LANGUAGE SERIES

PASCAL NEWSLETTER

SYSTEMS

Table of Contents

(C) Copyright ALCOR SYSTEMS 1982

Table of Contents

(C) Copyright ALCOR SYSTEMS 1982

History of the Alcor Pascal Compiler

The Alcor Pascal compiler grew out of the desire four years ago to have a quality Pascal compiler for CP/M use. At that time there were no professional quality compilers on the popular CP/M operating systems that were very memory efficient. After two years of development a completely operational compiler emerged that saw several hundred thousand lines of Pascal code run through it. At this time it was decided that this compiler formed an ideal basis for the first professional quality Pascal compiler on the TRS80. Since a void existed, and Pascal's popularity was ever increasing, it was decided to enter the TRS80 market.

A project team was formed and 1 1/2 man years later saw the Alcor Pascal compiler introduced as a commercial product. This underlines the amount of work required to take a completed system and make it a viable commercial software product. The Pascal compiler consists of 17,500 lines of assembly language runtime support and over 8,500 lines of Pascal source code. Not a minimal amount of work for a product in this price catagory !

The Difficulties of Integrating Languages With Different Operating System Environments

Alcor Systems has taken a particular approach to integrating into operating systems environments. We have chosen to implement our products on many computers with DIFFERENT operating systems. This is a tough job that can become very time consuming, however users typically have preferential tastes for the various operating systems on different computers. To force an operating system such as U.C.S.D. ($600 and $800 per copy) on the user for the sake of portability is a poor compromise.

It is true that operating systems will eventually become standardized, but the micro world is not quite there yet. An excellent example is CP/M. This is the most standardized operating system for Z-80 micros in existence, but as any TRS80 / CP/M user will tell you, CP/M is less sophisticated than many TRS80 operating systems such as LDOS,NEWDOS,DOSPLUS etc. In order to understand a few different disk formats requires extensive assembly language modification to the BIOS. Most TRS80 users are not used to this lack of sophistication.

(C) Copyright ALCOR SYSTEMS 1982

Alcor Systems chose the TRSDOS operating systems as the lowest common denominator in the link to all TRS80 users. There are a number of other excellent operating systems available to the TRS80 user that are more sophisticated than TRSDOS. We have made a concerted effort to assure compatibility of our system with most of these operating systems. The simplest way to achieve this in a language system is to make documented operating system calls to perform the various functions. This is the approach that Alcor took in implementing Pascal on the TRS80. The primary difference between the various TRSDOS like operating systems from an application programs viewpoint is the way the operating system handles the end of file mark in the directory for the various files. The end result is that if a program is assembled or compiled, then loaded into a command file on one operating system and transferred to another, it may not execute correctly if it performs file I/O through the operating system. To alleviate this problem, Alcor modified the runtime support for the Pascal system so that it could be specially patched by the Alcor patch program to execute correctly on any of the supported TRSDOS like operating systems.

Generally when the Pascal system is ported to execute on any operating system, all I/O routines in the runtime support must be be rewritten to conform to the calling sequence for I/O in the new operating system environment. This is a substantial job and requires a significant amount of work.

Runtime Support

Most TRS80 users have a vague feeling for the term "runtime support" as used by the typical systems programmer. Runtime support for the Pascal system refers to the modules linked to a users program by the linking loader. These modules consist of the interpreter and all of the necessary modules to perform I/O. This represents over 12,500 lines of assembly language code and in fact is one of the most difficult parts of a good Pascal compiler. The compiler itself is an 8,500 line Pascal source program that also has the 12,500 lines of runtime support code linked to it. In fact, the interpreter part of the runtime support is the part of the system that determines how efficient and fast the final program will execute. When the Pascal system is ported to another operating system, it is the I/O part of the runtime support that must be modified to conform to the new I/O subroutine calling sequences.

(C) Copyright ALCOR SYSTEMS 1982

When target processors are changed, for efficient execution, the entire interpreter must be rewritten. This also is a significant task. Many companies have not rewritten their interpreters when transitioning to the new processors such as the 68000,8086 and 8088. They have simply translated the interpreters assembly language to the "closest fit" assembly language of the new processor. This almost ALWAYS causes the interpreter to become extremely inefficient. This is because for maximum efficiency, the interpreter must use all of the hardware features of the new processor such as new registers, hardware stacks, etc. A fine example of this is Microsofts IBM Basic. Reports are now indicating that programs execute slower on the IBM (8088 processor) than the Z-80 counterpart. This is extremely interesting since the Intel 8088 is a fine 16 bit processor that is inherently faster and more efficient than the Z-80. This simply proves how important it is to have language systems that are optimized for the processor and operating system.

Pascal 1.2A Release Notes

Overview

The changes between Pascal 1.2 and 1.2A are primarily patches to correct reported bugs. There are enhancements to the text editor which allow printable characters not on the TRS80 keyboard to be generated with clear key sequences. Pascal 1.2A has all the known bugs at this time corrected. Registered Pascal 1.2 owners may apply patches to version 1.2 with the supplied Alcor patch program.

Since this is not an upgrade requiring recompilation of the Pascal system, patches will be available in two different forms; printed (in this newsletter), or directly from Alcor on a diskette. The patches printed in this newsletter may be entered into a file with Blaise or any other editor that understands text or ASCII format files. This file is then used as the patch file for the Alcor patch program. There are a series of patches that should be applied to correct errors and several that are optional. If you do not wish to type them in, you may send $8.00 + shipping to Alcor Systems (Please note your system serial number) for a diskette containing all of the patches entered into files and copies of the Alcor patch program setup to patch under the various supported operating systems.

(C) Copyright ALCOR SYSTEMS 1982

(Ldos,Newdos,Dosplus,Trsdos) This diskette also contains the program examples from this newsletter.

********** NOTICE TO PASCAL 1.2A OWNERS **********

The 1.2A release disks have already been corrected and require no patching of any kind to correct bugs. The only patching necessary is for Pascal operation under the different operating systems. See the supplied Patch instruction sheet.

****

Known Bugs in Pascal 1.2

PASCAL COMPILER

A. Sets

<item> IN <set expression>

The IN operator sometimes returns true when <item> is not a member of the set when the ordinal of the <item> is greater than the ordinal of the highest member actually present in the <set expression>.

B. Procedure calls

When making procedure calls of the form:

IF <expression> THEN <procedure call> else <statement>

the compiler will incorrectly flag a syntax error if the procedure has no parameters.

C. Hexadecimal constants

If hexadecimal constants are declared greater than $8000 <hex 8000> the compiler incorrectly stores the value.

(C) Copyright ALCOR SYSTEMS 1982

D. File of Char

When a FILE OF <TYPE> is declared, and the generated file contains the control characters CR <#0D>, LF <#0A>, TAB <#0B> and Control Z <#1A>, they will be treated like they are in a text file and cause special processing when the is file read by Pascal. The correction causes them to be treated like any other character.

Known Bugs in Release 1.2

EDITOR

A. "RENAME FAILED CRASH"

Under certain conditions that are highly operating system and data file dependent, upon exiting from the editor, a "RENAME FAILED" error flag will be displayed. This occasionally results in a file loss. This is caused by an improper filename termination in the file descriptor <no CR termination of filename string> by the editor runtime.

Enhancements in Release 1.2A

EDITOR

A. I/O error detection and recovery

An enhancement to allow for better I/O error correction and recovery has been made in the text editor. When a disk error has been detected, the message:

IO ERROR

followed by the operating system error code is displayed. You may type "Q" to allow the error to pass or any other key to cause a retry.

(C) Copyright ALCOR SYSTEMS 1982

B. Screen update change

In 1.2 when the cursor is at the bottom of the display and the enter key is hit, the screen is completely cleared and rewritten. Release 1.2A causes the screen to scroll by sending a line feed to the screen driver. This increases the effective scroll rate.

C. Character generation

In 1.2A certain characters not available on the TRS80 keyboard may be generated by using the clear key sequences. They are:

Sequence	character
clear 1 -------	[↑
clear 2 -------	] ←
clear 3 -------	^ →
clear 4 -------	{
clear 5 -------	}
clear 6 -------	| ◆
clear 7 -------	/

D. Clear key definition

An optional patch is available to allow the clear key to be redefined to the: Shift Down Arrow (Model I only) or to the: / key on the Model I or Model III.

More Information About The Alcor Patch Program

The Alcor patch program is used to patch Pascal system files utilizing a control file that contains information about the patch to be applied. The control file is like any text or character file. It contains special commands and checks to assure that the patches are correctly applied. It is not designed for general use to patch non Pascal files. The control file may be built by using any text editor that can create normal ASCII text files. (Blaise)

(C) Copyright ALCOR SYSTEMS 1982

As supplied on diskette, the Alcor Pascal system requires a valid operating system to always be resident in drive 0. It may be one of the following:

MODEL I

* Trsdos 2.3, * Ldos 5.1, Newdos 2.0, Dosplus 3.3, 3.4

MODEL III

* Trsdos 1.3, Ldos 5.1, * Newdos 2.0, Dosplus 3.3, 3.4

* - The Pascal system will operate properly as delivered on diskette without patching. The others require patching with tha Alcor patch program using the control files supplied on diskette. The Pascal system files should be patched under one of the "*" systems and then copied to the desired system format.

Patch Command Format

(Example)

```
;
;  This patch will fix the rename failed error in the editor
;
F, ED/CMD, ALCOR1
P,5A8E,0B64,0006,10,10,06,04,03,10,01,06,00,03,22,01
P,5A94,0A4B,0005,4B,00,01,22,00,4B,12,02,F7,00
C,5A99,04C0,0001,22
W,E591
E
```

The ";" character is used as a comment specifier. Any text to the right of the comment specifier is ignored by the patch program.

The first data line of a patch file contains a file specification header: (Example)

F, ED/CMD, ALCOR1

"F" - File header prefix. (For /CMD files only)
("O" for /OV1 /OV2 /OV3 /OV4 overlay files)
"ED/CMD" - The filename of the file to be patched.
"ALCOR1" - Output prompt by patch program requesting the name of the diskette to be inserted into the drive.

Lines 2 through N contain the actual patch information:

(C) Copyright ALCOR SYSTEMS 1982

(Example)

```
P,5A8E,0B64,0006,10,10,06,04,03,10,01,06,00,03,22,01
```

```
"P"             - Patch header prefix.
"5A8E"          - Address of change relative to load point of pgm.
"0B64"          - Checksum for the line
"0006"          - Number of bytes to be changed
"10,10......"   - Old contents, new contents
```

This line contains checksum information. This is used to assure that previous patches have been correctly applied.
(Example)

```
C,5A99,04C0,0001,22
```

```
"C"         -   Checksum header prefix.
"5A99"      -   Address of data to be checked
"04C0"      -   Checksum of line
"0001"      -   Number of bytes checked
"22"        -   Expected contents of checked location
```

This line is the command to execute the patch.
(Example)

```
W, 5284
```

```
"W"         -  Write header prefix.
"5284"      -  Checksum for entire patch
```

The final line of the file.
(Example)

```
E
```

In general, multiple patches may be combined in a file. In this case, the "E" is placed at the end of the last patch.

(C) Copyright ALCOR SYSTEMS 1982

Special Notice about the Alcor Pascal Patch program

IT IS RECOMMENDED THAT ALL PATCHING BE DONE UNDER TRSDOS

The Alcor Patch program may be used to apply any patches to the system while operating under TRSDOS on the model I or III. This includes patching the system with the supplied conversion patches which allows the system to execute under the various supported operating systems such as LDOS, NEWDOS and DOSPLUS. After patching has been performed under the TRSDOS operating system, the Pascal system may then be copied to any of the ALCOR supported operating systems using the normal operating system utilities.

IF YOU MUST PATCH WHILE EXECUTING UNDER ANY OF THE THE ALCOR SUPPORTED OPERATING SYSTEMS OTHER THAN TRSDOS

Generally, if the Pascal system must be patched for execution under your computer and operating system combination, then the PATCH program should be patched first. This particular patch for the PATCH program may be applied while under ANY of the Alcor supported operating systems.
The following patch may be entered into a text file with any legal filename. IE; SPECIAL/PAT You may use the text editor if creating this file under TRSDOS and copy it to the desired operating system.

For Model I systems

```
F,  PATCH/CMD, ALCOR
P,0FA1,054E,0001,13,00
P,0B16,0529,0001,00,03
W,F589
E
```

For Model III systems

```
F,  PATCH/CMD, ALCOR
P,0FA1,054E,0001,00,13
P,0B16,0529,0001,03,00
W,F589
E
```

(C) Copyright ALCOR SYSTEMS 1982

Patches to Update 1.2 to 1.2A

MODEL I

```
;      IN STATEMENTS OF THE FORM:
;         <ITEM> IN <SET EXPRESSION>
;      THE IN OPERATOR SOMETIMES RETURNS TRUE WHEN <ITEM> IS
;         NOT A MEMBER OF THE SET WHEN THE ORDINAL OF <ITEM> IS
;         GREATER THAN THE ORDINAL OF THE HIGHEST MEMBER ACTUALLY
;         PRESENT IN THE <SET EXPRESSION>.
;      IN THE PASCAL COMPILER, STATEMENTS OF THE FORM:
;         IF <EXPRESSION> THEN <PROCEDURE CALL> ELSE <STATEMENT>
;         THE COMPILER INCORRECTLY FLAGS A SYNTAX ERROR IF THE
;         PROCEDURE CALL IS A CALL TO A PROCEDURE WITH NO PARAMETERS
;
F, RUN/CMD, ALCOR1
P,091F,0847,0003,6F,C3,26,F8,00,7D
P,2BF8,0D2F,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,2BFE,0F1C,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
W,DB6E
;
F, PASCAL/CMD, ALCOR1
P,091F,0842,0003,6F,C3,26,6C,00,D8
P,866C,0D19,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,8672,0E91,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
P,2313,065C,0002,20,7F,20,41
W,D5B8
;
F, LINKLOAD/CMD, ALCOR2
P,091F,0839,0003,6F,C3,26,DF,00,70
P,1EBF,0D60,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,1EC5,0ED8,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
W,DB8F
;
F, PASCALB/CMD, ALCOR2
P,091F,081B,0003,6F,C3,26,47,00,78
P,2647,0CDD,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,264D,0ECA,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
P,2443,0667,0002,20,7F,20,41
W,D5D7
;
;      PATCH TO FIX HEXADECIMAL CONSTANTS GREATER THAN #8000
;
```

(C) Copyright ALCOR SYSTEMS 1982

```
F, PASCALB/CMD, ALCOR2
P,1FB8,059E,0001,3E,24
P,2443,066B,0002,7F,41,41,20
W,F3F7
O, PASCAL/OV4, ALCOR2
P,2207,057B,0001,7F,FF
W,FA85
F,PASCAL/CMD,ALCOR1
P,1E88,057E,0001,3E,24
P,2313,0660,0002,7F,41,41,20
P,7BF1,05CA,0001,7F,FF
W,EE58
;
; PATCH TO CORRECT PROCESSING OF FILE OF CHAR
;    WHEN THE FILE CONTAINS CONTROL CHARACTERS
;
F,PASCAL/CMD,ALCOR1
P,7A8A,08C4,0003,9C,4D,5C,EC,6D,0B
P,8679,0D02,0006,FF,10,FF,0C,FF,5C,FF,6D,FF,27,FF,10
P,867F,0D41,0006,FF,10,FF,4A,FF,0C,FF,00,FF,0B,FF,5C
P,8685,0CFE,0006,FF,6F,FF,27,FF,42,FF,4D,FF,03,FF,F4
W,CFFB
O,PASCAL/OV4,ALCOR2
P,20A0,0874,0003,9C,4D,5C,EC,6D,0B
P,2C8F,0D4D,0006,FF,10,FF,0C,FF,5C,FF,6D,FF,27,FF,10
P,2C95,0D17,0006,FF,10,FF,4A,FF,0C,FF,00,FF,0B,FF,5C
P,2C9B,0D49,0006,FF,6F,FF,27,FF,42,FF,4D,FF,03,FF,F4
W,CFDF
;
; PATCH TO FIX THE OCCASIONAL "RENAME FAILED" ERROR IN
;    THE TEXT EDITOR
;
F, ED/CMD, ALCOR1
P,5A8E,0B64,0006,10,10,06,04,03,10,01,06,00,03,22,01
P,5A94,0A4B,0005,4B,00,01,22,00,4B,12,02,F7,00
C,5A99,04C0,0001,22
W,E591
;
;  EDITOR PATCH TO ADD TRANSLATIONS FOR CHARACTERS
;     NOT GENERATED BY THE TRS80 KEYBOARD   (optional)
;
F, ED/CMD, ALCOR1
P,46A7,0849,0003,03,4D,8A,5A,00,CD
P,1404,0BF9,0006,CD,10,BE,0A,62,4A,46,05,EB,00,21,5A
P,140A,0BD1,0006,A4,31,62,22,7E,02,23,5B,A7,0C,28,10
P,1410,0BC7,0006,1B,0A,B8,4A,28,05,07,00,5E,5A,16,32
P,1416,0BAE,0006,00,22,19,02,23,5D,18,0C,F1,10,4E,0A
P,141C,0BF5,0006,06,4A,00,05,03,00,DD,5A,5E,33,0A,22
```

(C) Copyright ALCOR SYSTEMS 1982

```
P,1422,0C24,0006,DD,02,56,5E,0B,0C,D5,10,13,0A,ED,4A
P,1428,0BBF,0006,B0,05,C3,00,AE,5A,66,34,21,22,BB,02
P,142E,0C42,0006,66,7B,7E,0C,A7,10,28,0A,5C,4A,21,05
P,1434,0BB8,0006,99,00,62,5A,7E,35,A7,22,28,02,43,7D
P,143A,0BB4,0006,B8,0C,28,10,06,0A,11,4A,05,05,00,00
P,1440,0BE3,0006,19,5A,18,36,F3,22,23,02,E5,7C,DD,0C
P,1446,0C08,0006,5E,10,0A,0A,DD,4A,56,05,0B,00,13,5A
P,144C,0C2C,0006,13,37,AF,22,21,02,BB,2F,66,0C,4E,03
P,1452,0A57,0005,23,8A,06,00,00,4D,81,53,ED,32
W, 5284
;
;  PATCH TO INCREASE SPEED OF SCREEN UPDATE
;
F, ED/CMD, ALCOR1
P,475A,0860,0003,4A,4D,61,FA,02,CC
P,1457,0B7C,0006,B0,12,E1,F9,01,4A,04,F1,00,01,81,03
P,145D,0BD4,0006,ED,01,B0,00,21,03,C4,0A,66,00,F5,24
P,1463,0BBC,0006,7E,22,23,17,A7,C8,28,00,12,11,4F,C8
P,1469,0BA6,0006,F1,00,81,02,06,01,00,0C,ED,11,B0,C8
P,146F,0C2B,0006,DD,00,5E,66,0A,03,DD,01,56,00,0B,04
P,1475,0B8C,0006,D5,0C,13,11,12,C8,18,00,34,67,F1,02
P,147B,0CB1,0006,18,1F,F2,0C,DD,12,6E,F9,0A,4A,DD,F1
P,1481,0B61,0006,66,01,0B,03,E5,02,23,00,36,03,02,0A
P,1487,0B74,0006,23,00,36,24,3A,22,23,17,36,C8,43,00
P,148D,0C0E,0006,18,11,1F,C8,DD,00,E5,02,E1,01,11,0C
P,1493,0B8C,0006,08,11,00,C8,CD,00,E6,66,67,03,21,01
P,1499,0BAC,0006,B8,00,66,04,11,0C,03,11,00,C8,CD,00
P,149F,0CA9,0006,E6,67,67,02,DD,1E,6E,0C,0A,12,DD,F9
P,14A5,0BF1,0006,66,4A,0B,F1,E5,01,36,03,40,0A,EB,00
P,14AB,0C49,0006,CD,03,F8,0A,67,00,D1,24,DD,22,E5,17
P,14B1,0BDE,0006,E1,C8,CD,00,C7,11,66,C8,C3,00,D2,02
P,14B7,0B95,0006,62,01,20,0C,3D,11,20,C8,00,00,14,66
P,14BD,0C03,0006,21,03,7C,01,14,00,CD,04,5A,0C,13,11
P,14C3,0C3D,0006,D1,C8,00,00,C9,67,FD,02,DD,0A,E5,0C
P,14C9,0C61,0006,C5,4A,D5,61,E5,02,E5,4D,DD,8E,E1,32
P,1804,0567,0001,AF,C9
W,040D
;
;  PATCH TO ALLOW RETRYS ON DISK ERRORS
;     WHEN A DISK ERROR IS DETECTED, THE MESSAGE:
;        IO ERROR:
;     FOLLOWED BY THE ERROR CODE IS DISPLAYED
;     YOU MAY TYPE "Q" TO ALLOW THE ERROR TO PASS
;     OR ANY OTHER KEY TO RETRY
;
F, ED/CMD, ALCOR1
P,127A,085F,0003,FE,C3,1D,CF,20,66
P,14CF,0CDD,0006,EB,FE,23,1D,7E,CA,A7,7E,28,64,15,F5
```

(C) Copyright ALCOR SYSTEMS 1982

```
P,14D5,0C4D,0006,23,CD,E6,F7,1F,66,4F,FE,06,51,00,28
P,14DB,0C7B,0006,7E,04,FE,F1,3A,C3,20,6E,1C,64,23,F1
P,14E1,09B8,0004,7E,C1,FE,C3,4C,AA,28,64
P,1377,0803,0003,DD,C3,E1,E5,20,66
P,14E5,0CB9,0006,0A,DD,FE,E1,44,CA,28,7B,0C,65,DD,F5
P,14EB,0CE5,0006,36,CD,0E,F7,1B,66,18,FE,1B,51,DD,CA
P,14F1,0C40,0006,36,C9,0E,5E,1A,F1,18,C3,15,65,DD,65
P,14F7,0CCB,0006,36,C5,0E,D5,1C,E5,18,DD,0F,E5,DD,FD
P,14FD,0C80,0006,5E,E5,08,F5,DD,21,56,A0,09,5E,7A,11
P,1503,0C25,0006,B3,0B,28,00,0B,CD,ED,E6,B0,67,3E,F1
P,1509,0C26,0006,0D,CD,12,29,E1,67,D1,CD,C1,49,DD,00
P,150F,0C9E,0006,E1,FD,C9,E1,DD,DD,36,E1,12,E1,FF,D1
P,1515,0693,0002,18,C1,F4,C9
P,0CA0,0C0B,0006,D1,0D,E1,49,D5,4F,5E,20,23,45,56,52
P,0CA6,0AC7,0005,23,52,E3,4F,E5,52,D5,3A,C3,20
P,0CC9,09E6,0004,0E,F1,FE,C3,1A,7F,28,65
W,34E4
;
;  INTERNAL EDITOR PATCH
;
F, ED/CMD, ALCOR1
P,5957,0804,0003,12,4D,F5,51,4A,B3
P,0CAB,0C39,0006,4C,10,5D,02,D1,58,D1,12,E1,F7,C5,22
P,0CB1,0BC4,0006,D5,0A,E5,68,01,20,00,10,01,02,CD,56
P,0CB7,0C07,0006,96,2B,60,42,EB,21,E1,07,D5,10,7A,02
P,0CBD,0BFB,0006,B3,58,28,15,44,02,01,1F,00,E9,01,12
P,0CC3,0C9A,0006,CD,F5,26,65,5E,08,FE,4D,0D,93,28,4C
W,BB63
E
```

Patches to update 1.2 to 1.2A

MODEL III

```
;         IN STATEMENTS OF THE FORM:
;         <ITEM> IN <SET EXPRESSION>
;
;      THE IN OPERATOR SOMETIMES RETURNS TRUE WHEN <ITEM> IS
;         NOT A MEMBER OF THE SET WHEN THE ORDINAL OF <ITEM> IS
;         GREATER THAN THE ORDINAL OF THE HIGHEST MEMBER ACTUALLY
;         PRESENT IN THE <SET EXPRESSION>.
;
;      IN THE PASCAL COMPILER, STATEMENTS OF THE FORM:
```

(C) Copyright ALCOR SYSTEMS 1982

```
;         IF <EXPRESSION> THEN <PROCEDURE CALL> ELSE <STATEMENT>
;
;         THE COMPILER INCORRECTLY FLAGS A SYNTAX ERROR IF THE
;         PROCEDURE CALL IS A CALL TO A PROCEDURE WITH NO PARAMETERS.
;
F, RUN/CMD, ALCOR1
P,091F,0847,0003,6F,C3,26,F8,00,7D
P,2BF8,0D2F,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,2BFE,0F1C,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
W,DB6E
;
F, PASCAL/CMD, ALCOR1
P,091F,0842,0003,6F,C3,26,6C,00,D8
P,866C,0D19,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,8672,0E91,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
P,2313,065C,0002,20,7F,20,41
W,D5B8
;
F, LINKLOAD/CMD, ALCOR2
P,091F,0839,0003,6F,C3,26,DF,00,70
P,1EBF,0D60,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,1EC5,0ED8,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
W,DB8F
;
F, PASCALB/CMD, ALCOR2
P,091F,081B,0003,6F,C3,26,47,00,78
P,2647,0CDD,0006,FF,BA,FF,30,FF,06,FF,6F,FF,26,FF,00
P,264D,0ECA,0007,FF,C3,FF,22,FF,5B,FF,AF,FF,C3,FF,36,FF,5B
P,2443,0667,0002,20,7F,20,41
W,D5D7
;
;       PATCH TO FIX HEXADECIMAL CONSTANTS GREATER THAN #8000
;
F, PASCALB/CMD, ALCOR2
P,1FB8,059E,0001,3E,24
P,2443,066B,0002,7F,41,41,20
W,F3F7
O, PASCAL/OV4, ALCOR2
P,2207,057B,0001,7F,FF
W,FA85
F,PASCAL/CMD,ALCOR1
P,1E88,057E,0001,3E,24
P,2313,0660,0002,7F,41,41,20
P,7BF1,05CA,0001,7F,FF
W,EE58
;
; PATCH TO CORRECT PROCESSING OF FILE OF CHAR
;     WHEN THE FILE CONTAINS CONTROL CHARACTERS
```

(C) Copyright ALCOR SYSTEMS 1982

```
;
F,PASCAL/CMD,ALCOR1
P,7A8A,08C4,0003,9C,4D,5C,EC,6D,0B
P,8679,0D02,0006,FF,10,FF,0C,FF,5C,FF,6D,FF,27,FF,10
P,867F,0D41,0006,FF,10,FF,4A,FF,0C,FF,00,FF,0B,FF,5C
P,8685,0CFE,0006,FF,6F,FF,27,FF,42,FF,4D,FF,03,FF,F4
W,CFFB
O,PASCAL/OV4,ALCOR2
P,20A0,0874,0003,9C,4D,5C,EC,6D,0B
P,2C8F,0D4D,0006,FF,10,FF,0C,FF,5C,FF,6D,FF,27,FF,10
P,2C95,0D17,0006,FF,10,FF,4A,FF,0C,FF,00,FF,0B,FF,5C
P,2C9B,0D49,0006,FF,6F,FF,27,FF,42,FF,4D,FF,03,FF,F4
W,CFDF
;
; PATCH TO FIX THE OCCASIONAL "RENAME FAILED" ERROR IN
;    THE TEXT EDITOR
;
F, ED/CMD, ALCOR1
P,5A8E,0B64,0006,10,10,06,04,03,10,01,06,00,03,22,01
P,5A94,0A4B,0005,4B,00,01,22,00,4B,12,02,F7,00
C,5A99,04C0,0001,22
W,E591
;
;  EDITOR PATCH TO ADD TRANSLATIONS FOR CHARACTERS
;     NOT GENERATED BY THE TRS80 KEYBOARD  (optional)
;
F, ED/CMD, ALCOR1
P,46A7,0849,0003,03,4D,8A,5A,00,CD
P,1404,0BF9,0006,CD,10,BE,0A,62,4A,46,05,EB,00,21,5A
P,140A,0BD1,0006,A4,31,62,22,7E,02,23,5B,A7,0C,28,10
P,1410,0BC7,0006,1B,0A,B8,4A,28,05,07,00,5E,5A,16,32
P,1416,0BAE,0006,00,22,19,02,23,5D,18,0C,F1,10,4E,0A
P,141C,0BF5,0006,06,4A,00,05,03,00,DD,5A,5E,33,0A,22
P,1422,0C24,0006,DD,02,56,5E,0B,0C,D5,10,13,0A,ED,4A
P,1428,0BBF,0006,B0,05,C3,00,AE,5A,66,34,21,22,BB,02
P,142E,0C42,0006,66,7B,7E,0C,A7,10,28,0A,5C,4A,21,05
P,1434,0BB8,0006,99,00,62,5A,7E,35,A7,22,28,02,43,7D
P,143A,0BB4,0006,B8,0C,28,10,06,0A,11,4A,05,05,00,00
P,1440,0BE3,0006,19,5A,18,36,F3,22,23,02,E5,7C,DD,0C
P,1446,0C08,0006,5E,10,0A,0A,DD,4A,56,05,0B,00,13,5A
P,144C,0C2C,0006,13,37,AF,22,21,02,BB,2F,66,0C,4E,03
P,1452,0A57,0005,23,8A,06,00,00,4D,81,53,ED,32
W, 5284
;
;  PATCH TO INCREASE SPEED OF SCREEN UPDATE  (optional)
;
F, ED/CMD, ALCOR1
P,475A,0860,0003,4A,4D,61,FA,02,CC
```

(C) Copyright ALCOR SYSTEMS 1982

```
P,1457,0B7C,0006,B0,12,E1,F9,01,4A,04,F1,00,01,81,03
P,145D,0BD4,0006,ED,01,B0,00,21,03,C4,0A,66,00,F5,24
P,1463,0BBC,0006,7E,22,23,17,A7,C8,28,00,12,11,4F,C8
P,1469,0BA6,0006,F1,00,81,02,06,01,00,0C,ED,11,B0,C8
P,146F,0C2B,0006,DD,00,5E,66,0A,03,DD,01,56,00,0B,04
P,1475,0B8C,0006,D5,0C,13,11,12,C8,18,00,34,67,F1,02
P,147B,0CB1,0006,18,1F,F2,0C,DD,12,6E,F9,0A,4A,DD,F1
P,1481,0B61,0006,66,01,0B,03,E5,02,23,00,36,03,02,0A
P,1487,0B74,0006,23,00,36,24,3A,22,23,17,36,C8,43,00
P,148D,0C0E,0006,18,11,1F,C8,DD,00,E5,02,E1,01,11,0C
P,1493,0B8C,0006,08,11,00,C8,CD,00,E6,66,67,03,21,01
P,1499,0BAC,0006,B8,00,66,04,11,0C,03,11,00,C8,CD,00
P,149F,0CA9,0006,E6,67,67,02,DD,1E,6E,0C,0A,12,DD,F9
P,14A5,0BF1,0006,66,4A,0B,F1,E5,01,36,03,40,0A,EB,00
P,14AB,0C49,0006,CD,03,F8,0A,67,00,D1,24,DD,22,E5,17
P,14B1,0BDE,0006,E1,C8,CD,00,C7,11,66,C8,C3,00,D2,02
P,14B7,0BC1,0006,62,01,20,0C,3D,11,20,C8,00,00,9F,66
P,14BD,0C22,0006,21,03,3A,01,1D,00,F5,04,3A,0C,3D,11
P,14C3,0C0F,0006,1D,C8,00,00,D6,67,01,02,DD,0A,E5,0C
P,14C9,0C61,0006,C5,4A,D5,61,E5,02,E5,4D,DD,8E,E1,32
P,1804,0567,0001,AF,C9
W,03F0
;
;  PATCH TO ALLOW RETRYS ON DISK ERRORS
;     WHEN A DISK ERROR IS DETECTED, THE MESSAGE:
;        IO ERROR:
;     FOLLOWED BY THE ERROR CODE IS DISPLAYED
;     YOU MAY TYPE "Q" TO ALLOW THE ERROR TO PASS
;     OR ANY OTHER KEY TO RETRY
;
F, ED/CMD, ALCOR1
P,127A,085F,0003,FE,C3,1D,CF,20,66
P,14CF,0CDD,0006,EB,FE,23,1D,7E,CA,A7,7E,28,64,15,F5
P,14D5,0C4D,0006,23,CD,E6,F7,1F,66,4F,FE,06,51,00,28
P,14DB,0C7B,0006,7E,04,FE,F1,3A,C3,20,6E,1C,64,23,F1
P,14E1,09B8,0004,7E,C1,FE,C3,4C,AA,28,64
P,1377,0803,0003,DD,C3,E1,E5,20,66
P,14E5,0CB9,0006,0A,DD,FE,E1,44,CA,28,7B,0C,65,DD,F5
P,14EB,0CE5,0006,36,CD,0E,F7,1B,66,18,FE,1B,51,DD,CA
P,14F1,0C40,0006,36,C9,0E,5E,1A,F1,18,C3,15,65,DD,65
P,14F7,0CCB,0006,36,C5,0E,D5,1C,E5,18,DD,0F,E5,DD,FD
P,14FD,0C80,0006,5E,E5,08,F5,DD,21,56,A0,09,5E,7A,11
P,1503,0C25,0006,B3,0B,28,00,0B,CD,ED,E6,B0,67,3E,F1
P,1509,0C26,0006,0D,CD,12,29,E1,67,D1,CD,C1,49,DD,00
P,150F,0C9E,0006,E1,FD,C9,E1,DD,DD,36,E1,12,E1,FF,D1
P,1515,0693,0002,18,C1,F4,C9
P,0CA0,0C0B,0006,D1,0D,E1,49,D5,4F,5E,20,23,45,56,52
P,0CA6,0AC7,0005,23,52,E3,4F,E5,52,D5,3A,C3,20
```

(C) Copyright ALCOR SYSTEMS 1982

```
P,0CC9,09E6,0004,0E,F1,FE,C3,1A,7F,28,65
W,34E4
;
;  INTERNAL EDITOR PATCH
;
F, ED/CMD, ALCOR1
P,5957,0804,0003,12,4D,F5,51,4A,B3
P,0CAB,0C39,0006,4C,10,5D,02,D1,58,D1,12,E1,F7,C5,22
P,0CB1,0BC4,0006,D5,0A,E5,68,01,20,00,10,01,02,CD,56
P,0CB7,0C07,0006,96,2B,60,42,EB,21,E1,07,D5,10,7A,02
P,0CBD,0BFB,0006,B3,58,28,15,44,02,01,1F,00,E9,01,12
P,0CC3,0C9A,0006,CD,F5,26,65,5E,08,FE,4D,0D,93,28,4C
W,BB63
E
```

(C) Copyright ALCOR SYSTEMS 1982

New Product Announcements

By September, Alcor will be delivering the Alcor Pascal system for the Osborne-1, Apple II (Z-80 softcard equipped) and traditional 8 inch diskette based CP/M systems.

Alcor Pascal on CP/M

Alcor Pascal on CP/M based computers is functionally the same as far as the Pascal language is concerned. There are two main differences:

(1) A library of external functions that are optimized for the CP/M operating system.

(2) The inclusion of the Blaise II text editing system that has many new and powerful features such as:

Hardware Reconfigurable

(a) May be reprogrammed through the use of text setup files to understand smart or dumb terminals and use their features to speed up execution of the text editor.

(b) Keyboard layout of the editor control keys may be specified in the setup file. Other popular keyboard layouts for word processors such as wordstar may be mimicked.

Powerful Macro Language

The macro language will allow macro commands to be built from primitive or other macro commands. They may be placed in the setup file or defined during the edit. The combination of commands for word processing is infinite. Commands to insert printer control codes are a simple example. Setup files may be generated that give the editor a Basic, Pascal or any other language personality desired.

Word Processing Additions

(a) Justify and fill.

(b) Block text functions.

(C) Copyright ALCOR SYSTEMS 1982

(d) Extract text to file.

(e) Delete word.

(f) Move cursor by word.

(g) Center line.

(h) More than one file may be edited at a time

Osborne-1 Version

The Osborne-1 version is supplied in the standard Osborne-1 single density, single sided CP/M format. Alcor Pascal is the first Pascal system that will truly impact program development on the Osborne-1. The TRS80 versions of Alcor Pascal are quickly becoming the industry standard for Pascal programmers even with single density disk systems as proven by many TRS80 Model I owners. Most other language systems of the scope and complexity as Pascal are hardly manageable on single density drives because of the many intermediate files generated in the course of a compile. A fine example is Pascal MT+ by Digital Research. It simply won't execute without performing the floppy shuffle on double density 5 inch disks. Performance is less than desirable on 8 inch single density disk based systems.

Alcor Pascal on the Apple

Recent estimates show that there are over 50,000 Apple II systems with the Z-80 softcard running the popular CP/M operating system. Apple users have bought the softcard in order to access the large base of CP/M software. Unfortunately, a common problem for the average Apple owner is that of restricted diskette storage. Since the average user doesn't own a winchester drive just yet, they have found an absence of an acceptable Pascal system just like the Osborne-1 users. They will now have a complete and powerful Pascal language development system which is a perfect mate for their Z-card equipped Apple II. Alcor Pascal will be delivered on the standard CP/M 5 -1/4 Apple II formated diskettes.

(C) Copyright ALCOR SYSTEMS 1982

Traditional CP/M Versions

The standard CP/M format for Alcor Pascal is 8 inch, single density, single sided, soft sectored for such systems as the California Computer System, TRS Model II and TRS Model 16. They are functionally the same as the Osborne-1 and Apple II versions.

Notice to Software Developers

Compatibility Between Versions

All CP/M and TRS80 versions are object code compatible. This means that the compiler output object code may be transferred between any of these versions and re-linked with the proper run-time support by the target machine linking loader for execution. The only restrictions for direct compatibility is that any machine dependent routines not be included in the compiled object. (Such routines as graphics) If they are used, the equivalent routine on the target machine must be compiled and/or linked to by the linking loader. This means that programs for sale can be developed and marketed on one machine such as the TRS80, then ported and sold on other Alcor supported machines, thus expanding your market base. As always, there is absolutely no royalty fees required by Alcor Systems. Such are the advantages of program development in an efficient high level language.

Random News Notes

Pascal Newsletter

This is the first Pascal newsletter. It appears at this time that it will probably be published 4 times a year. Registered owners will receive four issues regardless of the dating of their registration agreement.

Alcor is looking for individual contributions for the Newsletter in the form of donated programs, programming tips, etc. There is NO compensation involved at this time. Alcor will give full credit to the author. We can NOT guarantee acceptance and timing for publication of submitted information.

(C) Copyright ALCOR SYSTEMS 1982

The Next Alcor Pascal Version Upgrade

Alcor Pascal 2.0 will be released around the fourth quarter of 1982. Just a few of the planned additions are built in Random Access files and an Include feature. The Include feature will allow libraries of declarations or routines to be included in the source program during a compile by declaring the filename in a compiler option comment. ALL registered Pascal owners will receive this upgrade regardless of the dating of their service contract. There will be a nominal charge for diskette replacement.

Alcors' Policy On Documentation Upgrades

Documentation changes will be handled through the newsletter. Any errors found will be noted in an errata section of the newsletter. Any additional sheets added to the documents will also be included in the newsletter.

Clarification of Runtime Licensing Agreement

There has been a little confusion about what a user may distribute for resale. In plain english, if you develope programs with the Alcor Pascal compiler and build /CMD file programs on the TRS80, you may sell these programs freely. Alcor does not require any fees for the runtime support that is always included in /CMD files. (12,500 lines of Alcor assembler code) You may NOT distribute the Run, Pascal, Trslib, or any other Alcor programs or files beside the final user /CMD file.

Compiler Errata

There exists a non-patched bug in the compiler. It occurs when a READ statement with a subrange variable is used as an argument and the lower bound of the subrange type is not 0 . An example is as follows.

```
PROGRAM TEST;
TYPE
  SMALLINT = -10 .. 200;
VAR
  NUMBER   : SMALLINT;
BEGIN
  READLN(NUMBER);
  WRITELN(NUMBER);
END.
```

(C) Copyright ALCOR SYSTEMS 1982

Upon writing the NUMBER value, the output will be incorrect. The READ in value of NUMBER will be off by the lower bound. In this case a value entered by the user of 30 would be READ in as 20. A simple way of getting around the problem is to declare an INTEGER variable and READ into it, then assign it's value to the subrange variable. The following is an example:

```
PROGRAM TEST;
TYPE
  SMALLINT = -10..200;
VAR
  NUMBER    : SMALLINT;
  INTNUMBER: INTEGER;
BEGIN
  READLN(INTNUMBER);
  NUMBER:=INTNUMBER;
  WRITELN(NUMBER);
END.
```

Documentation Errata
(First and Second Printing)

Certain ommisions or errors in the documentation package have been noted. They are corrected as follows:

(1) When executing under LDOS, the Pascal RUN command must be renamed to prevent conflicts between the Pascal and LDOS RUN commands. If you try to RUN a Pascal object file with the LDOS RUN command you will probably get the message: Load File Format Error from LDOS. Simply RENAME the Pascal RUN command to anything desired.

(2) When executing under TRSDOS(M III) or LDOS, the Alcor Patch program should be renamed to prevent conflicts as in (1).

(3) There are no passwords on any Alcor supplied diskettes. Where passwords are required simply use a blank character terminated by the enter key.

(4) In the revised 1.2A documentation package, page 23 of the system manual shows an example program segment. In the declaration for the external procedure SET$ACNM, the parameter FILEID should be passed by reference. The correct declaration is as follows:

(C) Copyright ALCOR SYSTEMS 1982

```
PROCEDURE SET$ACNM(VAR F:TEXT; VAR FNAME:FILENAME;
          LEN:INTEGER;  VAR FILEID:ALPHA); EXTERNAL;
```

(5) The database program as supplied on diskette requires significant memory to compile. You may have to use the overlayed compiler PASCALB if you have any high memory drivers loaded. To run the database program requires a stack specification of 15k. If using the RUN command 15k must be specified on the command line as: RUN DATABASE 15K When building a command file with this object, specify 15k as the stack specifier in the build command.

(6) Tutorial program variable names don't necessarily match the supplied source on diskette. (Alcor Pascal uses 8 character unique variable names)

(7) On page 42 of the REFERENCE manual, in the program example's main body, the line:

```
tran@.link := translist;
```

should read:

```
trans@.link:=translist;
```

(8) Please note that Pascal only (p.99 of reference manual) scans to column 72 of the source program. Extending comments or statements over col 72 can cause unpredictable results and cause a fatal error to the compiler.

(9) Error code 401 is not documented. It means that an open comment was encountered in a comment. Nested comment statements are not allowed.

(10) Error code 403 is not documented. Too many procedure nesting levels. The nesting limit is 16.

(11) Error code 404 is not documented. Array bounds must be scalar.

(12) On page 14 of the Tutorial manual there is a typo in listing 4.2. The last writeln statement reads:

```
WRITELN(OUTPUT,' Business I.D.    =',ID);
```

(C) Copyright ALCOR SYSTEMS 1982

Should read:

```
WRITELN(OUTPUT,' Business I.D.    =',ssnumber);
```

(13) The Tutorial program T92/PCL doesn't match the listing 9.2 in the manual. The manual is correct.

(14) Page 59 of the Tutorial. Set inequality should read:

setl <> set2	Set inequality- If all members of the first set are in the second set, and all members of the second set are in the first set: Then return false.

Answers To Some Of The Commonly Asked Questions

Question
Is Alcors pcode compitable with U.C.S.D. ?

Answer
The answer is unequivocally no. Alcors pcode design is totally our own. This is not to thwart anyones desire for compatibility, but was simply necessary for efficient design. Alcors' p-code design is one of the primary reasons that a complete and efficient implementaion of Pascal was possible on a 48k equipped computer such as the TRS80 Model I or III.

Question
Will programs written in U.C.S.D. Pascal execute when recompiled with Alcor Pascal ?

Answer
U.C.S.D. has language features that are not common to Standard Pascal as defined by Jensen & Wirth. If the non-standard language features of U.C.S.D. are avoided, then source programs may be recompiled on Alcor pascal for execution. It must be warned that such extensions as string manipulation are not defined in the Standard. However, Alcor Pascal handles string functions also, but not necessarily in the same manner as

(C) Copyright ALCOR SYSTEMS 1982

U.C.S.D. . For a complete description of Standard Pascal, see the "Pascal User Manual And Report" by Katheleen Jensen and Niklaus Wirth, published by Springer-Verlag, New York. Nicklaus Wirth is the person who designed and implemented the first Pascal compiler in 1971.

Question
Does Alcor Pascal understand my lowercase modification on the Model I ?

Answer
Yes, if your operating system does. Pascal only makes operating system calls to perform I/O to the screen and keyboard. If you have a lowercase modification and load the appropriate high memory driver before invoking the editor, it will understand lowercase.

Question
Will the Pascal system work with my TRS80 Model I clock speed up modification ?

Answer
In most cases it will, however we can't guarantee it for all combinations. The Pascal system only makes standard operating system calls to perform disk related functions. There is no timing dependent code in the Pascal system, therefore if your clock speed up modification and operating system work together CORRECTLY, Alcor Pascal should also. We have had a few complaints about home brew mods that didn't always work correctly with particular operating system combinations. Since we don't have all possible hardware combinations at Alcor, it is impossible for us to GUARANTEE anything about clock speed up mods.

Question
Does Alcor Pascal support 8 inch foppies and Winchester type drives on the TRS80 ?

Answer
Yes it does. As delivered the Pascal system will work properly with 8 inch floppies and hard disk units as long as they are already properly integrated into the operating system environment with the required software drivers. The Pascal system treats these devices as logical devices and makes operating system calls to perform I/O. In simple terms, if other user application programs work correctly, Alcor Pascal should.

(C) Copyright ALCOR SYSTEMS 1982

Question
Is Alcor Pascal supplied in CP/M format for TRS80 systems that have the Omikron CP/M adapter. Also what about the other CP/M adapters ?

Answer
Not at the present time. The only CP/M formats supported at this time are Osborne-1, Apple II (Z-80 softcard) and traditional 8 inch soft sectored formats. We are looking into the possibility of supporting these other formats if we get enough requests.

Question
What about other TRSDOS like operating systems for the TRS80 ? Do you support any others than TRSDOS, LDOS, DOSPLUS and NEWDOS ?

Answer
No, not at this time. We don't have any plans for doing so unless we get enough requests.

Question
Where can I call for technical assistance ?

Answer
You should only call our technical assistance number after completely reading the manuals. In most cases the information requested is clearly described in the manuals. Many people exclaim that I haven't had time to read them yet so I just called because it was easier. If you truly have a problem and can't find the answer after thoroughly reading the manuals, call our technical assistance people at 214-494-1316. This is our computer facility in Garland,Texas. Normal hours are 9-12, 2-5 P.M. Monday thru Friday.

Getting Command Line Parameters From Pascal On The TRS80

When writing an application program you may add a touch of class to the finished program by allowing users to specify parameters on the command line when bidding the program. This is typical of most TRSDOS commands. When a user types the /CMD filename, the operating system loads and executes the appropriate program, then stores the command line that invoked the program in a predetermined place in memory. Once the program is invoked, the program may retreive the contents of this memory location and use it to determine flow of control in the program.

The command line arguments are stored beginning at hex address 4318 on the model I and 4225 on the model III. A pointer

(C) Copyright ALCOR SYSTEMS 1982

variable may be declared as a pointer to an array of CHAR. Through the use of the type transfer operator, the pointer variable may be assigned the appropriate value. The pointer variable may then be used to access the buffer containing the command line.

Example:

```
PROGRAM GETCOMMANDLINE;
TYPE
   POINTER = @BUFFER;
   BUFFER = ARRAY(.1..64.)OF CHAR;
VAR
   BUFPTR : POINTER;
BEGIN
  BUFPTR::INTEGER := #4318;
  FOR I := 1 TO 64 DO WRITE(BUFPTR@(.I.));
END.
```

An excellent program example which uses this technique is a print command. The following print program illustrates command line parsing.

```
(*$NO INOUT*)
PROGRAM PRINT;

TYPE  ALPHA=PACKED ARRAY(.1..8.) OF CHAR;
      NAME=PACKED ARRAY[1..64] OF CHAR;
      POINTER=^NAME;

VAR   CH :CHAR;
      LEN,START,STOP :INTEGER;
      FILENAME,PRINTER :TEXT;
      ID :ALPHA;
      FN :NAME;
      BUFPTR :POINTER;

PROCEDURE SET$ACNM(VAR FNAME:TEXT;VAR FN:NAME;LEN:INTEGER;
                   VAR ID:ALPHA); EXTERNAL;

BEGIN
  (* DIRECT OUTPUT TO THE LINE PRINTER *)
  FN[1]:=':';  FN[2]:='L';
  ID:='PRINTER ';
  SET$ACNM(PRINTER,FN,2,ID);
  REWRITE(PRINTER);
```

(C) Copyright ALCOR SYSTEMS 1982

```
  (* GET FILENAME FROM COMMAND LINE
     EXAMPLE COMMAND LINE -->  PRINT FILE/EXT
     GIVEN ABOVE EXAMPLE: THIS SECTION PUTS FILE/EXT INTO FN *)
  BUFPTR::INTEGER := #4225; (* BUFPTR POINTS TO COMMAND LINE *)
  START:=1;
  (* FIND THE FIRST BLANK IN THE COMMAND LINE *)
  WHILE BUFPTR^[START] <> ' ' DO START:=START+1;
  (* THROW AWAY ANY EXTRA BLANKS BEFORE FILE NAME *)
  WHILE BUFPTR^[START] = ' ' AND START<64 DO
      START:=START+1;
  IF START<64 THEN
    BEGIN
    STOP:=START;
    (* PUT FILE NAME IN ARRAY FN *)
    WHILE STOP<64 AND BUFPTR^[STOP] <>' ' AND BUFPTR^[STOP]<>'#0D' DO
      BEGIN
      FN[STOP-START+1]:=BUFPTR^[STOP];
      STOP:=STOP+1;
      END;
    LEN:=STOP-START;

    (* ASSIGN INPUT FROM FILE SPECIFIED ON COMMAND LINE *)
    ID:='FILENAME';
    SET$ACNM(FILENAME,FN,LEN,ID);
    RESET(FILENAME);

    (* PRINT THE FILE *)
    WHILE NOT EOF(FILENAME) DO
      BEGIN
      WHILE NOT EOLN(FILENAME) DO
        BEGIN
        READ(FILENAME,CH);
        WRITE(PRINTER,CH);
        END;
      READLN(FILENAME);
      WRITELN(PRINTER);
      END;
    END;
END.
```

(C) Copyright ALCOR SYSTEMS 1982

Details on Alcors Object Code Format

Some of the following format information has been extracted from the Advanced Development Package for those people who don't own a copy.

D. Object Format

Alcor Systems uses its own format for object code. The main reason for this is that support for many of the features of Alcor Pascal are not present in existing object formats. For example, Alcor Pascal supports common blocks for statically allocated variable storage and the object format must in turn allow for this.

The pseudo-code (pcode) generated by the compiler is address independent. That is, it contains no absolute memory addresses and can execute without change when loaded anywhere in memory. All branching and calling of procedures within the pcode is done relative to the current program counter. Since procedures are compiled into separate modules, calculation of these relative addresses must be done when the code is loaded. The object format supports external references that are program counter relative.

The object code is tagged hexadecimal and is emitted in a line oriented stream that is compatible with a pascal text file. In particular, the object code is character oriented and contains only printable ASCII characters. This allows the object to be manipulated by text editors or transmitted over modems. This is not possible with bit oriented formats.

Each item in the object file begins with a tag which is usually an upper case letter. The tag defines the type of item and the number and size of the fields to follow. Tags are followed by one or more fields that specify the information to be loaded. Three types of fields exist. Bytes are specified with a two character hexadecimal number. Words consist of a four character hexadecimal number with the most significant byte first. Labels consist of eight character names that are the names of external symbols, common blocks, etc...

Following is a table which lists all the tags used in an object file. All tags are followed by one to three fields of information, each field being either a byte, word, or label. The meaning of each tag is also shown.

(C) Copyright ALCOR SYSTEMS 1982

Tag	Field1	Field2	Field3	Meaning
A	byte			Absolute(non-relocatable byte)
E				End of module
F				End of line
G	word	label		Definition of external symbol
I	word	label		External reference declaration
J	label			Module name
K	word			Reference to external symbol
M	word	word	label	Definition of common block
N	word			Reference to common
O	word	word		Overlay definition
P	word			Code (PC) origin
Q	word			Relative reference to external
W	word			Relocatable word
X	word			Absolute word
Y	word			Entry point definition
:				End of file

E. Splitting Object Modules

Since object files are in ASCII format, they may be edited with a text editor or used as input to a Pascal program. The following is a list of the pure pcode output (/OBJ) file for the LOOP procedure (Page 11 Advanced Development Pkg. manual) in the mixed mode operation example. Following it is a listing of the object (/COD) file which results from running the pcode object through the code generator. As you can see, the code generation has caused approximately a factor of 2 increase in size.

Pure Pcode Listing (/OBJ)

```
JLOOP    P0000G0000LOOP    A01X0000A38A02A03X0001A15A04A10A04A03X2710A07F
A15A06A2BA4EX0000A03X0001A03X0002A22A03X0003A22A03X0004A22A03X0005A22A03F
X0006A22A03X0007A22A03X0008A22A03X0009A22A03X000AA22A03X000BA22A03X000CF
A22A03X000DA22A03X000EA22A03X000FA22A15A02A10A04A30X0004A10A06A27A21AB9F
P0014X0047P005DA3AP0001X0006E
:
```

(C) Copyright ALCOR SYSTEMS 1982

' Native Code Listing (/COD)

```
JLOOP    G0003LOOP    AC1AEBAE9A01X0006A38A02A55A21X0001ADDA75A04ADDA74F
A05ADDA6EA04ADDA66A05AE5A21X2710AE5ADDA75A06ADDA74A07AC1AE1A78AACAEDA42F
A28A09A47A3FA1FAA8A07AE6A01A18A02A3EA00AA7AC2X0000A21X0001AE5A21X0002AC1F
A09AE5A21X0003AC1A09AE5A21X0004AC1A09AE5A21X0005AC1A09AE5A21X0006AC1A09F
AE5A21X0007AC1A09AE5A21X0008AC1A09AE5A21X0009AC1A09AE5A21X000AAC1A09AE5F
A21X000BAC1A09AE5A21X000CAC1A09AE5A21X000DAC1A09AE5A21X000EAC1A09AE5A21F
X000FAC1A09ADDA75A02ADDA74A03ADDA6EA04ADDA66A05AE5ADDAE5AE1A01X0004A09F
A4EA23A46A03A70A2BA71ADDA6EA06ADDA66A07AC1AAFAEDA42A20A01A3CAA7ACAW003AF
F0038W00BDF00BDACDW0000A3AF
:
```

Each module in an object file begins with the module name. Therefore, it is possible to split a file containing several modules into several files, each containing one module. This is an alternate method of segmenting large programs where it is desired to perform code generation on only selected parts.

There are two ways to split the object modules. One is to text edit them. The other more desirable method is to write a Pascal program to split them. A simple program may be written to read the pcode (/OBJ) file. Each time a module is encountered, open a file of the same name as the module and write the module to that file. Once all the modules are separated into different files, selected modules may be input to the code generator and translated to native machine instructions. The linking loader may then be used to link the individual modules and build an executable command (/CMD) file.

Management Of Object Modules

One of the more useful features of Alcor Pascal object code is its ease of manipulation with normal programs or programming tools. Since it is plain ASCII text, and is relocatable, large programs may be broken up into the various object modules that correspond to source procedures and functions. The utility of this may not be obvious, but is very important to the serious programmer. When writing large Pascal programs that are well modularized, (broken down to many procedures and functions) it is convenient to only recompile the parts of the program that are necessary during the development phase. There are two different ways of performing this. The first method is to always

(C) Copyright ALCOR SYSTEMS 1982

compile procedures and functions separately using the compiler nullbody option as described in the Reference manual, and then to link the desired modules together with the linking loader to produce a final program. Indeed, that is how all development work on the Pascal compiler is done at Alcor Systems. However, when first writing a large program, it is usually inconvenient to separately compile and link all modules in the beginning, because the time required to perform the separate compiles and link edit is greater than the time saved by not compiling the entire program at once. The typical mode is that the program starts out as one large program with many procedures and functions. The entire program is compiled and one object file is produced. Usually after the program reaches 500 lines or more, the time required to do a complete compile is significant. At this point, a good practice is to split the compiled object file such that all sections of the object file that represent procedures and functions reside in different files. A COPY of the source program is then modified by removing all the actual procedure and function statements, and by leaving external procedure or function declarations in their place. (see the Alcor Pascal Reference Manual) At this point the modified source program may then be compiled much faster because the resultant program will be shorter. After the program has been successfully recompiled, it may then be linked with the linking loader to the previously split object files to satisfy the linking loaders external references. From then on, only the main program and new procedures or functions need to be recompiled. The following Pascal program will split an object file into different object files that have the object code for different procedures and functions. It does so by recognizing the previously described object code tag field information. This is a handy little program that will serve as an excellent utility for the advanced Pascal programmer.

```
(*$NO INOUT*)
PROGRAM SPLIT_OBJECT;
TYPE    FILENAME = ARRAY(.1..72.) OF CHAR;
        ALPHA = ARRAY(.1..8.) OF CHAR;

VAR     INPUT,OUTPUT,OBJFILE,SPLITOBJ     :TEXT;
        EXT                               :PACKED ARRAY(.1..5.) OF CHAR;
        MODULENAME,ID                     :ALPHA;
        CH,DRIVE                          :CHAR;
        FIRSTIME,FIRSTCHAR                :BOOLEAN;
        FN                                :FILENAME;
        I,LN                              :INTEGER;
```

(C) Copyright ALCOR SYSTEMS 1982

```
PROCEDURE SET$ACNM(VAR F :TEXT; VAR FN :FILENAME; LEN :INTEGER;
                   VAR ID :ALPHA); EXTERNAL;
FUNCTION CONC(S1,S2 :STRING): STRING; EXTERNAL;
FUNCTION LEN(S :STRING): INTEGER; EXTERNAL;
PROCEDURE GOTOXY(X,Y:INTEGER); EXTERNAL;
PROCEDURE CLEARSCREEN; EXTERNAL;
PROCEDURE NOBLANK(FLAG:BOOLEAN); EXTERNAL;

BEGIN
  (*SET INPUT AND OUTPUT TO THE TERMINAL*)
  FN(.1.):=':'; FN(.2.):='C';
  NOBLANK(TRUE);
  ID:='INPUT   '; SET$ACNM(INPUT,FN,2,ID); RESET(INPUT);
  ID:='OUTPUT  '; SET$ACNM(OUTPUT,FN,2,ID); REWRITE(OUTPUT);
  CLEARSCREEN;
  GOTOXY(0,0);
  (* PROMPT FOR NEEDED INFORMATION*)
  WRITELN(OUTPUT,
  '          $$ ALCOR PASCAL SPLIT OBJECT UTILITY $$');
  WRITELN(OUTPUT);
  WRITELN(OUTPUT,'      ENTER OBJECT FILENAME TO BE SPLIT: ');
  WRITELN(OUTPUT,'ENTER DISK DRIVE FOR SPLIT OBJECT FILES: ');
  GOTOXY(41,2);
  READLN(INPUT,FN);
  GOTOXY(41,3);
  READLN(INPUT,DRIVE);
  GOTOXY(0,4);
  WRITELN(OUTPUT,'  * SPLIT IN PROGRESS *');
  WRITELN(OUTPUT,'  Modulename        Filename');
  (*OPEN OBJECT FILE TO BE SPLIT*)
  ID:='OBJFILE ';
  LN:=72; WHILE FN(.LN.)=' ' DO LN:=LN-1;
  SET$ACNM(OBJFILE,FN,LN,ID);
  RESET(OBJFILE);
  (*SPLIT /OBJ FILE INTO MULTIPLE FILES*)
  ID:='SPLITOBJ';
  EXT:='/OBJ:';
  WHILE NOT EOF(OBJFILE) DO
    BEGIN
    READ(OBJFILE,CH);
    (*IF AT BEGNNING OF MODULE*)
    IF CH='J' THEN
       BEGIN
       CLOSE(SPLITOBJ);
       READ(OBJFILE,MODULENAME);
       LN:=0;
       FOR I:=1 TO 8 DO
          IF (MODULENAME(.I.)<>' ')AND(MODULENAME(.I.)<>'$')
             AND(MODULENAME(.I.)<>'_') THEN
             BEGIN
             LN:=LN+1;
             FN(.LN.):=MODULENAME(.I.);
             END;
```

(C) Copyright ALCOR SYSTEMS 1982

```
      FOR I:=1 TO 5 DO
         FN(.LN+I.):=EXT(.I.);
      FN(.LN+6.):=DRIVE;
      LN:=LN+6;
      SET$ACNM(SPLITOBJ,FN,LN,ID);
      REWRITE(SPLITOBJ);
      WRITE(SPLITOBJ,'J');
      WRITE(OUTPUT,'  ',MODULENAME,'  =====>  ');
      FOR I:=1 TO LN DO WRITE(OUTPUT,FN(.I.));
      WRITELN(OUTPUT);
      WRITE(SPLITOBJ,MODULENAME);
      END
   ELSE
      WRITE(SPLITOBJ,CH);
   WHILE NOT EOLN(OBJFILE) DO
      BEGIN
      READ(OBJFILE,CH);
      WRITE(SPLITOBJ,CH);
      END;
   IF NOT EOF(OBJFILE) THEN
      READLN(OBJFILE);
   WRITELN(SPLITOBJ);
   END;
 WRITELN(OUTPUT,'  * SPLIT COMPLETE *');
 END.          (*SPLIT_OBJ*)
```

Building TRSDOS CMD Files With Assembly Language Subroutines

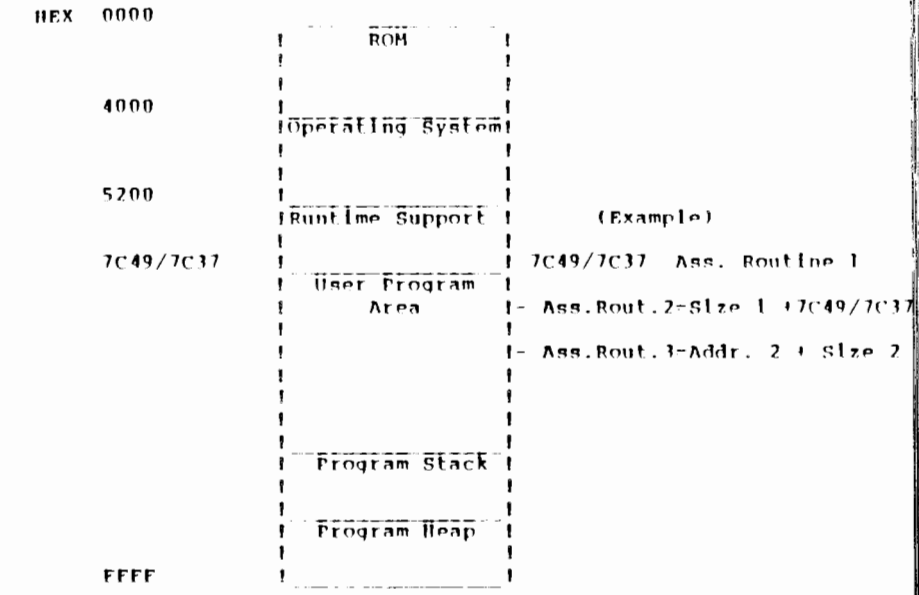
An important requirement for any serious language system is the ability to build CMD files that include assembly languange subroutines. Alcor Pascal is one of the most efficient high level language systems available today. A vast majority of programming tasks may be handled with ease in Pascal, while at the same time producing programs that are highly efficient. However there always exist tasks which are better suited for assembly language.

Radio Shack Assembly Language

The R.S. assembler produces non relocatable code that must be origined at the time of assembly. The technique for building CMD files with code produced by it and Pascal code will briefly be described. There are many variations on how to handle subroutine entry points, but the simplest will be shown.

(C) Copyright ALCOR SYSTEMS 1982

TRS80
The Memory Map



The user program area starts at 7C49 on the Model III and 7C37 on the Model I. . This is the origin that user assembly programs should start at. The basic steps are:

(1) Origin all assembly language routines at 7C49/7C37 and assemble using the RS or similar assembler.

(2) Compute the length of the assembler load modules by subtracting the beginning instruction address from the ending instructions address and adding the length of the last instruction in bytes. This will be the size in bytes of each separately assembled module.

(3) Reassemble routines 2..N, re-origining each module to the proper load address as computed from a 7C49/7C37 starting point. Example: #7C49/7C37 + size of assembler routine 1 = origin of assembler routine 2, etc...

(C) Copyright ALCOR SYSTEMS 1982

(4) Create a text file with an editor for each assembly language module that contains:

PxxxxE

Where xxxx is the size (HEX) in bytes of this particular assembly as computed from step 2. Create one file for each routine. They may be named as desired. They are used as dummy routines by the linking loader to reserve user program area for the assembler routines.

(5) Use the normal system LOAD command to load all desired assembly language routines into memory.

(6) Invoke LINKLOAD. Use the LOAD command to load each one of the dummy modules created in step 4. Load dummy modules in the desired memory order from #7C49/7C37 to #FFFF. Normally Pascal programs will start loading at #7C49/7C37 to #FFFF in sequence. Loading the dumy modules will reserve areas of memory for non Pascal programs starting at #7C49/7C37. This will force Pascal programs to load after the assembly language routines.

(7) Load Pascal programs and procedures, build CMD file.

Note - In the Pascal source program that calls the users assembly language routines, insert constants for the entry point of every routine.

The following assembly language routine plots an array. It may be assembled with the R.S. editor/assembler. Once assembled the output file should be transferred from TRSDOS 2.3B to 2.3 on the Model I as Alcor Pascal is not compatible with 2.3B unless the Pascal system is patched with the supplied patch file labeled NEWDOS/PAT on the Model I. Thats right folks, TRSDOS 2.3B on the Model I behaves like NEWDOS does. The following Pascal program simply illustrates the linkage between Pascal and assembler routines.

```
00100    ;
00110    ;  Sample assembly language program for linkage
00120    ;     to pascal.  This subroutine plots a series
00130    ;     of points on the screen from an array.
00140    ;
```

(C) Copyright ALCOR SYSTEMS 1982

```
00150           ORG     07C49H  ; Load point
00160   ;
00170   ;  Subroutine to plot a series of points on the screen.
00180   ;  INPUTS:  HL contains the address of the array of
00190   ;             points.  Each point is represented by two
00200   ;             bytes containing the x and y coordinates.
00210   ;           DE contains the count of the number of points
00220   ;             to be plotted.
00230   ;
00240   PLOT    LD      A,D     ; CHECK FOR ZERO COUNT
00250           OR      E
00260           RET     Z       ; RETURN ON ZERO COUNT
00270           LD      B,(HL)  ; GET X COORDINATE
00280           INC     HL
00290           LD      C,(HL)  ; GET Y COORDINATE
00300           INC     HL
00310           PUSH    HL      ; SAVE POINTER TO POINTS
00320           CALL    POINT   ; DETERMINE SCREEN LOCATION
00330           OR      (HL)    ; MERGE WITH PREVIOUS CONTENTS
00340           OR      080H    ; GUARANTEE GRAPHICS MODE
00350           AND     0BFH    ; MASK OUT SPECIAL CHARACTERS
00360           LD      (HL),A  ; STORE INTO SCREEN
00370           POP     HL      ; RECOVER POINTER TO DATA
00380           DEC     DE      ; DECREMENT COUNT
00390           JR      PLOT    ; AND REPEAT
00400   ;
00410   ;
00420   ;  Subroutine to compute the location of a point
00430   ;    on the screen.
00440   ;
00450   ;  INPUTS:  B contains the x coordinate
00460   ;           C contains the y coordinate
00470   ;  OUTPUTS: HL contains the memory address within
00480   ;              the screen memory
00490   ;           A contains a bit mask with a bit set in
00500   ;             the position of the addressed point
00510   ;
00520   ;  The memory address is:
00530   ;       (y div 3)*64 + (x div 2) + screenorigin
00540   ;  The bit number is:
00550   ;       (y mod 3)*2 + (x mod 2)
00560   ;
00570   POINT   LD      A,C
00580           AND     03FH    ; MASK TO 0..63 range
00590           CP      030H    ; GREATER THAN 47?
00600           JP      C,PT001
00610           LD      A,02FH  ; SET TO 47 IF > 47
00620   PT001   LD      C,0FFH  ; DIVIDE Y BY 3
```

(C) Copyright ALCOR SYSTEMS 1982

```
00630   PT002   INC     C       ; SUBTRACT 3 UNTIL NEGATIVE
00640           SUB     3
00650           JR      NC,PT002
00660           ADD     A,3     ; C IS Y DIV 3
00670           LD      L,B     ; SAVE X COORDINATE
00680           LD      B,A     ; B IS REMAINDER (Y MOD 3)
00690           LD      A,L     ; GET X COORDINATE
00700           ADD     A,A     ; A IS X*2
00710           LD      L,A
00720           LD      H,C     ; HL IS (Y/3*256+X*2)
00730           SRL     H       ; DIVIDE HL BY 4
00740           RR      L
00750           SRL     H
00760           RR      L       ; HL IS (64*Y/3) + X/2
00770           RLA             ; LSB OF A IS (X MOD 2)
00780           AND     1       ; MASK LSB
00790           LD      C,A
00800           LD      A,H
00810           AND     3
00820           OR      03CH    ; UPPER BYTE OF SCREEN ADDRESS
00830           LD      H,A     ; HL IS BYTE ADDRESS
00840           LD      A,B     ; GET Y MOD 3
00850           RLCA            ; MULTIPLY BY TWO
00860           ADD     A,C     ; ADD X MOD 2
00870           AND     7
00880           LD      B,A     ; B IS BIT NUMBER
00890           INC     B       ; SHIFT COUNT
00900           XOR     A
00910           SCF             ; SHIFT IN CARRY
00920   PT003   RLA
00930           DJNZ    PT003
00940           RET
00950           END
```

The 1 line dumy loader file used to reserve user are is as follows:

```
        P004DE
```

```
{*$NO INOUT*}
PROGRAM DIAGONALS;

(*  Sample program to plot diagonal lines on the screen *)
(*  This program calls an assembly language subroutine  *)
(*  to display a set of points contained in an array    *)
```

(C) Copyright ALCOR SYSTEMS 1982

```
CONST
   plot = #7C49;  (* load point of plotting routine *)
   arsize = 95;   (* size of array of points *)
TYPE
   byte = 0..255;
   point = packed record
      x : byte;
      y : byte;
      end;
   plotarray = packed array[0..arsize] of point;

VAR
   line : plotarray;
   A    : byte;
   BC, DE, HL, IX, IY : INTEGER;

PROCEDURE CALL$(ADDRESS : INTEGER; VAR A:BYTE;
      VAR BC, DE, HL, IX, IY : INTEGER); EXTERNAL;
PROCEDURE CLEARGRAPHICS; EXTERNAL;

BEGIN
   CLEARGRAPHICS;
   FOR I := 0 TO 47 DO BEGIN  (* LEFT TO RIGHT DIAGONAL *)
      LINE[I].X := I+I;
      LINE[I].Y := I;
      LINE[I+48].X := I+I+1;
      LINE[I+48].Y := I;
      END;
   HL := LOCATION(LINE);
   DE := 96;
   CALL$(PLOT,A,BC,DE,HL,IX,IY);
   FOR I := 0 TO 47 DO BEGIN   ( LOW RIGHT TO UPPER LEFT )
      LINE[I].X := I+I;
      LINE[I].Y := 47-I;
      LINE[I+48].X := I+I+1;
      LINE[I+48].Y := 47-I;
      END;
   HL := LOCATION(LINE);
   DE := 96;
   CALL$(PLOT,A,BC,DE,HL,IX,IY);
END.
```

(C) Copyright ALCOR SYSTEMS 1982

Loading Short Assembly Language Routines Into Pascal Arrays And Executing

While pascal is a versatile language, sometimes there's an operation that pascal can't do. In this case, it is necessary to go outside of the language. There are many ways to do this, and that is what this column is about. We'll discuss some of the ways that Alcor Pascal provides for the knowledgeable programmer to perform feats of great daring.

Using tricks in pascal is like performing stunts in an airplane: You can do it, but you had better know what you're doing. Once you fool the system, it no longer protects you. Alcor pascal provides mechanisms to escape from pascal's restrictions. Among these are the type transfer operator, and the ability to call assembly language routines that are stored in Read only memory or elsewhere.

In this column, we will show you how to execute assembly language routines entirely within pascal. The example that we will use provides bit manipulation functions for your pascal programs. The programs listed on the next page demonstrate this capability. The machine language code is loaded into a pascal array and executed by the program. It is not necessary to leave pascal or invoke other programs to do this. Let's see how the magic happens.

Alcor pascal allows characters within strings to be specified by their ascii codes. This is done by using a number sign (#) followed by a two digit hexadecimal (base 16) number. In this way, we can insert any sequence of bytes into a string. In particular, the contents of a string can represent a program in the native code of the processor on which we are executing.

The actual machine instructions are encoded as hexadecimal numbers and loaded into an array with an assignment statement. This subroutine can then be called from pascal. To call the routine we will use the library routine CALL$. CALL$ requires the address of the code to be executed. This can be determined in pascal by use of the built in function LOCATION. LOCATION returns the address of any variable. If we perform a call to the location of the array, the processor will execute the contents of the array as instructions. All that is necessary is to load the registers (via CALL$) with the proper contents and execute a subroutine contained within the array. The subroutine returns to pascal by executing a return instruction.

(C) Copyright ALCOR SYSTEMS 1982

The assembly language source for the logical AND routine is listed below. Similar routines can be written for other purposes. The array that you use can be any size, it need only be large enough to contain the code that you wish to execute. This particular routine was assembled with Alcors' Multiprocessor assembler although any assembler may be used such as Radio Shacks Editor Assembler combination.

```
MULTIPROCESSOR ASSEMBLER   VERSION: 1.3    18:26:17 04/08/82      PAGE 1

 1                ;
 2                ;    Z80/8080 subroutine to generate the logical
 3                ;         AND of two 16 bit numbers
 4                ;  INPUTS:
 5                ;      DE, HL - operands
 6                ;  OUTPUTS:
 7                ;      HL     - result
 8                ;
 9 0000 7C              MOV   A,H
10 0001 A2              ANA   D
11 0002 67              MOV   H,A
12 0003 7D              MOV   A,L
13 0004 A3              ANA   E
14 0005 6F              MOV   L,A
15 0006 C9              RET
```

The string of assembler output that is to be loaded into the Pascal array is the third column group or '#7C#A2#67#7D#A3#6F#C9' .

The following program may be compiled and stored as an object file. You may then declare any of the functions or procedures in your application program as external declarations. When you build a /CMD file with the linking loader, link to the file containing the object file containing this output.

```
PROGRAM LOGICALOPERATIONS;
TYPE
CODEARRAY = PACKED ARRAY[1..8] OF CHAR;
BYTE = 0..255;

PROCEDURE CALL$(ADDRESS : INTEGER; VAR A : BYTE;
                VAR BC, DE, HL, IX, IY : INTEGER); EXTERNAL;

FUNCTION AND16(OP1, OP2 : INTEGER): INTEGER;
VAR
   CODE : CODEARRAY;
```

(C) Copyright ALCOR SYSTEMS 1982

```
    A     : BYTE;
    BC, DE, HL, IX, IY : INTEGER;
BEGIN
    DE := OP1;
    HL := OP2;
    CODE := '#7C#A2#67#7D#A3#6F#C9 ';
    CALL$(LOCATION(CODE),A,BC,DE,HL,IX,IY);
    AND16 := HL;
END;  (* AND16 *)

FUNCTION SHIFTLEFT(OPERAND : INTEGER): INTEGER;
(* shift the operand one position left *)
VAR
    CODE : CODEARRAY;
    A     : BYTE;
    BC, DE, HL, IX, IY : INTEGER;
BEGIN
    HL := OPERAND;
    CODE := '#29#C9       ';
    CALL$(LOCATION(CODE),A,BC,DE,HL,IX,IY);
    SHIFTLEFT := HL;
END;  (* SHIFTLEFT *)

(* Other functions can be written using the same patterns *)
(*    The only change required is the actual code loaded   *)
(*    into the code array.  The following strings may be   *)
(*    used to perform other logical operations.            *)

(*    Inclusive Or            ==> '#7A#B4#67#7B#B5#6F#C9 '    *)
(*    Exclusive Or            ==> '#7A#AC#67#7B#AD#6F#C9 '    *)
(*    Bit inversion           ==> '#7C#2F#67#7D#2F#6F#C9 '    *)
(*    Shift right             ==> '#7C#A7#1F#67#7D#1F#6F#C9'  *)
(*    Arithmetic shift right==> '#CB#2C#CB#1D#C9    '         *)
(*    Swap bytes              ==> '#7C#65#6F#C9     '         *)

BEGIN
   (*$NULLBODY*)  (* The nullbody option prevents code generation *)
                  (* for the main program.  The resulting compiler*)
                  (* output can be used as a procedure library.   *)
END.
```

A simple example follows:

```
PROGRAM EXAMPLE;
VAR
   BYTE1,BYTE2,RESULT:INTEGER;
```

(C) Copyright ALCOR SYSTEMS 1982

```
FUNCTION AND16(OP1,OP2:INTEGER) : INTEGER ; EXTERNAL;
BEGIN
   FOR I := 1 TO 5 DO
      BEGIN
      WRITELN('ENTER TWO INTEGER NUMBERS TO BE ANDED');
      READLN(BYTE1,BYTE2);
      RESULT := AND16(BYTE1,BYTE2);
      WRITELN('ANDED RESULT = ',RESULT)'
      END;
END.
```

Random Access Files

Random Access files refers to a file access method where any record may be READ or WRITTEN to in any order. As most Pascal programmers know, Pascal does not define the Random file type.

The following Pascal procedures and functions will allow random access to files on the TRS80. The following Pascal program and routines should be compiled and left in object format. When using random access files, these routines should be declared as external in the main program. Then simply link to the previously compiled random access routines with the linking loader to satisfy any external references.

All source and complied object files for random access files are on the patch disk available to update 1.2 owners to 1.2A. They are NOT in the 1.2A release system. There are also other misc. files on this diskette. (Registered owners send $ 8.00 + shipping to Alcor) Built in random access files will be included in Pascal version 2.0 to be released in late fourth quarter this year. ALL registered Pascal owners will be eligible for this update, regardless of ,licensing date.

The following declarations should be included in the source program.

RANDOM FILE ROUTINES

```
PROCEDURE OPENRAND(VAR F:FILETYPE; RECORDLEN:INTEGER;  PATHNAME:STRING;
                   VAR STATUS:INTEGER); EXTERNAL;
```

The purpose of this routine is to open a random file. The F variable is of any file type. Random file types are fixed in length and should be declared as a FILE OF DATATYPE. A text file is not a particularly useful DATATYPE. The filetype may be any structure such as an ARRAY, RECORD, etc... RECORDLEN must be the size required for the filetype. The SIZE(J) function may be used to determine the RECORDLEN. PATHNAME is the physical

(C) Copyright ALCOR SYSTEMS 1982

filename on disk. You must prompt the user if it is to be changed at runtime. STATUS is a code returned by PASCAL and the operating system. The status code returns the status of an operation on a random file.

```
PROCEDURE READRAND(VAR F:FILETYPE; RECORDNUM:INTEGER;
          VAR DAT:DATATYPE; VAR STATUS:INTEGER); EXTERNAL;
```

This routine is used to READ data from a random file. The RECORDNUM is the record number to be read. DAT is the buffer for the data and is declared to be of the same type the file declared DATATYPE in the OPENRAND routine.

```
PROCEDURE WRITERAND(VAR F:FILETYPE; RECORDNUM:INTEGER;
          VAR DAT:DATATYPE; VAR STATUS:INTEGER); EXTERNAL;
```

This routine is used to WRITE data to a random file. The RECORDNUM is the record number to be written. DAT is the buffer for the data and is declared to be of the same type the file declared DATATYPE in the OPENRAND routine.

```
PROCEDURE CLOSERAND(VAR F:FILETYPE); EXTERNAL;
```

Random files on TRSDOS are required to be closed before program termination. Failure to do so may result in a loss of data.

As with random files on any operating system, there are some peculiarities about random files. For example:

(1) If you WRITE record number 1 and WRITE record number 100, and then read any record from 2 to 99, the returned buffer will contain trash. The data will be whatever was previously on the diskette, probably the contents of an old file. This is because the operating system does not keep that much context. It is up to the user to keep track of unwritten records so they are not READ.

(2) Random file record sizes may be from 1 to 256 only. All blocking is taken care of by the system.

(3) The standard functions EOLN, EOF have no meaning for random files. The status codes as returned by the above routines perform those fuctions where applicable.

(C) Copyright ALCOR SYSTEMS 1982

(4) The procedure OPENRAND is used to open a file for reading and writing to. Opening an empty file and reading is perfectly legal. It is up to the user to check the returned status on all random file operations.

(5) Random file record numbers are defined from 0..32,767 .

(6) As with normal files, if a file is declared LOCALLY within a procedure and opened, (Not passed in as a parameter) once the procedure is exited, Pascal will automatically close the file using the standard CLOSE file routine for non random files and postion the EOF mark in the directory at the last record read or written to. This may not be the correct position as desired by the programmer. An explicit call to CLOSERAND should be used to close the random file and position the EOF. This will always correctly place the EOF mark.

(7) You may declare a file to be:

```
(*WHERE XX IS ANY RECORD LENGTH FROM 1 TO 256*)
TYPE  LINE = ARRAY(.1..XX.) OF CHAR;
VAR   F:FILE OF LINE;
```

Once the file has been opened, you may access it by using the READRAND and WRITERAND external procedures even if the file was not created by Pascal. There is only one procedure for opening random files. (no reset and rewrite) You may read or write to a random file.

Random File Error Codes
Returned By External Procedures

```
 15 - DISK WRITE PROTECTED
 24 - FILE NOT FOUND
 27 - DISK FULL
 28 - END OF FILE
 29 - RECORD NOT FOUND (PAST EOF)
128 - PATH NAME IS NULL OR TOO LONG
129 - RECORD LENGTH IS NOT BETWEEN 1 AND 256
130 - FILE IS ALREADY OPEN
131 - FILE IS NOT OPEN
```

(C) Copyright ALCOR SYSTEMS 1982

The following source Pascal may be compiled and the object appended to the TRSLIB/OBJ file or kept separate. These routines must be linked to by the linking loader if they are used in the main program. (The source and compiled object file is available from Alcor for $ 8.00 + shipping on the 1.2A update diskette)

```
PROGRAM RANDOM;

(*  External procedures to implement random access *)
(*    files in pascal.  Logical record lengths are *)
(*    limited to 256 bytes maximum.                *)

(* Error codes: *)
(*   128 = path name is null or too long *)
(*   129 = record length is not between 1 and 256 *)
(*   130 = File is already open *)
(*   131 = File is not open *)

TYPE
   REC = PACKED ARRAY[1..20] OF INTEGER;
   RANDFILE = FILE OF REC;
   BYTE = 0..255;
   ALPHA = PACKED ARRAY[1..8] OF CHAR;
   FILEBUFFER = PACKED ARRAY[0..255] OF BYTE;
   BUFPTR = @FILEBUFFER;
   DCB = PACKED RECORD   (* disk control block *)
      CASE BOOLEAN OF
         FALSE : (NAME : PACKED ARRAY[1..50] OF CHAR);
         TRUE  : (
            FILL1    : PACKED ARRAY[1..3] OF BYTE;
            BUFFER   : BUFPTR;  (* address of buffer *)
            OFFSET   : BYTE;    (* offset for end of record *)
            DRIVE    : BYTE;    (* disk drive *)
            FILL2    : BYTE;    (* reserved *)
            EOFLOC   : BYTE;    (* end of file offset *)
            LRL      : BYTE;    (* logical record length *)
            NEXT     : INTEGER; (* next record number *)
            LAST     : INTEGER; (* last record in file *)
            FILL3    : PACKED ARRAY[1..18] OF BYTE;
            );
         END;

   FILEDESC = RECORD
      NAME : ALPHA;      (* Pascal name of file *)
      DCBPTR : @DCB;     (* pointer to dcb *)
      BUFR   : BUFPTR;   (* pointer to physical buffer *)
      FILL   : PACKED ARRAY[1..12] OF BYTE;
      RECLEN : INTEGER; (* logical record length *)
      END;
```

(C) Copyright ALCOR SYSTEMS 1982

```
   TWOBYTES = PACKED RECORD
      LB : BYTE;
      UB : BYTE;
      END;

PROCEDURE CALL$(ADDRESS : INTEGER; VAR A : BYTE;
          VAR BC, DE, HL, IX, IY : INTEGER); EXTERNAL;
FUNCTION LEN(S : STRING) : INTEGER; EXTERNAL;

PROCEDURE OPENRAND(VAR F : RANDFILE; RECORDLEN : INTEGER;
            PATHNAME : STRING; VAR STATUS : INTEGER);
CONST
   CALLOPEN = '#CD#24#44#F5#E1#C9';
   CALLINIT = '#CD#20#44#F5#E1#C9';
VAR
   EXECUTE : PACKED ARRAY[1..6] OF CHAR;
   A       : BYTE;
   BC, DE, HL, IX, IY : INTEGER;
BEGIN
   STATUS := 0;
   IF F::FILEDESC.NAME[1] = CHR(0) THEN BEGIN
      (* File is not already open *)
      WITH F::FILEDESC DO BEGIN
         NAME := 'RANDOM  ';  (* Set file name *)
         NEW(DCBPTR);         { Allocate DCB for operating sys }
         NEW(BUFR);           { Allocate a physical buffer }
         RECLEN := RECORDLEN;
         END;
      GETSTR(PATHNAME,F::FILEDESC.DCBPTR@.NAME);  (* Set file name *)
      IF LEN(PATHNAME) = 0 OR LEN(PATHNAME) > 50 THEN
         STATUS := 128
      ELSE F::FILEDESC.DCBPTR@.NAME[LEN(PATHNAME)+1] := '#0D';

      IF RECORDLEN > 256 OR RECORDLEN < 1 THEN
         STATUS := 129
      ELSE IF STATUS = 0 THEN BEGIN
         HL := LOCATION(F::FILEDESC.BUFR@);  (* Physical buffer *)
         DE := LOCATION(F::FILEDESC.DCBPTR@); (* Disk control block *)
         BC::TWOBYTES.UB := RECORDLEN;  (* Record length *)
         EXECUTE := CALLOPEN;
         CALL$(LOCATION(EXECUTE),A,BC,DE,HL,IX,IY);
         (* Callopen is an machine language program   *)
         (* executed from an array.  It calls the system *)
         (* routine and returns the contents of the status *)
         (* register in register L.  This is necessary since *)
         (* NEWDOS80 does not return with the A register and *)
         (* the status register in a consistent state. *)
         IF NOT ODD(HL DIV 64) THEN STATUS := A;
```

(C) Copyright ALCOR SYSTEMS 1982

```
            { Check the Z flag in the returned status register}
          IF STATUS = 24 THEN BEGIN
             (* File does not exist - attempt to create it *)
             STATUS := 0;
             HL := LOCATION(F::FILEDESC.BUFR@);
             DE := LOCATION(F::FILEDESC.DCBPTR@);
             BC::TWOBYTES.UB := RECORDLEN;
             EXECUTE := CALLINIT;
             CALL$(LOCATION(EXECUTE),A,BC,DE,HL,IX,IY);
             IF NOT ODD(HL DIV 64) THEN STATUS := A;
             END;
          END; (* else *)

       IF STATUS <> 0 THEN F::FILEDESC.NAME[1] := CHR(0);
       END  (* If not already open *)
    ELSE STATUS := 130;
 END; (* OPENRAND *)

 PROCEDURE CLOSERAND(VAR F : RANDFILE);
 VAR
    A : BYTE;
    BC, DE, HL, IX, IY : INTEGER;
 BEGIN
    WITH F::FILEDESC DO BEGIN
       IF NAME[1] <> CHR(0) THEN BEGIN (* File is open *)
          (* position to end of file to preserve file size *)
          BC := DCBPTR@.LAST+1;
          DE := LOCATION(DCBPTR@);
          CALL$(#4442,A,BC,DE,HL,IX,IY);
          (* close the file *)
          DE := LOCATION(DCBPTR@);
          CALL$(#4428,A,BC,DE,HL,IX,IY);
          IF DCBPTR <> NIL THEN DISPOSE(DCBPTR);
          IF BUFR <> NIL THEN DISPOSE(BUFR);
          NAME[1] := CHR(0);
          END;
       END; (* with *)
 END; (* CLOSERAND *)

PROCEDURE READRAND(VAR F:RANDFILE; RECORDNUM:INTEGER;
          VAR DAT:REC; VAR STATUS:INTEGER);
CONST
    CALLPOSN = '#CD#42#44#F5#E1#C9';
    CALLREAD=  '#CD#36#44#F5#E1#C9';
 VAR
    EXECUTE:PACKED ARRAY[1..6]OF CHAR;
    A : BYTE;
    BC, DE, HL, IX, IY : INTEGER;
```

(C) Copyright ALCOR SYSTEMS 1982

```
BEGIN
   WITH F::FILEDESC DO BEGIN
      IF NAME[1] = CHR(0) THEN STATUS := 131
      ELSE BEGIN
         BC := RECORDNUM;
         DE := LOCATION(DCBPTR@);
         EXECUTE := CALLPOSN;
         CALL$(LOCATION(EXECUTE),A,BC,DE,HL,IX,IY);
         IF NOT ODD(HL DIV 64) THEN STATUS := A
         ELSE BEGIN
            HL := LOCATION(DAT);
            DE := LOCATION(DCBPTR@);
            EXECUTE := CALLREAD;
            CALL$(LOCATION(EXECUTE),A,BC,DE,HL,IX,IY);
            IF NOT ODD(HL DIV 64) THEN STATUS := A;
            IF RECLEN = 256 THEN DAT::FILEBUFFER := BUFR@;
            END;
         END;
      END; (* with *)
END; (* READRAND *)


PROCEDURE WRITERAND(VAR F: RANDFILE; RECORDNUM : INTEGER;
          VAR DAT : REC; VAR STATUS : INTEGER);
CONST
   CALLPOSN = '#CD#42#44#F5#E1#C9';
   CALLWRITE = '#CD#39#44#F5#E1#C9';
VAR
   EXECUTE : PACKED ARRAY[1..6] OF CHAR;
   A : BYTE;
   BC, DE, HL, IX, IY : INTEGER;
BEGIN
   WITH F::FILEDESC DO BEGIN
      IF NAME[1] = CHR(0) THEN STATUS := 131
      ELSE BEGIN
         BC := RECORDNUM;
         DE := LOCATION(DCBPTR@);
         EXECUTE := CALLPOSN;
         CALL$(LOCATION(EXECUTE),A,BC,DE,HL,IX,IY);
      *  IF NOT ODD(HL DIV 64) THEN STATUS := A
         ELSE BEGIN
            HL := LOCATION(DAT);
            IF RECLEN = 256 THEN BUFR@ := DAT::FILEBUFFER;
            DE := LOCATION(DCBPTR@);
            EXECUTE := CALLWRITE;
            CALL$(LOCATION(EXECUTE),A,BC,DE,HL,IX,IY);
            IF NOT ODD(HL DIV 64) THEN STATUS := A;
            END;
```

(C) Copyright ALCOR SYSTEMS 1982

* IF A=28 OR A=29 THEN HL := 64;

```
        END;
      END; (* with *)
END; (* WRITERAND *)
BEGIN (*$NULLBODY*) END.
```

The following is one of our random file test programs. It should adequately illustrate the use of random files in an application program.

```
PROGRAM RANDTEST;
TYPE R1 = PACKED ARRAY[1..32] OF CHAR;
     R2 = PACKED ARRAY[1..256] OF CHAR;
     RF1 = FILE OF R1; RF2 = FILE OF R2;
VAR  FILE1 : RF1; FILE2 : RF2;
     REC1 : R1; REC2 : R2; REC1A: R1; REC2A: R2;
     I,STATUS : INTEGER;

PROCEDURE OPENRAND(VAR F : RF1; RECORDLEN : INTEGER;
      PATHNAME : STRING; VAR STATUS : INTEGER); EXTERNAL;
PROCEDURE CLOSERAND(VAR F : RF1); EXTERNAL;
PROCEDURE READRAND(VAR F : RF1; RECORDNUM : INTEGER;
       VAR DAT : R1; VAR STATUS : INTEGER); EXTERNAL;
PROCEDURE WRITERAND(VAR F : RF1; RECORDNUM : INTEGER;
       VAR DAT : R1; VAR STATUS : INTEGER); EXTERNAL;

PROCEDURE CHECK(TEST : INTEGER; STATUS : INTEGER);
BEGIN
   IF STATUS <> 0 THEN BEGIN
      WRITE(OUTPUT,'ERROR AT TEST: ',TEST:3);
      WRITELN(OUTPUT,' STATUS= ',STATUS:3);
      END;
END; (* CHECK *)

BEGIN
   REC1 := ' ABCDEFGHIJKLMNOPQRSTUVWXYZ01234';
   OPENRAND(FILE1,SIZE(R1),BLDSTR('RAND1/DAT:2'),STATUS);
   CHECK(1,STATUS);
   FOR I := 50 TO 260 DO BEGIN
      REC1[1] := CHR((I MOD 26) + ORD('A'));
      WRITERAND(FILE1,I,REC1,STATUS);
      CHECK(2,STATUS);
      END;
   CLOSERAND(FILE1);
   OPENRAND(FILE1,SIZE(R1),BLDSTR('RAND1/DAT:2'),STATUS);
   CHECK(3,STATUS);
   FOR I := 49 DOWNTO 0 DO BEGIN
      REC1[1] := CHR((I MOD 26) + ORD('A'));
      WRITERAND(FILE1,I,REC1,STATUS);
      CHECK(4,STATUS);
      END;
   CLOSERAND(FILE1);
```

(C) Copyright ALCOR SYSTEMS 1982

```
    OPENRAND(FILE,SIZE(R1),BLDSTR('RAND1/DAT:2'),STATUS);
    CHECK(5,STATUS);
    REC1A := REC1;
    FOR I := 0 TO 260 DO BEGIN
       REC1A[1] := CHR((I MOD 26) + ORD('A'));
       READRAND(FILE1,I,REC1,STATUS);
       CHECK(2,STATUS);
       IF REC1 <> REC1A THEN BEGIN
          WRITELN(OUTPUT,'READ VERIFY ERROR AT ',I);
          WRITELN(OUTPUT,REC1);
          WRITELN(OUTPUT,REC2);
          END;
       END;
    CLOSERAND(FILE1);
END.
```

Fixing Physical Filenames Into Pascal Programs

Alcor Pascal normally prompts the user for physical (names of files on diskette) filenames to be linked to Pascal logical (variable filenames in Pascal) filenames when a RESET or REWRITE statement is executed in a program. Pascal stops execution and displays the Pascal logical filename on the screen followed by the "=" symbol, and waits for a user reply terminated by the ENTER key. This is to provide a simple mechanism for filename associations. All files in Alcor Pascal are treated the same. The obvious advantage is that the program does not have to be recompiled to change the physical filename associations.

Turning Off The INPUT And OUTPUT Prompts

To meet the Jensen and Wirth definition of standard Pascal, INPUT and OUTPUT are predeclared. When the program starts execution, a RESET on INPUT and REWRITE on OUTPUT is performed resulting in prompts to the screen for physical filenames. This predeclaration is required so programs written in standard Pascal may be compiled and executed on Alcor Pascal without any modification. However there are times when a user does not want this predeclared feature of Pascal. There is a compiler option that will prevent INPUT and OUTPUT from being predeclared. The first statement of the program should contain: (*$NO INOUT*) as a comment. This means that you may no longer use INPUT and OUTPUT as predeclared files. However you may declare them just like any other file in the VAR section of the program, and perform RESET and REWRITE statements on them as desired. The only side effect is that INPUT and OUTPUT must be included as an argument in READ and WRITE statements.

(C) Copyright ALCOR SYSTEMS 1982

To prevent a prompt to the screen from occurring when a file is used in a REWRITE or RESET statement requires that a procedure call to SET$ACNM be performed sometime prior to the RESET or REWRITE statement. The purpose of the SET$ACNM call is to perform the usual physical to logical filename association. The reason for the procedure call instead of a compiler option is to allow mixing of SET$ACNM associations with normal file declarations that prompt upon the RESET or REWRITE. A word of warning. If the parameters for SET$ACNM are not correct, (No such physical filename, etc) the SET$ACNM will not be performed and a prompt for the physical filename will still occur.

The Following Is An Extract From The 1.2A Manuals

```
TYPE
   FILENM = PACKED ARRAY[1..XX] OF CHAR;
   ALPHA  = PACKED ARRAY[1..8] OF CHAR;
(Where XX is any length long enough for the filename)

PROCEDURE SET$ACNM(VAR F : TEXT; VAR file name : FILENM;
  NAMELENGTH : INTEGER; VAR FILEID : ALPHA); EXTERNAL;
```

SET$ACNM is used to set the name of the physical file or device to be associated with a pascal file. It allows a program to compute file names internally. For example, a database program may know the name of the file containing the database. This procedure allows the program to specify the file name rather than requesting it from the keyboard.

The parameter F can be a file of any type. The external declaration of SET$ACNM that is included in the source program must specify a type for F that matches the actual file type to be used.

File name is a string containing the text of the file name. This string must be compatible with the operating system syntax for file names. The physical devices: lineprinter (:L), crt (:C) and dummy (:D) may also be used. NAMELENGTH is an integer that specifies the length of the file name.

FILEID is an 8 character string that is used to identify the Pascal name for the file, such as INPUT or OUTPUT.

If SET$ACNM is called prior to a RESET or REWRITE on a file, then Pascal will not prompt the CRT for the file name. All subsequent RESET or REWRITES will not cause a prompt unless a CLOSE(file name) is performed on the file. The file name association will remain as previously defined by SET$ACNM.

(C) Copyright ALCOR SYSTEMS 1982

(Example program segment)

```
TYPE
FILENAME = PACKED ARRAY [1..15]OF CHAR;
ALPHA    = PACKED ARRAY [1..8]OF CHAR;
VAR  FNAME :FILENAME;
     FILEID:ALPHA;
     F     :TEXT;
PROCEDURE SET$ACNM(VAR F:TEXT; VAR FNAME:FILENAME; LEN:INTEGER;
                   VAR FILEID:ALPHA); EXTERNAL;
BEGIN
   (* THIS ASSIGNMENT STATEMENT REQUIRES THE NAME TO BE LEFT *)
   (* JUSTIFIED, AND BLANK PADDED TO THE CORRECT ARRAY LENGTH *)
   FNAME:='DATA/TXT:0      ';
   FILEID:='F       ';
   SET$ACNM(F,FNAME,10,FILEID);
   RESET(F);
   READ(F,CH);
   (* AND ETC..... *)
```

A few users have complained that SET$ACNM isn't easy enough to use. The following procedure will make the calling sequence for SET$ACNM simpler. It may be separately compiled and then linked to programs with the linking loader.

```
PROGRAM SETACCESS;
{ The following procedure makes the SET$ACNM procedure  }
{   easier to use.  It may be separately compiled and   }
{   linked to programs that use it                      }
PROCEDURE SETACNM(VAR F : TEXT; NAME : STRING);
{ Arguments are:                                        }
{     F is a file.  The external declaration used       }
{       for this procedure may use any type of file     }
{     NAME is the physical name of the file.  The       }
{       string is disposed in this procedure.  This     }
{       makes it convenient to use a string constant    }
{       as an argument with no loss of memory.          }
TYPE
   ALPHA = PACKED ARRAY[1..8] OF CHAR;
   FILENAME = PACKED ARRAY[1..50] OF CHAR;
VAR
   NAMELENGTH : INTEGER;
   FILEID     : ALPHA;
   FNAME      : FILENAME;
   FUNCTION LEN(S : STRING) : INTEGER; EXTERNAL;
   PROCEDURE SET$ACNM(VAR F : TEXT; VAR FNAME : FILENAME;
             LEN : INTEGER; VAR FILEID : ALPHA); EXTERNAL;
BEGIN
   NAMELENGTH:=LEN(NAME); {determine length of filename}
   IF NAMELENGTH > 50 THEN NAMELENGTH := 50; {stay in array bounds}
   GETSTR(NAME,FNAME);
   FILEID := 'FILE    ';
   SET$ACNM(F,FNAME,NAMELENGTH,FILEID);
```

(C) Copyright ALCOR SYSTEMS 1982

```
   DISPOSE(NAME);
END;
BEGIN
   (*$NULLBODY *)    {No main program}
END.
```

The next example shows a sample of the use of the above procedure. s you can see, a call to SETACNM requires only one line.

```
PROGRAM EXAMPLE;
{  a sample of the SETACNM procedure for simple }
{  specification of fixed file names from a program }
TYPE
   DATARECORD = RECORD
      ID : INTEGER;
      { other stuff }
      END;
   DATAFILE = FILE OF DATARECORD;
VAR
   DATABASE : DATAFILE;
PROCEDURE SETACNM(VAR F : DATAFILE; NAME : STRING); EXTERNAL;
BEGIN
   SETACNM(DATABASE,BLDSTR('DATABASE/DAT:1'));
   RESET(DATABASE);
   { do something useful here }
END.
```

Program Chaining In Pascal

Programs may be chained together in Pascal. This is accomplished by making the standard TRSDOS operating system call COMDOS. COMDOS requires a textstring argument that is the text of any legal TRSDOS command or /CMD file name. A command string may be built in Pascal and passed to COMDOS. The ability to perform operating system calls to execute tasks is a very important feature of Alcor Pascal. There are many special language extensions for the serious programmer that allow convenient data manipulation for systems programming. Such operators as TYPE TRANSFER are invaluable. The only caution is that many system calls may wipe out your Pascal program. Care must be taken to prevent a system program from loading on top of your Pascal program. Needless to say, results can be unpredictable. Program chaining can be accomplished by building a string with the appropriate command line for invoking the desired Pascal or Basic program. If the called Pascal program includes the previously detailed routines for getting command line parameters from the system, command line parameters may be passed to the called or chained to program.

The following program illustrates Pascal program chaining with the ability to pass information to the called program on the command line. Once invoked, the program will parse the command

(C) Copyright ALCOR SYSTEMS 1982

line and look for the parameter after the original filename field. The parameter field will have its ASCII value decremented, then used as a new command line parameter for another program chaining call. The new call will be to the same program, which will cause a new copy of the program to be invoked. This chaining process will continue until the value of the command line parameter is the ASCII character "A".

This program simply builds a command line string and places it at the address that TRSDOS uses for command line storage. The COMDOS routine is passed this address for the TRSDOS command line string. In this way, COMDOS executes the command line just as though it was entered from the keyboard, leaving the command line information at the usual memory address for use by the program.

```
(*$NO INOUT*)
PROGRAM TEST;
CONST
       (*THIS IS THE LOCATION OF THE COMMAND LINE STORAGE*)
       (*BUFFER FOR THE PARTICULAR OPERATING SYSTEM AND  *)
       (*COMPUTER COMBINATION.  TRSDOS 1.3/MODEL III    *)
   TRSCMDBUF = #4225;
       (*TRSDOS 1.3  EXECUTE TRSDOS CMD VECTOR ADDRESS  *)
   COMDOS    = #4299;
TYPE  CMDLINE = ARRAY(.1..64.)OF CHAR;
VAR   LINE          : CMDLINE;
      STRLINE       :STRING;
      BLKLOC        :INTEGER;
      CH            :CHAR;
   PROCEDURE GETCMD(VAR CMD:CMDLINE);
       (*THE PURPOSE OF THIS PROCEDURE IS TO RETREIVE THE COMMAND*)
       (*LINE FROM THE TRSDOS STORAGE LOCATION *)
   TYPE
      POINTER = @CMDLINE;
   VAR
      BUFPTR:POINTER;
   BEGIN
           (*LOAD THE POINTER ADDRESS WITH THE COMMAND*)
           (*BUFFER LOCATION*)
      BUFPTR::INTEGER:=TRSCMDBUF;
           (*LOAD THE RETURN BUFFER WITH THE COMMAND LINE*)
      CMD:=BUFPTR@;
   END;
   PROCEDURE EXECUTECMD(CMD:STRING);
   TYPE PTRCMDLINE = @CMDLINE;
```

Copyright ALCOR SYSTEMS 1982

```
VAR  TRSCMDLOC:PTRCMDLINE;
               I:INTEGER;

     (*THIS PROCEDURE IS IN THE EXTERNAL TRSLIB AND MAY*)
     (*BE LINKED TO WITH THE LINKING LOADER*)
PROCEDURE USER(ADDRESS:INTEGER; VAR PARMPTR:INTEGER);
                EXTERNAL;

BEGIN
   TRSCMDLOC::INTEGER:=TRSCMDBUFF;
       (*LOAD THE CONTENTS OF THE DESIRED COMMAND INTO THE*)
       (*NORMAL TRSDOS COMMAND BUFFER*)
   GETSTR(CMD,TRSCMDLOC@);
   I:=64;
       (*ADD A TRAILING CARRIAGE RETURN AS REQUIRED BY TRSDOS*)
   WHILE (TRSCMDLOC@(.I.)=' ')DO I:=I-1;
   IF (I>0) THEN TRSCMDLOC@(.I+1.):=CHR(13);
       (*CALL COMDOS TO EXECUTE COMMAND*)
   USER(COMDOS,TRSCMDLOC::INTEGER);
END;

(*ALL OF THESE STRING ROUTINES MAY BE LINKED TO IN THE*)
(*STRING LIBRARY*)

FUNCTION LEFT$(S : STRING; POSITION : INTEGER) : STRING; EXTERNAL;
FUNCTION CHARACTER(S : STRING; POSITION : INTEGER) : CHAR; EXTERNAL;
FUNCTION CONC(S1, S2 : STRING) : STRING; EXTERNAL;
FUNCTION CPYSTR(S : STRING) : STRING; EXTERNAL;
FUNCTION FIND(SUBSTRING, S : STRING) : INTEGER; EXTERNAL;



BEGIN
          (*GET THE COMMAND LINE FROM THE TRSDOS BUFFER*)
   GETCMD(LINE);
          (*BUILD A STRING FROM THE COMMAND BUFFER*)
   STRLINE:=BLDSTR(LINE);
          (*FIND THE BLANK THAT SEPARATES THE COMMAND NAME FROM *)
          (*THE COMMAND PARAMETER*)
   BLKLOC:=FIND(BLDSTR(' '),STRLINE);
          (*GET THE COMMAND PARAMETER CHARACTER*)
   CH:=CHARACTER(STRLINE,BLKLOC+1);
   MESSAGE(CH);
   IF(CH>'A')THEN
      BEGIN
            (*STRIP OUT THE COMMAND NAME FROM THE COMMAND LINE*)
            (*AND PUT A COMMAND PARAMETER THAT IS ITS PREDECESSOR*)
            (*FROM THE ASCII CHARACTER SET*)
```

(C) Copyright ALCOR SYSTEMS 1982

```
      STRLINE:=CONC(LEFT$(STRLINE,BLKLOC),BLDSTR(PRED(CH)));
              (*EXECUTE THIS PROGRAM WITH NEW PARAMETER FIELD*)
              (*ON THE COMMAND LINE*)
      EXECUTECMD(STRLINE);
      END;
END.
```

The procedures EXECUTECMD and GETCMD may be compiled separately with the use of the NULLBODY feature and then linked to as needed by Pascal programs. The TRSDOS command buffer address and COMDOS vector must be changed in the CONST sections for the proper operating system and computer combinations.
They are set up for separate compilation as follows:

```
PROGRAM DUMY;
CONST
       (*THIS IS THE LOCATION OF THE COMMAND LINE STORAGE*)
       (*BUFFER FOR THE PARTICULAR OPERATING SYSTEM AND  *)
       (*COMPUTER COMBINATION.  TRSDOS 1.3/MODEL III     *)
TRSCMDBUF = #4225;
       (*TRSDOS 1.3  EXECUTE TRSDOS CMD VECTOR ADDRESS   *)
COMDOS    = #4299;
TYPE CMDLINE = ARRAY(.1..64.)OF CHAR;
   PROCEDURE GETCMD(VAR CMD:CMDLINE);
       (*THE PURPOSE OF THIS ROUTINE IS TO RETREIVE*)
       (*A COMMAND LINE FROM THE TRSDOS STORAGE LOCATION *)
   TYPE
      POINTER = @CMDLINE;
   VAR
      BUFPTR:POINTER;
   BEGIN
            (*LOAD THE POINTER ADDRESS WITH THE COMMAND*)
            (*BUFFER LOCATION*)
      BUFPTR::INTEGER:=TRSCMDBUF;
            (*LOAD THE RETURN BUFFER WITH THE COMMAND LINE*)
      CMD:=BUFPTR@;
   END;

   PROCEDURE EXECUTECMD(CMD:STRING);
   TYPE
        PTRCMDLINE = @CMDLINE;
   VAR  TRSCMDLOC:PTRCMDLINE;
                I:INTEGER;

       (*THIS PROCEDURE IS IN THE EXTERNAL TRSLIB AND MAY*)
       (*LINKED TO WITH THE LINKING LOADER*)
   PROCEDURE USER(ADDRESS:INTEGER; VAR PARMPTR:INTEGER);
```

(C) Copyright ALCOR SYSTEMS 1982

```
                    EXTERNAL;

    BEGIN
       TRSCMDLOC::INTEGER:=TRSCMDBUFF;
            (*LOAD THE CONTENTS OF THE DESIRED COMMAND INTO THE*)
            (*NORMAL TRSDOS COMMAND BUFFER*)
       GETSTR(CMD,TRSCMDLOC@);
       I:=64;
            (*ADD A TRAILING CARRIAGE RETURN AS REQUIRED BY*)
            (*TRSDOS*)
       WHILE (TRSCMDLOC@(.I.)=' ')DO I:=I-1;
       IF (I>0) THEN TRSCMDLOC@(.I+1.):=CHR(13);
            (*CALL COMDOS TO EXECUTE COMMAND*)
       USER(COMDOS,TRSCMDLOC::INTEGER);
    END;

  (*ALL OF THESE STRING ROUTINES MAY BE LINKED TO IN THE*)
  (*STRING LIBRARY*)

FUNCTION LEFT$(S : STRING; POSITION : INTEGER) : STRING;
    EXTERNAL;
FUNCTION CHARACTER(S : STRING; POSITION : INTEGER) : CHAR;
    EXTERNAL;
FUNCTION CONC(S1, S2 : STRING) : STRING; EXTERNAL;
FUNCTION CPYSTR(S : STRING) : STRING; EXTERNAL;
FUNCTION FIND(SUBSTRING, S : STRING) : INTEGER; EXTERNAL;


BEGIN
(*$NULLBODY*)
END.
```

Miscellaneous Patches To Change Pascal Characteristics

TRS80 MODEL I and MODEL III

```
; BLAISE TEXT EDITOR PATCH     (MODEL I AND III)
; OPTIONAL PATCH TO CHANGE THE DEFINITION OF THE CLEAR
;     KEY TO "/" .    THE SLASH CHARACTER IS GENERATED BY
;     "//" OR BY "/7"
;
F, ED/CMD, ALCOR1
P,459A,05AC,0001,1F,2F
P,45B2,058B,0001,1F,2F
W,F4C9
E
```

(C) Copyright ALCOR SYSTEMS 1982

```
; BLAISE TEXT EDITOR PATCH        (MODEL I AND III)
; OPTIONAL PATCH TO CHANGE THE CHARACTER GENERATED BY
;    "CLEAR 7" FROM "/" TO "_".
F, ED/CMD, ALCOR1
P,144F,05B0,0001,2F,5F
W,FA50
E


;  OPTIONAL PATCH TO SUPPRESS THE STACK AND
;     HEAP USED MESSAGE IN THE LINKING LOADER
;
F, LINKLOAD/CMD, ALCOR2          (MODEL I AND III)
P,2315,07DB,0003,21,C3,A9,8C,75,75
W,F825
E
```

TRS80 MODEL I ONLY

```
;
; OPTIONAL PATCH TO CHANGE THE DEFINITION OF THE CLEAR
; KEY TO THE SHIFT DOWN ARROW KEY (MODEL I ONLY)
;
F, ED/CMD, ALCOR1
P,459A,05A1,0001,1F,1A
P,45B2,0580,0001,1F,1A
W,F4DF
E
```

(C) Copyright ALCOR SYSTEMS 1982

Using KSM Filters with LDOS

Some users of LDOS 5.1.2 have reported having trouble using KSM filters with the Blaise text editor. The problem is related to the use of the "SHIFT CLEAR" key as a prefix to editor commands. The optional patch which changes the definition of the CLEAR key to the "/" key should fix the problem. The use of the CLEAR key does not cause any problems if you are using LDOS 5.1.3.

The use of KSM filters with LDOS can make the Blaise text editor much easier to use as well as provide macro command capability. A KSM filter may be created which maps the 26 alphabetic keys to editor commands. Each alphabetic key may correspond to a single editor command or to a sequence of editor commands. Using a KSM filter, the commands which are mapped to alphabetic keys are invoked by holding down the CLEAR key while pressing the appropriate alphabetic key. The commands are also accessible in the normal manner using "SHIFT CLEAR". In the normal accessing of commands, the "SHIFT CLEAR" keys should not be held down. They should be pressed and then released before pressing the key associated with a particular command.

KSM filters may be created using the LDOS BUILD command or a Pascal program can be written to generate the filter. Once created, the FILTER command is used to load the filter and the SYSTEM command is used to save the new configuration on your LDOS system disk.

Example:

```
BUILD ED/KSM (HEX)
define keys A --> Z as prompted
FILTER *KI TO KSM/FLT USING ED/KSM
SYSTEM (SYSGEN)
```

The following Pascal program is an alternative to using the LDOS BUILD command. It generates a file which when loaded as a KSM filter will provide new ways of invoking editor commands via the 26 alphabetic keys. This is an example of one way to configure the keyboard. The program may be modified to configure the keyboard in other ways if desired.

(C) Copyright ALCOR SYSTEMS 1982

```
PROGRAM GENERATE_LDOS_KSM_FILTER;
BEGIN
{THIS PROGRAMS CREATES A 26 LINE FILE WHICH MAY BE USED AS
      A KSM FILTER TO ALTER THE WAY COMMANDS ARE INVOKED IN
      THE BLAISE TEXT EDITOR.  A COMMAND IS INVOKED BY
      HOLDING DOWN THE CLEAR KEY WHILE PRESSING ONE OF THE
      26 ALPHABETIC KEYS.  THE COMMANDS MAY ALSO STILL BE
      INVOKED IN THE USUAL WAY USING <SHIFT> <CLEAR> WHICH
      GENERATES #1F.                                      }

{A}   WRITELN('#1FA');                    {TOGGLE AUTO INDENT}
{B}   WRITELN('#1FB');                    {BACK TAB}
{C}   WRITELN('#1FC');                    {COMMAND MODE}
{D}   WRITELN('#1FD');                    {DUPLICATE LINE}
{E}   WRITELN('#1FCEXIT;;');              {EXIT AND SAVE}
{F}   WRITELN('#1FF');                    {FIND NEXT}
{G}   WRITELN('#1FG');                    {MERGE}
{H}   WRITELN('#1FH');                    {HOME}
{I}   WRITELN('#1FI');                    {INSERT MODE}
{J}   WRITELN('#1FCA 66;');               {APPEND 66 LINES}
{K}   WRITELN('#1FK');                    {CLEAR TO END OF LINE}
{L}   WRITELN('#60#60#60#60#60#60#60#60#60',{INSERT 15 BLANK LINES}
            '#60#60#60#60#60#60#60#60');   {SHIFT @ GENERATES #60}
{M}   WRITELN('#1F#09');                  {CURSOR TO END OF LINE}
{N}   WRITELN('#1F#08');                  {CURSOR TO BEG OF LINE}
{O}   WRITELN('#1FO');                    {OPEN LINE AT CURSOR}
{P}   WRITELN('#1FCW 66;#1FCA 66;');      {NEXT PAGE}
{Q}   WRITELN('#1FCHS 1;');               {SCROLL LEFT}
{R}   WRITELN('#1FR');                    {REPLACE NEXT}
{S}   WRITELN('#1FS');                    {SET TAB}
{T}   WRITELN('#1FT');                    {TAB}
{U}   WRITELN('#1FCSHOWLINE 1;');         {CURSOR TO TOP OF TEXT}
{V}   WRITELN('#1FCSHOWLINE 9999;');      {CURSOR TO BOT OF TEXT}
{W}   WRITELN('#1FCHS 17;');              {SCROLL RIGHT}
{X}   WRITELN('#1F#0A');                  {SCROLL DOWN}
{Y}   WRITELN('#1FY');                    {CLEAR TAB}
{Z}   WRITELN('#1F#5B');                  {SCROLL UP}
END.
```

(C) Copyright ALCOR SYSTEMS

ALCOR SYSTEMS
800 W. Garland Avenue
Suite 100
Garland, Texas 75040