sure that the objectives can be accomplished in a reasonable time frame. You need to know is going to happen because you gave the presentation. Know where your talk is going. You don't want to be like Alice talking to the Cheshire cat:

"Would you tell me please which way I ought to walk from here?"

"That depends a great deal on where you want to get to," said the cat.

"Oh, I don't care much where," said Alice.

"Then it doesn't much matter which way you walk," said the cat.

"So long as I get somewhere," Alice added as an explanation.

"Oh, you're sure to do that," said the cat. "If you only walk long enough."

Our talks are the same way.

5. DO NOT ASSUME THAT ALL OF YOUR AUDIENCE UNDERSTANDS THE ACRONYMS AND BUZZWORDS THAT YOU USE. It is quite possible that all of you present have had a speaker use a buzzword that you did not know at this symposium.

6. BEWARE OF THE PODIUM. As you approach the podium, take it easy. Watch for wires that could trip you. Remember that the podium hides you from the audience and limits what you can do. If you do use the podium, stand erect behind the podium and don't slouch. Don't do things that might distract your audience. Don't pace. Let your hands be natural and, if you must stick them in your pocket, make sure your pockets are empty. Jingling coins or keys is distracting to the audience. Don't play with the telescoping pointer.

7. REHEARSE, REHEARSE, REHEARSE. Do a dry run with someone before hand. It isn't enough to simply go over your notes, you need to get on your feet and vocally rehearse your talk. You should be thoroughly familiar with your talk. Rehearse enough so that you feel you will be able to roll with the punches.

8. BE FAMILIAR WITH THE AUDIO/VISUAL EQUIPMENT. Know where the on/off switch is on the overhead projector. Make sure that all of the equipment is present. Have a backup. Try the equipment out. Anticipate problems.

9. BE SURE THAT YOUR AUDIENCE CAN SEE THE SCREEN. If you will block the audience's view of the screen, be prepared to step aside when you use the screen. Will any lights wash out the screen?

10. EXPEND TIME PREPARING YOUR VISUAL MATERIALS. Use white space effectively. Be careful of character sizes. Characters that are to small can not be seen. Keep the visuals simple by following the K.I.S.S. (Keep It Simple Stupid) policy. The more complicated your presentation, the more things that can go wrong. Keep the attention where you want it. After your audience has seen your visual, remove it and draw the attention back to you. If you must use a "busy" visual, use dark strips to introduce the visual to the audience in sections. Alternatively, you could highlight the important features of the visual. Put the audience's attention where you want it. After spending all of that time preparing a presentation, don't send it through the airport baggage. Carry it on with you. You may lose your luggage this way, but you won't lose your presentation materials.

11. AVOID DATED SLIDES OR FOILS. Keep your materials up-to-date or avoid putting dated items on the visuals.

12. REHEARSE WITH YOUR VISUALS. Make sure that they are in the proper order. Know when to use your visuals. Make sure that they come at the right time in your talk and that you are not talking about something and you are displaying another visual.

13. USE VISUALS AS A TOOL.

14. KEEP THINGS SIMPLE.

Mr. Mullen also had some additional remarks:

Don't talk in a monotone. Speed it up and slow it down. Add inflection and pauses to your talk. If you don't think you have the audience's attention, stop talking. Silence is a good way of getting their attention again. There are problems if handouts are passed out at the start of a talk. You will find that your audience will spend most of their time reading the handouts and missing much of what you have to say.

In a dark room, speak louder. For some reason, people cannot hear as well when the lights are dimmed.

Beware of the "opening joke" or the "opening story". You could kill your entire talk if it is not done well. In the opening part of your presentation, you want to capture your audience, you want to get their attention.

DECUS Library

Ardoth A. Hassler
The Catholic University of America
Washington, DC

Reported by Phil Beene, DECUS Scribe Service

During the recent DECUS Symposium held in St. Louis, Ms. Ardoth Hassler, library coordinator for the DECUS Library Board, led an informative working session designed to update interested users on DECUS Library activities.

Following her brief status report on how the Library has been operating since the previous DECUS Symposium, SIG representatives from the U.S. Program Library Committee delivered short reports on how their individual groups are progressing.

Ardoth began her report by explaining the DECUS Library's new incentive/reward program to encourage program contributions from DECUS users. Any member contributing a program will receive a plaque from the Library acknowledging their efforts. Although announcement brought a general sense of approval from members of the audience, many thought the program would be more successful if the contributor were offered the alternative of receiving credit towards one of the Library's existing listed programs. This suggestion will be referred to the Library Committee.

Following Ardoth's announcements, Larry Hicks gave a brief report on his work with the Library catalog. Since the last Symposium, the previous three existing versions of the catalog have been compiled together into one document. A free copy of this new version should be mailed to all DECUS members around June.

The Library will begin treating this catalog as its main marketing tool. It will be given away in order to make more DECUS members and others aware of the wide selection of programs available through the Library System.

In addition to the catalog distribution the Library will continue to improve user awareness through posters, buttons, booth displays and the implementation of advertising.

A discussion of the Library's current taping programs and

strategies elicited considerable input from members of the audience. One suggestion which came up during this portion of the discussion was the Library's need to make a better general abstract listing of available tapes and the information contained within them. The point brought up, was that many potential Library customers can't justify the tape purchase price without a better knowledge of what they are buying. Ardoth said the group would consider this in the future and try to come up with a better list.

Another taping problem currently being experienced by the Library is their need for a better copying system. A new mass-producing copying unit is one of the Library's main immediate objectives.

For those users and SIG members unable to attend the Symposium, or just wishing to obtain copies of SIG sessions, master tapes will be available. SIGs having copies made at the St. Louis Symposium included: RSX, RT-11, DECsystem-10/20, VAX, and RSTS.

Copies of these tapes are available to users for about $112. The RSX and VAX tapes are quite lengthy, and are expected to include two tapes.

The last portion of the formal half of the session included a discussion of long range planning goals. The main Library goal is to acquire a method of determining what items and interests users will have in the future. According to committee members, this knowledge will enable the Library to better plan and prepare for these needs, saving everybody time and money in the future.

Following the formal first half of the session, Ardoth and other committee members took questions from the audience.

Many of the questions concerned the alternative methods available to create Library donor incentives. Most DECUS members present agreed a credit system would be more effective than the current plaque proposal. Citing other successful library programs, even Ardoth agreed, but said she was not certain such a program would work in the DECUS system. She feels the Library could easily become overwhelmed with nonusable programs offered by users wanting free programs. Although the DECUS Library is not a profit organization, she said such results could destroy the system's financial stability through increased operating costs.

One alternative to the plaque system already being discussed, is the possibility of offering a RAINBOW 100 system through a lottery which would include all program contributors.

In other pertinent discussion, some users felt that DECUS should obtain stronger rules applying to users' copying and distribution of programs obtained through the library. Though

there is no law which protects the programs, which are public property, members said it would be nice if DECUS created a head or logo least recognize their contributions to program users. This would not only be self-rewarding to the program authors, but would also create a greater awareness among non-DECUS programmers.

In concluding the meeting, Ardoth promised all suggestions would be carefully discussed among Library managers, and promised to let users know results of these discussions as soon as possible.

NEWPOKER: Video Poker Game

Version: V1.0, July 1982

Author: Charles G. Davis, British Petroleum North America Trading, Houston, TX

Operating System: VAX/VMS

Source Language: BASIC

Memory Required: 248B

Special Hardware Required: VT100 or compatible terminal

NEWPOKER is a video poker game that plays more like a real game than any of its predecessors. It uses a full deck of 52 cards, displays the cards instead of telling you about them, and cannot see what is in your hand (until time to compare the hands). It was created for VAX/VMS (any version), but with a few modifications will run on PDP's as well.

It works on any VT100 compatible terminal and uses little memory. Includes a documentation file to instruct the players, the source code (so you can tinker with it) and the executable.

Restrictions: Currently NEWPOKER can not "check" the bet.

Documentation on magnetic media.

Media (Service Charge Code): 600' Magtape (MA)

Format: VAX/ANSI (Blocked at 2048)

Keywords: Game
Category Index: 13
Operating System Index:
VAX/VMS

Symposium Tape from the RSX SIG, Spring 1981, Miami

Version: Spring 1981

Author: Various

Submitted By: Glenn C. Everhart, RCA, Mt. Holly, NJ

Operating System: IAS, RSX-11D, RSX-11M

Source Language: BASIC, BASIC-PLUS, FORTRAN IV, FORTRAN IV-PLUS, MACRO-11, etc.,

The following is a brief description of some the programs to be found on the tape:

New release of RSX/IAS BASIC, including virtual arrays.
DOB - object module disassembler MRH version update
MAS - a CCL type program for IAS systems not using the full DCL
TCS services.
LUT - List logical unit table of a task
FCB - List FCB chain, giving open files on a device
WHE - show where on F4P task built with /TR:ALL is executing (locate loops, etc.)
TTPOOL - displays free space in TT driver pool
HLP - makes MCR HELP facility FORTRAN callable
help files, plotting utilities, etc.
IAS TCP subtasking from high level languages
11M MACRO preprocessor and C runtime library for UNIX standard I/O pack
11M front end to Command Line Interpreter software
11M - program to make multiuser programs
Help file for Evans and Sutherland PS2 program
TEDI - A formatting text editor (does justification, etc.)
LBL software tools (UNIX like Virtual OS in RATFOR) (Release 1)
Runoff for Diablo printers
RNP - Runoff preprocessor for include files, TI: prompts, macros...
DDT22 - a symbolic debugger/core image zapper for any PDP-11 able to operate from a separate task or within task. Can debug any task symbolically with as little as 200 words of space in the task.
VD - package for RSX-11M index
VD Driver
VD - package for RSX-11M-PLUS V1 And much, much more.

No guarantees are made as to the completeness, usabiltity, or quality of the programs on this tape. The material has not been checked or reviewed and documentation may or may not be included for each program.

Restrictions: Available in 1600 bpi only.
Complete sources not included. Documentation on magnetic media.
Media (Service Charge Code): 2400' Magtape (PS)
Format: BRU

Keywords: Symposia, Spring 1981, Miami, RSX-11
Category Index: 18
Operating System Index: RSX-11/IAS

SPEAKR.BAS: A Vented-Box Speaker Enclosure Calculator

Version: October 1982

Author: Jeff Reyer, Digital Equipment Corporation, Hudson, NH

Operating System: OS/8

Source Language: BASIC

Memory Required: 4KW

The SPEAKR program allows you to design a vented (ie. ported) box speaker enclosure provided you know three parameters about the driver:

1. Qts (total driver Q factor)
2. Fs (driver resonant frequency)
3. Vas (compliance equivalent volume)

It calculates the volume of the box (Vb), the diameter of the port (Dv), the length of the port (Lv), the peak or sag in the cabinet response (R), the -3dB cutoff frequency of the box (F3), and the resonant frequency of the enclosure (Fb). All measurements are in International Units: volume in liters, length in centimeters, and response in dB.

This program uses Keele's approximation method.

Restrictions: Originally written for VT100 (ANSI mode) but user may patch one line in sources for either VT52 or hard-copy mode.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA)

Format: OS/8

S1024
830407/

Keywords: Vented-Enclosure, Calculator
Category Index: 17
Operating System Index: OS/8

SPEAKR.BAS: A Vented-Box Speaker Enclosure Calculator

Version: October 1982

Author: Jeff Reyer, Digital Equipment Corporation, Hudson, NH

Operating System: OS/8

Source Language: BASIC

Memory Required: 4KW

The SPEAKR program allows you to design a vented (ie. ported) box speaker enclosure provided you know three parameters about the driver:

1. Qts (total driver Q factor)
2. Fs (driver resonant frequency)
3. Vas (compliance equivalent volume)

It calculates the volume of the box (Vb), the diameter of the port (Dv), the length of the port (Lv), the peak or sag in the cabinet response (R), the -3dB cutoff frequency of the box (F3), and the resonant frequency of the enclosure (Fb). All measurements are in International Units: volume in liters, length in centimeters, and response in dB.

This program uses Keele's approximation method.

Restrictions: Originally written for VT1ØØ (ANSI mode) but user may patch one line in sources for either VT52 or hard-copy mode.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA)

Format: OS/8

Keywords: Vented-Enclosure, Calculator
Category Index: 17
Operating System Index: OS/8

LST: A Paging Utility for Non-Form Feed Devices

Version: V3.0, February 1983

Author: Howard F. Graubart-Cervone

Operating System: RSTS/E V7.0

Source Language: BASIC-PLUS-TWO

Memory Required: 14KW

Other Software Required: RMS file support (optional).

LST is a system utility for printing out text files, in a paged format on non-form feed devices; it also works on form-feed devices. The only format command that need be added to the actual input files is ".PAGE", at the places where a new page is desired. Multiple input files are allowed, as many as will fit on a command line. Special option commands are available to suppress page numbering, set special form lengths, print a title, denote a cover page, pause between page output, and print the number of standard words (5-letters per word) in a file.

The program does not perform any modification of the file itself, i.e. filling, justification, footnoting, etc.

Media (Service Charge Code): Write-Up and Listing (DA)

Keywords: Device Driver, Form Feed
Category Index: 12
Operating System Index: RSTS

new
11-607

MEMO

Version: V1a, February 1983

Author: Mark J. Gilmore, California State University at Long Beach, Long Beach, CA

Operating System: RSTS/E V7

Source Language: BASIC-PLUS

Memory Required: 9KW

MEMO was written to help people who are usually (or constantly) at or near a terminal during the course of a work day. It acts as a computerized 'note box', giving users a quick, convenient way of storing notes about good ideas and/or things to do where they will not be lost (as is frequently the case with paper notes written to oneself). Features include listing memos by subject material, appending to previously entered memos and output to a file.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up and Listing (DA), 600' Magtape (MA)

Format: DOS-11

Keywords: Utility, Memo
Category Index: 7
Operating System Index: RSTS

DCW Menu for RSTS/E Systems

Version: December 1982

Author: Mark DeMoss, Dallas Computer Works Corporation,
Irving, TX

Operating System: RSTS/E V7.0 or later

Source Language: BASIC-PLUS

Memory Required: 16KW

The DCW Menu supports the creation, interactive editing and use of menus with RSTS/E systems.

The programs are written in BASIC-PLUS to run on any RSTS/E system. The DCW Menu includes programs to initialize menu files, add, delete and edit menus, change menu control parameters. The program Menu is used to access menu files.

Each Menu may contain up to 36 items. Longer menus may be divided into 2 or more linked menus, or organized into nested sub menus.

Menu has proven to be easy to learn and use for both user and manager. It includes features to aid development and management, such as:

- password protection of menu items
- disengaging ctrl/c in Menu
- return from program to previous menu
- enforcement of private logical names
- user-private default keyboard monitors

Restrictions: Requires Echo Control Mode

Documentation on magnetic media.

Media (Service Charge Code): 600' Magtape (MA)

Format: DOS-11

Keywords: Menu
Category Index: 7
Operating System Index: RSTS

new
V-SP-10

Library Miscellaneous Package

Version: V1.0, October 1982

Author: Glenn C. Everhart et. al., RCA, Mt. Holly, NJ

Operating System: IAS, RSX-11D, RSX-11M, RT-11, VAX/VMS

Source Language: APL, BASIC, FOCAL, FORTRAN IV, FORTRAN IV-PLUS, MACREL, MACRO-11, PASCAL, TECO

Keywords: Miscellaneous

This package contains a collection of approximately 31 individual DECUS Library program offerings. They are for the PDP-11 and VAX, mainly RSX and VMS applications, with some RT-11 packages as well. For a further description of these programs please refer to the individual program abstracts within this software catalog. These abstracts may or may not express the versions of the programs included within this package. Some of the programs have been or may be revised since the creation of this collection. The following is a list of the DECUS numbers and titles of the programs on the tape:

11-126	ECAP
11-207	MRMLIB
11-280	VBS: IBM to PDP-11 VBS-Format Magtape
11-307	Stage 2 for the PDP-11 Operating under RT-11
11-312	LALR(1) Parser Constructor to Translate Computer Languages
11-322	MARGOT: A MACRO-Based Generator of Command Language
11-337	EXMT: A General Purpose Magnetic Tape Handler
11-369	CALC: An Interactive Computer Language with Unlimited Numerical Precision
11-385	(Partial) PL-11: High Level Assembler Language for PDP-11
11-413	ORC: Object to Macro Conversion
11-415	EXFILE: Exchange File Program
11-433	LISP for RSX-11
11-437	PIP10
11-447	FOCAL RT
11-455	DUPLEX: Serial Communications Between Computers
11-459	ISAM Subroutines Library
11-461	FEP2: A Finite Element Program for Two-Dimemsional and Axisymmetric Three-Dimemsional Continua
11-464	SPACE WAR: For Cursor Addressing CRTs
11-473	FILES
11-475	ALGEB: A Language for Algebra and Number Theory
11-487	DV11/3271 Driver for RSX-11M V3.0
11-512	NDTRAN2
11-SP-6	(Partial)DDT22: Virtual Debugger and Systems Package
11-SP-25	APL-11 for RSX-11M and RSX-11M PLUS
VAX-5	STAT
VAX-15	CALC: A Calculator Program
VAX-16	NDTRAN2

Also included:

11-296 Fast Signal Processing Software Package for the PDP-11 (Note: This program is no longer available seperately through the DECUS Library).

VAX-6 SPICE2 (Note: U.S. Government export regulations prohibit distribution of this program outside of the United States without appropriate export licenses).

LISP for RT-11 is also included, plus some other routines and useful items not present elsewhere (like virtual disk for RSX).

Complete sources not available. Documentation on magnetic media.

Media (Service Charge Code): 2400' Magtape (PC)

Format: RMSBCK with ANSI Labels (Blocked at 2048)
S1041
821227/830216

Keywords: Miscellaneous
Index Operating System:
RSX-11/IAS, RT-11, VAX/VMS
Category: 17

MUMPS

MUMPS Language Tutorial

Dianne Brown
Digital Equipment Corporation
Marlboro, MA

Jim Bernard, Session Chairperson
Kettering Medical Center
Dayton, OH

Reported by Frank Fluke, DECUS Scribe Service

The MGH Utility Multi Programming System (MUMPS) is a high-level interpretive language. Although it is compiled at some stages, it is for the most part interpretive. Its primary focus is on data and string manipulation, therefore it is not very good with numbers. MUMPS is very easy to learn and encourages high programmer productivity.

MUMPS was first developed at Massachusetts General Hospital (MGH) in 1966 as a patient information system language. In 1971, MUMPS-11 was introduced as an operating system and a programming language. In 1972, the MUMPS User Group (MUG) was formed. In 1976 the MUMPS Device Communication (MDC) Standard was accepted; in 1977 it received ANSI recognition. In 1978, DSM-11 was introduced as a replacement for MUMPS-11 and in 1980, MUMPS was introduced for the VAX family.

The Syntax for MUMPS is much like BASIC. It is possible to have multiple command argument sequences on a line up to 255 characters long. Line labels are optional. Commands can also have multiple arguments separated by commas. Some of the more interesting features of the MUMPS language are reviewed below.

Unlike other languages, MUMPS evaluates operators left-to-right regardless of any predefined hierarchy. There are no data types to be concerned about because all input variables are strings. In string comparison with the IF statement, MUMPS uses a "[" symbolizing a CONTAINS command or a "]" to symbolize a FOLLOWS command.

One of the particularly strong features of MUMPS is Pattern Verification, symbolized by a "?", for checking input. It uses the following symbols:

N	Numerics Ø to 9
U	Upper Case Alphabetics A to Z
L	Lower Case Alphabetics a to z
P	Punctuation
A	Any Alphabetics A to Z and a to z
C	Control Characters
E	Everything

For example:

```
IF SSN ?3N1"-"2N1"-"4N WRITE "VALID SOCIAL SECURITY NUMBER"
```

checks to see if the given SSN is written correctly.

Another interesting feature of MUMPS is that all commands can be abbreviated to the first character of the command. Although this may make the code seem very cryptic, it can speed up programming once the basic commands are mastered.

Here are a few more characteristics of the language. MUMPS provides a Post Conditional Syntax, symbolized by a ":", which implies the "IF" command, but insures that the rest of the line will be executed even if the implied IF is false. There is a system variable, "$TEST" (all system variables begin with "$"), which contains the truth-value of the most recent IF. It is set to 1 if the value is true and Ø if the value is false. Since declared local variables remain until the program finishes, a KILL command is provided to delete local variables. You may delete all local variables, only one variable, or all of the variables but one.

MUMPS provides for multilevel subscripting and tree-structured arrays. Nodes in the tree do not necessarily have to contain data, the array size does not have to be defined, and when any new node is created the storage space is allocated and the subscripts are sorted. For this mode of data handling, the $NEXT function is used to return the subscript of the next sibling (the next node to the right on the same level) to a specified node. If no next sibling exists, then a -1 is returned. (Because of the danger of using a -1 as a flag, a new function, $ORDER, has been defined. It will return the null string if no sibling exists.) $NEXT basically walks around the tree structure. The $DATA function returns a value indicating whether or not the specified node has a defined value and/or descendents.

One final interesting attribute of MUMPS is the $ZTRAP system variable. It eliminates the problem of cryptic error messages by "trapping" a message within the system variable so that when the error occurs the "trapped" message will appear. The remaining system variables can only be read, not written to.

Writing Reports with VAX-11 DATATRIEVE

Seymour Kellerman
Digital Equipment Corporation
Nashua, NH

Robert R. Lott, Session Chairperson
E.I. DuPont
Nashville, TN

Reported by Susan Miller, DECUS Scribe Service

Seymour Kellerman of Digital Equipment Corporation said that reports are needed to display a subset of data, to format data, to group records and to summarize data. DATATRIEVE reports can feature a report name, current date, page numbers, column headings, detail lines and summary lines. Kellerman gave a number of examples which showed that DATATRIEVE can produce almost any type of report a company might require.

Kellerman gave examples of five reports based on a fictional company. He used two domains, NEWHIRE and SALES83. NEWHIRE consisted of data on new employees, such as ID, name, company division, company location, starting date, months employed, birth date and age. The SALES83 domain used ID, sales date and sales amount.

The following field structure was used for a NEWHIRE record with the ID index key providing querying.

```
DTR>  SHOW NEWHIREREC
RECORD NEWHIREREC USING
Ø1 NEWHIREREC.

        1Ø  ID  PIC IS 9(5)
                EDITSTRING IS 9(5).

        1Ø  EMPLOYEENAME          PIC IS X(2Ø).

        1Ø DIVISION          PIC X(1Ø)
                                 QUERYNAME IS DIV.

        1Ø  LOCATION         PIC X(12)
                                 QUERYNAME IS LOC.

        1Ø  STARTDATE  USAGE IS DATE
```

```
                        DEFAULT VALUE IS "TODAY"
                        EDITSTRING IS MM/DD/YY.

            10  BIRTHDATE  USAGE IS DATE
                        EDITSTRING IS MM/DD/YY.
```

Data was displayed from the contents file for the NEWHIRE domain. Output examples have been shortened to save space.

```
DTR> READY NEWHIRE
DTR> PRINT NEWHIRE

ID     EMPLOYEE NAME       DIVISION    LOCATION     START      BIRTH

ØØØ12 CHARLOTTE SPIVA      SOFTWARE    MILWAUKEE    9/12/72    1/23/4Ø
ØØ891 FRED HOWL            SOFTWARE    ST LOUIS     4/Ø9/76    4/2Ø/32
Ø2943 CASS TERRY           MARKETING   PORTLAND     1/Ø2/8Ø    6/28/2Ø
11111 ANNE DINNAN          SALES       MILWAUKEE    4/Ø1/82    1/24/25
```

Because the output was not easy to read, it was broken down into a subset of records which were sorted to make it "a little bit more comprehensible", said Kellerman.

```
DTR> PRINT NEWHIRE WITH DIVISION = "SOFTWARE", "MARKETING"
SORTED BY
CON>    DIVISION, LOCATION

ID     EMPLOYEE NAME       DIVISION    LOCATION     START      BIRTH

32432 THOMAS SCHWEIK       MARKETING   MILWAUKEE   11/Ø7/31   1Ø/1Ø/39
78923 LYDIA HARRISON       MARKETING   MILWAUKEE    6/19/79    8/29/54
87465 ANTHONY IACOBONE     MARKETING   MILWAUKEE    1/Ø2/73    9/2Ø/39
Ø2943 CASS TERRY           MARKETING   PORTLAND     1/Ø2/8Ø    6/28/2Ø
```

To produce a control group report, identify and sort data. Then divide the detail lines into groups and print the detail lines. Finally, summarize data on each group and for the entire report. The following specification was used to divide employees into subsets selected division and location:

Report Specification: Employees by DIVISION and LOCATION

```
DTR> SHOW NEWHIREREPORT
PROCEDURE NEWHIREREPORT
REPORT NEWHIRE WITH DIVISION = "SOFTWARE",---------------->(1)
    "MARKETING" SORTED BY DIVISION, LOCATION
SET COLUMNSPAGE = 7Ø
SET REPORTNAME = "EMPLOYEE REPORT"
AT TOP OF DIVISION PRINT DIVISION ------------------------->(2)
AT TOP OF LOCATION PRINT LOCATION ------------------------->(2)
PRINT ID, EMPLOYEENAME, STARTDATE ------------------------->(3)
AT BOTTOM OF DIVISION PRINT SKIP, ------------------------->(4)
     "TOTAL EMPLOYEES IN"|||DIVISION|":",
     SPACE, COUNT USING Z9, SKIP
```

```
AT BOTTOM OF REPORT PRINT SKIP, "TOTAL EMPLOYEES:", ------->(5)
     SPACE, COUNT (-) USING Z9
ENDREPORT
ENDPROCEDURE

DTR> :NEWHIREREPORT
```

The output on the employee report begins as follows:

EMPLOYEE REPORT

DIVISION	LOCATION	ID	EMPLOYEE NAME	START
MARKETING				
	MILWAUKEE			
		32432	THOMAS SCHWEIK	11/Ø7/81
		78923	LYDIA HARRISON	6/19/79
		87465	ANTHONY IACOBONE	1/Ø2/73

Another example shown was the employee age range report. The preliminary work included defining a COMPUTED BY field. One trick of the procedure report is to define an age range, and use a variable to compute the decade of an age. Sort the records by AGE and then group them according to AGERANGE.

Report Specification: Age Range Report

```
DTR> SHOW EMPLOYEEAGE
PROCEDUKRE EMPLOYEEAGE
DECLARE AGERANGE COMPUTED BY ---------------------------->(1)
     FN$FLOOR (AGE/1Ø) EDIT STRING IS 9Ø.
REPORT NEWHIRE WITH DIVISION = "SOFTWARE", -------------->(2)
     "MARKETING" SORTED BY DIVISION, AGE
SET COLUMNSPAGE = 7Ø
SET REPORTNAME = "AGE OF EMPLOYEES"
AT TOP OF DIVISION PRINT DIVISION
AT TOP OF AGE RANGE PRINT AGERANGE ("EMPLOYEES"/OVER") --->(3)
PRINT EMPLOYEENAME, BIRTHDATE
AT BOTTOM OF DIVISION PRINT SKIP, COL 18,
     "AVERAGE AGE OF EMPLOYEES IN"|||DIVISION|":",SPACE,
     AVERAGE AGE USING 99, SKIP
ENDREPORT
ENDPROCEDURE

DTR> :EMPLOYEEAGE
```

Then the output is grouped by employee age ranges with the report based on dates.

AGE OF EMPLOYEES

```
DIVISION     EMPLOYEES OVER        EMPLOYEE NAME        BIRTH DATE

MARKETING
                  20
                                    DAN ROBERTS           3/21/59
                                    LOUISE DEPALMA       11/20/54
                                    LYDIA HARRISON        8/29/54
```

The SALES83 domain used the following field structure:

```
DTR> SHOW SALES83REC
RECORD SALES83REC USING
01 SALES83REC.

    05 SALEDATA.
       10 ID            PIC IS 9(5).
       10 SALESDATE    USAGE DATE
                        EDITSTRING IS MM/DD/YY
                        DEFAULT VALUE IS "TODAY".

    05 AMOUNT        PIC IS 9(5)V99
                     EDITSTRING IS $$$,$$$.99.
```

To do the sales division report, define a related record. A flat or hierarchial record may be used. Kellerman recommends using a flat record. "Queries come out much easier," with flat records, he said, that it's "easier to access data and output it." After choosing a record, the user should decide if he wants a simple or compound key. He should also decide how the various fields should be arranged.

To produce a sales report ordered by location and employee you should: identify the data, join two domains, and sort by LOCATION and ID, set up a heading for each location, group the records, and calculate summary data for each employee. Finally, calculate summary data for each location and all employees.

Data is needed from both domains to make an "all totals" report. Following is an example of the steps involved:

Report Specification: An All Totals Report

```
REPORT SALES83 CROSS NEWHIRE OVER ID SORTED BY LOCATION,
ID ------->(1)
SET COLUMNSPAGE = 70
SET REPORTNAME = "EMPLOYEE SALES REPORT"
AT TOP OF LOCATION PRINT COL 1, LOCATION, SKIP ---------->(2)
(Column headers should also be specified)
AT BOTTOM OF ID PRINT COL 13, ID, COL 20, EMPLOYEENAME, -->(3)
    COL 41, COUNT ("M"/"O"/"S") USING Z9,
    COL 46, TOTAL AMOUNT USING $$$,$$$.99,
```

```
    COL 60, AVERAGE AMOUNT USING $$$,$$$.99
AT BOTTOM OF LOCATION PRINT SKIP, COL 25, LOCATION|||"TOTALS;", ----->(4
)
    COL 45, TOTAL AMOUNT (-) USING $$$$,$$$.99,
    COL 60, AVERAGE AMOUNT (-) USING $$$,$$$.99, SKIP
AT BOTTOM OF REPORT PRINT SKIP, COL 30, "SALES FORCE:", --->(5)
    COL 45, TOTAL AMOUNT (-) USING $$$$,$$$.99
    COL 60, AVERAGE AMOUNT (-) USING $$$,$$$.99
ENDREPORT
```

One peculiarity of this report is that it doesn't include a detail line, since we are only interested in each employees total sales. The output results are the employee total sales report:

EMPLOYEE SALES REPORT

LOCATION	ID	EMPLOYEE NAME	MOS	TOTAL AMOUNT	AVERAGE AMOUNT
MILWAUKEE					
	11111	ANNE DINNAN	4	$10,096.31	$2,524.08
	22222	NANCY ROTHBLATT	4	$23,130.17	$5,782.54
	34567	LYDIA BARNET	4	$22,221.20	$5,555.30
	44444	WAYNE SMITH	4	$40,943.69	$10,235.92
	88888	JOSEPH FREDERICK	4	$53,944.00	$13,486.00
		MILWAUKEE TOTALS:		$150,335.37	$7,516.77

Following is the procedure to follow for a sales commission report:

PROCEDURE FOR A SALES COMMISSION REPORT

```
DTR> SHOW SALESCOMM
PROCEDURE SALESCOMMV2
READY SALES83, NEWHIRE
FIND SALES83 WITH FN$MONTH (SALESDATE) = *."month in numbers"
DECLARE COMMPCT COMPUTED BY
    CHOICE
        MONTHSEMP LE 10 AND AMOUNT > 5000 THEN 10
        MONTHSEMP LE 10 THEN 5
        AMOUNT > 10000 THEN 12
        ELSE 7
    ENDCHOICE
         EDITSTRING IS Z9%.
DECLARE RATING COMPUTED BY
    CHOICE
        MONTHSEMP LE 10 AND AMOUNT > 5000 THEN "ABOVE QUOTA"
        AMOUNT > 10000 THEN "ABOVE QUOTA"
        ELSE "BELOW QUOTA"
    ENDCHOICE.
DECLARE COMMISSION COMPUTED BY (COMMPCT * AMOUNT)/100
         EDITSTRING IS $$,$$$.99.
```

```
REPORT CURRENT CROSS NEWHIRE OVER ID SORTED BY COMMPCT
SET COLUMNSPAGE = 70
AT TOP OF PAGE PRINT COL 10,
     "SALES COMMISSION FOR MONTH OF"|||FORMAT SALESDATE USING
     MMM-YYYY, SKIP 2, COLUMNHEADER
AT TOP OF COMMPCT PRINT COMMPCT, RATING
PRINT EMPLOYEENAME ("NAME"), MONTHSEMP, AMOUNT, COMMISSION
AT BOTTOM OF COMMPCT PRINT SKIP 2, COL 10,
     "TOTAL AT"|||COMMPCT|"%:", SPACE, COUNT (-) USING Z9,
     COL 40, "TOTAL COMMISSION:",
     TOTAL COMMISSION USING $$$,$$$.99, SKIP 2
AT BOTTOM OF REPORT PRINT COL 10,
     "************************************************",
     SKIP, COL 10, "TOTAL SALESMEN:", SPACE, COUNT (-) USING Z9,
     COL 35, "GRAND TOTAL COMMISSION:",
     TOTAL COMMISSION USING $$$$,$$$.99
ENDREPORT
ENDPROCEDURE

DTR> :SALESCOMM
Enter month in numbers: 5
```

After completing the above procedure, a reports of employee sales commissions was produced. The report was sorted in decreasing order of commissions.

SALES COMMISSION FOR MONTH OF MAY-1983

COMM PCT	RATING	NAME	MONTHS EMP	AMOUNT	COMM
5%	BELOW QUOTA				
		ANNE DINNAN	9	$2,389.90	$119.50
		JOSEPH FREDERICK	10	$5,000.00	$250.00
		RICK LANGHART	9	$4,999.99	$250.00
		LYDIA BARNET	7	$2,598.79	$129.94
	TOTAL AT 5%:	4	TOTAL COMMISSION:		$749.43

A total sales report can also be created. First, define a temporary file to accept the totals data for each employee. The SUPERSEDE option will delete the previous file. Then generate a record stream of unique values for ID. Store the totals for all the records of SALES83 matching on ID. After that, sort the totals records by descending values for TOTSALES, so that the top salespeople will be listed first. Finally, report on the totals records of SALESTOTALS joined with the detail records of SALES83.

Field Test

Field Test: What is it and How Do We Pick Sites

Stephen R. Beason
Digital Equipment Corporation
Maynard, MA

Angela J. Cossette
Digital Equipment Corporation
Maynard, MA

Thomas E. Davis
Digital Equipment Corporation
Maynard, MA

Emily Kitchen, Session Chairperson
A.H. Robins Company
Richmond, VA

Reported by Margaret Watters, DECUS Scribe Service

Three DIGITAL representatives gave a successful talk on what is a field test, and how companies can become field test sites. Stephen R. Beason, the Central Quality Group Manager began the session by giving a general discussion on what constitutes a field test, why they exist, and why a customer would want to become involved in one.

A field test is the last step in the qualification period. The main objective of a field test is to increase the quality of DIGITAL's products. What cannot be tested in the DIGITAL environment, can be tested in the field. The field test also is helpful, because there are uses of products that DIGITAL had not considered which may create some problems in the product that DIGITAL can then look into. DIGITAL can also evaluate to what extent the customer's requirements and expectations have been met by the new product.

The customer would benefit from a field test at his site, because he would be able to be directly involved in the development of a product that he uses every day, and thereby to increase the quality of that product. Also, there is the possibility that the site will experience a competitive edge by having the product in use up to a year earlier than the shipping date.

Thomas Davis, a Software Services Representative, and another speaker, felt that the major advantage to field tests is the increased customer satisfaction they foster. He also mentioned the general guidelines that DIGITAL uses to choose test sites. Since DIGITAL wants real testing of their products, the main criteria is that the site will test the product well, and that it have a technical staff that will be able to communicate effectively with the DIGITAL engineers. It is also important for the site to have the necessary resources for the product to be tested (hardware etc.). He also said that the site must have a good relationship with DIGITAL.

Angela Cossette, the Field Test Administrator, discussed how the field test works. The three major segments of the process are setting up, testing, and termination. During the testing period the people working at the site send QARs to DIGITAL which are worked on immediately. After the termination period, a questionnaire must be filled out. In terms of the cost to a chosen site, the site must spend the time using, testing, and providing feedback, but the use of the product is otherwise free. One member of the audience said that the site must also pay transportation costs associated with attending a training seminar, but there was no comment from the speakers, so the matter remained unclear.

A site that wishes to become a field test site should get in touch with its local Software Services Manager or with Angela Cossette directly: Digital Equipment Corporation, P.O. BOX F, Maynard, MA, 01754.

Writing User Manuals

Writing User Manuals-A Workshop

Patti A. Petry
The Singer Company
Silver Spring, MD

Douglas Bickford, Session Chairperson
University of Vermont
Burlington, VT

Reported by Phil Beene, DECUS Scribe Service

The preparation of in-house user manuals in a frequently undertaken and sometimes disastrously performed task. When writing user manuals, careful planning and efficient allocation of time can result in a written product which will be better understood and read by more people.

Helping other DECUS members gain a better understanding of structured documentation during the recent symposium in St. Louis was Patti A. Petry, Principal Document Developer for the Singer Company's Link Simulation Systems Division.

In her work with the Singer Company, Petry produces in-house user manuals for various military and private users, some of them adding up to 30,000 pages and requiring 50 writers.

The key to success in producing such mega-documents, Petry says, is careful completion of a formal outline before ever writing a single word of copy. "Planning is the most crucial step in smoothly and efficiently producing quality documentation," she said.

Petry says Singer always follows the below plan in producing outlines for their documents:

1. Initial Planning. Always know what you are going to do.

2. First Draft. During this process, the project engineer edits the initial plan and sends his results to the EDG.

3. Cursory Edit. All people involved in the project go over the finished outline.

4. Word Processor. Outline is actually typed into a printed form.

5. Complete in-depth edit. The staff go over the printed document, making changes where needed.

6. Walkthrough. In this stage, the author, project engineer and section manager go through the copy together, discussing the total overview of the outline and making changes.

7. Edit Changes. Editors edit the changes made in the walkthrough.

8. Word Processor. The new version of the outline is printed into hard copy.

9. Project Engineer Review. P.E. is in charge of the total project, and he reviews the copy to make sure the outline is following the original strategy.

10. Printed/Shipped. The outline is printed and shipped to the management of the company or government branch producing the manual.

11. Government Edit. During this stage, the outline is checked and edited by the government branch, etc. producing the manual. (Singer does most of its work for the U.S. Navy.)

12. During this final stage, the outline is finally ready to be written into the manual. All changes must be made and approved before any copy is written.

By following the above outline, Petry says both time and manhours are saved. This is especially important when dealing with engineers and other technical writers who often require long hours and much effort to produce properly written copy. According to Petry, outlining the manual will also eliminate the wasteful problem of overwriting.

Another practice Petry says is successful within the Singer company is the creation of an author's manual for each project. When writing a thousand-page document with several authors, keeping the entire work in track can be difficult. The author's manual will standardize goals of the project, abbreviations included in the work and keep the work within government standards. This constantly updated manual will save many mistakes and keep the staff organized, resulting in time and money saved.

Petry discussed several other ways to keep staff efficient. Among her suggestions were creating an in-house style book and sending company engineers through technical writing refresher

courses at periodic intervals.

Following Petry's presentation, Douglas Bickford, session chairman from the University of Vermont, led the audience in an open discussion to create new ideas and input.

The question raised most often was how to create user interest in the manual itself. Good indexing, a good table of contents, bright graphics, running examples throughout the text and a good understanding of the user audience were all listed as key factors to the manual's success and continued use.

LN01 Page Printer

Ron Iversen
Digital Equipment Corporation
Maynard, MA

Don Feinberg
Digital Equipment Corporation
Maynard, MA

Louis Benton, Session Chairperson
Staff Computer Technology Corporation
San Diego, CA

Reported by Gene Mitchell, DECUS Scribe Service

The session focused on the DIGITAL LN01 page printer. According to DIGITAL's Ron Iversen, this printer falls into the third major category of available printers, those that produce less than 20,000 pages per month. The LN01 itself produces about 15,000 pages per month.

Iversen listed some applications for the LN01, including data processing, centralized word processing, and distributed cluster printing. DIGITAL's priorities are as follows:

1. High speed, reliability, two-sided printing, and little need to have someone attending the machine at all times.

2. High quality, some copying.

3. Compact size and convenience.

Market trends today, according to Iversen, are pointing towards non-impacting, high quality printers. In addition, laser technology is in demand, with capabilities in graphics integration.

LN01 hardware was discussed, features including:

1. Low-noise due to the non-impact printing (<55dB)

2. Approximately 12 pages per minute.

3. Compact resolution:300x300 dots/in.

4. Several paper sizes including 8.5x11 and 8.5x14.

5. High speed, making it a good replacement for multiple LQPs.

6. LP11 or DMF-32 interface compatibility.

7. Greater graphics functionality.

8. Two ROM resident mono-spaced electronic fonts.

9. Two printing modes: (a) Portrait mode--Titan style (SERIF); point, 10 pitch. (b) Landscape mode--sans SERIF style; 6-72 point, 13.6 pitch.

10. Type-faces: Regular, italic, bold, bold italic, 6-36 pt.

The machine's major limitation is its limited capability with sophisticated graphics. According to co-speaker Don Feinberg of DIGITAL, this should improve in the future.

The support date for the following operating systems is projected for sometime in June of 1983: VMS V3.3, RSX-11 V4.1, RSX-11+V2.1, RSTS/E V7.2 and V8.0.

The software package was summarized by Feinberg, who he emphasized that the LN01 is a page-printer, printing only in full-page increments. Fonts are stored as arrays of arbitrary characters. 1700 unique characters can be printed on each page. The microprocessor in the LN01 reads input band-by-band. This prohibits font-loading during paper motion. The software required falls into two types:

1. "Creator" software offers direct user interaction, producing an "image" ready to send to the printer.

2. System software manages the printer itself and separates jobs.

Feinberg summarized by saying that the development of DIGITAL's printers is directed toward high-quality capabilities with laser technology and strong graphics.

LA12/LA100 Functions & Features

John L. Davis
Digital Equipment Corporation
Maynard, MA

E. Jory
Digital Equipment Corporation
Maynard, MA

Reported by Frank Fluke, DECUS Scribe Service

This article is a discussion of the LA12 and LA1ØØ, which covered the construction of characters using dot matrix techniques. Particular emphasis was places on what can be done on the LA1ØØ and the construction of personal fonts, binary image mode graphics or pixels, and new technical aspects of the LA12 and LA1ØØ products. The first part of the discussion centered on dot matrix techniques for the LA1ØØ. These techniques consist of two parts, the decomposition of characters into small dots to get the image and the printing of those images.

The next entity introduced was the character cell, including its placement and definition. The size of a character cell is called the pitch. Pitch refers to both horizontal and vertical character size and is often confused with spacing. Resolution is the distance between possible dot positions within the character cell. For example, resolution influences how well diagonals can be drawn and whether they look like straight lines or steps. The LA1ØØ has a resolution which is 1/14 of the cell size in the vertical direction. The LNØ1 has a resolution of 1/3Ø. The VT1ØØs is 1/1Ø. When discussing printer resolutions you are talking about much greater resolutions because of the importance of hardcopy clarity which is often used for public display as opposed to video output. In the horizontal direction, the LA1ØØ (1/33) has a finer resolution than the LNØ1 (1/3Ø), while the VT1ØØ remains at 1/1Ø.

The next important parameter is dot size. On the LA1ØØ the dot size is 13 mil. On the LNØ1 it is 3 1/3 mil. The dot size has two influences:

1. The finer the dots, the more detail that can be shown.

2. The finer the dots, the more dots that are needed.

Certain printer limitations must be faced, since the printer which puts the most dots on paper most augments the stress. It also increases the load on the power supply, heat dissipation requirements, storage requirements for additional features, CPU requirements (to process all the dots), repetition rate for mechanical devices, and how fast the laser beam in the should be shut on and off. Thus, a compromise must be found between resolution, dot size, and the stress put on the printer.

One of the ways to save storage space is to avoid storing the white space which is at the right or at the bottom of the character. This is not always suitable for situations where extremely high resolution is required, since the quality of the output can be adversely affected. Another way to reduce storage requirements is to use algorithms which reduce the number of dots needed to form a character. For instance, the number of necessary dots can be cut in half by leaving a space to the right of each dot instead of having two dots adjacent to each other. This would not affect vertical quality, diagonals, or curves yet it cuts the stress on the printer in half, thus allowing speed to double. This can be carried one step further for even less quality and more speed. For medium resolution, start dropping dots in the vertical direction. Although this which does affect diagonals and curves, it permits a further doubling of speed. The purpose of all this is to open up an infinite number of characters.

This brings up the notion of a Dot Pattern Set (DPS). The DPS consists of a self-contained part (94 characters plus the auxiliaries) and a secondary part which contains a list of differences to be applied to the primary. In the DPS are stored all character patterns, all character set tables, all the character set selection data, and all supported attributes. This allows support of any new fonts. When a new character set is created, its character set table is placed in the DPS. Therefore, any existing machine is able to support it. Dot patterns can be modified in two ways:

1. New patterns can be defined(gives best quality).

2. Space saving algorithms can be used (saves on number of designs needed).

Q. What are the differences between the RA versions and the PC versions?

A. There are a lot of misconceptions going around. The differences between them are as follows:

1. We changed the code on the PC to 8-bit code. The standard code that we use today is a subset, so if you have either one of those and use 7-bit code

there should be absolutely no change. The only place that should show a difference is in Europe.

2. The second difference is in the graphics aspect ratios. In the PC version we have included a choice of, I think, ten different graphics aspect ratios. The only one of our personal computers that wasn't quite right was the professional. It works, it prints, but circles came out a little flat and squares came out a little rectangular. The PC version corrects that.

3. The third difference is the EMIRFI standards. The RA version is FCC Class A and the PC is FCC Class B which is more stringent. Of the changes that can be made, the ROMs are interchangeable. The reason that people tell you that you can't make it into a PC is because you still haven't made it FCC Class B. You can make it work the same but its not FCC Class B because that takes some power supply changes and that's a little more extensive.

Concerning fonts, we are going to make custom fonts which I believe is the most cost-effective way to do it. The turnaround time is fast but we cannot be in the business of creating ones and twos for everybody. For someone with a little bit of volume we can do it. For people with any custom fonts or symbol sets, we should have the capability to make those for you within the next month or two.

Some terms must be defined before discussing binary image graphics. A pixel is defined as the smallest displayable dot. It is a spot of light on the screen and a blob of ink on the printer. The grid is the positions where you can place the pixels. On the printer we talk about dots per inch and on the video we talk about dots per screen. Resolution (previously defined) is understandable in hardcopy devices but is not understandable in video screens. Aspect ratio is the ratio between width and height. Overlap (when the spot size is significantly larger than the pixel) does not happen on the video screen because the dot on the screen is the same size as the pixel. Protocol is a way of encoding.

Some key characteristics have come into play in the graphics area and they turn out to be 'gotchas' when it comes to drawing things, so the differences need to be understood. With pixels, in the case of the printer, there exists a real problem. On DIGITAL terminals, the size of the wire can't be changed very easily on the fly, so it's pretty much constrained. The pixel is defined by the size of the wire and the amount of ink that gets put on paper. In the video, the pixel size is relative to the screen so if there is a large screen there is a large pixel. Horizontal resolution is determined by physical measurement and

on the video it is a simple electron beam. The vertical resolution is in fact fixed by the vertical step (How much can the stepper motor be moved?), and that is 1/48" which turns out to be 72 dots per inch. There is an overlap between lines in that if 72 dots per inch is divided out, it is slightly smaller than the wire diameter. So there is a little bit of an overlap. There are some accuracy constraints from how the paper is moved around and the video has none of these problems because it is all relative to the screen. These are all key constraints.

In developing the protocol, it was to be; compatible with supported standards, compatible with the syntax, and able to pass information. The Device Control String (DCS) was chosen and the syntax was filled inside the DCS. This simply outlines what cannot be done. The printable data portion is a simple transformation where the received character code is transformed into binary data. The leading parameter in the DCS is used to define the aspect ratio. That is a key piece of information when interpreting pixels or any image like them. There are some aspect ratios that go from 5 to 1 down to 1 to 1. Pixels, then, are nothing more than a way of passing around a nice, clean image. This is great from the printer side because no bit map is needed.

Q. I have an LA12 and I tried to print something off of my VT125 with it, but I couldn't find a way to tell the printer using the keyboard to print out to the printer port. How can this be done?

A. The VT125 option plugs into the serial line of a VT100. There is no way, if you plug into a serial line, to get to a key and redefine it to be "print the screen". What they have provided is a kind of kludge mechanism so it can be done. What you do is simply enter the command screen to print the screen in your answer-back message, and then hit control-break whenever you want to print the screen image and it will come out. That's the hack.

Q. But only graphics?

A. Right, it only prints the graphics portion.

The LA100 is very flexible and can adapt to almost any kind of environment. It can be; desk-top or stand-mounted, fitted with an acoustic cover to reduce noise, and fed using rear or bottom tractors or with roll paper in the back. The power supply is an international power supply that can be run between 85 and 250 volts, and between 43 and 67 hertz. The LA100 is Class B certified so it can be used in the home.

Communications are also flexible. The LA100 can be connected to the host using any EIA connection, a 20 mA connection, a modem, or the parallel interface. It has 14

stretch speed, 8 split speed, 8 character format, and can handle any kind of stop bit. The transmittal rate is limited to 60 CPS so it will not overload any host. It also has programmable answer-back, enabling or disabling of automatic answer-back, and coded disconnect. It supports XON/XOFF and busy. It has an input buffer which goes between 400 and 4K and supports long and short breaks for communication ending in a manner which is compatible with both U.S. and European requirements.

Flexibility is also found in form handling where the LA12 and the LA100 are much improved over the LA34 and the LA120.

Printing flexibility allows four print modes: draft, memo, letter, and graphic. There are 8 horizontal pitches in draft, 2x2 horizontal pitches in letter, and 8 graphic pitches. It has an infinite number of print styles and character sets. There are 270 resident characters and 180 characters either resident or plugged in, for a total of 450. The style as well as the quality may be selected either from the host or from the front panel. To improve speed, the LA100 handles auto-underline and does underlining on the fly.

There is flexibility in the way it handles the keyboard. The LA100 comes in both models RO and KSR. In KSR you can have: auto-repeat or not, keyclick or not, 8 keyboard configurations, disabled break key or not, and the RETURN can be a standard carriage return or a new line command.

The LA100 can be used as a line printer, a word processor, a standard console, a graphic printer, or a personal computer, each with many of the features adaptable to the chosen environment. It is compatible with the LA120, LA34, LA12, and the LA15, and feeds on the DECmate, PC, VT100, and the VT125. It works under both RSTS and VMS.

Some of the changes in the new generation - LA12 and LA100 - were stressed, such as form handling and the ability to handle the new DIGITAL 8-bit protocol which mainly doubles the extension of a repertory of controls and extends the number of characters that can be accessed with a single code by another 94. This reduces the data flow and increases the throughput of the system.

In order to set up the LA12, do the following:

1. Type the first character of the major piece, like "C" for Communications, etc.

2. Type the first character of the subcommand that is desired to understand or change. If you hit RETURN, it will tell you what can be done with that command, what the variations are, and then it will tell you the way it is set at the moment.

3. The last thing is the dialer, and if you type "D" you

can start entering the numbers. The essential thing about the dialer is that it gives auto-dialing capability and storage for up to 26 numbers. The amount of characters per number varies depending on the numbers that you have. There is a fixed limit of 128 characters and there is an extra character needed for every number that you want to put in there.

The other things added in the version 2 code for the LA12 are; some bold printing capability, the variable aspect ratio, graphics to be compatible with the PCs, and an updated APL character set.

Keyboard dialing is actually very easy to do. To edit the keys a little, a simple editor is provided along with the facility to enter the digits, some separators so it looks pretty, and some wait characters for secondary dial tone detects. The keyboard dialer is pulse dial only and we did that to provide the most universal base thought possible.

Q. Is that a new ROM set?

A. The keyboard dialer is combination logic board and new ROM set. It was not the base board, it was the modem board we had to go in and change to put in the dialer circuitry.

Q. When is that coming out on the LA12?

A. About the first of July. The way the dialer works is that any time you are online and not in set-up mode, if you type "D", you go into dialer mode and it prompts and says, "You're in dialer mode. What do you want to do?" It's that simple.

Writing C for VMS & UNIX Systems

David Moore
Digital Equipment Corporation
Nashua, NH

Jim Livingston, Session Chairperson
Measurex Corporation
Cupertino, CA

Reported by Todd Spangler, DECUS Scribe Service

There are several problems when writing portable C code. There is no formal standard for C. Confusion exists between boundaries of C and UNIX and there is a general lack of awareness of the problem. David Moore of Digital Equipment Corporation described the differences between UNIX C and VAX/VMS C. VAX/VMS does not use UNIX-style file specifications, fork and exec sort utilities on standard I/O, command line parsing (piping and file redirection), and other routines not in the emulation set. On VMS, RMS provides an RTL stream which is compatible with that of the UNIX stream system. Exec-family RTL routines only provide sharable reads between parent/child directories and only initialization with "=" is supported. Also supported are the += and *= operators.

To help maintain the portability of C code, it is advisable to keep track of size of data items. On the PDP-11, a long variable is equivalent to a short variable as far as memory is concerned. On VAX, a short variable is half the size of a long variable. The memory order and continuity must also be kept track of as VMS can have variables that exist but that do not exist in storage. When refering to this variable with a pointer, there will be an error message created since the variable cannot be found. On VMS, the layout of program address space is important, especially uninitialized pointers, end, edata, and etext. In UNIX, there is a zero pointer, but on VMS this pointer is protected. Unlike UNIX, characters are signed on VMS. Pointer/integer exchanges are possible, but not portable due to size conflicts. External identifiers on VMS are 31 bits long and on UNIX are 6 to 8 bits long. Unlike VMS, it is possible to have holes in the structure alignment. The order of operation on VMS is only forced when using the COMMA, logical AND,and bitwise or logical OR. To be safe, one should not rely on character set dependencies (VMS uses 7 bit ASCII). VMS does not have an ASM program.

Basically, rules to follow in making C code as portable as possible are to use DEFINE as much as possible and stay away from manifest constants. Make use of common header (.h) files in order to keep some record of system dependent constructs, use SIZE OF to keep track of data, and use common routines with all C libraries.

When writing non portable C, one should choose the right support environment. Making use of all the compiler capabilities is important. Using symbols like 'vax', 'vms', and 'vaxlic' will identify the environment to the compiler, while using include text libraries will improve optimization. Allowing the compiler to manage temporary files and assign registers will increase portability. Using constant folding, DEFINE, MODULE, INCLUDE libraries, passing constants by reference, compiler listings including symbol tables, cross references, and preprocessor substitutions will also be helpful. The proper use of PERROR RTL routines to diagnose errors is desirable. With these things in mind, one can consider the system, use the suggestions, and be on the road to writing portable and non-portable C code.

UNIX Hints & Kinks

Armando P. Stettner
Digital Equipment Corporation
Merrimack, NH

Dorothy Geiger, Session Chairperson
Cal Poly
San Luis Obispo, CA

Reported by Dorothy Geiger, DECUS Scribe Service

The UNIX Hints and Kinks session was led by Armando P. Stettner and included panelists Joe Sventek, Norman Wilson, Bill Burns, and Vance Vaughn. Questions were as follows:

Q. Does UNIX have support for networking?

A. VMS/UUCP is being examined, as is DECnet for VAX/UNIX. Software is available to support "tar" under VMS and TCP/IP under UNIX.

Q. Does 4.2Bsd have new networking software?

A. 4.2Bsd provides rich support for interprocess communications between machines, including named sockets for servers, TCP/IP, and Ethernet. In addition, processes do not require the same parent for interprocess communications.

Q. What is the Software Tools Group's Virtual Operating System?

A. VOS is NOT UNIX, it is an entire program development environment which is based on the Software Tools book by Kernighan and Plauger. It provides a variety of shells, tools and utilities, and has been implemented for various operating systems such as RSX and VMS. The VOS software is available on the DECUS SIG tapes.

Q. Will DIGITAL's release of UNIX have system performance and monitoring tools?

A. DIGITAL's current release includes "vmstat", "iostat" and kernel profiling. There are no firm current plans for more.

Q. Can files be accessed across the Network?

A. 4.2Bsd provides for symbolic "links" across network nodes. In addition, work has been done elsewhere to allow file systems to be "mounted" across nodes. 4.3Bsd has transparent file access as a design goal.

Q. Will DIGITAL's VAX/UNIX release have the same basic "goodies" as 4.2Bsd?

A. No decision has been made on 4.1Bsd vs 4.2Bsd. Because 4.2Bsd is a new release, assessments of suitability and reliability are premature at this time.

Q. How does one get 4.xBsd?

A. 4.xBsd may only be supplied to holders of Bell UNIX source licenses.

Q. Is DIGITAL shipping UNIX source licenses?

A. No, pending resolution of legal questions with Bell.

Q. Does DIGITAL support UNIX in the VAX cluster world?

A. Not at the present time.

Q. It would be highly desirable for DIGITAL to provide support in this area.

A. Noted. (APS)

Q. Are there bugs in the 4.2Bsd DMF-32 driver?

A. None are known to the panel. However, be aware that the DMF-32 driver only uses the serial asynchronous portion of the DMF-32, and that the DIGITAL board have modem control on only two lines of the eight on the board.

Q. Will DIGITAL continue to provide free source to device drivers as has been done in the past?

A. Hopefully.

Q. What is the state of Berkeley INGRESS?

A. Good. The VAX architecture necessitates fewer processes per user than the PDP-11 architecture. This greatly enhances performance. INGRESS is available on the current 4.xBsd tape.

Q. Are there any changes in 4.2Bsd from 4.1Bsd?

A. Many. For example, Bsd4.2 has a new file system which is much better for high I/O bandwidth applications such as VLSI design graphics.

Q. What about the new F77 compiler on 4.2Bsd?

A. The old compiler produced code which executed about half as fast as the VMS Fortran compiler. With the new compiler, execution speed is about the same as VMS Fortran, but compile times are SLOW.

Q. Are there incompatibilities between 4.2Bsd and 4.1Bsd?

A. Executable images will run on both systems. Problems arise when munging on directories, since directory formats have changed.

Q. Does 2.9Bsd include job process control via C shell?

A. Yes.

Q. How does System V differ from 4.xBsd?

A. System V supports a two UNIBUS configuration on the VAX-11/780 with distinct restrictions on device placement. 4.2Bsd supports a four UNIBUS configuration with no restrictions on device placement. In addition, VAX-11/750 support is new on System V but not on 4.2Bsd.