BASIC SIG

WAR CORRESPONDENTS

The BASIC SIG is looking for a few good pens

There are alot of you out there in the trenches. We need your war stories.

The BASIC SIG will pay you five dollars for each BASIC war story that you send in and is printed in the news letter.

So for five fast bucks (a fast fin) jot down a quick line and send it in.

JANUARY 1984 ISSUE

Printed in the U.S.A.

BE A WAR CORRESPONDENT

THE BASIC SIG IS LOOKING
FOR A FEW GOOD PENS....
AND WE ARE WILLING TO PAY
FOR THEM !!!!!!

WE NEED <u>YOUR</u> BASIC WAR STORIES

$5

THERE ARE A BUNCH OF YOU OUT
THERE IN THE TRENCHES.

THE BASIC SIG WILL PAY YOU $5
FOR EACH BASIC WAR STORY THAT
GIVE TO US.

SO FOR FIVE QUICK BUCKS (FAST FIN)
GET IN THOSE WAR STORIES.

TED A. BEAR

NEWSLETTER
EDITOR

BASIC SIG

Looking for a Logo

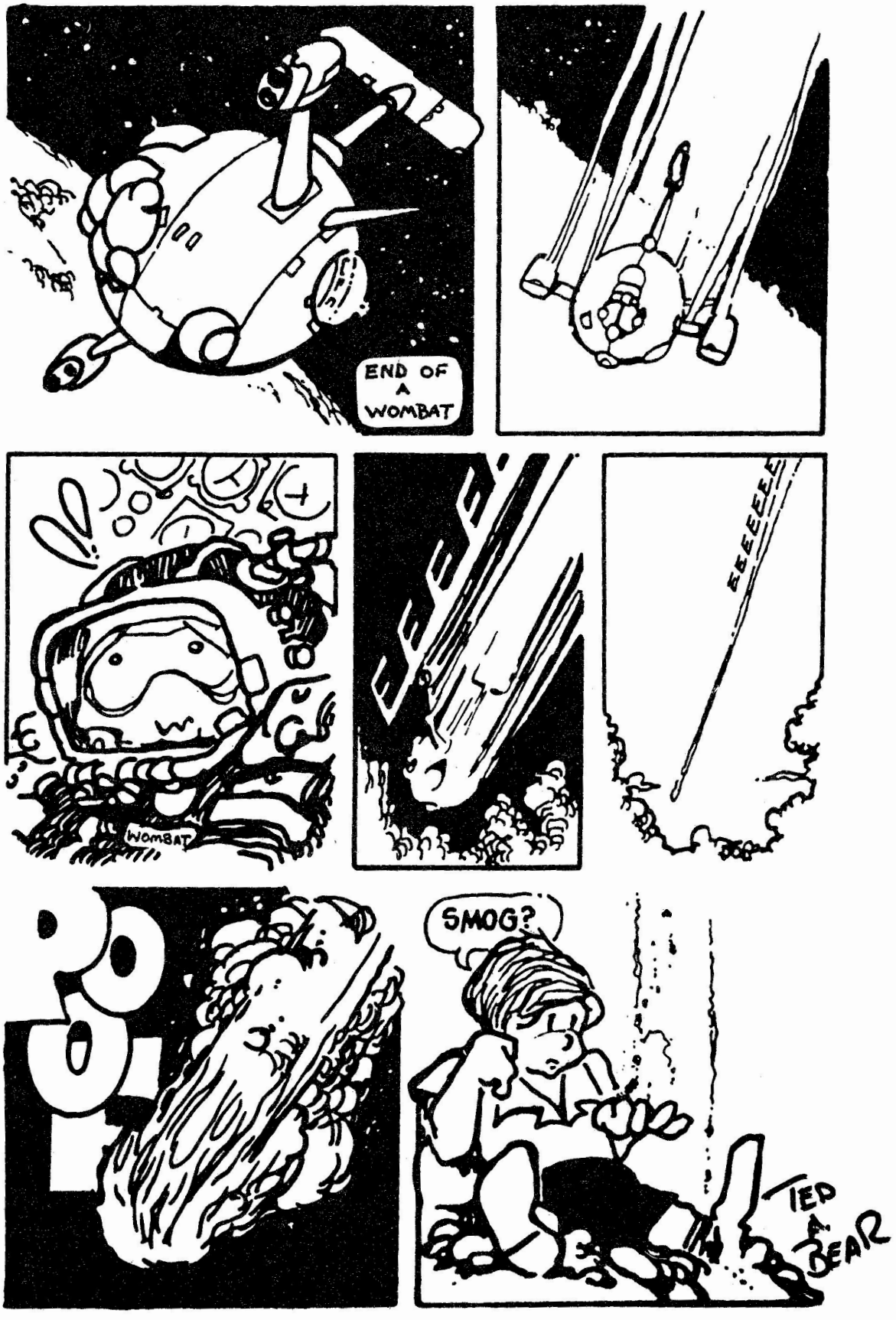
The BASIC SIG decided that a Logo was in order, so we set out on the trek.

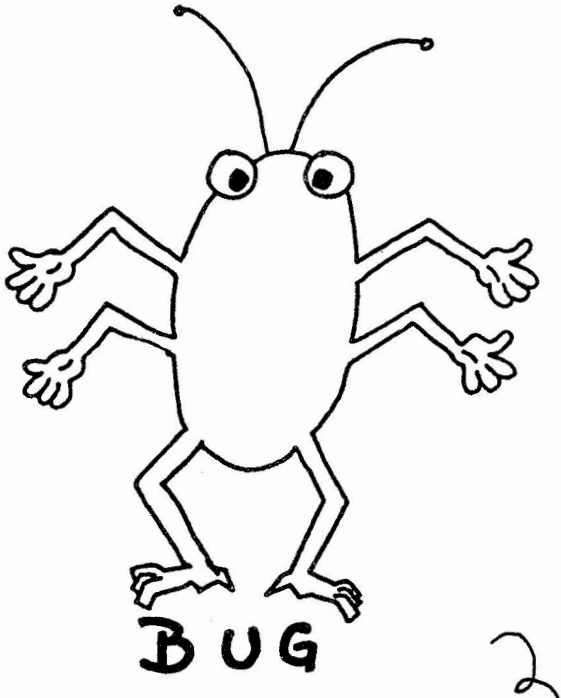
First we looked at character from ~~Lewis Carroll's~~ Alice in Wonderland, but we didn't like the look and it was already taken.

Then we looked at a furry animal, but that just didn't fly.

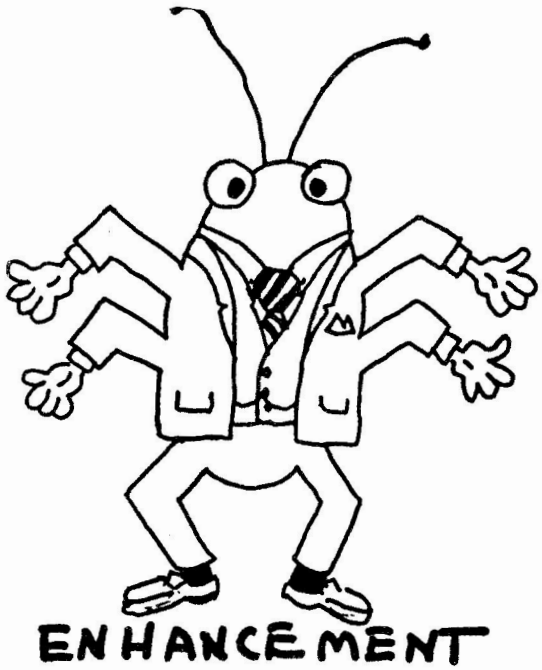
Then we looked at how we come up with enhancements by dressing up our Boss.

But then we decided on your BASIC programmer. So you will see B.P. in forth comming issues.


END OF A WOMBAT
EEEEEEE
WOMBAT
SMOG?
TIED A BEAR


BUG


BUG
TED A BEAR


ENHANCEMENT

DATA COMMUNICATIONS

GLOSSARY

ADAPTIVE EQUALIZATION	Busing
AMPLITUDE MODULATION	Feast or famine
ASCII	A Chinese question
AUTOMATIC CALLING UNIT	Teenager with a telephone
BAUD	Lady of the evening
BIT	12 1/2 cents
BIT RATE	How often you're bitten
BROAD BAND	An all girl orchestra
BYTE RATE	How often you bite
ENVELOPE DELAY	The U.S. Mail System
ERROR RATE	Result of Frequency Modulation
FREQUENCY MODULATION	The Rhythm System
FULL DUPLEX	No Vacancies
HIGH SPEED LINE	Romeo in a hurry
HOLLERITH	What thou dost when thy phone is out of order
KILOCYCLE	A 1,000 wheeled vehicle
MICROWAVE	Signal from a friendly Micro
MODEM	Southern for "more of them"
PARITY CHECK	Agricultural subsidy
PARITY ERROR	The check is late
PHASE JITTER	Nervous reaction to the full moon
POISSON DISTRIBUTION	Serving line at a fish fry
REAL TIME	See "BAUD"
SEMICONDUCTOR	Part-time railroad employee
SYNCH CODE	S.O.S. from the Titanic
TELETYPEWRITER	Talk it over with your Smith-Corona
TIELINE	The latest in neckwear
TWISTED PAIR	A couple of perverts

23-Sep-1983

Corning Glass Works
MP-BH-5
Corning, N. Y. 14831

Editor, BASICSIG Newsletter
DECUS
One Iron Way, MR02-1/C11
Marlboro, Mass 01752

Dear Editor,

I believe the patch contained in the August issue (allowing BP2 V1.6 tasks to run under RSTS/E V8.0) will not function properly. As I recall, I tried this once while installing a new, patched version of RMSRES. Not wanting to break all my current programs, I wanted to have both the original unpatched version and the new patched version available at the same time. To do this, I altered the library name in the file headers of all the old tasks. I found when RMSRES tries to attach to a given portion of its code using its dynamic APR, it had a hard-coded entry of 'RMSRES' in the attach command. The program will then crash with a 'Library not resident' error. It will be necessary to also patch the 'old' version of RMSRES to change the library reference to RMSOLD.

Unfortunately, I no longer have access to my old RSTS/E system and so perhaps Bill Tabor (or some other wizard) can supply the necessary patch.

Sincerely yours,

Pasquale F. Scopelliti

October 5, 1983

DECUS
1 Iron Way, MR02-1/C11
Marlboro, MA 01752

Gentlemen:

I have just received a first copy of the BASIC/SIGG and find that it illustrates clearly a problem which I find intensely frustrating. Basic + 2 for RSTS and RSX is an excellent language, well supported and enthusiastically received. Basic for RT-11 is archaic, inadequate and lacks notable features which are clearly identified in some of the articles in this BASIC/SIGG issue. Variable names, inadequate local references and a small list of other factors which are serious, indeed almost fatal flaws.

CPM, which has taken the personal computer world by storm, supports a much better version of basic. RT-11 is a vastly superior operating system to CPM and in fact served as the model for CPM. If only Digital would provide an adequate version of basic to run on RT-11, they could offer serious competition to CPM because of the relatively large program libraries available.

As an RT-11 user, responsible presently for five LSI-11's with planned purchases of more computers (which may or may not be Digital machines depending upon whether or not support is provided) I would like to register a very strong vote in favor of upgrading Basic under RT-11 to a competitive language. I believe that if this were done, the relative friendliness and quality of RT-11 would serve as a superb entry into more elaborate operating systems which the corporation has made its primary areas of support.

I thank you for your attention in this matter.

Sincerely,

Allen K. Ream, M.D.
Associate Professor of Anesthesia
Chief, Cardiovascular Anesthesia
Director, Institute of Engineering
Design in Medicine

AKR/vg

DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY

ONE IRON WAY, MARLBORO, MASSACHUSETTS 01752
TEL. (617) 481-9511 ext. 4100/TWX 710-3470212 TELEX 948457

November 3, 1983

Mr. Ted Bear
2185 Cox Road
Aptos, CA 95003

Dear Ted:

I just received the September issue of the BASIC SIG Newsletter and was very pleased to see that you had included abstracts of programs from the DECUS Library. I appreciate your including them as I feel that the SIG Newsletters are the best vehicle DECUS has for publicizing the Library.

Keep up the good work!

Sincerely,

Ardoth A. Hassler

(Ms.) Ardoth A. Hassler
Decus U.S. Program
Library Coordinator

Assistant Director for
Academic Services
Computer Center
Catholic University of America
Washington, D.C.

AAH/dl

cc: Dan Esbensen
Charles Mustain
Brent Lapham
Paula Morin
Lee Otsubo

ATTACHMENT B

DECUS PROCEEDINGS

For your convenience and information listed below are the current DECUS Proceedings that are available and can be ordered through the DECUS office in Marlboro, Massachusetts. As availability changes this list will be updated.

			DECUS Part No.	Media Service Codes
Europe	1980	Amsterdam, Holland	PRO-81/V07.1	YA
U.S. Fall	1980	San Diego, California	PRO-81/V07.2	YA
Canada	1981	Montreal, Quebec	PRO-81/V07.3	YA
U.S. Spring	1981	Miami, Florida	PRO-81/V07.4	YA
Australia	1981	Brisbane, Australia	PRO-81/V07.5	YA
Europe	1981	Hamburg, Germany*	PRO-82/V08.1	YA
U.S. Fall	1981	Los Angeles, California	PRO-82/V08.2	YA
Canada	1982	Toronto, Canada	PRO-82/V08.3	YA
U.S. Spring	1982	Atlanta, Georgia	PRO-82/V08.4	YA
Europe	1982	Warwick, United Kingdom	PRO-EUR-82	YA
U.S. Fall	1982	Anaheim, California	PRO-ANA-82	YA
U.S. Spring	1983	St. Louis, Missouri	PRO-STLO-83	YA

* Available from Geneva only. None available until further notice.

PLEASE NOTE: The DECUS Proceedings are no longer grouped together in one volume; they are each listed separately. European, Canadian and Australian Proceedings will be listed by the year, date and place of the symposium. U.S. Proceedings will be listed by the year, season (Spring or Fall) and place of the symposium.

BASIC Business Package

Version: May 1983

Author: Glenn C. Everhart et. al., RCA Corporation, Mt. Holly, NJ

Operating System: IAS, RSX-11D, RSX-11M, VAX/VMS

Source Language: BASIC, FORTRAN-IV PLUS, MACRO-11

This submission contains a grab-bag of utilities for general use after some mods. Included is Michael Reese BASIC for RSX (or compatibility mode VMS) from the RSX SIG tapes, and several utility sets with their own documentation. Two business packages and a DBMS seed package in PASCAL from the CP/M User Group are included. The better of these includes G/L, A/P, A/R, payroll, etc. Source files are present; binary files pertaining to CP/M have been generally excluded, though images of the CP/M floppies are provided. These can be used directly by those with The Bridge.

The Reese BASIC.MAC files are corrected by the .cmp files already and should be ready to build. The writeup with the kit describes where the submissions are.

No guarantees are made as to the completeness, usability, or quality of the programs on the tape. The material has not been checked or reviewed, and documentation may or may not be included.

Note: CPMUG Vol. 43 is not intact and Vols. 43-45 only include most sources of Osborn package. Businessmaster II package is complete.

Restrictions: Some work will be needed to convert CBASIC and MBASIC dialects to DEC BASIC.

Documentation on magnetic media.

Media (Service Charge Code): 2400' Magtape (PC)

Format: RMSBCK with ANSI Labels (Blocked at 2048)

Keywords: Finance, BASIC, Languages
Category Index: 15
Operating System Index: RSX-11/IAS, VAX/VMS

new
11-SP-55

KERMIT and CPMUG Grab Bag

Version: V1.0, June 1983

Author: Glenn C. Everhart, RCA Corporation, Mt. Holly, NJ

Operating System: RSX-11D, RSX-11M, RT-11, VAX/VMS, CP/M, M5DOS, UNIX

Source Language: BASIC, BLISS, MACRO-10, MACRO-11, PASCAL, C

This submission contains a Kermit distribution package for reliable communications over terminal lines between PDP-11, VAX, CP/M-80 based micros, IBM PC's, DEC10s, DEC20s, IBM 370s and/or Apples. It comes from Columbia University and appears reliable. Note that there isn't an RSX Kermit yet, but one may be buildable with the contents here. Also a good deal of CP/M User Group software (sources only, no binaries) is included. Enough of it is in dialects of C, PASCAL, or BASIC to be used in non-CP/M environments.

For those with VAXes, there is an 8080 emulator and CP/M hooks for VAX/VMS on the Australian VAX SIG '82 DECUS tape, available through the Library, it will let you use these packages directly. Also there is a replacement for COMLIB in the RSX11M V4 BRU utility to (hopefully) allow BRU to be used to already initialized disks under VMS. It is untried but should work.

No guarantees are made as to the completeness, usability, or quality of the programs on the tape. The material has not been checked or reviewed and documentation may or may not be included.

Note: Some elements have only the HEX files. However, most items are in source.

Restrictions: CPMUG files may or may not be complete. Most appear to be. You will need Kermit or something similar to move files to CP/M. Any binaries here are useless and most are deleted.

Documentation on magnetic media.

Media (Service Charge Code): 2400' Magtape (PC)

Format: RMSBCK with ANSI Labels (Blocked at 2048)

Keywords: Communication, Utility
Category Index: 7
Operating System Index:
RSX-11, RT-11, VAX/VMS, CP/M

new
11-643

Extension Routines for MU-BASIC

Version: July 1983

Author: Harald Wiessmann, Wiessmann, Schaltenwurte, Ing. Buro, Reutlingen, Germany

Operating System: RT-11 V4.Ø or later

Source Language: MACRO-11

Memory Required: 27KW

Other Software Required: MU-BASIC V2.Ø or later

The extension routines enable additional functions in MU-BASIC such as: set time and date, signal wait, input/output of any installed DL line with device time out capability, pack and unpack float values (single precision) to octal and vice versa, clear ring buffer. Except for multiuser I/O functions the extension routines can also be applied under BASIC-11.

Restrictions: The above functions add about 5 blocks to the Basic interpreter. All comments in source are made in German.

Documentation on magnetic media.

Media (Service Charge Code): Listing (German) (BA), Floppy Diskette (KA), 6ØØ' Magtape (MA)

Format: RT-11

Keywords: Extension Routines, BASIC, MU-BASIC
Category Index: 7
Operating System Index: RT-11

November 7, 1983

new
11-653

RENUM/PRENUM BASIC Renumberer

Version: V1.Ø, July 1983

Author: William B. Leng, Southern Connecticut University, New Haven, CT

Operating System: RSTS/E V6 or later

Source Language: BASIC-PLUS2

Memory Required: 19KW

Special Hardware Required: VTlØØ or Control Sequence-Compatible Video

RENUM numbers .BAS or .B2S programs starting with any new line number and by any increment. All statement references are also translated.

PRENUM renumbers any .BAS or .B2S programs using any increment, but only between statements 1ØØØ and 18999. Statements 1ØØØ, 1ØØØØ, 15ØØØ and 19ØØØ are not changed, following the convention given in the PDP-11 BASIC+2 Language Reference Manual, Section E.2. Renumbering starts at line 1ØØØ and restarts at lines 1ØØØØ and 15ØØØ.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA) Format: RT-11, 6ØØ' Magtape (MA) Format: DOS-11

Keywords: Renumber, BASIC
Category Index: 9
Operating System Index: RSTS

November 7, 1983

new
11-654

Student Terminal Management System

Version: V1.Ø, March 1983

Author: William B. Leng, Southern Connecticut State University, New Haven, CT

Operating System: RSTS/E V6 or later

Source Language: BASIC-PLUS2

Memory Required: 17KW

Other Software Required: Uses RSTS/E System Calls

Special Hardware Required: Uses VT1ØØ cursor commands

A Terminal management system to automatically handle scheduling of student terminals on a first-come, first served (one-hour-on, one-hour-off) basis. Provisions are made to send messages to all STUDENT terminals and to ascertain who's on them and what they are running. Terminal usage can be formatted for printout to teachers or usage percentage can be plotted on a VT1ØØ with hard-copy backup to use for justification of resource changes. The available terminal list can be dynamically changed at any time.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up (AA), Floppy Diskette (KA) Format: RT-11, 6ØØ' Magtape Format: DOS-11.

Keywords: Terminal Management, Utility
Category Index: 7
Operating System Index: RSTS

November 7, 1983

new
11-654

Student Terminal Management System

Version: V1.Ø, March 1983

Author: William B. Leng, Southern Connecticut State University, New Haven, CT

Operating System: RSTS/E V6 or later

Source Language: BASIC-PLUS2

Memory Required: 17KW

Other Software Required: Uses RSTS/E System Calls

Special Hardware Required: Uses VT1ØØ cursor commands

A terminal management system to automatically handle scheduling of student terminals on a first-come, first served (one-hour-on, one-hour-off) basis. Provisions are made to send messages to all STUDENT terminals and to ascertain who's on them and what they are running. Terminal usage can be formatted for printout to teachers or usage percentage can be plotted on a VT1ØØ with hard-copy backup to use for justification of resource changes. The available terminal list can be dynamically changed at any time.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up (AA), Floppy Diskette (KA) Format: RT-11, 6ØØ' Magtape Format: DOS-11.

Keywords: Terminal Management, Utility
Category Index: 7
Operating System Index: RSTS

November 14, 1983

November 4, 1983

Ted Bear
BASIC SIG Newsletter Editor
2185 Cox Road
Aptos, CA 95003

Dear Ted:

Please accept this contribution to the BASIC SIG newsletter. It's a couple of programs to answer Wish #10. It works on both RSTS/E BASIC PLUS and RSTS/E BASIC+2 V1.6 .

Sincerely,

John F. Priebe

John F. Priebe

WISH GRANTED

by John F. Priebe

Wish number 00010 on the BASIC SIG wishlist wanted the MID function to work on the left side of the equal sign; an example being

```
MID(A$,5,3) = "CAT"
```

This is certainly possible, but makes BASIC kludgier than it already is, and it's no mean trick to write some code yourself to do it. I submit the following two functions which do the same thing (only differently).

The first function replaces a substring within a string starting at a given character position.

```
10        def fnreplace1$( oris.str$, replace.str$, p% )          &
      !    entry          oris.str$ = original string              &
      !                   replace.str$ = the new substrins         &
      !                   p% = char position to start replacins    &
      !    exit           fnreplace1$ = the new strins             &
      \     fnreplace1$ = left(oris.str$, p%-1%) +                 &
                          replace.str$ +                           &
                          right(oris.str$, p%+len(replace.str$))   &
      \ fnend                                                      &

20        my$ = "NO ONE LIKES DEAD CATS."                          &
      \ my2$ = fnreplace1$( my$, "RAT", 19% )                      &
      \ print my2$                                                 &

99        end
```

In the example above, function "replace1" would set and return the strings:

```
NO ONE LIKES DEAD CATS.
NO ONE LIKES DEAD RATS.            (CAT ---> RAT)
```

The second function replaces a substring within a string by searching for a search string. This is handy when you don't know a particular character position, but you do know that you want to exchange one substring with another.

```
10        def fnreplace2$( oris.str$, search.str$, replace.str$ ) &
      !    entry          oris.str$ = original string              &
      !                   search.str$ = the string to be replaced  &
      !                   replace.str$ = the new substring         &
      !    exit           fnreplace2$ = the new string             &
      \    p% = instr(1%, oris.str$, search.str$ )                 &
      \    fnreplace2$ = oris.str$                                 &
               if p% = 0%       ! if search.str$ not found         &
      \    fnreplace2$ = left(oris.str$, p%-1%) +                  &
                          replace.str$ +                           &
                          right(oris.str$, p%+len(search.str$))    &
               if p% > 0%                                          &
      \ fnend                                                      &

20        my$ = "NO ONE LIKES DEAD CATS."                          &
      \ my2$ = fnreplace2$( my$, "LIK", "RAT" )                    &
      \ print my2$                                                 &

99        end
```

In this second example, function "replace2" would set and return the strings:

```
NO ONE LIKES DEAD CATS.
NO ONE RATES DEAD CATS.            (LIKes ---> RATes)
```

digital

SOFTWARE PERFORMANCE REPORT

FIELD NO.: | CORPORATE SPR NO.:

812872

TO SET UP FOR PROPER ALIGNMENT, START AT MARK BELOW.

PAGE ____ OF ____

OPERATING SYSTEM	VERSION	SYSTEM PROGRAM OR DOCUMENT TITLE	VERSION OR DOCUMENT PART NO.	DATE
VMS	V3.4	BASIC.EXE	V2.2	11/29/82

NAME: Joseph W. Kmoch
FIRM: Milwaukee Public Schools
2525 N. Sherman Blvd.
Milwaukee, WI 53210
ADDRESS: c/o Washington High School
CUST. NO.: N6DS-191-3

DEC OFFICE AND CONTACT PERSON: Brookfield, WI M. Booker

DO YOU HAVE SOURCE: YES [ ] NO [X]

REPORT TYPE/PRIORITY

- [X] PROBLEM/ERROR
- [ ] SUGGESTED ENHANCEMENT
- [ ] OTHER

1. [ ] HEAVY SYSTEM IMPACT
2. [ ] MODERATE SYSTEM IMPACT
3. [XX] MINOR SYSTEM IMPACT
4. [ ] NO SIGNIFICANT IMPACT
5. [ ] DOCUMENTATION/SUGGESTION

SUBMITTED BY: same PHONE: (414) 449-9400

CAN THE PROBLEM BE REPRODUCED AT WILL? YES [XX] NO [ ]

ATTACHMENTS
MAG TAPE [ ] FLOPPY DISKS [ ] LISTING [ ] DECTAPE [ ]
OTHER:

COULD THIS SPR HAVE BEEN PREVENTED BY BETTER OR MORE DOCUMENTATION? PLEASE EXPLAIN IN PROVIDED SPACE BELOW. YES [ ] NO [XX]

CPU TYPE	SERIAL NO.	MEMORY SIZE	DISTRIBUTION MEDIUM	SYSTEM DEVICE	DO NOT PUBLISH
11/750	82C46401H	4.25 mb	mt 1600	rm 80	[ ]

[redacted]

USE OF /REAL=GFLOAT AND /REAL=HFLOAT produces unusable code

Any program which uses real values and which is com_piled with the GFLOAT or HFLOAT qualifiers will fail.

Here is a program and associated commands that will demonstrate this problem:

```
$ CREATE TEST.BAS
10 I = 0
20 END
^Z
[redacted]
$ BASIC/REAL=HFLOAT TEST
$ LINK TEST
$ RUN TEST
%SYSTEM-F-OPCDEC, opcode reserved to DIGITAL
```

A similar message occurs when using the GELOAT option.

ALL SUBMISSIONS BECOME THE PROPERTY OF DIGITAL EQUIPMENT CORPORATION.

SHORT NAME	MNT. CAT.	MNT. GRP.	XFER GRP.	PL	PRB. TYPE

DATE RECEIVED (MAIL)	DATE TO MAINTAINER	XFER DATE	LOGGED ON

DATE RECEIVED (ASG)	DATE RECEIVED FROM MAINTAINER	DATE ANSWERED	LOGGED OFF

FOR IMMEDIATE RELEASE..... September 23, 1983

DECUS ANNOUNCES PRO-350 SOFTWARE

(Marlboro, MA) The Digital Equipment Computer Users Society (DECUS) recently announced the first Professional-350 software to become available from their user program library. This software package is a developer's kit for the PRO-350 which includes an advanced text editor (TECO) a sophisticated directory listing utility (SRD) a utility for reproducing floppy diskettes (COPY), and a "command line interface" (MCR) which provides a software development environment similar to Digital's RSX-11M operating system.

The command line interface provides the folloiwng fourteen functions; SET terminal attributes, SHOW terminal attributes, SHOW partitions, SHOW commons. SHOW tasks, SHOW active, SHOW memory, SHOW logicals, RUN filespec, INSTALL filespec, REMOVE filespec, SHOW TRANSLATIONS logname, ASSIGN logname value, and DEASSIGN logname. Although MCR was written for use with P/OS Version 1.0, source files are provided, along with hints for adapting it for use with future versions of Digital's P/OS operating system. TECO and SRD are equivalent to the corresponding programs commonly used on Digital's PDP-11 computers.

The developer's kit was written by Richerd J.D. Kirkman of Filetab Support Services, London, England. It has been used with P/OS Version 1.5, and as expected, many of the MCR functions require modifications in order to be operable. The other utilities, however, appear to work normally under P/OS V1.5. Extensive, built in "HELP" messages are included with the developer's kit; still experience with RSX-11 software is extremely helpful in using these programs. Sources are included only for MCR.

The Digital Equipment Computer Users Society was established in 1961 to advance the effective use of DIGITAL computers. DECUS is a computer user group supported in part by Digital Equipment Corporation. A major activity of DECUS is the Program Library which distributes software written and submitted by DECUS members. A wide range of software is available for various Digtial computers including compilers, utilities, and application packages. Programs and software are distributed for nominal service charges; however, the DECUS Program Library is a clearing house only -- it does not sell or test programs. All programs and information are provided "AS IS", and no software support is provided by DECUS or Digital Equipment Corp.

To order the PRO-350 Developer's Kit (DECUS Part No. PRO-101) call (617) 467-4135. For additional information about DECUS and the DECUS Program Library, write to DECUS Program Library, One Iron Way, MR02-1/C11, Marlboro, MA 01752.

ONE IRON WAY, (MR02-1/C11), MARLBORO, MA 01752 TWX 710-3470212 TELEX 948457
TELEPHONE: U.S. Activities (617) 467-4879 or 4875 Library (617) 467-4178 Finance & Admin. (617) 467-4881

October 24, 1983

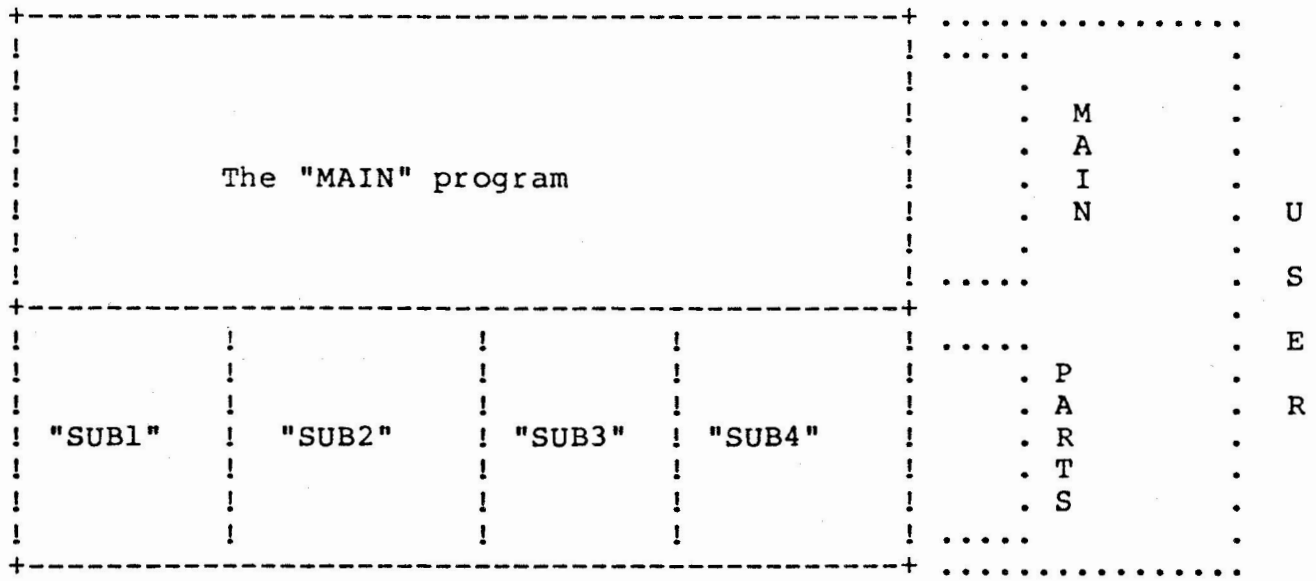
Understanding COTREEs

This document will explain the COTREE overlay structure and its .ODL file requirements. The reader should be familiar with .ODL file directives. The discussion will begin with a simple .ODL file example, then progress to an example using a COTREE.

A typical .ODL file:

```
        .ROOT USER
USER:   .FCTR MAIN-PARTS
PARTS:  .FCTR *(PART1, PART2, PART3, PART4)
PART1:  .FCTR SUB1
PART2:  .FCTR SUB2
PART3:  .FCTR SUB3
PART4:  .FCTR SUB4
        .END
```

This .ODL file would result in a task image that looks like the following:



This kind of simple task image structure works well in most cases. However, lets look at a more complex situation. Lets say that both "SUB1" and "SUB2" need to call "SUB3" and

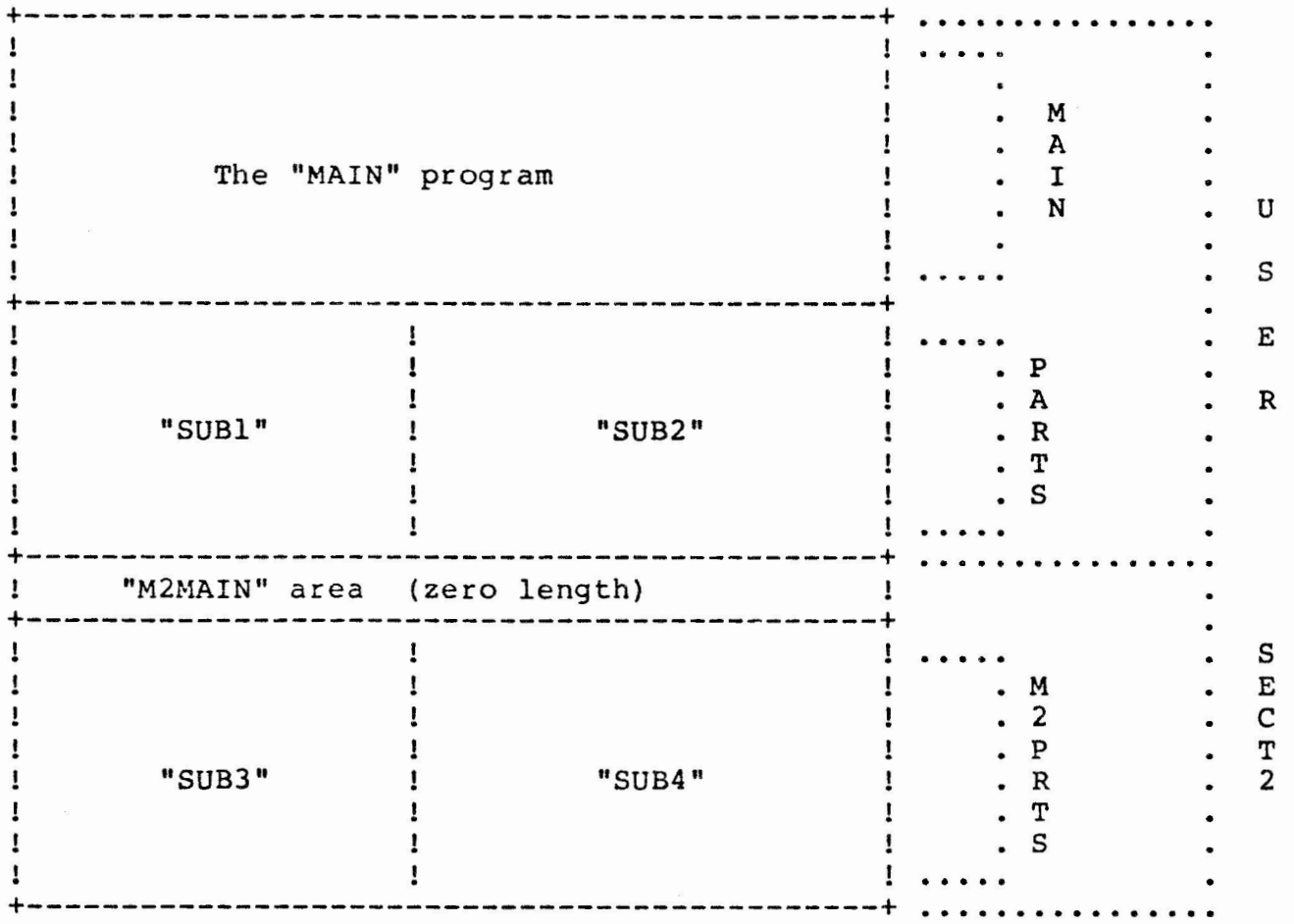
sometimes "SUB4". In this case we want "SUB1" and "SUB2" to use one section of memory, while "SUB3" and "SUB4" use a different section of memory. To do this we need to define a "COTREE", or named area of memory, in addition to the "USER" area that we already have. The .ODL file could look like the following:

```
        .ROOT   USER, SECT2
USER:   .FCTR   MAIN-PARTS
PARTS:  .FCTR   *(PART1, PART2)
PART1:  .FCTR   SUB1
PART2:  .FCTR   SUB2
;
SECT2:  .FCTR   *MEM2
        .NAME   M2MAIN
MEM2:   .FCTR   M2MAIN-M2PRTS
M2PRTS: .FCTR   *(M2PRT1, M2PRT2)
M2PRT1: .FCTR   SUB3
M2PRT2: .FCTR   SUB4
;
        .END
```

In this sample .ODL file, "SECT2" is the name of a "COTREE" section. This section of memory is to follow the "USER" section of memory. The "COTREE" section consists of one main area called "MEM2". The "*" in front of the name "MEM2" (the autoload indicator) is required for any COTREE section.

Each "COTREE" section needs a "root" name to attach its parts to. In this example, the COTREE root is a null segment named M2MAIN. This means that the segment only exists for .ODL structural purposes and does not represent a subroutine. We have attached to this root the two subroutines, "SUB3" and "SUB4". These two subroutines will take turns occupying the "MEM2" COTREE section of memory.

This .ODL file would result in a task image that looks like the following:



Structuring the .ODL file in this manner allows handling even complex COTREEs and other overlay structures, without coding lengthy perenthetical expressions. The payoff is in more readable, understandable .ODL files that clearly describe the task image to be built.

Interfacing to SORT-11 from BASIC-PLUS II

By

William I. Tabor

Computer Products, Inc.

Fort Lauderdale, Florida

1.0 Introduction

The SORT-11 Manual describes to a programmer how to call several subroutines to sort data. The descriptions use FORTRAN as the reference language, this cause an information gap to occur leaving the BASIC-PLUS II programmer in the dark until he understands how BASIC-PLUS II passes data to a MACRO subroutine.

The purpose of this paper is to fill in the gaps and describe how to interface and use SORT-11 as a callable subroutine from a BASIC-PLUS II program.

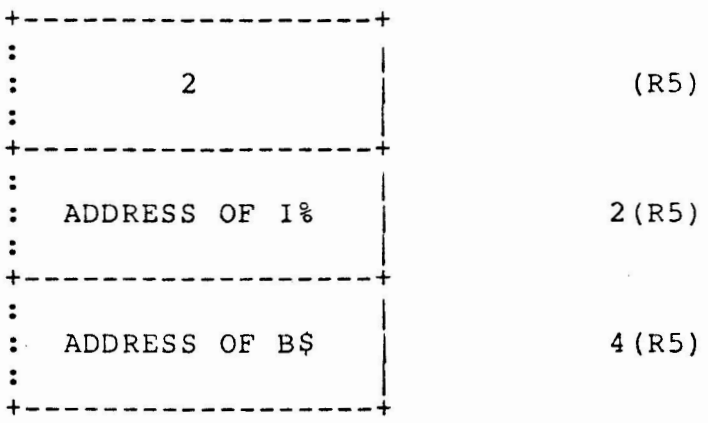
2.0 Call by reference

In the BASIC-PLUS II users guide there is a discussion on how to call a MACRO subroutine from BASIC-PLUS II (D.E.C does not support the use of MACRO subroutines however). The example in the manual uses a mechanism know as "CALL BY REF". CALL BY REF will pass the address of strings and arrays instead of the address of the descriptors in the R5 (register 5) protocol packet.

For example if I had the statement in my basic program;

```
CALL EXAMPL BY REF ( I% , B$ )
```

the R5 Packet would look like this



3.0 Callable Routines of SORT-11

There are four subroutines in SORT-11, they are RSORT, RELES RETRN, end ENDS.

3.1 RSORT - Initializing the sort package.

RSORT sets up the internal variables for use by the rest of the SORT-11 subroutines. The calling format to RSORT is:

```
CALL RSORT BY REF ( ERR.CODE%,              &
                    KEY.SIZE%,              &
                    MAX.REC.SIZE%,          &
                    KEY.ADDRESS%,           &
                    WORK.SPACE%(),          &
                    WORK.SPACE.SIZE%,       &
                    NUM.WORK.FILE%,         &
                    BUFF.SIZE%,             &
                    CLUS.WINDOW%,           &
                    ALLOC.WORK%,            &
                    FIRST.CHANNEL% )
```

WHERE:

ERR.CODE%	= returns error code (Ø = no error else the value is the error number see error table in Appendix C)
KEY.SIZE%	= size of key in bytes (must be even and positive)
MAX.REC.SIZE%	= size in bytes of largest record to be passed to sort (must be even, positive and not exceed 16,383 when added to the key size)
KEY.ADDRESS%	= address of the most significant word in the key.
WORK.SPACE%()	= address of first word in work space
WORK.SPACE.SIZE%	= size in bytes of work space
NUM.WORK.FILE%	= number of scratch files to use in sort (must be more than two and less than eleven)
BUF.SIZE%	= number of 512 byte buffers to allocate to each scratch file (refer to Appendix D. of the SORT-11 manual for more information on this value)
CLUS.WINDOW%	= for a RSTS/E system this value is the clustersize to be used in opening the scratch files. for RSX systems this is the number of retrieval pointers to be used in opening the scratch files.
ALLOC.WORK%	= number of blocks to allocate to each scratch file when it is opened.
FIRST.CHANNEL%	= channel number to open the first scratch file on.

3.2 RELES - Passing input records to SORT-11

RELES will pass a record to SORT-11 for sorting. The calling format for RELES is:

```
CALL RELES BY REF ( ERR.CODE%,          &
                    RECORD.SIZE%,       &
                    RECORD$ )
```

WHERE:

ERR.CODE%	= returns error code (Ø = no error else the value is the error number see error table in Appendix C)
RECORD.SIZE%	= the size of the record being passed to SORT-11.
RECORD$	= the record being passed to passed to SORT-11.

3.3 MERGE - Merging the Scratch Files

Complete the Sorting process by calling MERGE to merge the scratch files. The calling format for MERGE is:

```
CALL MERGE BY REF ( ERR.CODE% )
```

WHERE:

ERR.CODE%	= returns error code (Ø = no error else the value is the error number see error table in Appendix C)

3.4 RETRN - Get a record back form SORT-11 in sort sequence

RETRN will get a single record back from SORT-11 in sort sequence. The calling format for RETRN is:

```
CALL RETRN BY REF ( ERR.CODE%,          &
                    RECORD.SIZE%,       &
                    RECORD$ )
```

WHERE:

ERR.CODE%	= returns error code (Ø = no error, negative number to indicate end of sorted data, else the value is the error number see error table in Appendix C)
RECORD.SIZE%	= the size of the record being returned to the calling routine from SORT-11.
RECORD$	= the record being returned to the calling routine from SORT-11

3.5 ENDS - Clean up and terminate the SORT-11 routines.

The calling format for ENDS is:

```
CALL ENDS BY REF ( ERR.CODE% )
```

WHERE:

ERR.CODE%	= returns error code (Ø = no error else the value is the error number see error table in Appendix C)

4.0 Internal work space

Now that I have gone over calls to the SORT-11 subroutines, the only other information necessary to access SORT-11 is how to set up the internal memory space for sorting.

4.1 Declaring the work space

The simplest way to declare the internal work space for SORT-11 is to use the MAP statement. For example:

```
MAP     (SRTWRK)        WORK.SPACE%(11999%)
```

in the above example a work space of 12,000 words or 24,000 bytes has been set up.

The calculation for the minimum size of the work space is:

```
SIZE = NUMBER.OF.SCRATCH.FILES
       * ( 512. * NUMBER.OF.BUFFERS
       + ( 100. + RECORD.SIZE + KEYSIZE + 10. ))
```

4.2 KEY SPACE

In addition to the work space required by SORT-11, the location of the key has to set into a static area. This can also be done by using the MAP statement. For example:

```
MAP     (SRTKEY)        KY$(511%) =   1%
MAP     (SRTKEY)        FILL$     = 510% &
                        KY%
```

5.0 Building the key

The most significant character in the sort key must be the right most byte of the key. For example:

A Key of "ABC" must be loaded into the key space as

```
KY$(511%) = "A"
KY$(510%) = "B"
KY$(509%) = "C"
```

To build a Key that contains a numeric packed field you have to modify the data.

For example an integer in CVT%$ format

to convert this to a sortable string you must first reverse the sign bit so negative numbers are sorted in the proper order. The following lines of code will accomplish this:

```
TEMP% = ASCII(INP.STR$)
 IF TEMP% < 128%
  THEN
   OUT.STR$ = CHR$(TEMP%+128%)
  ELSE
   OUT.STR$ = CHR$(TEMP%-128%)
 END IF
OUT.STR$ = OUT.STR$ + SEG$(INP.STR$,2%,2%)
```

note that the high and low order bytes have been reversed so to place the string into the key work area the first byte would go to the right most byte of the key. For example:

```
KY$(511%) = SEG$(OUT.STR$,1%,1%)
KY$(510%) = SEG$(OUT.STR$,2%,2%)
```

The following code will generate an eight byte string to be placed into the key work area for packed floating point numbers, this routine will work for both single and double precession values.

```
INP.STR$ = STRING$(4%,0%)+INP.STR$
                IF LEN(INP.STR$)=4%
TEMP% = ASCII(SEG$(INP.STR$,7%,7%))
IF TEMP% < 128%
 THEN
  OUT.STR$=CHR$(TEMP%+128%)
                + SEG$(INP.STR$,8%,8%)
                + SEG$(INP.STR$,5%,6%)
                + SEG$(INP.STR$,3%,4%)
                + SEG$(INP.STR$,1%,2%)
  ELSE
   OUT.STR$ = CHR$(255% XOR (TEMP%-128%))
                + CHR$(255% XOR ASCII(RIGHT$(INP.STR$,8%)))
   OUT.STR$ = OUT.STR$
                + CHR$(255% XOR ASCII(RIGHT$(INP.STR$,5%-Z%)))
                + CHR$(255% XOR ASCII(RIGHT$(INP.STR$,6%-Z%)))
                                FOR Z% = 0% TO 4% STEP 2%
```

6.0 Building the Task

To include SORT-11 into your task space the modules SORTS and SIORMS must be included in to your task as well as the RMS sequential library code.

SORTS and SIORMS are both found in the SORT.OLB in the SORT-11 distribution kit. To include them in your Task image include the following line in your ODL file:

```
SORT:   .FCTR  LB:SORT/LB:SORTS:SIORMS
```

For example:

```
        .ROOT  BASIC2-RMSROT-USER,RMSALL
USER:   .FCTR  SY:USEREP-SORT-LIBR
LIBR:   .FCTR  LB:BP2OTS/LB
SORT:   .FCTR  LB:SORT/LB:SORTS:SIORMS
@LB:BP2IC3
@LB:RMSRLX
        .END
```

APPENDIX A

SAMPLE PROGRAM

```
    00001 1     %TITLE 'USEREP V01.00'
    00001       %IDENT 'V01.00'
    00001       OPTION SIZE = REAL DOUBLE
    00002       DECLARE STRING CONSTANT PROG = 'USEREP V01.00'
    00003       ON ERROR GO TO 19000
    00003
    00001 801   ! INCLUDE THE MAP FOR USER.RMS FILE
    00001       %INCLUDE 'USER.REC'
I1  00001       MAP     (USERAC)        USER.NAME$              = 20%,      &
I1  00001                               PROJECT.NUMBER$         =  3%,      &
I1  00001                               PROGRAMER.NUMBER$       =  3%,      &
I1  00001                               DATE.OF.ACCESS$         =  6%,      &
I1  00001                               TIME.OF.ACCESS$         =  4%,      &
I1  00001                               KEYBOARD.OF.ACCESS$     =  2%,      &
I1  00001                               JOB.NUMBER.OF.ACCESS$   =  2%
I1  00001
    00001       %PAGE
```

```
00001
00001 802    ! ============================================================
00001        !
00001        !                S E T   U P   F O R   S O R T
00001        !
00001        ! ============================================================
00001        !
00001        MAP      (SORT  )        KY$(511%) =   1%
00002        MAP      (SORT  )        FILL$     = 510%,                    &
00002                                  KY%
00003        MAP      (WORK  )        WORK%(11999%)
00004        MAP      (XXXX  )        WORK.REC$ =   8%
00005        MAP      (XXXX  )        REC.COUNT
00005        %PAGE
```

```
00005
00001 1000  ! ============================================================
00001       !                                                            I
00001       !                 S T A R T   O F   P R O G R A M
00001       !
00001       ! ============================================================
00001       !
00001       THE.START.TIME0 = TIME(0%)
00002       THE.START.TIME1 = TIME(1%)
00003       OPEN 'DR0:[1,254]USERYR.RMS' AS FILE 1%,                      &
00003                        ORGANIZATION RELATIVE FIXED,                 &
00003                        MAP USERAC
00004       KEY.LEN% = LEN(USER.NAME$) + LEN (DATE.OF.ACCESS$) +          &
00004                        + LEN(TIME.OF.ACCESS$)
00004
00004
00001 1010  ! ============================================================
00001       !
00001       !                 C A L L   R S O R T   T O   S E T
00001       !                 T H E   P A R A M E T E R S   F O R
00001       !                 S O R T I N G
00001       !
00001       ! ============================================================
00001       !
00001       CALL RSORT BY REF (IERR%, KEY.LEN%, 8%, KY%, WORK%(),         &
00001              24000%, 5%, 4%, 128%, 400%, 5% )
00002       !
00002       !        IERR%     = ERROR CODE RETURNED FROM RSORT ( 0 = NO ERROR )
00002       !        KEY.LEN%  = MAXIMUM LENGTH OF KEY
00002       !        8%        = LENGTH OF RECORD ( JUST RELATIVE RECORD NUMBER )
00002       !        KY%       = ADDRESS OF MOST SIGNIFICANT WORD IN KEY
00002       !        WORK%()   = ADDRESS OF FIRST WORD OF WORK AREA
00002       !        24000%    = SIZE IN BYTES OF WORK AREA ( (11999+1)*2)
00002       !        5%        = NUMBER OF SCRATCH FILES
00002       !        4%        = NUMBER OF 512-BYTE BUFFERS TO BE ALLOCATED TO EACH
00002       !                    SCRATCH FILE
00002       !        128%      = RSTS/E CLUSTERSIZE FOR SCRATCH FILES
00002       !                    (IN RSX IS THE NUMBER OF RETRIEVAL POINTERS)
00002       !        400%      = NUMBER OF BLOCKS TO ALLOCATE FOR EACH SCRATCH FILE
00002       !                    ON OPEN
00002       !        5%        = STARTING LUN (CHANNEL NUMBER) FOR WORK FILES
00002       IF IERR%
00003        THEN
00003         PRINT '?ERROR IN RSORT -- ERROR NUMBER =';IERR%
00004         GO TO 32767
00004       %PAGE
```

```
00004
00001 2000  ! ============================================================
00001       !
00001       !                R E A D   I N P U T   F I L E
00001       !                A N D   R E L E A S E   D A T A
00001       !                T O   S O R T
00001       !
00001       ! ============================================================
00001       !
00001       START.TIME0 = TIME(0%)
00002       START.TIME1 = TIME(1%)
00003       REC.COUNT = 1
00004       EOF.FLAG% = 0%
00005       GET #1%, BLOCK REC.COUNT
00005
00005
00001 2010  ! ============================================================
00001       !
00001       !                M A I N   L O O P   F O R
00001       !                I N P U T   P R O C E S S I N G
00001       !
00001       ! ============================================================
00001       !
00001       UNTIL EOF%
00002       KEY.LOC%=511%
00003       THE.KEY$ = USER.NAME$ + DATE.OF.ACCESS$ + TIME.OF.ACCESS$
00004       !
00004       ! BUILD THE KEY
00004       !
00004       !
00004       !       NOW LOAD KEY INTO KEY BUFFER BACKWARDS
00004       !
00004       FOR I% = 1% TO KEY.LEN%
00005        KY$(KEY.LOC%) = SEG$(THE.KEY$,I%,I%)
00006        KEY.LOC% = KEY.LOC% - 1%
00007       NEXT I%
00008       !
00008       CALL RELES BY REF (IERR%, 8%, WORK.REC$)                        &
00008       !
00009       !       IERR%     = ERROR CODE ( 0 = NO ERROR )
00009       !       8%        = SIZE OF WORK.REC$
00009       !       WORK.REC$ = RELATIVE RECORD NUMBER (REFER TO MAP SECTION
00009       !
00009       IF IERR%
00010        THEN
00010         PRINT '?ERROR IN RELES -- ERROR NUMBER =';IERR%
00011         GO TO 32767
00011        ELSE
00012       REC.COUNT = REC.COUNT + 1
00013       GET #1%, BLOCK REC.COUNT
00013
00001 2020  NEXT
00002       END.TIME0 = TIME(0%)
00003       END.TIME1 = TIME(1%)
00004       PRINT 'Release time for ';NUM1$(REC.COUNT-1);' records'
00005       GOSUB 10000
00005       %PAGE
```

```
00005
00001 3000  ! ============================================================
00001        !                                                            !
00001        !                D O   A   M E R G                           !
00001        !                                                            !
00001        ! ============================================================
00001        !
00001        START.TIME0=TIME(0%)
00002        START.TIME1=TIME(1%)
00003        CALL MERGE BY REF (IERR%)
00004        !
00004        !       IERR% = ERROR CODE ( 0 = NO ERROR )
00004        !
00004        IF IERR%
00005         THEN
00005          PRINT "?ERROR IN MERGE -- ERROR NUMBER =";IERR%
00006          GO TO 32767
00006         ELSE
00007        END.TIME0 = TIME(0%)
00008        END.TIME1 = TIME(1%)
00009        PRINT
00010        PRINT "Merge Time"
00011        GOSUB 10000
00011        %PAGE
```

```
00011
00001 4000  ! ============================================================
00001       !
00001       !              G E T   D A T A   B A C K   F R O M
00001       !              S O R T   A N D   G E N E R A T E
00001       !              T H E   R E P O R T
00001       !
00001       ! ============================================================
00001       !
00001       OPEN "DR2:REPORT.LST" FOR OUTPUT AS FILE 2%,                    &
00001                    ORGANIZATION SEQUENTIAL
00002       START.TIME0=TIME(0%)
00003       START.TIME1=TIME(1%)
00004       CURRENT.PAGE.NUMBER% = 0%
00005       CURRENT.LINE.ON.PAGE% = 66%
00006       MAX.LINE.ON.PAGE% = 60%
00007       CALL RETRN BY REF (IERR%, 8%, WORK.REC$, 0%)
00008       !
00008       !       IERR%      = ERROR CODE ( 0 = NO ERROR)
00008       !       8%         = SIZE OF WORK.REC$
00008       !       WORK.REC$  = RETURNED RELATIVE RECORD NUMBER IN SORTED SEQUENCE
00008       !       0%         = DUMMY ARGUMENT
00008       !
00008       GET #1%, BLOCK REC.COUNT
00009       OLD.USER.NAME$ = USER.NAME$
00009
00001 4010  UNTIL IERR%
00002       GET #1%, BLOCK REC.COUNT
00003       IF OLD.USER.NAME$ <> USER.NAME$
00004        THEN
00004          CURRENT.PAGE.NUMBER% = 0%
00005          GOSUB 11000
00006          OLD.USER.NAME$ = USER.NAME$
00007          CURRENT.LINE.ON.PAGE% = CURRENT.LINE.ON.PAGE% + 1%
00008       END IF
00009       IF CURRENT.LINE.ON.PAGE% > MAX.LINE.ON.PAGE%
00010        THEN
00010          GOSUB 11000
00011       END IF
00012       PRINT #2%, USER.NAME$;"   [";PROJECT.NUMBER$;                     &
00012                             ",";PROGRAMER.NUMBER$;"]  ";               &
00012                             SEG$(DATE.OF.ACCESS$,3%,4%);"/";           &
00012                             SEG$(DATE.OF.ACCESS$,5%,6%);"/";           &
00012                             SEG$(DATE.OF.ACCESS$,1%,2%);"   ";
00013       HR$ = SEG$(TIME.OF.ACCESS$,1%,2%)
00014       TIME.FLAG$ = "am"
00015       IF HR$ > "12"
00016        THEN
00016          HR$ = NUM1$(VAL(HR$)-12%)
00017          TIME.FLAG$ = "pm"
00018       END IF
00019       IF HR$ = "12"
00020        THEN
00020          TIME.FLAG$ = "pm"
00021       END IF
00022       HR$ = STRING$(2%-LEN(HR$),ASCII("0"))+HR$
00023       PRINT #2%, HR$;":";SEG$(TIME.OF.ACCESS$,3%,4%);TIME.FLAG$;        &
```

```
00023                                   '    KB';                                &
00023                                   KEYBOARD.OF.ACCESS$;'';    ';             &
00023                                   'Job number = ';JOB.NUMBER.OF.ACCESS$
00024         CURRENT.LINE.ON.PAGE% = CURRENT.LINE.ON.PAGE% + 1%
00025         CALL RETRN BY REF (IERR%, 8%, WORK.REC$, 0%)
00026         NEXT
00027         END.TIME0 = TIME(0%)
00028         END.TIME1 = TIME(1%)
00029         PRINT
00030         PRINT 'Return from sort'
00031         GOSUB 10000
00032         CALL ENDS BY REF (IERR%)
00033         START.TIME0 = THE.START.TIME0
00034         START.TIME1 = THE.START.TIME1
00035         END.TIME0 = TIME(0%)
00036         END.TIME1 = TIME(1%)
00037         PRINT
00038         PRINT 'Total time for Job'
00039         GOSUB 10000
00040         GOTO 32000
00040         %PAGE
```

```
00040
00001 10000 ! ============================================================
00001       !
00001       !       T I M E   P R I N T E R
00001       !
00001       ! ============================================================
00001       !
00001       END.TIME0 = END.TIME0 + (24*60*60)                              &
00001                         IF END.TIME0 < START.TIME0
00002       ELAP.TIME=END.TIME0-START.TIME0
00003       HR = INT(ELAP.TIME/3600.)
00004       MN = INT((ELAP.TIME-(HR*3600.))/60.)
00005       SE = INT(ELAP.TIME -((HR*3600.)+MN*60.))
00006       HR$=NUM1$(HR)
00007       HR$=STRING$(2%-LEN(HR$),ASCII("0"))+HR$
00008       MN$=NUM1$(MN)
00009       MN$=STRING$(2%-LEN(MN$),ASCII("0"))+MN$
00010       SE$=NUM1$(SE)
00011       SE$=STRING$(2%-LEN(SE$),ASCII("0"))+SE$
00012       PRINT "Elapsed time = ";HR$;":";MN$;":";SE$
00013       PRINT USING "CPU time     = ###.# SECONDS",                     &
00013                         (END.TIME1-START.TIME1)/10
00014       RETURN
00014       %PAGE
```

```
00014
00001 11000 ! ============================================================
00001       !
00001       !                  P R I N T   P A G E   H E A D E R
00001       !
00001       ! ============================================================
00001       !
00001       PRINT #2%, CHR$(12%)
00002       PRINT #2%, DATE$(0%);
00003       HEADER$ = 'U S E R   A C C E S S   L I S T'
00004       PRINT #2%, TAB((80%-LEN(HEADER$))/2%);HEADER$;TAB(70%);
00005       CURRENT.PAGE.NUMBER% = CURRENT.PAGE.NUMBER% + 1%
00006       PRINT #2% USING 'Page ###', CURRENT.PAGE.NUMBER%
00007       PRINT #2%, TIME$(0%)
00008       PRINT #2%
00009       CURRENT.LINE.ON.PAGE% = 4%
00010       RETURN
00010       %PAGE
```

```
00010
00001 19000 ! ============================================================
00001       !
00001       !       O N   E R R O R   H A N D L E R
00001       !
00001       ! ============================================================
00001       !
00001       IF ERR = 155%
00002        THEN
00002         IF ERL = 2000 OR ERL = 2010
00003          THEN
00003           EOF% = -1%
00004           RESUME 2020
00005          END IF
00006           END IF
00006
00006
00001 19900 E% = ERR
00002       L% = ERL
00003       PRINT '?Unexpected error'
00004       PRINT ERT$(E%);' at line ';NUM1$(L%);' in ';PROG
00005       RESUME 32767
00005       %PAGE
```

```
00005
00001 32000 ! ============================================================
00001       !
00001       !                 N O R M A L   E X I T   P O I N T
00001       !
00001       ! ============================================================
00001       !
00001       PRINT #2%, CHR$(12%)
00002       CLOSE #2%
00003       CLOSE #1%
00003
00001 32767 END
```

APPENDIX B

MANGLE SUBROUTINE

```
00001 1     SUB MANGLE ( INP.STR$, OUT.STR$ )
00002       %TITLE "MANGLE V02.00"
00002       %IDENT "V02.00"
00002       OPTION SIZE = REAL DOUBLE
00003       DECLARE STRING CONSTANT PROG = "MANGLE V02.00"
00004       !
00004       !       SUBROUTINE MANGLE
00004       !
00004       !       WRITTEN :       20-MAY-83
00004       !       AUTHOR  :       W. TABOR
00004       !
00004       !       PURPOSE :       SET NUMERIC VARIABLE TO SORTABLE STRING
00004       !
00004       !       INPUT   :       INP.STR$ = THE INPUT STRING
00004       !
00004       !       OUTPUT  :       OUT.STR$ = THE OUTPUT STRING READY FOR RELEASE
00004       !                                  TO SORT-11
00004       !
00004       ! ---------------------------------------------------------
00004       !
00004       !       THANK YOU GARY M. BERG FOR SHOWING THE WAY.
00004       !
00004       ! ---------------------------------------------------------
00004       !
00004       !               M O D I F C A T I O N   H I S T O R Y
00004       !
00004       ! ---------------------------------------------------------
00004       !
00004       ! 01-OCT-83  W.I.T  UPGRADE TO VERSION 2.1 OF BASIC-PLUS II
00004       !
00004       ! ---------------------------------------------------------
00004       !
00004       ON ERROR GO TO 19000
00004       %PAGE
```

```
00004
00001 1000  ! ============================================================
00001       !
00001       !                  HANDLE FOR INTEGER
00001       !
00001       ! ============================================================
00001       !
00001       IF LEN(INP.STR$) = 2%
00002        THEN
00002         TEMP% = ASCII(INP.STR$)
00003          IF TEMP% < 128%
00004           THEN
00004            OUT.STR$ = CHR$(TEMP%+128%)
00004           ELSE
00005            OUT.STR$ = CHR$(TEMP%-128%)
00006          END IF
00007         OUT.STR$ = OUT.STR$ + SEG$(INP.STR$,2%,2%)
00007       %PAGE
```

```
00007
00001 1010  ! ============================================================
00001        ! ' ' ' '
00001        !                 HANDLE FOR FLOATING POINT
00001        !
00001        ! ============================================================
00001        !
00001        INP.STR$ = STRING$(4%,0%)+INP.STR$                              &
00001                         IF LEN(INP.STR$)=4%
00002        IF LEN(INP.STR$) = 8%
00003         THEN                                  ! THIS AN EIGHT BYTE FLOAT
00003          TEMP% = ASCII(SEG$(INP.STR$,7%,7%))
00004          IF TEMP% < 128%
00005           THEN
00005            OUT.STR$=CHR$(TEMP%+128%)                                   &
00005                         + SEG$(INP.STR$,8%,8%)                         &
00005                         + SEG$(INP.STR$,5%,6%)                         &
00005                         + SEG$(INP.STR$,3%,4%)                         &
00005                         + SEG$(INP.STR$,1%,2%)
00005           ELSE
00006            OUT.STR$ = CHR$(255% XOR (TEMP%-128%))                     &
00006                         + CHR$(255% XOR ASCII(RIGHT$(INP.STR$,8%)))
00007            OUT.STR$ = OUT.STR$                                        &
00007                         + CHR$(255% XOR ASCII(RIGHT$(INP.STR$,5%-Z%))) &
00007                         + CHR$(255% XOR ASCII(RIGHT$(INP.STR$,6%-Z%))) &
00007                                   FOR Z% = 0% TO 4% STEP 2%
00007
00001 9000  GO TO 32000
00001        %PAGE
```

```
00001
00001 19000 ! ============================================================
00001       !
00001       !                 ON ERROR ROUTINES
00001       !
00001       ! ============================================================
00001
00001 19990 ET$=ERT$(ERR)
00002       PRINT '??Error ';ET$;' in ';PROG;' at line';ERL
00003       RESUME 32767
00003       %PAGE
```

```
00003
00001 32000 ! ============================================================
00001       !
00001       !                   STANDARD EXIT POINT
00001       !
00001       ! ============================================================
00001
00001 32767 SUBEND
```

APPENDIX C

SORT-11 ERROR CODES

00 - No errors

01 - Device input error

03 - Device output error

04 - Open(in) failure

05 - Size of current record is greater than maximum size

06 - Not enough work area

07 - RETRN was called after it had exited with a negative error code (end of sort)

08 - SORT routine called out of order (the order of the calls should be RSORT, RELES, MERGE, RETRN, ENDS).

09 - Sort already in progress (To do a second sort, ENDS must be called to clean up the first sort).

10 - Key size is not positive, SORTS detected a zero or negative key size in its calling parameter

11 - Record size is not positive

12 - Key address is not even (the keys must start at an even address because SORT uses word movers).

13 - Record address is not even

14 - Scratch records will be too large (the size of the keys plus the size of the largest record must be less than 37776 octal).

15 - Too few scratch files are given (a minimum of 3 scratch files must be specified).

16 - Too many scratch files are given (a maximum of 10 scratch files may be specified).

17 - End of string record was detected where none was expected

18 - Unexpected end-of-file

19 - SORT found a record larger than expected

20 - Record length is not standard for SORTT, SORTA, SORTI.

STEPS IN STRUCTURED PROGRAMMING

1. PROBLEM STATEMENT
2. INPUT/OUTPUT SPECIFICATIONS
3. PROGRAM DESIGN
4. PROGRAM CODING
5. TESTING AND DEBUGGING
6. DOCUMENTATION AND COMMENTS
7. MAINTENANCE

10 - Key size is not positive, SORTS detected a zero or negative key size in its calling parameter

THE STEPS INVOLVED IN STRUCTURED DESIGN ARE:

1. FIRST LOOK	A WHOLE VIEW OF THE PROBLEM
2. ANALYSIS	EXAMINE WHAT THE MAJOR TASKS ARE.
3. SUBROUTINE BREAKDOWN	ASSIGN EACH TASK INTO A MODULE
4. UNIT BREAKDOWN	EACH MODULE IS BROKEN INTO MAJOR UNITS
5. PROCEDURE BREAKDOWN	EACH UNIT IS BROKEN INTO MAJOR SUBROUTINES
6. ACTIVITY BREAKDOWN	EACH SUBPROCEDURE IS BROKEN INTO ACTIVITIES
7. SYNTHSIS	COMBINING ALL MODULES TOGETHER

BREAKDOWN OF PROGRAM COMMANDS

PROGRAM COMMANDS

1. CONTEXT
2. SYNTAX
3. EXAMPLE
4. FUNCTIONS
5. RULES AND CHARACTERISTICS
6. ILLUSTRATIONS
7. APPLICATION

IMPLEMENTATION OF STRUCTURED PROGRAMMING

For the sake of simplicity, the program we choose is very small and less complicated. The idea is to illustrate most of the concepts and techniques discussed in this chapter in the development of a structured program.

The development of a structured program can be viewed as undergoing what is sometimes known as the cycle of birth, death, and resurrection. The birth of the structured program takes place through the process of the input, output specifications of the problem--the STEP 1. This is the FIRST LOOK at structured programming. At this stage, we do not have a clear perception of its parts. We simply see the problem as a whole, and as such it can not be tackled. Hence, we want to have a closer look at it by dividing it into well defined parts. Thus, in structured programming, the gradual death process occurs through progressive and systematic breakdowns of the problem. This breakdown begins with an ANALYTIC VIEW of the problem--the STEP 2. Here we examine the complexity of the problem and we try to adopt the "divide and conquer" principle. We delineate the major tasks involved in the problem. Once we delineate the tasks, we introduce the technique of modularity, namely, we assign each task to functions or subroutines. This marks the first-level breakdown, the SUBROUTINE BREAKDOWN--the STEP 3. Once each module has been defined, it is easy to introduce the TOP-DOWN DESIGN to each module. In this stage, usually there is a general partitioning of each unit into three major units. These three units in each module can usually be identified as Preparation, Process, and Conclusion.

The preparatory unit introduces into the specified task. The process unit does the necessary calculations and computations. In the conclusion unit, the task is wrapped-up. This process may be called the UNITS BREAKDOWN--the STEP 4.

Each of this unit is further broken down into procedures--the STEP 5. This process can be called the PROCEDURE BREAKDOWN. At this stage, care must be taken to choose appropriate program structures such as sequence, selection, or looping. In STEP 6, the procedures are further broken down into subprocedures if necessary. These procedures or sub-procedures, in turn, are broken down into activities. This may be called the ACTIVITY BREAKDOWN. The death processes ends with it.

Thus, the activities are translated into the particular codes. Obviously, in BASIC, these activities are translated into BASIC statements. After this process, we make sure that each of the modules works as desired through testing and debugging. Comments, documentation, and indentations are also inserted as deemed appropriate. These are the cosmetic processes for the funeral. Finally, we combine each of these finished modules together and make it one single program. This is the SYNTHESIS. This synthesis brings about resurrection--structured program. This is STEP 8.

STEPS IN STRUCTURED PROGRAMMING

STEP	EVENT	DESCRIPTION	DIAGRAM
1	FIRST-LOOK	A whole view of the problem without knowing what the parts are.	
2	ANALYSIS	Examine what the mayor tasks are.	
3	SUBROUTINES BREAKDOWN	Assign each task into each module (modularization).	
4	UNIT BREAKDOWN	Each module is broken down into major units. (top down design begins)	
5	PROCEDURE BREAKDOWN	Each unit is broken down into major procedures or sub-procedures. (selection of program structures).	
6	ACTIVITY BREAKDOWN	Each procedure or subprocedure is broken down into activities translatable to language codes.	
7	SYNTHESIS	Combining all modules together. (Appropriate program structures, comments, documentation, indentation, and remarks are necessary).	

ILLUSTRATION OF A STRUCTURED PROGRAM

STEP 1: FIRST LOOK

The problem is to generate a multiple choice quiz program which will allow the user to answer the questions and will give out the result of the quiz.

STEP 2: ANALYSIS

Obviously, the program must contain the set of multiple choice questions, it must receive the answers as input from the user, it must examine its rectitude and validity if necessary, it must assess the number of right and wrong answers and finally, it should print out the result. It is also desirable to explain to the user the nature and purpose of the program in the beginning.

STEP 3: FIRST-LEVEL BREAKDOWN: SUBROUTINES

In this stage we assign the major tasks delineated in the analysis stage into different modules in the proper sequence. Thus, we might arrive at:

MODULE 1: Subroutine explaining the nature and purpose of the program.

MODULE 2: Subroutine to present the current question.

MODULE 3: Subroutine to answer the current question, to make a validity check.

MODULE 4: Subroutine to verify the answer.

MODULE 5: Subroutine to print out the results.

STEP 4, SETP 5, AND STEP 6 (UNIT PROCEDURE AND ACTIVITY BREAKDOWNS)

Step 4, Step 5, and Step 6 are combined in one table shown below. After the modules dealing with different levels of breakdown and coding are well defined, each module is tackled individually.

MODULE 1

	STEP 4 PROCEDURE	STEP 5 ACTIVITY	STEP 6 BASIC STATEMENT
PREPARATION UNIT	Select a subroutine	Call a subroutine	100 GOSUB 1000
PROCESS UNIT	Explain purpose and nature	printout purpose	100 PRINT "PURPOSE"
		printout nature.	100 PRINT "NATURE"
CONCLUSION UNIT	End of sub-routine.	Return to main line.	1020 RETURN

MODULE 2

	PROCEDURE	ACTIVITY	BASIC STATEMENT
PREPARATION UNIT	Select a subroutine	Call a subroutine	120 GOSUB 2000
PROCESS UNIT	1. Present question #1.	Present the ques-tion.	120 PRINT "5X2 IS"
		Present choice 1	210 PRINT, 4
		Present choice 2.	220 PRINT, 6
		Present choice 3.	230 PRINT, 8

		Present choice 4.	240 PRINT, 10
	2. Present question #2.	. . . . .	. . . . .
	3. Present question #3.	. . . . .	. . . . .
	4. Present question #4.	. . . . .	. . . . .
	5. Present question #5.	. . . . .	. . . . .
CONCLUSION UNIT	End the subroutine.	Return to main line.	999 RETURN

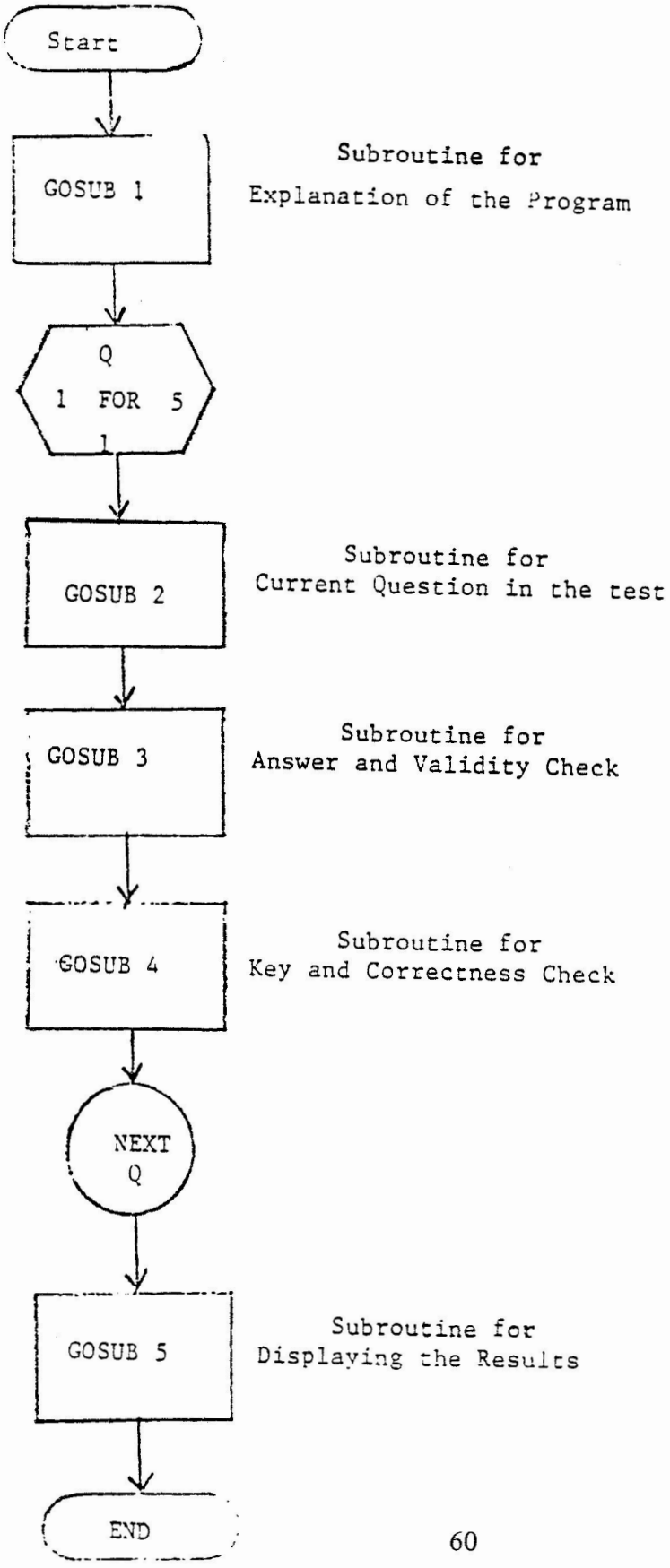
As described above, the rest of the modules, namely, module 3, module 4, and module 5 can be developed in a similar fashion.

After we develop each module, STEP 7: SYNTHESIS, they should be combined into one single program. This constitutes the Main Line of the program. This process marks STEP 7--THE SYNTHESIS. The main line for this program may be as follows:

```
100   GOSUB 100           ! SUBROUTINE FOR EXPLANATION
110   FOR Q = 1 TO 5      ! Q = QUESTION
120     GOSUB 2000        ! SUBROUTINE FOR CURRENT
                          ! QUESTION.
130     GOSUB 3000        ! SUBROUTINE FOR ANSWER AND
                          ! VALIDITY CHECK.
140     GOSUB 4000        ! SUBROUTINE FOR KEY AND
                          ! CORRECTNESS.
150   NEXT Q
160   GOSUB 5000          ! SUBROUTINE FOR RESULTS.
170   STOP
```

The complete program is given below along with the flowchart.

FLOWCHART FOR THE MAIN LINE OF THE PROGRAM



```
                    QUIZ PROGRAM
-----------------------------------------------------
10    ! THE PURPOSE OF THIS PROGRAM IS GO GENERATE A
20    ! FIVE QUESTION MULTIPLE CHOICE QUIZ WHICH WILL
30    ! TELL THE USER IF THE QUESTION IS ANSWERED
40    ! CORRECTLY AND WILL ALSO GIVE THE NUMBER RIGHT
50    ! AT THE END OF THE QUIZ.
60    !
70    GOSUB 200       !SUBROUTINE FOR EXPLANATION
80    FOR Q = 1 TO 5 !Q = QUESTION
90      GOSUB 300     !SUBROUTINE FOR CURRENT QUESTION
100     GOSUB 870     !SUBROUTINE FOR ANSWER AND VALIDY CHECK
110     GOSUB 990     !SUBROUTINE FOR KEY AND CORRECTNESS
120   NEXT Q
130   GOSUB 1070      !SUBROUTINE FOR RESULTS.
140   !-------------------------------------------------
150   REM -  SUBROUTINE FOR EXPLANATION
160   !-------------------------------------------------
200   PRINT "*****************************************"
210   PRINT
220   PRINT "THE FOLLOWING IS A BASIC MATH QUIZ.
230   PRINT "ANSWER EACH QUESTION WITH THE LETTER OF"
240   PRINT "CHOICE YOU FEEL ANSWERS THE QUESTION"
250   RETURN
260   !-------------------------------------------------
270   REM - SUBROUTINE FOR CURRENT QUESTION.
280   !-------------------------------------------------
300   ON Q GO TO 310,420,530,630,740
310   PRINT
320   PRINT
330   PRINT "QUESTION #1"
340   PRINT "WHICH IS THE ANSWER TO THE FOLLOWING FOR X?"
350   PRINT
355   PRINT "X = (2*3) + ((5-1)*2)"
360   PRINT
370   PRINT ,"A) 36.6"
380   PRINT ,"B) 9"
390   PRINT ,"C) 14"
400   PRINT ,"D) -14"
410   RETURN
420   PRINT
430   PRINT
440   PRINT "QUESTION #2"
450   PRINT "WHICH OF THE FOLLOWING IS THE CORRECT"
460   PRINT "SOLUTION FOR Y IN THE EQUATION BELOW?"
465   PRINT
470   PRINT "Y = (((3*2)-1)-2) + 1"
475   PRINT
480   PRINT ,"A) 4"
```

```
490  PRINT ,"B) -4"
500  PRINT ,"C) 3"
510  PRINT ,"D) 26"
520  RETURN
530  PRINT
540  PRINT
550  PRINT "QUESTION #3"
560  PRINT "WHICH OF THE FOLLOWING IS THE CORRECT SOLUTION"
570  PRINT "FOR Z IN THE EQUATION BELOW?"
580  PRINT
585  PRINT "Z = ((((3*2)*(3-1)/2)-1)/1)"
590  PRINT
595  PRINT ,"A) 0"
600  PRINT ,"B) 5"
605  PRINT ,"C) 4"
610  PRINT ,"D) 2"
620  RETURN
630  PRINT
640  PRINT
650  PRINT "QUESTION #4"
660  PRINT "WHICH OF THE BELOW IS NOT A PROPER"
670  PRINT "MATHEMATICAL EXPRESSION IN VAX BASIC?"
680  PRINT
690  PRINT ,"A) (A*B-4*X+Y)*1-3/4*(1)"
700  PRINT ,"B) 1*2*3*4*5*6/1*1-1"
710  PRINT ,"C) 222/1+0"
720  PRINT ,"D) 3*(5/1(2*3))"
730  RETURN
740  PRINT
750  PRINT
760  PRINT "QUESTION #5"
770  PRINT "WHICH OF THE BELOW IS A CORRECT VERSION"
780  PRINT "OF THE QUADRATIC FORMULA?"
790  PRINT
800  PRINT ,"A) (-B + SQRT(B**2-4*A*C))/(2*A)"
810  PRINT ,"B) B-4*A*C"
820  PRINT ,"C) B**2-4*A/2*A"
830  PRINT ,"D) SQRT(B**2-4*A*C)"
840  RETURN
850  !-----------------------------------------------
860  REM - SUBROUTINE FOR VALIDITY CHECK AND ANSWER
865  !-----------------------------------------------
870  PRINT
880  PRINT "WHAT IS YOUR CHOICE";
890  INPUT A$
900  IF A$ = "A" THEN 960
910  IF A$ = "B" THEN 960
920  IF A$ = "C" THEN 960
930  IF A$ = "D" THEN 960
940  PRINT "INVALID RESPONSE, PLEASE RETYPE ENTRY.
950  GO TO 880
```

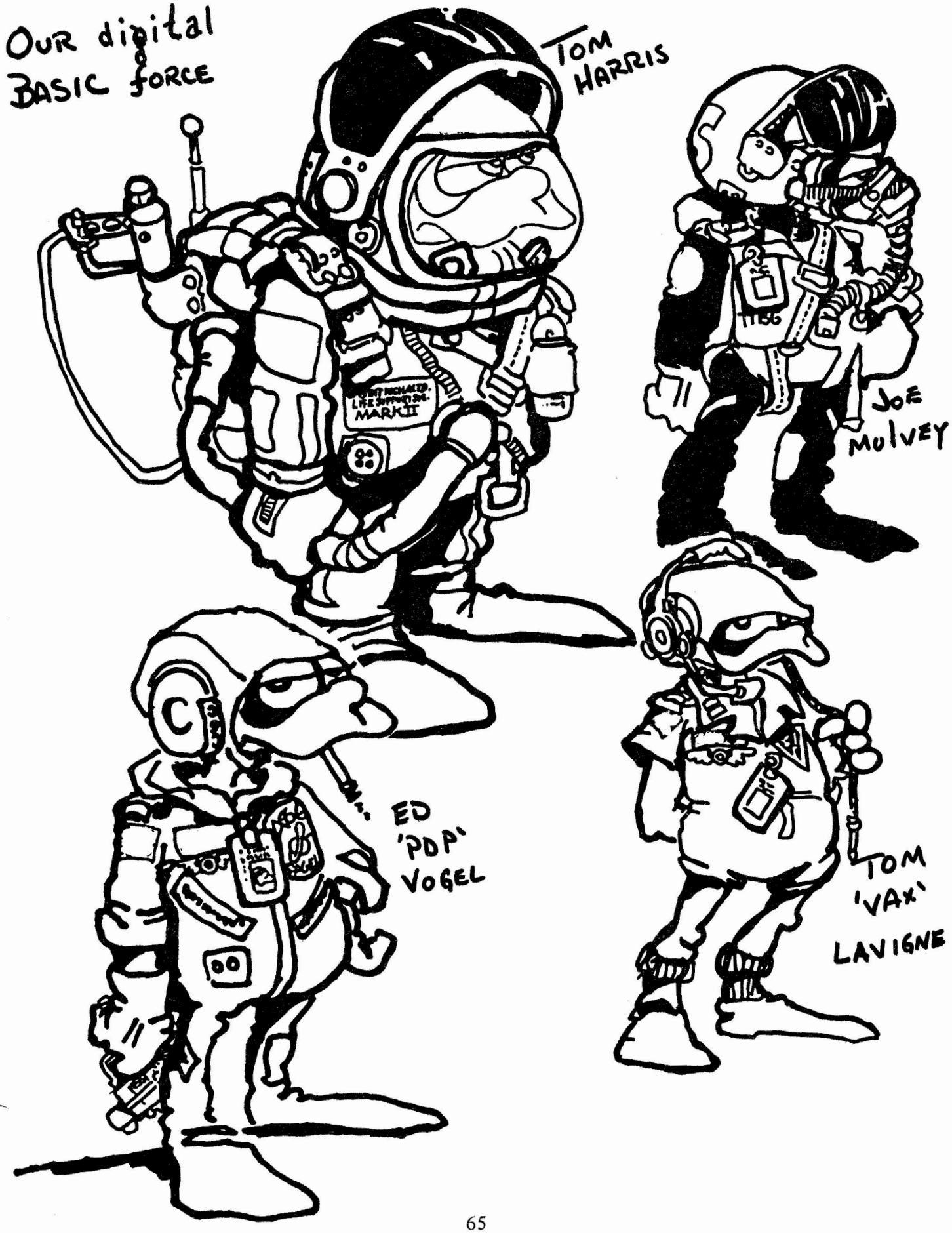
```
960  RETURN
970  !-------------------------------------------------
980  REM - SUBROUTINE FOR KEY AND CORRECTNESS
985  !-------------------------------------------------
990  READ K$
1000 IF A$ = K$ THEN 1025
1010 PRINT
1020 PRINT "INCORRECT, ";K$" WAS THE CORRECT ANSWER."
1022 GO TO 1040
1025 PRINT
1030 PRINT "CORRECT!!   ";K$;" IS THE CORRECT ANSWER."
1035 LET C = C+1         !C=NUMBER OF QUESTIONS CORRECT
1040 RETURN
1050 !-------------------------------------------------
1055 REM - SUBROUTINE FOR RESULTS
1065 !-------------------------------------------------
1070 LET P = C/.05       !P=PERCENTAGE CORRECT
1080 PRINT
1090 PRINT "THAT IS THE END OF OUR FIVE QUESTION QUIZ"
1100 PRINT "YOU HAD ";C;" OUT OF FIVE QUESTIONS CORRECT."
1105 PRINT "THAT IS ";P;"%."
1110 PRINT
1115 IF C = 5 THEN 1160
1120 IF C = 4 THEN 1170
1130 IF C = 3 THEN 1180
1140 IF C < 3 THEN 1190
1150 PRINT
1160 PRINT "GREAT JOB, YOU GOT THEM ALL CORRECT!!"
1165 GO TO 1200
1170 PRINT "GOOD JOB, YOU ALMOST GOT THEM ALL!"
1175 GO TO 1200
1180 PRINT "FAIR JOB, THAT IS ABOUT AVERAGE."
1185 GO TO 1200
1190 PRINT "YOU DID NOT DO VERY WELL."
1200 RETURN
1210 !
1215 DATA "C","A","B","D","A"
1220 END
```

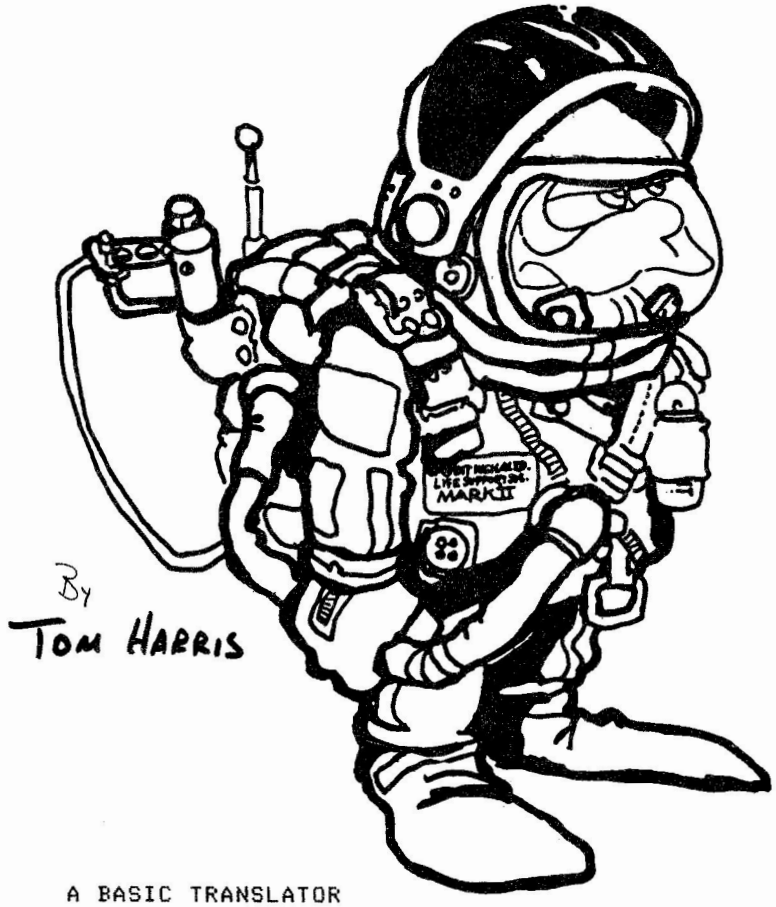
```
READY
RUNNH
```

PROGRAM RUN

```
THE FOLLOWING IS A BASIC MATH QUIZ.
ANSWER EACH QUESTION WITH THE LETTER OF
CHOICE YOU FEEL ANSWERS THE QUESTION
```

```
QUESTION #1
WHICH IS THE ANSWER TO THE FOLLOWING FOR X?

X = (2*3) + ((5-1)*2)

            A) 36.6
            B) 9
            C) 14
            D) -14

WHAT IS YOUR CHOICE ?I
INVALID RESPONSE, PLEASE RETYPE ENTRY.*
WHAT IS YOUR CHOICE ?C

CORRECT!!   C IS THE CORRECT ANSWER.


QUESTION #2
WHICH OF THE FOLLOWING IS THE CORRECT
SOLUTION FOR Y IN THE EQUATION BELOW?


Y = (((3*2)-1)-2) + 1

            A) 4
            B) -4
            C) 3
            D) 26

WHAT IS YOUR CHOICE ?A

CORRECT!!   A IS THE CORRECT ANSWER.


QUESTION #3
WHICH OF THE FOLLOWING IS THE CORRECT SOLUTION
FOR Z IN THE EQUATION BELOW?

Z = ((((3*2)*(3-1)/2)-1)/1)

            A) 0
            B) 5
            C) 4
            D) 2

WHAT IS YOUR CHOICE ?D

INCORRECT, B WAS THE CORRECT ANSWER.
```


Our digital
BASIC force
TOM
HARRIS
JOE
MULVEY
ED
'PDP'
VOGEL
TOM
'VAX'
LAVIGNE



A BASIC TRANSLATOR

The purpose of the BASIC translator is threefold: Help move code to V2 BASIC for BASIC-PLUS-2, convert MICROBASIC to V2, and to make the program pretty print. The program is written in V2 BASIC, and is easily modified because of heavy use of instr and data tables for MICROBASIC. Unfortunately it is not perfect but it is free to DECUS.

Traditional BASIC-PLUS-2 code

```
1               DIM A(10,10),B(10)
2               M,N = 10
3               B(I) = RND*200 FOR I = 1 TO N
10              PRINT "REPORT BEGINS"                                   &
        \       A(I,J) = B(I)*100 FOR J = 1 TO M FOR I = 1 TO N         &
        \       PRINT                                                   &
        \       FOR I = 1 TO N                                          &
        \       PRINT A(I,J) FOR J = 1 TO M                             &
        \       IF B < 100 THEN                                         &
        \       A$ = "_"                                                &
        \       ELSE A$ = " "
20              PRINT A$                                                &
        \       NEXT I                                                  &
        \       PRINT
90              END
```

```
                Same code - (old) CUSP convention

1               DIM A(10,10),B(10)
2               M,N = 10
3               B(I) = RND*200 FOR I = 1 TO N
10              PRINT 'REPORT BEGINS'                                   \       &
                A(I,J) = B(I)*100 FOR J = 1 TO M FOR I = 1 TO N \       &
                PRINT                                                   \       &
                FOR I = 1 TO N                                          \       &
                PRINT A(I,J) FOR J = 1 TO M                             \       &
                IF B < 100 THEN                                         \       &
                A$ = '_'                                                \       &
                ELSE A$ = ' '
20              PRINT A$                                                \       &
                NEXT I                                                  \       &
                PRINT
90              END

                Same code after 3 maintainers and 2 years

1               DIM A(10,10),B(10)
2               M,N = 10
3               B(I) = RND*200 FOR I = 1 TO N
10              PRINT 'REPORT BEGINS'                                   \       &
                A(I,J) = B(I)*100 FOR J = 1 TO M FOR I = 1 TO N \       &
                PRINT                                                           &
        \       FOR I = 1 TO N                                          \       &
                PRINT A(I,J) FOR J = 1 TO M                                     &
        \       IF B < 100 THEN                                                 &
        \       A$ = '_'                                                \       &
                ELSE A$ = ' '                                                   &
20              PRINT A$                                                \       &
                NEXT I                                                          &
        \       PRINT
90              END

                Same code - as it existed under RSTS/E V6

1               DIM A(10,10),B(10)
2               M,N = 10
3               B(I) = RND*200 FOR I = 1 TO N
10              PRINT 'REPORT BEGINS'                                           &
        \       A(I,J) = B(I)*100 FOR J = 1 TO M FOR I = 1 TO N                 &
        \       PRINT\FOR I = 1 TO N\PRINT A(I,J)                               &
                FOR J = 1 TO M\IF B < 100 THEN                                  &
        \       A$ = '_'\ELSE A$ = ' '
20              PRINT A$\NEXT I\PRINT
90              END

                Same code - as it started under RSTS/E V4
```

```
1  DIM A(10,10),B(10)
2  M,N=10
3  B(I)=RND*200FORI=1TON
10 &"REPORT BEGINS"&
   \A(I,J)=B(I)*100\FORJ=1TOMFORI=1TON&
   \&\FORI=1TON\&A(I,J)&
   FORJ=1TOM\IFB(I)<100THEN&
   A$="_"\ELSEA$=" "
20 &A$\NEXTI\&
90 END
```

Running the program

```
$ RUN FIXIT
FIXIT V 1.0 08:04 AM
What input file <TT:> (?=HELP) ? B747.B2S
What output file <TT:> ? B474.BAS
Is this a Microprocessor BASIC Program <NO> ? NO
Want to customize program conversion ? Answer Yes or No <N> ? NO
100 200 300 400 500 600 700 1000 1300 1400
(1210) Input Lines from B747.BAS, (1501) lines written to B747.BAS
(99) lines of DIM and MAP statements moved.
(30) seconds elapsed time...(2420)lines/minute
$
```

Running the program using the customize conversion option

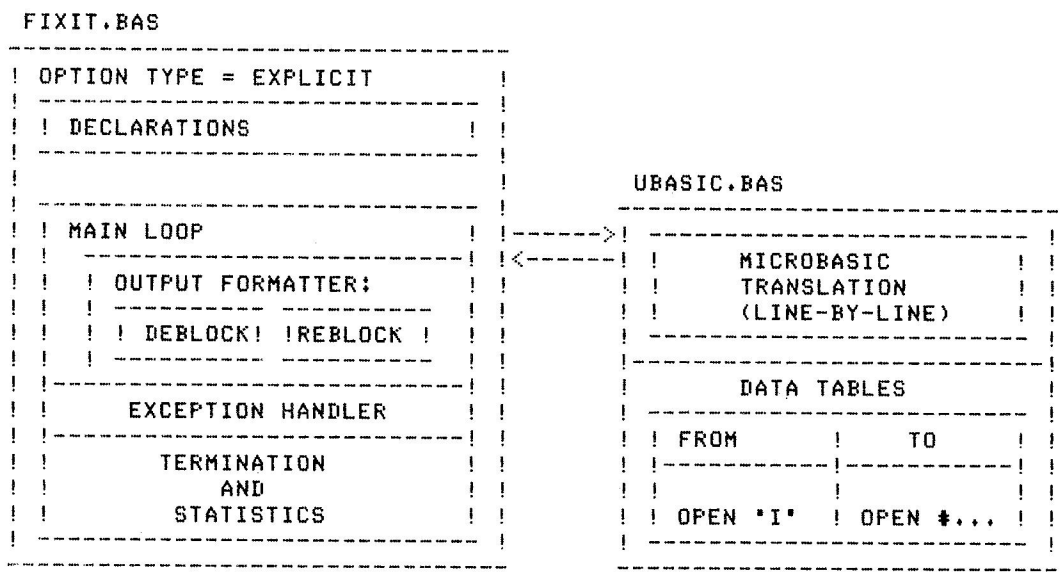
```
$ RUN FIXIT
FIXIT V 1.0 08:04 AM
What input file <TT:> (?=HELP) ? DATABASE.B2S
What output file <TT:> ? DATABASE.BAS
Is this a Microprocessor BASIC Program <NO> ? NO
Want to customize program conversion ? Answer Yes or No <N> ? YES
What line number should be used for moved DIM's and MAP's <3> ? -1
What column should be used to start comments in <16> ? 32
100
(137) Input Lines from DATABASE.B2S, (163) lines written to DATABASE.BAS
(0) lines of DIM and MAP statements moved.
(3) seconds elapsed time...(2740)lines/minute
$
```

Running the program to convert MICROBASIC to V2 BASIC

```
$ RUN FIXIT
FIXIT V 1.0 08:04 AM
What input file <TT:> (?=HELP) ? CPM.OLD
What output file <TT:> ? CPM.TMP
Is this a Microprocessor BASIC Program <NO> ? YES
Want to customize program conversion ? Answer Yes or No <N> ?
100 200 300 400 500
(222) Input Lines from CPM.OLD, (514) lines written to CPM.OLD
(1) lines of DIM and MAP statements moved.
(38) seconds elapsed time...(350)lines/minute
$
```

Some major syntactic issues are comments, statement modifiers, literals and long lines. Long line rule: If <= 132 characters, pass it through, if > 132 characters, force continuation breaking on: + , ;) AND OR and repeat as above.

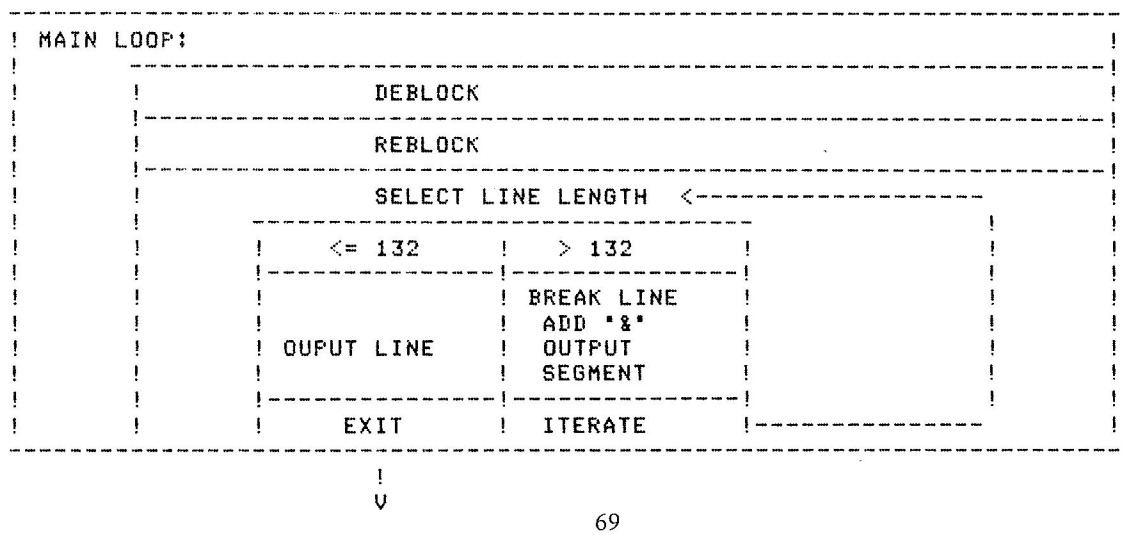
Translator structure



Subroutines

DEBLOCK: Invoke MICROBASIC translation
Removes "&"
Assemble entire (continued) line
Break at "\","THEN","ELSE", OR "!"

REBLOCK: Count block (IF,FOR,WHILE,UNTIL,UNLESS)
Change "REM" to "!"
Center comment text
Insert "END IF" as needed
Move COMMON/DIM/MAP



MICROBASIC conversion is done by DEF's added at line 30000 including: INKEY, AT, MKS, MKI, LOF and EOF. Spaces are inserted and many other transformations are performed.

Transformations

'	-------->	!
&	-------->	PRINT
/BAS	-------->	.BAS
/DAT	-------->	.DAT
LINE INPUT	-------->	LINPUT
LPRINT	-------->	PRINT #9,
LPRINT USING	-------->	PRINT #9 USING
REM	-------->	\REM
PRINT @	-------->	PRINT AT
:	-------->	\
WEND	-------->	NEXT
OPEN "I",4,"ABC/DAT"	-------->	OPEN "ABC.DAT" FOR INPUT AS FILE #4
OPEN "O",5,"ABC/BAS"	-------->	OPEN "ABC.BAS" FOR OUTPUT AS FILE #5
IF A>2 THEN 190 ELSE 200	-------->	IF A>2 THEN GOTO 190 ELSE GOTO 200
IF A>2 PRINT "OK"	-------->	IF A>2 THEN PRINT "OK"
IF A>2 INPUT B$	-------->	IF A>2 THEN INPUT B$
IF A>2 LET C=4	-------->	IF A>2 THEN LET C=4

Functions

INKEY	SINGLE CARACTER INPUT
MKI$	
MKS$	DATA CONVERSION
MKD$	
@121	A SPECIFIC LOCATION (121)

Not handled (yet)

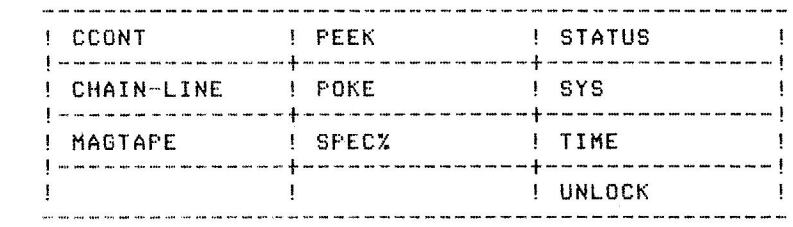
CMD	HOST MONITOR COMMAND
DEFDBL	
DEFFN	
DEFINT	
DEFSNG	DECLARATIVE CONTROLS
DEFUSR	
DEFSTR	
ERROR	SIGNAL AN ERROR
OUT	DEVICE (PORT) OUTPUT
PEEK	ADDRESS (READ)
POKE	ADDRESS (WRITE)

Some bonuses are RSTS/E dependency flasser, simpler than FIXIT, faster and the same price (free). The flasser takes file-name or @file outputs exceptions report (summary to terminal, details to <your> file). It is also very easily modified because it uses INSTR tests and GOSUB checks.

Using the BASIC-PLUS dependency filter

```
WHAT INPUT FILE <TT:>           ? BASICPLUS.BAS
WHAT REPORT FILE <TT:>          ? RSTS.RPT

BASICPLUS.BAS: 349 LINES PROCESSED, 40 WRITTEN TO RSTS.RPT
               3                PEEK
               3                POKE
               1                OPEN <RSTS dependency>
               12               KILL <RSTS dependency>
               2                NAME..AS <RSTS dependency>
               3                COUNT
               3                UNLOCK
               3                MAGTAPE function
               2                TIME <non-zero argument>
               2                CHAIN..LINE
               3                STATUS function
               2                SPEC% function
               1                SYS function
```

Dependencies



With all of this information you should be able to use the programs.

```
1       %TITLE  "FIXIT Utility for Old BASIC Programs"
        %SBTTL  "Declarations and Variable Directory"
        %IDENT  "FIXIT"

        OPTION TYPE = EXPLICIT

        !          Author:      Tom Harris   August 1, 1983
        !                       Digital Equipment Corp (ZK2-3/K06)
        !                       110 Spit Brook Road
        !                       Nashua, NH  USA     03062
        !       ------------------------------------------------------------
        !          Input:       an input file name, and conversion controls
        !                       input can be BASIC-PLUS-2 V1.6 or MicroBASIC
        !                       style programs. Keyboard input is allowed.
        !       ------------------------------------------------------------
        !           Output:     a BASIC program, formatted for V2 BASIC
        !       ------------------------------------------------------------
        !           Support:    Here it is, have fun, suggestions welcome,
        !                       but no guarantees. <user-supported>
        !       ------------------------------------------------------------
        !           THIS IS A HANDOUT DEVELOPED FOR FALL US DECUS 1983
        !       ------------------------------------------------------------
        DECLARE INTEGER CONSTANT                                            &
                         No_Tabs      = 16       ! EDIT$ code               &
                        ,No_Blanks    = 424      ! EDIT$ codes              &
                        ,Trim_Front   = 8        ! EDIT$ code               &
                        ,Trim_Back    = 128      ! EDIT$ code               &
                        ,TRUE         = -1

        DECLARE STRING CONSTANT Break = "\"      ! Backslash                &

        DECLARE STRING                                                      &
                         DIMs.MAP(200)           ! To move -> top/prog.     &
                        ,F.In                    ! Input file name          &
                        ,F.Out                   ! Output File Name         &
                        ,Main                    ! Working input buffer     &
                        ,Out                     ! Output line image        &
                        ,Source                  ! Input line image         &
                        ,S1 ,S2, S3              ! Temp String
        DECLARE LONG                                                        &
                         Bad.File                ! Error on File OPEN       &
                        ,Comment.Column          ! Comments to column x     &
                        ,DIMs.MAPs.Count         ! Count DIM/MAPs           &
                        ,TAB.Back                ! Fixup indents w/this     &
                        ,First.Break             ! THEN/ELSE/... tests      &
                        ,Endfile                 ! End-of-File flag         &
                        ,I                       ! General Use              &
                        ,IF.Count                ! Nesting counter          &
                        ,Lines.In                ! Counts # Lines input     &
                        ,Lines.Out               ! Lines output             &
                        ,Loop.Count              ! Loop nesting counter     &
                        ,MicroBASIC              ! TRUE if a micro BASIC    &
                        ,MAP.DIM.Line.Number     ! Line # for DIM/MAP's     &
                        ,Start.Time              ! Seconds since 00:00      &
                        ,T1 ,T2, T3              ! Temp working storage     &
                        ,TT.Output               ! TRUE if F.Out="TT:"

        %PAGE
```

```
	!	The following flags are used by the uBASIC FUNCTION
	!	as a type of "OWN" storage, i.e. static storage
	!	that retains values between invocations. They are
	!	initialized to 0 as the main program starts running.

	!	The flags correspond to uBASIC statements or functions
	!	for which DEC BASIC functions or subroutines must
	!	be generated (i.e. no easy 1-1 transform exists).

	!	e.g. PRINT @128,"Hi"  becomes  PRINT AT(128);"Hi"

	!	and a BASIC function named AT has to be inserted
	!	into the front of the converted program.

	MAP (FLAGS) BYTE Action.Flag	! Caller must do something &
			,At_Flag	! Output PRINT AT functn   &
			,MKS_Flag	! Output MKS function      &
			,MKI_Flag	! Output MKI function      &
			,MKD_Flag	! Output MKD function      &
			,CVS_Flag	! Output CVS function      &
			,CVI_Flag	! Output CVI function      &
			,CVD_Flag	! Output CVD function      &
			,INKEY_Flag	! Output INKEY$ function

	MAP (FLAGS) BYTE All_Flags(10)  ! This makes initializing easier


	!	Below, we test the %VARIANT value (use the SET VARIANT command in the BASIC environment)
	!	to see whether we should minimize the size of this program by excluding HELP text and by
	!	skipping the call on UBASIC thus omitting that code from the executable program. Minimizing
	!	in this manner lets one run FIXIT on a PDP-11 system without having to overlay any code
	!	thus getting best possible performance.
	!
	!	The default is NOT to minimize. To request the smaller program:  SET VARIANT:1  before compiling
	!

	%LET	%Small	= 1%		! SET VARIANT = 1 when compiling, and the MicroBASIC code drops out,
	%LET	%Large	= 2%		!	and then you don't have to overlay the translator

	%IF %VARIANT = 0%
	    %THEN
		%LET %Size = %Large
		EXTERNAL STRING FUNCTION uBASIC(STRING) ! This EXTERNAL happens only on %Large systems
	    %ELSE
		%LET %Size = %Small
	%END %IF
%PAGE
```

```
        1 = CTRLC
        ON ERROR GOTO Oops

        GOTO Bye IF Bad.File    ! Error Handler "OOPS" resumes here when unable to OPEN the *INPUT* file

        PRINT "FIXIT V1.0-AA ";TIME$(0)
        PRINT " "

Begin_Processing:

        LINPUT "What input  file <TT:>  (? = HELP) ";F.In

        ! Get an input file name, and append .BAS to it
        ! as a default extension - unless the file name
        ! might be a device spec, e.g. TT:  or  TTA3:

        F.In = EDIT$(F.In,No_Blanks)

        IF F.In = "?"
        THEN
            GOSUB Help
            GOTO  Begin_Processing
        END IF

        F.In = F.In + ".BAS"    IF (F.In <>"") AND 0=INSTR(1,F.In,".") AND 0=INSTR(1,F.In,":")
        F.In = "TT:"            IF LEN(F.In) = 0

        OPEN F.In FOR INPUT AS FILE #1, ACCESS READ, VARIABLE, RECORDSIZE 132

        LINPUT "What output file <TT:> ";F.Out

        ! Do the same thing for the output file name

        F.Out = F.Out + ".BAS"  IF EDIT$(F.Out,No_Blanks)<>"" AND 0=INSTR(1,F.Out,".") AND 0=INSTR(1,F.Out,":")

        IF EDIT$(F.Out,No_Blanks)=""
        THEN
            F.Out="TT:"
            TT.Output = True
        END IF

        OPEN F.Out FOR OUTPUT AS FILE #2,VARIABLE, RECORDSIZE 132

        %PAGE
```

```
MAP.DIM.Line.Number     = 3      ! Default: move MAP/DIM's to line "3"
Comment.Column          = 16     ! Default: try to start comments in column 16
DIMs.MAPs.Count         = 0      ! Global counter: tels how many DIM/MAP's saved up to move when program ends.

All_Flags(I)            = 0        FOR I = 0 to 10

ON ERROR GOTO Hiccup

LINPUT "Is this a Microprocessor BASIC Program <NO>";S1
MicroBASIC = TRUE                  IF EDIT$(Left(S1+"N",1),No_Blanks)="Y"

LINPUT "Want to customize program conversion ? Answer Yes or No <N> ";S1

IF EDIT$(LEFT(S1+"N",1),No_Blanks) = "Y"
THEN
    INPUT  "What line number should be used for moved DIM's and MAP's <3>";MAP.DIM.Line.Number
    MAP.DIM.Line.Number = 3 IF MAP.DIM.Line.Number = 0

    INPUT  "What column should be used to start comments in <16>";Comment.Column
    Comment.Column = 16 IF Comment.Column < 2  OR  Comment.Column > 60

    S1,S2,S3 = ""
END IF

Start.Time = TIME(0%)              ! Start timing the conversion
%PAGE
```

```
        %SBTTL "Main Loop"

Main_Loop:

        WHILE TRUE

            GOSUB DeBlock_Line

        EXIT Main_Loop IF Endfile AND LEN(Main)=0

            GOSUB ReBlock_Line

            Break.Lines:

               IF LEN(Out) > 132                        !  This is where we break a line that is too long ...
               THEN
                   T1 = INSTR(78,Out,";")
                   T2 = INSTR(78,Out,",")               !  We'll break on a comma, semicolon, plus, AND, OR, or paren
                   T3 = INSTR(78,Out,"+")
                   T3 = INSTR(78,Out," AND ")  IF T3 = 0
                   T3 = INSTR(78,Out," OR ")   IF T3 = 0

                   T1 = T2      IF T1 = 0   OR   (T2<>0 AND T2<T1)
                   T1 = T3      IF T1 = 0   OR   (T3<>0 AND T3<T1)
                   T1 = INSTR(78,Out,")")        IF T1 = 0

                   PRINT #2, MID(Out,1,T1)+"   &"
                   Lines.Out = Lines.Out + 1

                   Out = SPACE$(7)+EDIT$(RIGHT(Out,T1+1),Trim_Front)
                   GOTO Break.Lines
               ELSE
                   IF LEN(Out) <> 0
                   THEN
                       PRINT #2, Out                    !       Here, the line fits OK - no need to break it
                       Lines.Out = Lines.Out + 1
                   END IF
               END IF

           PRINT Lines.Out;    IF Lines.Out = 100*INT(Lines.Out/100)  AND  NOT TT.Output ! Show progress...
        NEXT

        PRINT   IF TT.Output

        GOTO Done

        %PAGE
```

```
	%SBTTL  "HELP Text"

Help:	PRINT " "
	%IF %Size = %Large
	%THEN
		PRINT " "
		PRINT " This program converts V1 BASIC programs to a  possibly  more"
		PRINT " readable  and  executable format under V2 BASIC.  It handles"
		PRINT " BASIC-PLUS-2 V1.x source as well as elements  of  MicroBASIC"
		PRINT " source code."
		PRINT " "
		PRINT " NOTE:  This is a sample  program  which  itself  illustrates"
		PRINT " capabilities  of  DEC BASIC.  The program has purposely been"
		PRINT " kept to a simple  no-brains  approach  -  it  does  not  use"
		PRINT " sophisticated parsing techniques and thus does not perfectly"
		PRINT " convert all  of  the  possible  program  formats  which  the"
		PRINT " BASIC's  allow.   However, it has proven useful for a number"
		PRINT " of different programs, and can no doubt be altered to handle"
		PRINT " even more."
		PRINT " "
		PRINT " The program moves MAPs and DIMs, changes REM  statements  to"
		PRINT " '!',  inserts  'END  IF'  statements  where  necessary,  and"
		PRINT " <incidentally> indents statement blocks to indicate  control"
		PRINT " structures while performing minor clean-ups on code."
		PRINT " "
		PRINT " You will be asked  for  input  and  output  file  names  and"
		PRINT " whether  you  wish  to supply customizing directions or take"
		PRINT " default processing parameters.  Input and Output file  names"
		PRINT " are  assumed  to have a .BAS extension, so you need not type"
		PRINT " it in (other extensions are permitted - just type 'em in!)."
		PRINT " "
		PRINT " You  will be  requested to  tell the FIXIT  program  whether"
		PRINT " the  input  text is in space-compressed MicroBASIC format so"
		PRINT " that expansions can be performed  (and  syntax  peculiar  to"
		PRINT " that implementation can be massaged into forms acceptable on"
		PRINT " this system."
		PRINT " "
		PRINT " Customizing means telling the FIXIT program whether to  move"
		PRINT " COMMON/DIMENSION/MAP statements and what line number to move"
		PRINT " them to (if you give a line  number  less  than  zero,  they"
		PRINT " won't  be moved - may be advisable if you have truly complex"
		PRINT " MAP or COMMON statements), you will be asked what column  to"
		PRINT " TRY  to align comments on (numbers between 8 and 70 are OK)."
		PRINT " "
		PRINT " The  program  then  runs,  reporting  each  100  <output>"
		PRINT " statements  processed.  Termination reporting tells how many"
		PRINT " input  statements  and  how  many  output  statements  were"
		PRINT " processed."
		PRINT " "
	%ELSE
		PRINT "Help Text is not available"
	%END %IF
 RETURN
	%PAGE
```

```
        %SBTTL  "Source Input Routine"

DeBlock_Line:                                   ! Return 1 stmt/line in *Source*

        Out,Source,S1,S2 = ""

        IF LEN(Main)=0 AND NOT Endfile          ! *Main* holds working text
        THEN
            IF LEN(S3) = 0
            THEN
                LINPUT  #1, Main
                Lines.In = Lines.In + 1
                Main = EDIT$(Main,No_Blanks)

                %IF %Size = %Large
                %THEN
                    Main = uBASIC(Main) IF MicroBASIC
                %END %IF

            ELSE
                Main = S3
                S3 = ""
            END IF
        END IF


  Handle.Continuation:

        IF MID(Main,LEN(Main),1)="&"            ! Continuation lines -> *Main*
        THEN
            T1 = INSTR(1,Main,"!")
            Main  = EDIT$(LEFT(Main,LEN(Main)-1),Trim_Back)+" "

            LINPUT #1, S1
            S1    = EDIT$(S1,No_Blanks)
            Lines.In = Lines.In + 1

            IF T1= 0
            THEN
                S1    = Break+" "+S1                IF LEFT(S1,1)="!"
                Main  = Main + EDIT$(S1,Trim_Front)
                GOTO Handle.Continuation
            ELSE
                S3 = S1
                S1 = ""
            END IF
        END IF

        %PAGE
```

```
        T1    = INSTR(1,Main,Break)             ! NOW, get a stmt into *Source*

        IF T1 <> 0
        THEN
            Source = EDIT$(LEFT(Main,T1-1),Trim_Front)
            Main   = EDIT$(RIGHT(Main,T1+1),Trim_Front)
            Main   = Break+Main IF LEN(Source) <> 0
            GOTO     DeBlock_Line       IF LEN(Source) = 0
        ELSE
            Source = EDIT$(Main,Trim_Front)
            Main   = ""
        END IF

        T1 = INSTR(1,Source,"!")          ! Break lines on THEN, ELSE and
        T2 = INSTR(1,Source,"THEN")       ! work to avoid being fooled by
        T3 = INSTR(1,Source,"ELSE")       ! comments containing THEN/ELSE

RETURN  IF (T2 + T3) = 0                  ! <none of the above>

RETURN  IF (T1<T2) AND (T1<T3) AND (T1<>0) ! <comment precedes the THEN or the ELSE>

        First.Break = 999                 ! Aha, something(s).. pick first one

        First.Break = T2 IF T2 <> 0 AND T2 < First.Break AND ( (T1 = 0) OR (T1<>0 AND T2<T1) )
        First.Break = T3 IF T3 <> 0 AND T3 < First.Break AND ( (T1 = 0) OR (T1<>0 AND T3<T1) )

        S1     = EDIT$(LEFT( Source,First.Break-1),Trim_Front)
        Main   = EDIT$(RIGHT(Source,First.Break  ),Trim_Front)+" "+Main
        Source = S1

        IF LEN(Source) = 0                        ! Handle THEN or ELSE in Column 1
        THEN
            S1     = LEFT(Main,4)
            Main   = EDIT$(RIGHT(Main,5),Trim_Front)
            Main   = "GOTO " + Main       IF 0 <> INSTR(1,"0123456789",LEFT(Main,1)) AND LEN(Main) <> 0
            Source = S1
        END IF

RETURN

        %PAGE
```

```
      %SBTTL  "Output Editing Subroutine"

ReBlock_Line:                                   ! Do pretty tabs & text fixups...

          Source = EDIT$(Source,Trim_Back)
RETURN    IF LEN(Source) = 0

          T1 = INSTR(1,Source,"!")              ! Handles RSTS CUSP convention
          S1 = EDIT$(Left(Source,T1-1),Trim_Back+No_Tabs)
          T3 = LEN(S1)

          Source = S1+SPACE$(Comment.Column-T3) + RIGHT(Source,T1)  IF T1<Comment.Column AND T1>0

          S1     = LEFT(Source,1)
          T1     = ASCII(S1)
          S2     = ""

          SELECT T1                                     ! Look for Line numbers
              CASE 48,49,50,51,52,53,54,55,56,57        ! Start text in column 9

                   T2  = INSTR(1,Source," ")    ! Space
                   T3  = INSTR(1,Source,"      ")       ! TAB

                   T2  = T3             IF  (T2 = 0)
                   T2  = T3             IF  (T3 > 0) AND (T3 < T2)
                   T2  = LEN(Source)+1          IF T2 = 0

                   Out = LEFT(Source,T2-1) + SPACE$(8-T2) + EDIT$(RIGHT(Source,T2),Trim_Front)

                   Out = Out + "!"              IF T2 = LEN(Source)+1

                   IF IF.Count > 0
                   THEN
                       PRINT #2, SPACE$(7+4*(Loop.Count+(IF.Count-I)));"END IF" FOR I=1 TO IF.Count
                       Lines.Out = Lines.Out + IF.Count
                       IF.Count = 0
                   END IF

              CASE ELSE
                   Out = SPACE$(7)+EDIT$(Source,Trim_Front)
          END SELECT

          ! *WHEW* BASIC source in col-8
          ! Now, do fixups...

          Out = LEFT(Out,7)+"! "+EDIT$(RIGHT(Out,12),Trim_Front)  IF MID(Out,8,3)="REM"
          Out = LEFT(Out,7)     +EDIT$(RIGHT(Out,12),Trim_Front)  IF MID(Out,8,3)="LET"

          %PAGE
```

```
%SBTTL "Pretty Print Formatting"

Loop.Count = Loop.Count - 1 IF MID(Out,8,4)="NEXT"
Loop.Count = 0              IF Loop.Count < 0

IF.Count = IF.Count - 1     IF 0 <> INSTR(1,Out,"END IF") AND IF.Count > 0
IF.Count = 0                IF IF.Count < 0

T1  = INSTR(1,Out,"ELSE")
T3  = INSTR(1,Out,"!")

TAB.Back = 0                ! <make it appear same level as IF>
TAB.Back = 1                IF T1 > 0 AND ((T3=0) OR (T3<>0 AND T1<T3))

S2  = Out

Out = LEFT(Out,7) + SPACE$(4*(IF.Count-TAB.Back)) + SPACE$(4*Loop.Count) + RIGHT(Out,8)

T3  = INSTR(1,Out,"!")      ! Try to Center comments...
S1  = EDIT$(Left(Out,T3-1),Trim_Back)
T2  = LEN(S1)

Out = S1+SPACE$(Comment.Column-T2)+EDIT$(RIGHT(Out,T3),Trim_Front) IF T3<Comment.Column AND T3>0

T1  = INSTR(1,Source,"!")   ! Key off THEN (not stmt modifier)
T2  = INSTR(1,Source,"THEN")

IF.Count = IF.Count + 1     IF T2<>0 AND (T1=0 OR (T1<>0 AND T2<T1))

Loop.Count = Loop.Count + 1 IF MID(S2,8,4)="FOR "
Loop.Count = Loop.Count + 1 IF MID(S2,8,6)="UNLESS"
Loop.Count = Loop.Count + 1 IF MID(S2,8,5)="UNTIL"
Loop.Count = Loop.Count + 1 IF MID(S2,8,5)="WHILE"

%PAGE
```

```
            %SBTTL "Handle COMMON/DIM/MAP Statements"

            T1  = MID(S2,8,3)="DIM"      ! Test for DIMension
            T2  = MID(S2,8,3)="MAP"      ! Test for MAP
            T3  = MID(S2,8,3)="COM"      ! Test for COMMON

RETURN      IF T1+T2+T3 = 0  OR  MAP.DIM.Line.Number < 0

            S1 = LEFT(Out,7)
            Out= EDIT$(RIGHT(Out,8),16+128+256+8)

            I  = INSTR(1,Out,",")

            IF  I <> 0 AND T1=0
            THEN
                DIMs.MAPs.Count = DIMs.MAPs.Count + 1
                DIMs.MAP(DIMs.MAPs.Count) = SPACE$(7)+LEFT(Out,I)+SPACE$(58-I)+" &"
                Out = EDIT$(RIGHT(Out,I+1),Trim_Front)

            Find.Comma:

                I = INSTR(1,Out,",")
                IF I > 0
                THEN
                    GOSUB Add.Item
                    GOTO Find.Comma
                END IF

                DIMs.MAPs.Count = DIMs.MAPs.Count + 1
                DIMs.MAP(DIMs.MAPs.Count) = SPACE$(15)+Out
            ELSE
                DIMs.MAPs.Count = DIMs.MAPs.Count + 1
                DIMs.MAP(DIMs.MAPs.Count) = SPACE$(7)+Out
            END IF

            Out = LEFT(S1,7)+SPACE$(Comment.Column-7)+"! ** MOVED COMMON, DIM, or MAP ** "

RETURN


Add.Item:
            DIMs.MAPs.Count = DIMs.MAPs.Count + 1

            DIMs.MAP(DIMs.MAPs.Count) = SPACE$(15) +                    &
                                        EDIT$(LEFT(Out,I),Trim_Front)+  &
                                        SPACE$(50-I)+" &"

            Out = EDIT$(RIGHT(Out,I+1),Trim_Front)

RETURN

      %PAGE
```

```
        %SBTTL  "Exception Handling and Program Termination Code"

    Hiccup:     Endfile = TRUE
                PRINT                                   IF F.Out <> "TT:"
                PRINT ERT$(ERR);" Error ";ERR           IF ERR <> 11
    RESUME  9

    Oops:       PRINT "Sorry, unable to open that file, program ends"
                Bad.File = TRUE
    RESUME  1

9       !

    Done:
        IF IF.Count > 0                         ! Put any pending END IF's
        THEN
            PRINT #2, SPACE$(7+4*(IF.Count-I));"END IF" FOR I=1 TO IF.Count
            Lines.Out = Lines.Out + IF.Count
        END IF

        IF DIMs.MAPs.Count <> 0                 ! Also, dump the DIM/MAP's
        THEN
            PRINT #2, NUM1$(MAP.DIM.Line.Number);TAB(Comment.Column); "! ** COMMON, DIM, and MAP's have been moved here ** "
            PRINT #2, DIMs.MAP(I) FOR I = 1 TO DIMs.MAPs.Count
            Lines.Out = Lines.Out + DIMs.MAPs.Count + 1
        END IF

        %PAGE
```

```
    %SBTTL  "Generated DEF's (for MicroBASIC Operations)"

%IF %Size = %Large
%THEN

    All_Flags(10) = All_Flags(10) + All_Flags(I)    FOR I = 1 TO 9

    IF All_Flags(10) <> 0
    THEN
        PRINT
        PRINT #2, "4";TAB(Comment.Column);"! ** Added functions here **"
        PRINT     "Added Line 4 ("; IF F.Out <> "TT:"

        IF CVI_Flag   <> 0%
        THEN
            PRINT " CVI";   IF F.Out <> "TT:"
            PRINT #2,  "    DEF WORD CVI(STRING CVI_IN)"
            PRINT #2,  "             MAP (CVIMAP) STRING CVI_STRING = 2"
            PRINT #2,  "             MAP (CVIMAP) WORD   CVI_WORD"
            PRINT #2,  ""
            PRINT #2,  "             CVI_STRING = CVI_IN"
            PRINT #2,  "             CVI        = CVI_WORD"
            PRINT #2,  "    END DEF"
            PRINT #2,  ""
            Lines.Out = Lines.Out + 8
        END IF

        IF CVS_Flag   <> 0
        THEN
            PRINT " CVS";   IF F.Out <> "TT:"
            PRINT #2,  "    DEF SINGLE CVS(STRING CVS_IN)"
            PRINT #2,  "             MAP (CVSMAP) STRING CVS_STRING = 4"
            PRINT #2,  "             MAP (CVSMAP) SINGLE CVS_SINGLE"
            PRINT #2,  ""
            PRINT #2,  "             CVS_STRING = CVS_IN"
            PRINT #2,  "             CVS        = CVS_SINGLE"
            PRINT #2,  "    END DEF"
            PRINT #2,  ""
            Lines.Out = Lines.Out + 8
        END IF

        IF CVD_Flag   <> 0
        THEN
            PRINT " CVD";   IF F.Out <> "TT:"
            PRINT #2,  "    DEF SINGLE CVD(STRING CVD_IN)"
            PRINT #2,  "             MAP (CVDMAP) STRING CVD_STRING = 8"
            PRINT #2,  "             MAP (CVDMAP) SINGLE CVD_DOUBLE"
            PRINT #2,  ""
            PRINT #2,  "             CVD_STRING = CVD_IN"
            PRINT #2,  "             CVD        = CVD_DOUBLE"
            PRINT #2,  "    END DEF"
            PRINT #2,  ""
            Lines.Out = Lines.Out + 8
        END IF
        %PAGE
```

```
IF MKI_Flag   <> 0
THEN
    PRINT " MKI";   IF F.Out <> "TT:"
    PRINT #2,  "    DEF STRING MKI(WORD MKI_IN)"
    PRINT #2,  "            MAP (MKIMAP) STRING MKI_STRING = 2"
    PRINT #2,  "            MAP (MKIMAP) WORD   MKI_WORD"
    PRINT #2,  ""
    PRINT #2,  "            MKI_WORD = MKI_IN"
    PRINT #2,  "            MKI      = MKI_STRING"
    PRINT #2,  "    END DEF"
    PRINT #2,  ""
    Lines.Out = Lines.Out + 8
END IF

IF MKS_Flag   <> 0
THEN
    PRINT " MKS";   IF F.Out <> "TT:"
    PRINT #2,  "    DEF STRING MKS(SINGLE MKS_IN)"
    PRINT #2,  "            MAP (MKSMAP) STRING MKS_STRING = 4"
    PRINT #2,  "            MAP (MKSMAP) SINGLE MKS_SINGLE"
    PRINT #2,  ""
    PRINT #2,  "            MKS_SINGLE = MKS_IN"
    PRINT #2,  "            MKS        = MKS_STRING"
    PRINT #2,  "    END DEF"
    PRINT #2,  ""
    Lines.Out = Lines.Out + 8
END IF

IF MKD_Flag   <> 0
THEN
    PRINT " MKD";   IF F.Out <> "TT:"
    PRINT #2,  "    DEF STRING MKD(DOUBLE MKD_IN)"
    PRINT #2,  "            MAP (MKDMAP) STRING MKD_STRING = 8"
    PRINT #2,  "            MAP (MKDMAP) SINGLE MKD_DOUBLE"
    PRINT #2,  ""
    PRINT #2,  "            MKD_DOUBLE = MKD_IN"
    PRINT #2,  "            MKD        = MKD_STRING"
    PRINT #2,  "    END DEF"
    PRINT #2,  ""
    Lines.Out = Lines.Out + 8
END IF

IF At_Flag <> 0
    THEN
    PRINT     " AT";        IF F.Out <> "TT:"
    PRINT #2, "      DEF STRING  At( WORD At.Line, At.Column) = ESC + '[' + NUM1$(At.Line)+';'+NUM1$(At.Column)+'f'"
    PRINT #2, "      DEF STRING CLS( WORD CLS.Line, CLS.Column) = AT(CLS.Line,CLS.Column) + Clear.Below"
    PRINT #2, "     "
    Lines.Out = Lines.Out + 3
END IF
%PAGE
```

```
IF INKEY_Flag <> 0
THEN
    PRINT      " INKEY";      IF F.Out <> "TT:"
    PRINT #2, "     %PAGE"
    PRINT #2, "     %SBTTL  'Keyboard Input Routine (VMS Only)' "

    PRINT #2, "     EXTERNAL LONG CONSTANT                                  &"
    PRINT #2, "                       IO$_READVBLK                          &"
    PRINT #2, "                      ,IO$M_NOECHO                           &"
    PRINT #2, "                      ,SS$_NORMAL"
    PRINT #2, ""

    PRINT #2, "     EXTERNAL LONG FUNCTION SYS$ASSIGN (                     &"
    PRINT #2, "                       STRING BY DESC                        &"
    PRINT #2, "                      ,WORD   BY REF                         &"
    PRINT #2, "                      ,LONG   BY VALUE                       &"
    PRINT #2, "                      ,LONG   BY VALUE  )"
    PRINT #2, ""

    PRINT #2, "     EXTERNAL LONG FUNCTION SYS$QIOW (                       &"
    PRINT #2, "                       LONG   BY VALUE                       &"
    PRINT #2, "                      ,WORD   BY VALUE                       &"
    PRINT #2, "                      ,LONG   BY VALUE                       &"
    PRINT #2, "                      ,LONG   BY REF                         &"
    PRINT #2, "                      ,LONG   BY REF                         &"
    PRINT #2, "                      ,LONG   BY VALUE                       &"
    PRINT #2, "                      ,WORD   BY REF                         &"
    PRINT #2, "                      ,LONG   BY VALUE                       &"
    PRINT #2, "                      ,LONG   BY VALUE                       &"
    PRINT #2, "                      ,LONG   BY REF                         &"
    PRINT #2, "                      ,LONG   BY REF                         &"
    PRINT #2, "                      ,LONG   BY VALUE  )"
    PRINT #2, ""

    PRINT #2, "     DECLARE STRING CONSTANT                                 &"
    PRINT #2, "                       Clear.Screen   = ESC + '[2J'          &"
    PRINT #2, "                      ,Clear.Below    = ESC + '[0J'          &"
    PRINT #2, "                      ,Clear.Right    = ESC + '[K'           &"
    PRINT #2, "                      ,Clear.Line     = ESC + '[2K'          &"
    PRINT #2, "                      ,Col.132        = ESC + '[?3h'         &"
    PRINT #2, "                      ,Black.Screen   = ESC + '[?51'         &"
    PRINT #2, "                      ,Normal.Chars   = ESC + '[0m'          &"
    PRINT #2, "                      ,Bold.Chars     = ESC + '[1m'          &"
    PRINT #2, "                      ,Reverse.Chars  = ESC + '[7m'          &"
    PRINT #2, "                      ,Underscore.On  = ESC + '[4m'          &"
    PRINT #2, "                      ,Blink          = ESC + '[5m'          &"
    PRINT #2, "                      ,Scroll.Down    = ESC + 'M'"
    PRINT #2, ""
    %PAGE
```

```
PRINT #2, "     DECLARE  INTEGER CONSTANT                                       &"
PRINT #2, "                      True          =  -1                            &"
PRINT #2, "                     ,False         =   0                            &"
PRINT #2, "                     ,Up            =   1                            &"
PRINT #2, "                     ,Down          =   0                            &"
PRINT #2, ""

PRINT #2, "     DECLARE STRING                                                  &"
PRINT #2, "                      Hold.Value                                     &"
PRINT #2, "                     ,Key.Value                                      &"
PRINT #2, "                     ,Key.Pad                                        &"
PRINT #2, "                     ,Text"
PRINT #2, ""

PRINT #2, "     DECLARE WORD                                                    &"
PRINT #2, "                      Chan                                           &"
PRINT #2, "                     ,QIO.Char                                       &"
PRINT #2, "                     ,Direction                                      &"
PRINT #2, ""

PRINT #2, "     DECLARE LONG                                                    &"
PRINT #2, "                      S.Status                                       &"
PRINT #2, ""

PRINT #2, "     %PAGE"
PRINT #2, "     %SBTTL  'Keypad (One Character) Input Function'"
PRINT #2, "     DEF STRING InKey"
PRINT #2, "         DECLARE BYTE  Arrow.Key"
PRINT #2, "         S.Status = SYS$ASSIGN('TT',Chan,,)        ! <ready for QIOs>"
PRINT #2, "         PRINT 'ERROR ON SYS$ASSIGN '  IF S.Status <> SS$_NORMAL"
PRINT #2, "         Arrow.Key = False"
PRINT #2, "         Key.Pad   = CHR$(0)"
PRINT #2, ""
PRINT #2, "      C1: WHILE TRUE"
PRINT #2, "             S.Status = SYS$QIOW(                                    &"
PRINT #2, "                       ,Chan                                         &"
PRINT #2, "                       ,(IO$_READVBLK OR IO$M_NOECHO)                &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                       ,QIO.Char                                     &"
PRINT #2, "                       ,1%                                           &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                       ,                                             &"
PRINT #2, "                                )"
PRINT #2, "             PRINT 'ERROR ON SYS$QIOW' IF S.Status <> SS$_NORMAL"
PRINT #2, "             Arrow.Key = True          IF QIO.Char = 27 AND Arrow.Key = False"
PRINT #2, "             EXIT C1                   IF NOT Arrow.Key"
PRINT #2, "             ITERATE C1                IF QIO.Char = ASCII('[') OR QIO.Char = ASCII('O')"
PRINT #2, ""
PRINT #2, "             IF QIO.Char <> ASCII('[') AND QIO.Char <> 27"
PRINT #2, "             THEN"
PRINT #2, "                     SELECT QIO.Char"
PRINT #2, "                         CASE 65"
PRINT #2, "                             QIO.Char = ASCII('(')   !  ESC [ A  is up-arrow"
PRINT #2, "                         CASE 66"
PRINT #2, "                             QIO.Char = ASCII('v')   !  ESC [ B  is down-arrow"
PRINT #2, "                         CASE 67"
```

```
            PRINT #2, "                             QIO.Char = ASCII('>')   ! ESC [ C  is right-arrow"
            PRINT #2, "                         CASE 68"
            PRINT #2, "                             QIO.Char = ASCII('<')   ! ESC [ D  is left-arrow"
            PRINT #2, "                         CASE ELSE"
            PRINT #2, "                             Key.Pad  = CHR$(QIO.Char)"
            PRINT #2, "                             QIO.Char = ASCII('?')   ! ESC [ other  is probably a keypad key"
            PRINT #2, "                     END SELECT"
            PRINT #2, "                 EXIT C1"
            PRINT #2, "                 END IF"
            PRINT #2, "             NEXT"
            PRINT #2, "             InKey = CHR$(QIO.Char)"
            PRINT #2, "             CALL SYS$DASSGN (Chan BY VALUE)"
            PRINT #2, "         END DEF"
            PRINT #2, "         %PAGE"
            PRINT #2, "         %SBTTL  'Program Text...'"
            Lines.Out = Lines.Out + 113
            END IF
        PRINT " )"  IF F.Out <> "TT:"
    END IF
%END %IF
%PAGE
```

```
%SBTTL "Program Termination and Statistics"

    CLOSE #2
    CLOSE #1

    I = TIME(0%)                                ! Compute elapsed <wall> time

    IF Start.Time < I
    THEN
        I = I - Start.Time                      ! Daytime run
    ELSE
        I = I + (24*60*60 - Start.Time)         ! Midnight run
    END IF

    PRINT
    PRINT   "("; NUM1$(Lines.In); ") Input Lines from " ; F.In ; ", ("; &
                    NUM1$(Lines.Out) ; ") lines written to " ; F.Out

    PRINT   "("; NUM1$(DIMs.MAPs.Count);") lines of DIM and MAP statements moved."

    PRINT   "("+NUM1$(I)+")Seconds elapsed time ... ("+NUM1$(Lines.In/(I/60))+") lines/minute"

Bye:
    END
```

```
400 %SBTTL "Fixup MicroBASIC Funnies"

    FUNCTION STRING uBASIC(STRING Source)

        OPTION TYPE = EXPLICIT

        DECLARE LONG CONSTANT   TRUE = -1
        DECLARE LONG            I, J, K, T1, T2, T3, T4, T5, Action_Code, X
        DECLARE STRING          Keyword, New.word, S1, S2

        !       The following flags are used by the uBASIC FUNCTION
        !       as a type of "OWN" storage, i.e. static storage
        !       that retains values between invocations. They are
        !       initialized to 0 as the main program starts running.

        !       The flags correspond to uBASIC statements or functions
        !       for which DEC BASIC functions or subroutines must
        !       be generated (i.e. no easy 1-1 transform exists).

        !       e.g. PRINT @128,"Hi"  becomes  PRINT AT(128);"Hi"

        !       and a BASIC function named AT has to be inserted
        !       into the front of the converted program.

        MAP (FLAGS)     BYTE     Action.Flag    ! Caller must do something &
                                ,At_Flag        ! Output PRINT AT functn   &
                                ,MKS_Flag       ! Output MKS function      &
                                ,MKI_Flag       ! Output MKI function      &
                                ,MKD_Flag       ! Output MKD function      &
                                ,CVS_Flag       ! Output CVS function      &
                                ,CVI_Flag       ! Output CVI function      &
                                ,CVD_Flag       ! Output CVD function      &
                                ,INKEY_Flag     ! Output INKEY$ function

        MAP (FLAGS)     BYTE    All_Flags(10)   ! This makes initializing easier

        GOTO Colon              IF LEN(Source) = 0      ! White Space

        GOSUB Lin.Num           IF 0 <> INSTR(1,"0123456789",LEFT(Source,1))

        GOTO Set                IF "REM" = LEFT(EDIT$(RIGHT(Source,J),-1),3)
        RESTORE

        %PAGE
```

```
        !       Here, we read from a table of DATA statements, looking for transforms
        !       to perform on the line of text sent into this FUNCTION. The table consists
        !       of:  thing-to-find, thing-to-change-to, and a special-action-code. The
        !       BASIC function INSTR does the lookup. The action code simply tells us
        !       to set one of the flags checked by the calling main program as it
        !       determines whether to emit DEF's at the end of the program being translated.

Look: WHILE TRUE
          READ Keyword, New.Word, Action_Code
      EXIT Look         IF Keyword = "..."
          I = 1
        Again:
          WHILE TRUE
              I =  INSTR(I,Source,Keyword)              !       The lookup

      ITERATE Look              IF I = 0                !       <unsuccessful>
              J = INSTR(1,Source,'"')                   !       <success - possibly..>
              K = INSTR(J+1,Source,'"')                 !            its no success if the find is inside a literal
          EXIT Again            IF J<I AND K>I          !            in that case, ignore the find. EXIT off end-of-line

              SELECT Action_Code                        !       <find>
                CASE 0                                  !               No Special Action Needed
                CASE 1,2,3,4,5,6,7,8                    !               Need action: set a flag
                      All_Flags(Action_Code)  = 1
                      Action.Flag             = 1       !               and the flag that says "some flags are set"
                CASE ELSE                               !
              END SELECT

              Source = LEFT(Source,I-1) + New.Word+ RIGHT(Source,I+Len(Keyword))      ! <insert the change-to text>
              I      = I + LEN(New.Word)                                              ! ... and scan for next word
              NEXT
          NEXT
Colon:                                  !  Lets turn colon-within-literal into a dash
          I = INSTR(1,Source,':"')      !       <inelegant trick>
          Source = MID(Source,1,I-1)+"-"+MID(Source,I+1,LEN(Source)-I) IF I <> 0
          GOTO Colon                    IF I <> 0

          I = INSTR(1,Source,': "')     !  ditto, for colon-space  into a dash
          Source = MID(Source,1,I-1)+"-"+MID(Source,I+1,LEN(Source)-I) IF I <> 0
          GOTO Colon                    IF I <> 0

          I = INSTR(1,Source,":")       !  If we see just a colon, that turns into a backslash

          Source = LEFT(Source,I-1)+"\"+RIGHT(Source,I+1) IF I <> 0
          GOTO Colon                                      IF I <> 0

          T1 = INSTR(1,Source," IF ")
          T2 = INSTR(1,Source," THEN ")         IF T1 <> 0

          GOSUB Fix.IF                          IF (T1 <> 0) AND (T2 = 0)       ! special handling for IF and OPEN
          GOSUB Fix.OPEN                        IF 0  <> INSTR(1,Source,"OPEN")
          GOSUB Print.At                        IF 0  <> INSTR(1,Source,"@")    ! and for PRINT @ to PRINT AT
      Set:
          Source = EDIT$(Source,8+16+32+128+256)                                ! discard any junk characters
          uBASIC = Source                                                       ! and exit: this line is done.
EXIT FUNCTION
%PAGE
```

```
      %SBTTL "Fixup MicroBASIC Syntax for DEC Systems"
      Lin.Num: J = 0                          ! inserts white space to the right of BASIC line numbers
      L.Scan:
          FOR I=2 TO LEN(Source)
              IF 0=INSTR(1,"0123456789",MID(Source,I,1))
              THEN
                  J = I
                  EXIT L.Scan
              END IF
          NEXT I
      Source = Left(Source,J-1)+" "+RIGHT(Source,J)   IF J <> 0
RETURN

Fix.IF:
      J  = T1                             !  some variants of IF omit the keyword "THEN"
      T1 = INSTR(J,Source,"GO")
      T2 = INSTR(J,Source,"PRINT")
      T3 = INSTR(J,Source,"INPUT")
      T4 = INSTR(J,Source,"LET")

      T5 = 999

      T5 = T1                             IF T1 < T5 AND T1 <> 0
      T5 = T2                             IF T2 < T5 AND T2 <> 0
      T5 = T3                             IF T3 < T5 AND T3 <> 0
      T5 = T4                             IF T4 < T5 AND T4 <> 0

      Source = LEFT(Source,T5-1)+" THEN "+RIGHT(Source,T5) IF T5 <> 999       ! insert a THEN if its needed
RETURN

Fix.OPEN:                                 ! the MicroBASIC OPEN has the right kind of components,
      T1 = INSTR(1,Source,'"O"')          !  OPEN FOR OUTPUT   and
      T2 = INSTR(1,Source,'"I"')          !  OPEN FOR INPUT    ... its just the spelling and the ordering...
RETURN     IF T1+T2 = 0

      S2 = "FOR INPUT"
      S2 = "FOR OUTPUT"                   IF T1 <> 0
      T1 = T2                             IF T2 <> 0 AND T1 = 0

      T3 = INSTR(T1,Source,",")           !  pickup the channel number
      S1 = " AS FILE #" + MID(Source,T3+1,1)

      T3 = INSTR(T3+1,Source,",")

      T2 = INSTR(T3,Source,"\")           ! and the file name
      T4 = INSTR(T3,Source,"ELSE")
      T2 = T4                             IF T4<T2 AND T4<>0
      T2 = LEN(Source)+1                  IF T2 =  0

      S1 = MID(Source, T3+1, T2-T3-1) + " " + S2 + S1

      Source = LEFT(Source,T1-1)+S1+RIGHT(Source,T2) ! and output a DEC-style OPEN statement
GOTO  FIX.OPEN
%PAGE
```

```
Print.At:              ! careful here, there are several variations of PRINT-at

      I = INSTR(1,Source,"@")                          !  Typical use ...
      J = INSTR(I,Source,",")                          !  PRINT @255+i3,"Hello"
      J = INSTR(1,Source,";")          IF J = 0        !  PRINT @255+i3;"Hello"
                                                       !        I      J
RETURN                                 IF J=0

      Source = LEFT(Source,I-1) + "AT(" + MID(Source,I+1,J-I-1) + ");"+RIGHT(Source,J+1)

      At_Flag          =  TRUE
      Action.Flag      =  TRUE
RETURN

%PAGE
```

```
        %SBTTL  "Conversion DATA Tables"

        !       GENERAL TABLE FORMAT IS SIMPLY...
        !
        !       <find this>     , <change to this>                              , Additional Action Code
        !
        !
1000    DATA    "AND"           ,  " AND "                                      , 0
1050    DATA    "OR"            ,  " OR "                                       , 0
1100    DATA    "'"             ,  " ! "                                        , 0
1150    DATA    "&"             ,  " PRINT"                                     , 0
1160    !
1200    DATA    "/BAS"          ,  ".BAS"                                       , 0
1250    DATA    "/DAT"          ,  ".DAT"                                       , 0
1300    !
2000    !
2200    DATA    "CLEAR"         ,  " ! *CLEAR* "                                , 0
2250    DATA    "CLOSE"         ,  " CLOSE "                                    , 0
2295    DATA    "CLR$"          ,  "C.LR$"                                      , 0
2300    DATA    "CLR"           ,  " PRINT FOR CLR. = 1 TO 24 ! *CLR* "         , 0
2350    DATA    "CLS"           ,  " PRINT FOR CLS. = 1 TO 24 ! *CLS* "         , 0
2360    DATA    "CMD"           ,  " PRINT '*CMD*' ! "                          , 0
2370    DATA    "CVI"           ,  "CVI"                                        , 6
2380    DATA    "CVS"           ,  "CVS"                                        , 5
2390    DATA    "CVD"           ,  "CVD"                                        , 7
2440    !
2450    DATA    "DATA"          ,  " DATA "                                     , 0
2500    DATA    "DEFDBL"        ,  " ! DECLARE DOUBLE ( *DEFDBL* ) "            , 0
2550    DATA    "DEFN"          ,  " ! (*DEFFN*)  "                             , 0
2600    DATA    "DEFINT"        ,  " ! DECLARE WORD   ( *DEFINT* ) "            , 0
2650    DATA    "DEFSNG"        ,  " ! DECLARE SINGLE ( *DEFSNG* ) "            , 0
2700    DATA    "DEFUSR"        ,  " ! (*DEFUSR*) "                             , 0
2750    DATA    "DEFSTR"        ,  " ! DECLARE STRING ( *DEFSTR* ) "            , 0
2800    DATA    "DELETE"        ,  " DELETE "                                   , 0
2850    DATA    "DIM"           ,  " DIM "                                      , 0
2855    !
2900    DATA    "ELSE"          ,  " ELSE "                                     , 0
2925    DATA    "WEND"          ,  " NEXT "                                     , 0
2950    DATA    "END"           ,  " END "                                      , 0
3000    DATA    "ERROR"         ,  " ERROR "                                    , 0
3005    !
3050    DATA    "FIELD"         ,  " FIELD ! *FIELD* ! "                        , 0
3100    DATA    "FOR"           ,  " FOR "                                      , 0
3105    !
3110    DATA    " F OR "        ,  " FOR "                                      , 0
3150    DATA    "GET"           ,  " GET "                                      , 0
3200    DATA    "GOSUB"         ,  " GOSUB "                                    , 0
3250    DATA    "GOTO"          ,  " GOTO "                                     , 0
3255    !
3300    DATA    "IF"            ,  " IF "                                       , 0
3350    DATA    "INKEY$"        ,  " INKEY"                                     , 8
3360    DATA    "INPUT"         ,  " INPUT "                                    , 0
3375    !
3400    DATA    "KILL"          ,  " KILL "                                     , 0
3405    !
3425    DATA    "LET"           ,  " LET "                                      , 0
3450    DATA    "LSET"          ,  " LSET "                                     , 0
3500    DATA    "RSET"          ,  " RSET "                                     , 0
3550    DATA   "LINE INPUT"     ,  " LINPUT "                                   , 0
3555    !
```

```
3600    DATA    "MAX"           ,  "MAXI"                          , 0
3605    DATA    "MKI$"          ,  "MKI"                           , 3
3610    DATA    "MKS$"          ,  "MKS"                           , 2
3615    DATA    "MKD$"          ,  "MKD"                           , 4
3645    !
3650    DATA    "NAME"          ,  " NAME "                        , 0
3700    DATA    "NEXT"          ,  " NEXT "                        , 0
3705    !
3800    DATA    "F OR"          ,  " FOR"                          , 0
3850    DATA    "OPEN"          ,  " OPEN "                        , 0
3860    DATA    "OPTION"        ,  " ! OPTION "                    , 0
3900    DATA    "OUT"           ,  " ! *OUT* "                     , 0
3905    !
3950    DATA    "PEEK"          ,  "  ! *PEEK* ! '?' * "          , 0
4000    DATA    "POKE"          ,  "? ! *POKE* "                   , 0
4050    DATA    "PRINT"         ,  "PRINT "                        , 0
4060    DATA    "WIDTH LPRINT"  , "MARGIN #9, "                    , 0
4100    DATA    "LPRINT"        ,  " PRINT #9,"                    , 0
4150    DATA    "PUT"           ,  "PUT "                          , 0
4155    !
4200    DATA    "RANDOM"        ,  " RANDOM "                      , 0
4250    DATA    "RANDOM IZE"    , " RANDOMIZE "                    , 0
4300    DATA    "READ"          ,  " READ "                        , 0
4350    DATA    "REM"           ,  "\ REM "                        , 0
4400    DATA    "RESET"         ,  " RESET "                       , 0
4450    DATA    "RESTORE"       ,  " RESTORE "                     , 0
4500    DATA    "RESUME"        ,  " RESUME "                      , 0
4550    DATA    "RETURN"        ,  " RETURN "                      , 0
4600    DATA    "RND("          ,  "RND*("                         , 0
4605    !
4650    DATA    "SET"           ,  " SET "                         , 0
4660    DATA    "SP"            ,  "SP."                           , 0
4700    DATA    "STEP"          ,  " STEP "                        , 0
4750    DATA    "STOP"          ,  " STOP "                        , 0
4755    !
4800    DATA    "THEN"          ,  " THEN "                        , 0
4850    DATA    "TO"            ,  " TO "                          , 0
4900    DATA    "GO TO"         ,  "GOTO"                          , 0
4905    !
4910    DATA    " S TO P "      ,  " STOP "                        , 0
4950    DATA    "USING"         ,  " USING "                       , 0
5000    DATA    "#9, USING"     ,  "#9 USING"                      , 0
5001    DATA    "#9,  USING"    ,  "#9 USING"                      , 0
5002    DATA    "#9,   USING"   ,  "#9 USING"                      , 0
5005    !
5050    DATA    "WHILE"         ,  " WHILE "                       , 0
5850    !
5900    DATA    "..."           ,  " ..."                          , 0
9000    !
9999    END FUNCTION
```

```
1       %TITLE "BASIC-PLUS System Dependency Flagger"
        %SBTTL "Declarations"
        %IDENT "BPFLAG"

        !               A BASIC-PLUS  System-Dependency Filter
        !               (Works for Basic+2/RSTS Source, too)
        !
        !          Author:      Tom Harris   August 1, 1983
        !                       Digital Equipment Corp (ZK2-3/K06)
        !                       110 Spit Brook Road
        !                       Nashua, NH  USA     03062
        !       ------------------------------------------------------
        !          Input:    either a filename, or an indirect @filename
        !
        !                    (indirect presumes one filename per line in
        !                     the indirect command file)
        !       ------------------------------------------------------
        !           Output:  Summary information to terminal:
        !                      -  file-by-file use of RSTS'isms
        !                      -  summary report
        !
        !                     Report file - lists *by file* each line
        !                           which has RSTS dependencies, along
        !                           with the (EDT) source line number.
        !                           Also gets the summary information.
        !       ------------------------------------------------------
        !           Support:  Here it is, have fun, suggestions welcome,
        !                     but no guarantees. <user-supported>
        !       ------------------------------------------------------
        !          THIS IS A HANDOUT  FOR FALL EUROPEAN DECUS,  1983
        !       ------------------------------------------------------

        DECLARE  INTEGER CONSTANT                                       &
                          True          =  -1                           &
                         ,False         =   0

        DECLARE  BYTE                                                   &
                          End.of.File                                   &
                         ,End.of.Job                                    &
                         ,Indirect

        DECLARE  STRING                                                 &
                          Answer                                        &
                         ,Text                                          &
                         ,Titles(50)

        DECLARE  LONG                                                   &
                          Category(50)          !   For summary report  &
                         ,Summary(50)

        PRINT  "BASIC-PLUS Dependency Filter " + TIME$(0%)
        PRINT

        %PAGE
```

```
%SBTTL "Initialization"

        Titles(1)       =       "NOEXTEND"
        Titles(2)       =       "PEEK"
        Titles(3)       =       "POKE"
        Titles(4)       =       "OPEN <RSTS dependency>"
        Titles(5)       =       "KILL <RSTS dependency>"
        Titles(6)       =       "NAME .. AS  <RSTS dependency>"
        Titles(7)       =       "CCONT"
        Titles(8)       =       "UNLOCK"
        Titles(9)       =       "MAGTAPE function"
        Titles(10)      =       "TIME <non-zero argument>"
        Titles(11)      =       "CHAIN .. LINE"
        Titles(12)      =       "STATUS function"
        Titles(13)      =       "SPEC% function"
        Titles(14)      =       "SYS function"


LINPUT "What Input File <TT:>",Input.File$

        Input.File$ = "TT:"     IF EDIT$(Input.File$,-1) = ""

        Indirect = False

        IF "@" = LEFT(Input.File$,1)
            THEN
                Control.File$ = MID(Input.File$,2,LEN(Input.File$)-1)
                Control.File$ = Control.File$ + ".COM"   IF 0 = INSTR(1,Control.File$,".")
                OPEN Control.File$ FOR INPUT AS FILE #3, VARIABLE, RECORDSIZE 132
                Indirect = True
        END IF

LINPUT "What Report File <TT:>",Output.File$

        Output.File$ = "TT:"    IF EDIT$(Output.File$,-1) = ""
        OPEN Output.File$ FOR OUTPUT AS FILE #2, VARIABLE, RECORDSIZE 132

End.of.Job = False
Prior.Dependencies = 0

%PAGE
```

```
%SBTTL "Main Loop, includes indirect file processing"

Main:
    WHILE True

    EXIT Main   IF End.of.Job

        ON ERROR GOTO Done
        LINPUT #3, Input.File$  IF Indirect

        ON ERROR GOTO Cant.Find
        OPEN Input.File$ FOR INPUT AS FILE #1, VARIABLE, RECORDSIZE 132

        PRINT #2,FF;                              IF Prior.Dependencies <> 0
        PRINT #2, "--------------------------- "+Input.File$+" -------------------------------"

        GOSUB Process.A.File
        GOSUB Report.Findings

        Input.Files = Input.Files + 1
        Total.Lines = Total.Lines + Input.Lines
        Total.Dependencies = Total.Dependencies + Dependencies
        Summary(I) = Summary(I) + Category(I)   FOR I = 1 TO 50
        Prior.Dependencies = Dependencies

    EXIT Main   IF NOT Indirect
    NEXT
    GOTO Finished

Report.Findings:

    PRINT
    PRINT Input.File$ + ":  " + NUM1$(Input.Lines)+" lines processed";

    IF Dependencies > 0
    THEN
        PRINT ",    " + NUM1$(Dependencies)+" written to "+Output.File$
        PRINT
    ELSE
        PRINT
    END IF
RETURN          IF Output.File$ = "TT:"

    PRINT #2
    PRINT #2, Input.File$ + ":  " + NUM1$(Input.Lines)+" lines processed";

    IF Dependencies > 0
    THEN
        PRINT #2, ",    " + NUM1$(Dependencies)+" written to "+Output.File$
        PRINT #2
        PRINT #2, ,Category(I),Titles(I)  IF Category(I)<> 0  FOR I=1 TO 50
    END IF

    PRINT #2    IF Dependencies = 0
RETURN
%PAGE
```

```
    %SBTTL "File Processing"

Process.A.File:

    End.of.File = False
    ON ERROR GOTO End.File
    Dependencies = 0
    Input.lines = 0
    Category(I) = 0                 FOR I = 1 TO 50

    Scan:
        WHILE True
            CLOSE #1                IF End.of.File
            EXIT Scan               IF End.of.File

            LINPUT #1, Answer
            Text = EDIT$(Answer,32)
            Input.Lines = Input.Lines + 1

            C = INSTR(1,Answer,"!")                         <omit comments>
            Text = MID(Text,1,C-1)                          IF C <> 0
            C = 1

            Drop.Literals:

                WHILE True
                    Q1 = INSTR(C,Text,'"')          !       Leading quote
                    Q2 = INSTR(Q1+1,Text,'"')       !       Trailing quote
                    Q2 = LEN(Text)                  IF Q2 = 0
                EXIT Drop.Literals                  IF Q1 = 0
                    Text = MID(Text,1,Q1) + MID(Text,Q2,LEN(Text)-Q2)
                    C = Q2+1
                NEXT

            GOSUB  Got.NOEXTEND     IF (0 <> INSTR(1,Text,"NOEXTEND"))      AND (Input.Lines < 10)
            GOSUB  Got.PEEK         IF  0 <> INSTR(1,Text,"PEEK")
            GOSUB  Got.POKE         IF  0 <> INSTR(1,Text,"POKE")
            GOSUB  Got.OPEN         IF (0 <> INSTR(1,Text,"OPEN"))          AND 0 <> (INSTR(1,Text,"AS"))
            GOSUB  Got.NAME         IF (0 <> INSTR(1,Text,"NAME"))          AND 0 <> (INSTR(1,Text,"AS"))
            GOSUB  Got.KILL         IF  0 <> INSTR(1,Text,"KILL")
            GOSUB  Got.SYS          IF  0 <> INSTR(1,Text,"SYS(")
            GOSUB  Got.CCONT        IF  0 <> INSTR(1,Text,"CCONT")
            GOSUB  Got.SPEC         IF  0 <> INSTR(1,Text,"SPEC%")
            GOSUB  Got.STATUS       IF  0 <> INSTR(1,Text,"STATUS")
            GOSUB  Got.UNLOCK       IF  0 <> INSTR(1,Text,"UNLOCK")
            GOSUB  Got.MAGTAPE      IF  0 <> INSTR(1,Text,"MAGTAPE")
            GOSUB  Got.TIME         IF  0 <> INSTR(1,Text,"TIME(")
            GOSUB  Got.CHAIN        IF (0 <> INSTR(1,Text,"CHAIN"))         AND (0 <> INSTR(1,Text,"LINE"))
        NEXT
    RETURN

    End.File:
        End.of.File = True
    RESUME
    %PAGE
```

```
%SBTTL "Dependency Analysis Subroutines"

Got.NOEXTEND:
        Category(1) = Category(1) + 1
        GOSUB  Report.Dependency
    RETURN

Got.PEEK:
        GOSUB  Report.Dependency
        Category(2) = Category(2) + 1
    RETURN

Got.POKE:
        GOSUB  Report.Dependency
        Category(3) = Category(3) + 1
    RETURN

Got.OPEN:
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,":")       ! Device Spec
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,">")       ! Protection Code
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,"MODE")    ! RSTS/E MODE's

        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,"[")       ! RSTS/E PPN
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,"]")       ! RSTS/E PPN
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,"'(")      ! RSTS/E PPN
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,")'")      ! RSTS/E PPN
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,'"(')      ! RSTS/E PPN
        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,')"')      ! RSTS/E PPN

        GOTO Report.OPEN        IF 0 <> INSTR(1,Text,"CLUSTERSIZE")
    RETURN

        Report.OPEN:
            Category(4) = Category(4) + 1
            GOSUB  Report.Dependency
    RETURN

Got.KILL:
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,":")       ! Device Spec
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,">")       ! Protection Code

        GOTO Report.KILL        IF 0 <> INSTR(1,Text,"[")       ! RSTS/E PPN
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,"]")       ! RSTS/E PPN
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,"'(")      ! RSTS/E PPN
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,")'")      ! RSTS/E PPN
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,'"(')      ! RSTS/E PPN
        GOTO Report.KILL        IF 0 <> INSTR(1,Text,')"')      ! RSTS/E PPN
    RETURN

        Report.KILL:
            Category(5) = Category(5) + 1
            GOSUB Report.Dependency
    RETURN
    %PAGE
```

```
Got.NAME:
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,":")       ! Device Spec
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,">")       ! Protection Code

        GOTO Report.NAME        IF 0 <> INSTR(1,Text,"[")       ! RSTS/E PPN
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,"]")       ! RSTS/E PPN
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,"'(")      ! RSTS/E PPN
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,")'")      ! RSTS/E PPN
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,'"(')      ! RSTS/E PPN
        GOTO Report.NAME        IF 0 <> INSTR(1,Text,')"')      ! RSTS/E PPN
    RETURN

        Report.NAME:
            Category(6) = Category(6) + 1
            GOSUB  Report.Dependency
    RETURN


Got.CCONT:
        Category(7) = Category(7) + 1
        GOSUB  Report.Dependency
    RETURN


Got.UNLOCK:
        Category(8) = Category(8) + 1
        GOSUB  Report.Dependency
    RETURN

Got.MAGTAPE:
        Category(9) = Category(9) + 1
        GOSUB  Report.Dependency
    RETURN


Got.TIME:
        GOTO Report.TIME        IF 0 <> INSTR(1,Text,"(1")      ! RSTS/E TIME
        GOTO Report.TIME        IF 0 <> INSTR(1,Text,"(2")      ! RSTS/E TIME
        GOTO Report.TIME        IF 0 <> INSTR(1,Text,"(3")      ! RSTS/E TIME
        GOTO Report.TIME        IF 0 <> INSTR(1,Text,"(4")      ! RSTS/E TIME
        GOTO Report.TIME        IF 0 =  INSTR(1,Text,"TIME(0")  ! RSTS/E TIME
    RETURN

        Report.TIME:
            Category(10) = Category(10) + 1
            GOSUB  Report.Dependency
    RETURN

   %PAGE
```

```
Got.CHAIN:
        Category(11) = Category(11) + 1
        GOSUB  Report.Dependency
    RETURN

Got.STATUS:
        Category(12) = Category(12) + 1
        GOSUB  Report.Dependency
    RETURN

Got.SPEC:
        Category(13) = Category(13) + 1
        GOSUB  Report.Dependency
    RETURN

Got.SYS:
        Category(14) = Category(14) + 1
        GOSUB  Report.Dependency
    RETURN

Report.Dependency:

        Dependencies = Dependencies + 1
        PRINT #2, FORMAT$(Input.Lines,"##### ") + LEFT(Answer,125)
    RETURN

    %PAGE
```

```
        %SBTTL "Program Termination"

    Done:

        End.of.Job = True
        RESUME

    Cant.Find:

        PRINT
        PRINT "*ERROR - Cannot OPEN file:  "+Input.File$
        PRINT "         "+ERT$(ERR)
        PRINT "*Program Now Terminates"
        RESUME 99
               !

99  Finished:   !

        Control.File$ = Input.File$             IF NOT Indirect

        PRINT
        PRINT  "--------------------------- Summary --------------------------------"
        PRINT
        PRINT  Control.File$ + "  " + NUM1$(Total.Lines)+" lines processed";
        PRINT  " from " + NUM1$(Input.Files) + " file(s)"
        PRINT  NUM1$(Total.Dependencies)+" dependencies written to "+Output.File$
        PRINT
        PRINT  ,,Summary(I),Titles(I)  IF Summary(I)<> 0  FOR I=1 TO 50
        PRINT

        IF Output.File$ <> "TT:"
        THEN
            PRINT #2
            PRINT #2, "--------------------------- Summary --------------------------------"
            PRINT #2
            PRINT #2, Control.File$ + "  " + NUM1$(Total.Lines)+" lines processed";
            PRINT #2, " from " + NUM1$(Input.Files) + " file(s)"
            PRINT #2, NUM1$(Total.Dependencies)+" dependencies written to "+Output.File$
            PRINT #2
            PRINT #2, ,,Summary(I),Titles(I)  IF Summary(I)<> 0  FOR I=1 TO 50
            PRINT
        END IF

        CLOSE #3
        CLOSE #2

        END
```

THREADED CODE PRODUCTION BY THE BASIC-PLUS-2 COMPILER

1.0 INTRODUCTION

This article introduces the reader to the concept of threaded code, and uses BASIC-PLUS-2 as an example of a compiler that produces threaded code. For those familiar with threads, and compilers that generate threaded code the first part of this article could be skipped, but the second part should still be of some interest.

2.0 THREADS AND THREADED CODE

This part of the article will discuss what threads and threaded code are, how threads work, what their advantages and disadvantages are, and why the BASIC-PLUS-2 compiler for the PDP-11 produces threaded code. In the strictest sense, a thread is the name of a routine that performs some action when the program is running. This routine may require certain arguments be in certain places, or it may leave a result of some kind in a particular place (on the stack for example). For every thread name there is an associated routine that will perform an action (in BASIC-PLUS-2 some routines can contain several threads, but the same thread can never point to more than one routine). At task build time these names are mapped to the addresses of the routines, and at runtime each of these routines is executed in an order as produced by the compiler, the combination of threads carries out the actions specified by the source program.

The exact mechanics of how all this is carried out at run-time is very language dependant, but a general approach would be to have a pointer to the threads (a pseudo PC). Each routine could then, when it was finished update the pointer, and transfer control to the routine pointed to next. How BASIC-PLUS-2 handles this will be talked about later in this paper.

There are two very important concepts with threaded code. First this type of code contains no (or very little) machine code. For example the code generated by the BASIC-PLUS-2 compiler contains one executable instruction, which is used to initialize the pseudo PC. The second thing to realize is that the routines that are executed at run-time may be executed many times in a given program execution. This is to say, anytime a given operation has to take place, the same thread will be produced by the compiler, and the same routine will be executed at run-time. It is important to realize that each routine exists in only one place in the task file, meaning the exact same code is executed every time a routine is needed.

It should be clear to the reader by now that a threaded code compiler really consists of two parts; the compiler itself, which generates the threaded code; and the collection of run-time routines, or OTS (object time system). The routines in the OTS are generally written in assembly code (MACRO in the case of the PDP-11). It is

interesting to note that the compiler need not know a lot about the OTS routines. All it must know is what the routines take as arguments (and where they are expected to be), and what they produce as results (and where the results are placed). The compiler never has to worry about what is happening at the machine code level. The OTS will make sure registers and other such low-level stuff are used properly. Threads may be viewed as a higher level, machine indepandant language. The OTS represents the low-level machine dependent side of the machine. For furhter discussion on this and other philosophical issues see Chapter 15 entitled "Turning Cousins into Sisters: an example of Software Smoothing of Hardware Differences" in the book "Computer Engineering" by Bell, Mudge, and McNamara. One thing that might be of interest is that it is theoretically possible for all threaded code compilers for a given machine to use the same OTS, more will said on this idea later.

At this point something more must be said on the organization of an OTS. Until now the OTS has been described as a set of routines, such that for every thread produced by an associated compiler, there is a routine corresponding to that thread. While this is true, there is more than just this to a typical OTS. An OTS will normally have two types of routines, one is the thread routine that has been talked about so far, the other is a support type routine that contains code used by many of the thread routines. That is, when a thread is executing, it may have to call a support routine to perform a given function. These functions are often required by many thread routines, and are usually do something at a low level (such as allocate space for a string). These function-type routines have no thread associated with them. They can be considered sub-routines in the OTS that are called only by other thread routines. Like thread routines, support routines expect arguments in a certain place, and return results in a certain place, and it is up to the calling routine to know exactly where these things are. It should also be noted that it is possible for a thread routine to call another routine, which will call another routine, and so forth. This is most common with threads that do complicated operations, for example the ROP$ thread in BASIC-PLUS-2 which does the file open for any type of file. Because the OTS works this way, its implementation can be very tricky. That is to say each routine must be sure that it doesn't kill or change any location that any other routine might require.

Now that the principals of threaded code have been discussed, the advantages and disadvantages of threaded code should be talked about. First the alternative to threaded code should be defined. Many complilers produce in-line code. In-line code consists largely of object code instructions that are actually executed at runtime, although calls to various system routines to perform complicated or system functions might also be present. This type of code is what most people think a compiler produces (and most of them do). A source code statement gets translated into machine code statements that carry out the desired operation.

Threaded code versus in-line code is an example of the old space/time trade-off problem. The main disadvantage of threaded code is that it is much slower than in-line code. The routines must keep updating the 'thread pointer' and jumping to the next thread routine when they are done. With in-line code none of this is necessary; the

code to do the next operation immediately follow the code that did the last task, no unnecessary jumping has to be done. In addition an intellegent in-line code compiler will make more efficient use of the target machine's hardware. Since an in-line code compiler produces actual machine code, it can, for example, allocate registers and other temporary data areas more efficiently than a threaded code compiler. For example arguments between threads are passed on the stack, and not in general registers this will tend to slow down the execution of threaded code relative to in-line code.

On the other hand threaded code will usually take-up less space than the same program compiled into in-line code. With in-line code every time an operation has to be performed, the same code is produced. As a result a program can have a section of code occuring many times within a program. Threaded code, on the other hand, keeps only one copy of the machine code around, and it will be executed everytime the operation is needed. While it is true that some small programs may produce larger tasks with threaded code, large programs that would exceed exceed memory if in-line code were produced, will fit using threaded code generation.

There are other advantages to threaded code, especially when transportability, or the ability for a compiler to work under different hardware or operating systems is necessary. For example on a PDP-11, where there are both different operation systems, and different hardware available (hardware math packages for instance), threaded code has many advantages. When in-line compilers produce object code, this code must be able to run on all of these configurations. There are three ways to possibly do this. First the compiler could produce code that would run on all possible combinations of hardware and operating systems. Even if this were possible it would be very inefficient (math operations would have to assume the minimum math hardware, and those systems with more advanced hardware would not benifit from it). Alternatively the compiler could be smart enough to generate the right code for the system it was running on. While this is possible, the compiler would have to be very large, and would run a lot slower, or at the very least have a very complicated installation procedure. This is not a very practical solution for a small machine like an 11. The last solution would be to have a bunch of different compilers, one for each possible configuration. However with BASIC-PLUS-2 on the PDP-11, there are now 3 major operating systems (RSTS, RSX, and PRO), and 2 different math packages (EIS, and FPU). This would mean 6 different compilers!!

Since no object code is brought into a threaded code program until it is linked by the task builder, only one compiler is necessary. The operating system, and hardware dependant code can be put in the OTS, and the proper routines will be brought in by the task builder. While this does mean there will be 6 different OTS's, as an OTS is just a library of routines, most of these routines will be the same accross all systems. Those routines that are different can be kept track of more easily. The basic idea is to tailor the OTS to the machine, but leave the compiler the same. To carry this point to an extreme, it should be possible to bring threaded code to any machine that has an OTS that will carry out the proper routines.

There are other reasons why a compiler might choose to produce threaded code. If someone was in a hurry to produce a working compiler, he might be able to 'steal' some working OTS routines from another compiler that ran on the same system. While not all routines can be used, things like math routines, that are similar accross many languages, and take a lot of time to write, can usually be used. This could cut down developement time considerably. This is not to say that such a practice has been done before, but it certainly is possible.

3.0 THREADED CODE A LA BASIC-PLUS-2

As the stated before, the PDP-11 BASIC-PLUS-2 compiler generates threaded code. The easiest way to see the threads that the compiler produces is to compile a program with the /MAC option. This will produce a macro output for the source program. If you examine this macro output it will not look much like any other macro source you have seen. For example the simple program:

```
10 PRINT "HELLO"
15 A=A+4
20 C$=C$+"BYE"
25 D%=SQR(C)
30 END
```

If you were to do a COM/MAC the code portion of the output would look like:

```
        .PSECT  $CODE
        JSR     R4,@ $INITM
        .WORD   10$
        .LIMIT
        .WORD   $$BP2
        .WORD   $FLAGR,$FLAGT-$FLAGR
        .WORD   0
        .WORD   $ICIO1
        .WORD   0
10$:    .WORD   20$
        .WORD   $PDATA,$PDATA+0
        .WORD   $IDATA+0,5
        .WORD   $STRNG,2
        .WORD   $TDATA,$ARRAY,0
        .ASCII  /EXAMPL/
20$:

L10:    LIN$    ,10             ;  10
        CLI$S
        IPT$
        RLI$M   ,$PDATA+16      ; "HELLO"
        PVS$AI  ,0              ;  0
        EOL$
```

```
L15:    LIN$     ,15              ;  15
        MOF$MS   ,$IDATA+6        ; A
        ADF$MS   ,$PDATA+0        ; #4
        MOF$SM   ,$IDATA+6        ; A

L20:    LIN$     ,20              ;  20
        RLI$M    ,$STRNG+0        ; C$
        RLI$M    ,$PDATA+4       ; "BYE"
        COS$AA
        MOS$SM   ,$STRNG+0        ; C$

L25:    LIN$     ,25              ;  25
        MOF$MS   ,$IDATA+0        ; C
        SQF$
        CIF$
        MOI$SM   ,$IDATA+4        ; D%

L30:    LIN$     ,30              ;  30
        END$
        .END     $CODE
```

As stated earlier, this is a strange looking macro program. The first line (JSR R4,@$INITM) is the one line of object code produced by the compiler. This causes control to go to the routine $INITM which does the program initializaion. The next several locations are arguments used by $INITM to do the initialization. As mentioned earlier in the article, a pointer is often kept that points to the threads. For BASIC-PLUS-2 this pointer is general register R4. More will be said on this later. The actual threaded code begins a label L10:.

These threads get mapped to their run-time routines by the task builder. The BASIC-PLUS-2 OTS is an object library that has entries who's names correspond to the names of the threads (SQF$, MOI$SM, etc.), and they do all the work.

The remainder of this article will deal with the BASIC-PLUS-2 thread naming conventions, and will explain briefly what each thread does.

The thread name is actually the address of the routine used to perform the required action. The routine is entered via an indirect jump (JMP @(R4)+) from the previous routine and exits via another indirect jump to the next routine. The PDP-11 GENERAL REGISTER 4 (R4) is used as the BASIC+2 program counter. Some arguments for a thread routine may also acquired through R4.

The first three letters of the thread name are arbitrarily distinct opcode names. the fourth letter is always a $. The letters following the $, when present, are descriptive combinations of sources and/or destinations. For example, COS$AS can be read as COncatenate String SOURCE1(A) with string SOURCE2(S) and leave the address of the result string on the stack.

Some threads require a mode. For example addition can be performed on many data types. If a mode is required, the third character of the thread name designates the mode of the thread. Therefore ADI$xx will be a word integer addition. The following table describes each of the modes and their abbreviations.

NAME	ABBREVIATION	DESCRIPTION
Byte	B	A one byte number
Integer	I	A one word Integer number.
Longword	L	A two word integer
Float	F	A two word single precision number .
Double	D	A four word double precision number
String	S	A list of ASCII characters.
RFA	R	An RMS RFA value

For thread that do not require a mode, the third character is usually used to better name the operation of the thread. IPU$, for example, is the initialize PRINT USING thread.

A number of thread require that their operand locations be specified in the thread name. The following tables describe the operand naming conventions.

OPERANDS

ABBR	NAME	DESCRIPTION
S	Stack	The source/destination operand is the stack.

M	Memory	The address of the source/destination is pointed to by R4.
P	Pointer	R4 points to the address of the address of an argument. This mode is used to handle sub-program arguments, and variables that appear in DYNAMIC MAPs
I	Immediate	The source is pointed to by R4. This is used only for word integer threads.
A	Address	The top of the stack contains the address of the destination, or in the case of strings the address of the source string descriptor or a 0 followed by a string descriptor.

It should be noted that many threads have arguments that are not specified by any thread suffix. Most BP2 built-in functions are of this type. The RIGHT$ function will produce a RIT$ thread. The arguments and result of this function are all on the stack. In addtion some threads have suffixes that specify only some of their arguments. For example VRI$M will return the value of a numeric array element. The "M" signifies that the array is a memory mode array, but the indicies into the array are on the stack, and the result is put on the stack, even thought there is no "S" suffix.

THE FOLLOWING IS A LIST OF THREADS PRODUCED BY THE BASIC-PLUS-2 COMPILER (VERSION 2.1), AND A BRIEF EXPLANATION OF EACH.

NOTE

The following list applies only to BASIC-PLUS-2 Version 2.1. The thread names and/or functions may change in any future version of BP2.

```
ABD$    - Absolute Value function, double precision, arg/res. on stack
ABF$    - Absolute Value function, single precision, arg/res. on stack
ABI$    - Absolute Value function, word integer, arg/result on stack
ABL$    - Absolute Value function, long integer, arg/result on stack
ADD$MS  - Double precision addition, memory + stack -> stack
ADD$PS  - Double precision addition, parameter + stack -> stack
ADD$SS  - Double precision addition, stack + stack -> stack
ADF$MA  - Single precision addition, memory + address -> address
ADF$MM  - Single precision addition, memory + memory -> memory
ADF$MP  - Single precision addition, memory + parameter -> par.
ADF$MS  - Single precision addition, memory + stack -> stack
ADF$PA  - Single precision addition, pointer + address -> address
ADF$PM  - Single precision addition, parameter + memory -> memory
ADF$PP  - Single precision addition, parameter + parameter -> par.
ADF$PS  - Single precision addition, parameter + stack -> stack
ADF$SA  - Single precision addition, stack + address -> address
ADF$SM  - Single precision addition, stack + memory -> memory
ADF$SP  - Single precision addition, stack + pointer -> pointer
ADF$SS  - Single precision addition, stack + stack -> stack
ADI$IA  - Word integer addition, immediate + address -> address
ADI$IM  - Word integer addition, immediate + memory -> memory
ADI$IP  - Word integer addition, immediate + parameter -> parameter
ADI$IS  - Word integer addition, immediate + stack -> stack
ADI$MA  - Word integer addition, memory + address -> address
ADI$MM  - Word integer addition, memory + memory -> memory
ADI$MP  - Word integer addition, memory + parameter -> parameter
ADI$MS  - Word integer addition, memory + stack -> stack
ADI$PA  - Word integer addition, parameter + address -> address
ADI$PM  - Word integer addition, parameter + memory -> memory
ADI$PP  - Word integer addition, parameter + parameter -> parameter
ADI$PS  - Word integer addition, parameter + stack -> stack
ADI$SA  - Word integer addition, parameter + address -> address
ADI$SM  - Word integer addition, stack + memory -> memory
ADI$SP  - Word integer addition, stack + parameter -> parameter
ADI$SS  - Word integer addition, stack + stack -> stack
ADL$MS  - Long integer addition, memory + stack -> stack
ADL$PS  - Long integer addition, parameter + stack -> stack
ADL$SS  - Long integer addition, stack + stack -> stack
AMI$M   - Begin array MOVE TO/FROM code loop, memory array
AMI$P   - Begin array MOVE TO/FROM code loop, parameter array
ANI$    - Word integer AND thread, args/result on stack
ANL$    - Long integer AND thread, args/result on stack
ARI$M   - Get addr of numeric array element, memory array, subs on stk
```

ARI$P - Get addr of num. array element, parameter array, subs on stk
ARI$V - Get addr of num. array element, virtual array, subs on stack
ARR$ - Get descriptor of remappable array element, subs on stack
ARS$C - Get addr of string array elem., common/map array, subs on stk
ARS$M - Get addr of string array element, memory array, subs on stack
ARS$P - Get addr of str array element, parameter array, subs on stk
ARS$V - Get addr of str array element, virtual array, subs on stack
ASC$ - ASCII function, arg/result on stack
ATD$ - Double precision ATN function, args/result on stack
ATF$ - Single precision ATN function, args/result on stack
BEQ$ - Branch equal, based on hardware condition codes
BGE$ - Branch greater than or equal to, based on condition codes
BGT$ - Branch greater than, based on condition codes
BLE$ - Branch less than or equal to, based on condition codes
BLT$ - Branch less than, based on condition codes
BNE$ - Branch not equal, based on condition codes
BRA$ - Branch (unconditional)
BUF$ - BUFSIZ function, channel number on stack, result to stack
CAL$ - CALL, #parameters and routine follow
CBI$ - Convert word integer to byte integer, arg/res on stack
CBR$ - CALL BY REF, #parameters and routine follow
CCD$ - CVT$F with /DOU, arg/result on stack
CCE$ - Enable ^C trapping (CTRLC)
CCF$ - CVT$F with /NODOU, arg/result on stack
CCP$ - CCPOS, arg/res on stack
CCX$ - Disable ^C trapping (RCTRLC)
CDF$ - Convert single precision number to double, arg/res on stack
CDI$ - Convert word integer to double precision, arg/res on stack
CDL$ - Convert long integer to double precision, arg/res on stack
CFD$ - Convert double precision to single, arg/res on stack
CFI$ - Convert word integer to double precision, arg/res on stack
CFL$ - Convert long integer to single precision, arg/res on stack
CHA$ - First thread in CHANGE string to array loop
CHN$ - CHAIN thread
CHR$ - CHR$ function
CHS$ - First thread in CHANGE array to string loop
CID$ - Convert double precision to word integer
CIF$ - Convert single precision to word integer
CIL$ - Convert long word to word integer
CIS$ - CVT$% function
CLB$M - Move 0 byte, memory argument
CLB$S - Move 0 byte to stack
CLD$ - Convert double precision to long integer
CLD$A - Move 0, double precision, address arg.
CLD$M - Move 0, double precision, memory arg.
CLD$P - Move 0, double precision, parameter arg.
CLD$S - Move 0, double precision to stack
CLF$ - Convert single precision to long integer
CLF$A - Move 0, single precision, address arg.
CLF$M - Move 0, single precision, memory arg.
CLF$P - Move 0, single precision, parameter arg.
CLF$S - Move 0, single precision to stack
CLI$ - Convert word integer to long integer
CLI$A - Move 0, word integer, address arg.
CLI$M - Move 0, word integer, memory arg.

CLI$P - Move 0, word integer, parameter arg.
CLI$S - Move 0, word integer to stack
CLL$A - Move 0, long integer, address arg.
CLL$M - Move 0, long integer, memory arg.
CLL$P - Move 0, long integer, parameter arg.
CLL$S - Move 0, long integer to stack
CLR$M - Move 0, RFA arg, memory arg.
CMD$MM - Compare double, memory and memory, set condition codes
CMD$MP - Compare double, memory and parameter, set condition codes
CMD$MS - Compare double, memory and stack, set condition codes
CMD$PM - Compare double, parameter and memory, set condition codes
CMD$PP - Compare double, parameter and parameter, set condition codes
CMD$PS - Compare double, parameter and stack, set condition codes
CMD$SM - Compare double, stack and memory, set condition codes
CMD$SP - Compare double, stack and parameter, set condition codes
CMD$SS - Compare double, stack and stack, set condition codes
CMF$MM - Compare single, memory and memory, set condition codes
CMF$MP - Compare single, memory and parameter, set condition codes
CMF$MS - Compare single, memory and stack, set condition codes
CMF$PM - Compare single, parameter and memory, set condition codes
CMF$PP - Compare single, parameter and parameter, set condition codes
CMF$PS - Compare single, parameter and stack, set condition codes
CMF$SM - Compare single, stack and memory, set condition codes
CMF$SP - Compare single, stack and parameter, set condition codes
CMF$SS - Compare single, stack and stack, set condition codes
CMI$II - Compare word, immediate and immediate, set cond. codes
CMI$IM - Compare word, immediate and memory, set condition codes
CMI$IP - Compare word, immediate and parameter, set condition codes
CMI$IS - Compare word, immediate and stack, set condition codes
CMI$MI - Compare word, memory and immediate, set condition codes
CMI$MM - Compare word, memory and memory, set condition codes
CMI$MP - Compare word, memory and parameter, set condition codes
CMI$MS - Compare word, memory and stack, set condition codes
CMI$PI - Compare word, parameter and immediate, set condition codes
CMI$PM - Compare word, parameter and memory, set condition codes
CMI$PP - Compare word, parameter and parameter, set condition codes
CMI$PS - Compare word, parameter and stack, set condition codes
CMI$SI - Compare word, stack and immediate, set condition codes
CMI$SM - Compare word, stack and memory, set condition codes
CMI$SP - Compare word, stack and parameter, set condition codes
CMI$SS - Compare word, stack and stack, set condition codes
CML$MM - Compare long, memory and memory, set condition codes
CML$MP - Compare long, memory and parameter, set condition codes
CML$MS - Compare long, memory and stack, set condition codes
CML$PM - Compare long, parameter and memory, set condition codes
CML$PP - Compare long, parameter and parameter, set condition codes
CML$PS - Compare long, parameter and stack, set condition codes
CML$SM - Compare long, stack and memory, set condition codes
CML$SP - Compare long, stack and parameter, set condition codes
CML$SS - Compare long, stack and stack, set condition codes
CMR$MM - Comapre RFA, memory and memory, set condition codes
CMR$MP - Comapre RFA, memory and parameter, set condition codes
CMR$MS - Comapre RFA, memory and stack, set condition codes
CMR$PM - Comapre RFA, parameter and memory, set condition codes
CMR$PP - Comapre RFA, parameter and parameter, set condition codes

CMR$PS - Comapre RFA, parameter and stack, set condition codes
CMR$SM - Comapre RFA, stack and memory, set condition codes
CMR$SP - Comapre RFA, stack and parameter, set condition codes
CMR$SS - Comapre RFA, stack and stack, set condition codes
CMS$AA - Compare string, address and address, set condition codes
CND$ - COS function, double precision, arg/res on stack
CNF$ - COS function, single precision, arg/res on stack
COI$IS - Compliment word integer, immediate to stack
COI$MS - Compliment word ingeter, memory to stack
COI$PS - Compliment word ingeter, parameter to stack
COI$SS - Compliment word ingeter, stack to stack
COL$MS - Compliment long ingeter, memory to stack
COL$PS - Compliment long ingeter, parameter to stack
COL$SS - Compliment long ingeter, stack to stack
COM$ - String arithmetic compare, args/result on stack
COS$AA - Concatenate string, address and address, result on stack
COS$AS - Concatenate string, address and stack, result on stack
COS$SA - Concatenate string, stack and address, result on stack
COS$SS - Concatenate string, stack and stack, result on stack
CPD$SM - Copy double, stack to memory
CPD$SP - Copy double, stack to parameter
CPF$SM - Copy single, stack to memory
CPF$SP - Copy single, stack to parameter
CPI$SM - Copy word, stack to memory
CPI$SP - Copy word, stack to parmeter
CSC$ - Return smaller of 2 arguments, args/result on stack
CSD$ - CVTF$ for double precision, arg/res on stack
CSF$ - CVTF$ for single, arg/res on stack
CSI$ - CVTF% arg/res on stack
CVT$ - CVT$$ and EDIT$ function, args/result on stack
DAT$ - DATE$ function, arg/res on stack
DCF$M - Decrement single precision, memory argument
DCI$A - Decrement word integer, address argument
DCI$M - Decrement word integer, memory argument
DCI$P - Decrement word integer, parameter argument
DCI$S - Decrement top of stack
DCL$ - Invoke DEF function
DFF$ - String arith. subtract, args/result on stack
DID$MS - Divide double precision, memory to stack
DID$PS - Divide double precision, parameter to stack
DID$SS - Divide double precision, stack to stack
DIF$MS - Divide single precision, memory to stack
DIF$PS - Divide single precision, parameter to stack
DIF$SS - Divide single precision, memory to stack
DII$IS - Divide word integer, immediate to stack
DII$MS - Divide word integer, memory to stack
DII$PS - Divide word integer, parameter to stack
DII$SS - Divide word integer, stack to stack
DIL$MS - Divide long integer, memory to stack
DIL$PS - Divide long integer, parameter to stack
DIL$SS - Divide long integer, stack to stack
DLN$ - DEBUG line thread
DPD$ - Duplicate stack, double precision
DPF$ - Duplicate stack, single precision
DPI$ - Duplicate stack

```
DTD$     - Double determinant, result on stack
DTF$     - Single determinant, result on stack
EAR$     - End of array MOVE FROM/TO loop
ECD$MM   - Approx compare, double, memory to memory, set cond codes
ECD$MP   - Approx compare, double, memory to parameter, set cond codes
ECD$MS   - Approx compare, double, memory to stack, set cond codes
ECD$PM   - Approx compare, double, parameter to memory, set cond codes
ECD$PP   - Approx comp, double, parameter to parameter, set cond. codes
ECD$PS   - Approx comp, double, parameter to stack, set condition codes
ECD$SM   - Approx compare, double, stack to memory, set condition codes
ECD$SP   - Approx comp, double, stack to parameter, set condition codes
ECD$SS   - Approx compare, double, stack to stack, set condition codes
ECF$MM   - Approx compare, single, memory to memory, set condition codes
ECF$MP   - Approx compare, single, memory to parameter, set cond codes
ECF$MS   - Approx compare, single, memory to stack, set condition codes
ECF$PM   - Approx compare, single, parameter to memory, set cond codes
ECF$PP   - Approx comp, single, parameter to parameter, set cond. codes
ECF$PS   - Approx comp, single, parameter to stack, set condition codes
ECF$SM   - Approx compare, single, stack to memory, set condition codes
ECF$SP   - Approx comp, single, stack to parameter, set condition codes
ECF$SS   - Approx compare, single, stack to stack, set condition codes
ECH$     - ECHO function, arg on stack
ECS$AA   - Exact string compare, args on stack, set condition codes
EDT$     - Enable 1 character input on channel, arg on stack
EFL$     - End of FIELD thread
EFV$     - Dynamic dimension thread, new dims on stack
END$     - END of program
EOL$     - End of I/O list thread
EPU$     - End of PRINT USING statement
EQI$     - EQV of word arguments, args/res on stack
EQL$     - EQV of long arguments, args/res on stack
ERL$     - ERL function
ERN$     - ERN$ function
ERR$     - ERR function
ERT$     - ERT$ function
EXD$     - Exponentiation, double precision, args/result on stack
EXF$     - Exponentiation, single, args/res on stack
FCL$     - DEF* function call
FDB$M    - Return from function with byte result
FDD$M    - Return from function with double result
FDF$M    - Return from function with single result
FDI$M    - Return from function with word result
FDL$M    - Return from function with long result
FDR$M    - Return from function with RFA result
FDS$M    - Return from function with string result
FFA$     - FIND by RFA
FID$     - FIX function for double precision
FIF$     - FIX function for single
FIL$     - FILL argument in MOVE TO/FROM
FIN$     - FSP$ function
FLD$     - FIELD statement
FLN$     - Function line thread
FMS$     - MOVE FROM with fixed length string
FSS$     - FSS$ function
FTD$     - FORMAT$ function, double precision
```

```
FTF$     - FORMAT$ function, single precision
FTI$     - FORMAT$ function, word
FTL$     - FORMAT$ function, long
FTS$     - FORMAT$ function, string
GFA$     - GET by RFA
GSC$     - Computed GO SUB (ON GOSUB)
GSU$     - GO SUB
ICI$A    - Increment word integer, address argument
ICI$M    - Increment word integer, memory argument
ICI$P    - Increment word integer, parameter arg.
ICI$S    - Increment top of stack
IFL$     - Initialize for FIELD
III$     - RE-initialize for INPUT, used for INPUT with prompt
IIN$     - Initialize for INPUT
ILI$     - RE-initialize for LINPUT, used for LINPUT with prompt
ILS$     - Initilaize for INPUT LINE
IMF$     - Initialize for MOVE FROM
IMI$     - IMP for word integer
IML$     - IMP for long integer
IMT$     - Initialize for MOVE TO
IND$     - INT function for double, arg/res on stack
INF$     - INT function for single, arg/res on stack
INS$     - INSTR function, args/res on stack
IOI$     - OR for word integer
IOL$     - OR for long integer
IPR$     - Initialize PRINT with RECORD
IPT$     - Initialize for PRINT
IPU$     - Initialize for PRINT USING
IRD$     - Initialize for READ
IRM$     - Initialize for REMAP
IVB$A    - INPUT byte, address of arg on stack
IVD$A    - INPUT double, address of arg on stack
IVF$A    - INPUT single, address of arg on stack
IVI$A    - INPUT word, address of arg on stack
IVL$A    - INPUT long, address of arg on stack
IVS$A    - INPUT string, address of arg on stack
JBB$     - DEBUG line thread, used a labels
JMC$     - Computed GOTO (ON GOTO)
KGE$     - End of CHANGE number to string
KIL$     - KILL thread
KTI$     - IMP thread for word integers
KTL$     - IMP thread for long integers
LCD$     - Common LOG (LOG10) function for double precision
LCF$     - Common LOG (LOG10) function for single
LEN$     - LEN function
LEQ$     - Load true if equal, input is condition codes, result on stack
LFK$     - Random FIND with KEYs
LFN$     - Sequential FIND
LFR$     - Random FIND
LFT$     - LEFT$ function
LGE$     - Load true if greater or equal, input is cond codes, res stack
LGK$     - GET with KEYs
LGN$     - Sequential GET
LGR$     - Random GET
LGT$     - Load true if greater than, input is cond codes, res on stack
```

```
LIN$     - LINE thread
LIS$     - Initialize for INPUT LINE
LIT$     - Re-initialize for INPUT LINE
LLE$     - Load true if less than or equal, input cond codes, stack res.
LLT$     - Load true if less than, input cond codes, result on stack
LND$     - Natural LOG function (LOG) double precision
LNE$     - Load true if not equal, input is cond codes, result on stack
LNF$     - Natural log function (LOG) single precision
LPC$     - Sequential PUT with count
LPN$     - Sequential PUT
LPR$     - Random PUT with count
LPT$     - Random PUT
LSS$AA   - LSET address to address
LSS$AM   - LSET address to memory
LSS$AP   - LSET address to parameter
LSS$MA   - LSET memory to address
LSS$PA   - LSET parameter to address
LUC$     - UPDATE with count
LUN$     - UPDATE without count
LYN$     - Same as ERL$
MAD$     - Convert string, second to top of stack, to address mode
MAR$     - Tail end of any matrix loop thread
MA1$     - Verify two matricies are same size (1 dimension)
MA2$     - Verify two matricies are same size (2 dimensions)
MFB$     - MOVE FROM byte variable
MFD$     - MOVE FROM double
MFF$     - MOVE FROM single
MFI$     - MOVE FROM word
MFL$     - MOVE FROM long
MFR$     - MOVE FROM RFA
MFS$     - MOVE FROM string
MGT$     - MAGTAPE function
MID$     - Matrix inversion double
MIF$     - Matrix inversion single
MII$     - Matrix inversion word
MIS$     - MID$ function
MM2$     - Verify that matrix multiplication is legal
MOB$MA   - Move byte, memory to address
MOB$MM   - Move byte, memory to memory
MOB$MP   - Move byte, memory to parameter
MOB$MS   - Move byte, memory to stack
MOB$PA   - Move byte, parameter to address
MOB$PM   - Move byte, parameter to memory
MOB$PP   - Move byte, parameter to parameter
MOB$PS   - Move byte, parameter to stack
MOB$SA   - Move byte, stack to address
MOB$SM   - Move byte, stack to memory
MOB$SP   - Move byte, stack to parameter
MOD$MA   - Move double, memory to address
MOD$MM   - Move double, memory to memory
MOD$MP   - Move double, memory to parameter
MOD$MS   - Move double, memory to stack
MOD$PA   - Move double, parameter to address
MOD$PM   - Move double, parameter to memory
MOD$PP   - Move double, parameter to parameter
```

MOD$PS - Move double, parameter to stack
MOD$SA - Move double, stack to address
MOD$SM - Move double, stack to memory
MOD$SP - Move double, stack to parameter
MOD$SS - Move double, stack to stack
MOF$MA - Move single, memory to address
MOF$MM - Move single, memory to memory
MOF$MP - Move single, memory to parameter
MOF$MS - Move single, memory to stack
MOF$PA - Move single, parameter to address
MOF$PM - Move single, parameter to memory
MOF$PP - Move single, parameter to parameter
MOF$PS - Move single, parameter to stack
MOF$SA - Move single, stack to address
MOF$SM - Move single, stack to memory
MOF$SP - Move single, stack to parameter
MOF$SS - Move single, stack to stack
MOI$IA - Move word, immediate to address
MOI$IM - Move word, immediate to memory
MOI$IP - Move word, immediate to parameter
MOI$IS - Move word, immediate to stack
MOI$MA - Move word, memory to address
MOI$MM - Move word, memory to memory
MOI$MP - Move word, memory to parameter
MOI$MS - Move word, memory to stack
MOI$PA - Move word, parameter to address
MOI$PM - Move word, parameter to memory
MOI$PP - Move word, parameter to parameter
MOI$PS - Move word, parameter to stack
MOI$SA - Move word, stack to address
MOI$SM - Move word, stack to memory
MOI$SP - Move word, stack to parameter
MOI$SS - Move word, stack to stack
MOR$MA - Move RFA, memory to address
MOR$MM - Move RFA, memory to memory
MOR$MP - Move RFA, memory to parameter
MOR$MS - Move RFA, memory to stack
MOR$PA - Move RFA, parameter to address
MOR$PM - Move RFA, parameter to memory
MOR$PP - Move RFA, parameter to parameter
MOR$PS - Move RFA, parameter to stack
MOR$SA - Move RFA, stack to address
MOR$SM - Move RFA, stack to memory
MOR$SP - Move RFA, stack to parameter
MOS$AA - Move string, address to address
MOS$AM - Move string, address to memory
MOS$AP - Move string, address to parameter
MOS$AS - Move string, address to stack
MOS$MA - Move string, memory to address
MOS$MM - Move string, memory to memory
MOS$MP - Move string, memory to parameter
MOS$MS - Move string, memory to stack
MOS$PA - Move string, parameter to address
MOS$PM - Move string, paremeter to memory
MOS$PP - Move string, parameter to parameter

MOS$PS - Move string, paremeter to stack
MOS$SA - Move string, stack to address
MOS$SM - Move string, stack to memory
MOS$SP - Move string, stack to parameter
MOS$SS - Move string, stack to stack
MRS$ - REMAP string with length
MR1$ - Used to start loop throught 1 dimensional matrix
MR2$ - Used to start loop throught 2 dimensional matrix
MSI$IM - Same as MOI$IM
MTB$ - MOVE TO byte
MTD$ - MOVE TO double precision
MTF$ - MOVE TO single precision
MTI$ - MOVE TO word integer
MTL$ - MOVE TO long
MTR$ - MOVE TO RFA
MTS$ - MOVE TO string
MUD$MS - Multiply double, memory to stack
MUD$PS - Multiply double, parameter to stack
MUD$SS - Multiply double, stack to stack
MUF$MS - Multiply single, memory to stack
MUF$PS - Multiply single, parameter to stack
MUF$SS - Multiply single, stack to stack
MUI$IS - Multiply word, immediate to stack
MUI$MS - Multiply word, memory to stack
MUI$PS - Multiply word, parameter to stack
MUI$SS - Multiply word, stack to stack
MUL$MS - Multiply long, memory to stack
MUL$PS - Multiply long, parameter to stack
MUL$SS - Multiply long, stack to stack
NCH$ - Thread to do NOECHO function
NDI$M - NEXT thread for word loops with step of -1, memory counter
NDI$P - NEXT thread for word loops with step of -1, parameter counter
NGD$MS - Negate double, memory to stack
NGD$PS - Negate double, parameter to stack
NGD$SS - Negate double, stack to stack
NGF$MS - Negate single, memory to stack
NGF$PS - Negate single, parameter to stack
NGF$SS - Negate single, stack to stack
NGI$MS - Negate word, memory to stack
NGI$PS - Negate word, parameter to stack
NGI$SS - Negate word, stack to stack
NGL$MS - Negate long, memory to stack
NGL$PS - Negate long, parameter to stack
NGL$SS - Negate long, stack to stack
NII$M - Word NEXT with step of 1, memory mode counter
NII$P - Word NEXT with step of 1, parameter mode counter
NMD$ - NUM$ function for double precision
NMF$ - NUM$ for single
NML$ - NUM$ for LONG
NMO$ - NUM function
NM2$ - NUM2 function
NOI$A - Move -1 (word) to address
NOI$M - Move -1 (word) to memory
NOI$P - Move -1 (word) to parameter
NOI$S - Move -1 (word) to stack

```
NSS$AA  - Null set (LET for virtual array strings), address to address
NSS$MA  - Null set (LET for virtual array strings), memory to address
NSS$PA  - Null set (LET for virtual array strings), parameter to addr
NVB$M   - NEXT, byte, memory counter
NVB$P   - NEXT, byte, parameter counter
NVD$M   - NEXT, double, memory counter
NVD$P   - NEXT, double, parameter counter
NVF$M   - NEXT, single, memory counter
NVF$P   - NEXT, single, parameter counter
NVI$M   - NEXT, word, memory counter
NVI$P   - NEXT, word, parameter counter
NVL$M   - NEXT, long, memory counter
NVL$P   - NEXT, long, parameter counter
N1D$    - NUM1$ function, double precision
N1F$    - NUM1$ function, single
N1L$    - NUM1$ function, long
OEA$    - ON ERROR GOTO 0 statement
OEG$    - ON ERROR GOTO statement
OGB$    - ON ERROR GO BACK statement
OGS$    - Special ON ERROR GO BACK for start of subprograms/DEFs
ONI$A   - Move word 1 to address
ONI$M   - Move word 1 to memory
ONI$P   - Move word 1 to parameter
ONI$S   - Move word 1 to stack
PLA$    - PLACE$ function
POS$    - POS function
PRO$    - PROD$ function
PUD$S   - PRINT USING, double precision
PUF$S   - PRINT USING, single
PUI$S   - PRINT USING, word
PUL$S   - PRINT USING, long
PUS$A   - PRINT USING, string
PVD$SI  - PRINT, double
PVF$SI  - PRINT, single
PVI$SI  - PRINT, word
PVL$SI  - PRINT, long
PVS$AI  - PRINT, string
QUO$    - QUO$ function
RAD$    - RAD$ function
RCL$    - CLOSE statement, channel on stack
RCO$    - RCTRLO function
RCT$    - RECOUNT function
RDI$M   - Matrix redimension array, memory mode array
RDI$P   - Matrix redimension array, parameter mode arrary
RDL$    - DELETE statement
REG$    - RETURN statement
RFA$    - GETRFA function
RFK$    - FIND with KEYs
RFL$    - REMAP FILL item
RFN$    - Sequential FIND
RFR$    - Random FIND
RGK$    - GET with KEYs
RGN$    - Sequential GET
RGR$    - Random GET
RIS$    - RESTORE with key
```

```
RIT$     - RIGHT$ function
RLI$I    - Move address of immediate operand to stack
RLI$M    - Move address of memory operand to stack
RLI$P    - Move address of parameter operand to stack
RMB$     - REMAP byte
RMD$     - REMAP double
RME$     - End of REMAP
RMF$     - REMAP single
RMI$     - REMAP word
RML$     - REMAP long
RMM$     - Clean-up dynamic arrays at end of subprogram
RMR$     - REMAP RFA
RMS$     - REMAP string
RND$     - RND function, double precision
RNF$     - RND function, single precision
RNZ$     - RANDOMIZE statement
ROP$     - OPEN statement
RPC$     - Sequential PUT with count
RPN$     - Sequential PUT
RPR$     - Random PUT with count
RPT$     - Random PUT
RSC$     - SCRATCH statement
RSI$M    - Same as MOI$MS
RSI$P    - Same as MOI$PS
RSM$     - RESUME with line number
RSR$     - RESTORE with channel number
RSS$AA   - RSET address to address
RSS$AM   - RSET address to memory
RSS$AP   - RSET address to parameter
RSS$MA   - RSET memory to address
RSS$PA   - RSET parameter to address
RST$     - RESTORE statement
RSU$     - RESUME statement
RUC$     - UPDATE with count
RUL$     - UNLOCK statement
RUN$     - UPDATE with no count
SBE$     - SUBEND statement
SEG$     - SEG$ function
SGD$     - SGN function for double precision
SGF$     - SGN function for single
SID$     - SIN function for double precision
SIF$     - SIN function for single
SLP$     - SLEEP function
SPC$     - SPACE$ function
SPK$     - SPEC% function
SQD$     - SQR function for double precision
SQF$     - SQR function for single
SSD$     - Swap two args on top of stack, double precision
SSF$     - Swap two args on top of stack, single
SSI$     - Swap two args on top of stack, word
SSL$     - Swap two args on top of stack, long
SSS$     - Swap two args on top of stack, string
STA$     - Statement thread (only /DEB)
STD$     - STR$ function for double precision
STF$     - STR$ function for single
```

```
STL$     - STR$ function for long
STN$     - Stmt thrd (/DEB) if control can get here from within line
STP$     - STOP statement
STR$     - STRING$ function
STS$     - STATUS function
SUD$MS   - Subtract, double precision, memory to stack
SUD$PS   - Subtract, double precision, parameter to stack
SUD$SS   - Subtract, double precision, stack to stack
SUF$MA   - Subtract, single, memory to address
SUF$MM   - Subtract, single, memory to memory
SUF$MP   - Subtract, single, memory to parameter
SUF$MS   - Subtract, single, memory to stack
SUF$PA   - Subtract, single, parameter to address
SUF$PM   - Subtract, single, parameter to memory
SUF$PP   - Subtract, single, parameter to parameter
SUF$PS   - Subtract, single, parameter to stack
SUF$SA   - Subtract, single, stack to address
SUF$SM   - Subtract, single, stack to memory
SUF$SP   - Subtract, single, stack to parameter
SUF$SS   - Subtract, single, stack to stack
SUI$IA   - Subtract, word, immediate to address
SUI$IM   - Subtract, word, immediate to memory
SUI$IP   - Subtract, word, immediate to parameter
SUI$IS   - Subtract, word, immediate to stack
SUI$MA   - Subtract, word, memory to address
SUI$MM   - Subtract, word, memory to memory
SUI$MP   - Subtract, word, memory to parameter
SUI$MS   - Subtract, word, memory to stack
SUI$PA   - Subtract, word, parameter to address
SUI$PM   - Subtract, word, parameter to memory
SUI$PP   - Subtract, word, parameter to parameter
SUI$PS   - Subtract, word, parameter to stack
SUI$SA   - Subtract, word, stack to address
SUI$SM   - Subtract, word, stack to memory
SUI$SP   - Subtract, word, stack to parameter
SUI$SS   - Subtract, word, stack to stack
SUL$MS   - Subtract, long, memory to stack
SUL$PS   - Subtract, long, parameter to stack
SUL$SS   - Subtract, long, stack to stack
SUM$     - SUM$ function
SWE$     - Initialize DEF* thread
SWI$     - SWAP% for words
SWL$     - SWAP% for long
SZI$M    - Get array dimensions, memory array
SZI$P    - Get array dimensions, parameter array
TAB$     - TAB function thread
TAP$     - TAPE function
TET$     - Tail of change string to number loop
TIM$     - TIME$ function
TJK$     - Initialize for DEFs thread
TMS$     - MOVE TO string with length
TND$     - TAN function for double precision
TNF$     - TAN function for single precision
TRM$     - TRM$ function
TSB$     - Test Byte, make sure word on stack is legal byte value
```

```
TSD$M   - Test double precision value, memory arg.
TSD$P   - Test double precision value, parameter arg.
TSD$S   - Test double precision value, stack arg.
TSF$M   - Test single value, memory arg.
TSF$P   - Test single value, parameter arg.
TSF$S   - Test single value, stack arg.
TSI$I   - Test word value, immediate arg.
TSI$M   - Test word value, memory arg.
TSI$P   - Test word value, parameter arg.
TSI$S   - Test word value, stack arg.
TSL$M   - Test long value, memory arg.
TSL$P   - Test long value, parameter arg.
TSL$S   - Test long value, stack arg.
TYD$    - TIME function, double precision
TYF$    - TIME function for single
ULK$    - UNLOCK thread
USE$    - Clean up at end of DEFs and DEF*s
VLD$    - VAL function for double precision
VLF$    - VAL function for single
VLI$    - VAL% function for word
VLL$    - VAL% functio for long
VRI$M   - Return value of numeric memory array, subscripts on stack
VRI$P   - Return value of numeric parameter array, subs on stack
VRI$V   - Return value of numeric virtual array, subs on stack
VRS$C   - Return value of string common/map array, subs on stack
VRS$M   - Return value of string memory array, subs on stack
VRS$P   - Return value of string parameter array, subs on stack
VRS$V   - Return value of string virtual array, subs on stack
WAT$    - WAIT thread
XDD$    - Exponentation thread, double ** double
XDI$    - Exponentation thread, double ** word
XFF$    - Exponentation thread, single ** single
XFI$    - Exponentation thread, single ** word
XII$    - Exponentation thread, word ** word
XLL$    - Exponentation thread, long ** long
XLN$    - Special line thread (/DEB only) used around DEFs and loops
XLT$    - XLATE$ function
XOI$    - XOR for word integers
XOL$    - XOR for long integers
```